С.Ю. ГАВРИЛЕНКО

ФОРМАЛЬНІ МОВИ, ГРАМАТИКИ ТА АВТОМАТИ

Харків 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

С.Ю. ГАВРИЛЕНКО

ФОРМАЛЬНІ МОВИ, ГРАМАТИКИ ТА АВТОМАТИ

Навчальний посібник
для студентів вищих навчальних закладів, які навчаються за спеціальністю
123 «Комп'ютерна інженерія»

Рекомендовано Вченою Радою Національного технічного університету «Харківський політехнічний інститут», протокол № 2, від 26 лютого 2021 року

Харків НТУ "ХПІ

2021

ББК 32.81

Г34
УДК 681.3.06

Рецензенти:
Можаєв О.О., д-р. техн. наук, професор, Харківський національний університет внутрішніх справ Міністерства внутрішніх справ України.
Коваленко А.А., д-р. техн. наук, професор, Харківський національний університет радіоелектроніки

Г34 Автори: С. Ю. Гавриленко

Г 34 Формальні мови, граматики та автомати: Навчальний посібник/ Гавриленко С.Ю. – Харків: НТУ « ХПІ », 2021. – 133 с. – на укр. мові.

Наведено класичні моделі, методи, алгоритми теорії формальних мов та граматик. Розглянуто *LL*(1)–граматики, *LR*(*k*) – граматики, граматики простого та операторного передування. Наведено моделі скінчених автоматів, а саме: автомати перетворювачі та автомати розпізнавачі. Розглянуто моделі низхідних та висхідних магазинних розпізнавачів. Теоретичний матеріал ілюстровано численними прикладами. Для самостійного виконання надано багато вправ.

Навчальний посібник орієнтовано для студентів, як навчаються за спеціальністю 123 «Комп'ютерна інженерія» та може бути корисним для студентів, які навчаються за спеціальністю 122 «Комп'ютерні науки».

Іл. 7. Табл. 33. Бібліогр. 16 назв

ББК 32.81

ЗМІСТ

ВСТУП

Теорія формальних мов, граматик та автоматів активно розвивається з 1950-х років. Основоположником даної теорії є американський вчений, професор Н. Хомський. Головним завданням теорії є розроблення математичного апарату для опису та аналізу природних і штучних мов. У наш час теорія формальних мов, граматик та автоматів є потужним засобом математичного моделювання, який активно застосовується, зокрема, в синтаксичному аналізі, перекладі, та у розробленні трансляторів [1-3].

У першому розділі підручника введено поняття формальної мови та формальної (погоджувальної) граматики; визначено основні операції над формальними мовами та описано ієрархію Хомського. Описано правила побудови граматик. Наведено приклади опису правил, що описують основні конструкції мов програмування. Введено поняття приведених граматик, описано алгоритми визначення непродуктивних та недосяжних символів, виключення ліворекурсивних та ланцюгових правил.

У другому розділі дано визначення скінченних автоматів. Розглянуто типи скінченних автоматів: автомат перетворювач та автомат розпізнавач (магазинний автомат) та способи їх задання. Наведено приклади синтезу скінченних автоматів .

У третьому розділі розглянуто побудову детермінованого спадного розпізнавача. Визначено правила побудови функцій ПЕРШ(μ), СЛІД(А) та множини ВИБІР(μ) та їх використання для побудови моделі магазинного автомата. Розглянуто побудову розпізнавача для слабкорозділених та $LL(1)$ – граматик. Наведено приклади побудови спадних розпізнавачів.

У четвертому розділі розглянуто побудову детермінованого висхідного розпізнавача. Визначено правила побудови таблиці переходів автомату та управляючої таблиці. Розглянуто побудову розпізнавача для $LR(0)$ та $LR(1)$ – граматики, наведено приклади.

У п’ятому розділі розглянуто граматики простого та операторного передування. Наведено алгоритм «зсув-згортання» для побудови висхідного розпізнавача, розглянуто приклади.

Наприкінці кожного розділу надано запитання та завдання для самостійної роботи.

1 ФОРМАЛЬНІ МОВИ І ГРАМАТИКИ

1.1 Визначення формальних мов і граматик

Математичні моделі, що використовують подання текстів у вигляді послідовності символів, називають формальними мовами і граматиками

Визначення. Кінцева множина символів, неподільних у даному розгляді, називається словником чи алфавітом, а символи, що входять у множину, – буквами алфавіту.

Наприклад, алфавіт A = {2, b, c, +, !} містить 5 букв, а алфавіт B = {00, 01, 10, 11} містить 4 букви, кожна з яких складається з двох символів.

Визначення. Послідовність букв алфавіту називається словом чи ланцюжком у цьому алфавіті. Число букв, що входять у слово, називається його довжиною.

Наприклад, слово а= 2bc в алфавіті A має довжину $l(a)$ = 3, а слово b = 0000110010 в алфавіті B має довжину l (b) = 5.

Якщо заданий алфавіт A, то позначимо A^* множину всяких ланцюжків, що можуть бути побудовані з букв алфавіту A. При цьому передбачається, що порожній ланцюжок, який позначимо знаком $, також входить у множину A^*.

Визначення. Формальною граматикою Г, що породжує множину символів, називається наступна сукупність чотирьох об'єктів:

$$\Gamma = \{ V_T, I, V_A, R \},$$

де V_T – термінальний алфавіт (словник); букви цього алфавіту називаються термінальними символами; з них будуються ланцюжки породжувані граматикою; V_A – нетермінальний, допоміжний алфавіт (словник); букви цього алфавіту використовуються при побудові ланцюжків; вони можуть входити в проміжні ланцюжки, але не повинні входити в результат породження; I – початковий символ граматики $I \in V_A$; R – множина правил

виведення вигляду, що $\alpha \rightarrow \beta$, де α і β – ланцюжки, побудовані з букв алфавіту $V_т \cup V_A$, що називають повним алфавітом (словником) граматики Г.

До множини правил граматики можуть також входити правила з порожньою правою частиною вигляду $E \rightarrow$. Щоб уникнути невизначеності через відсутність символу в правій частині правила, домовимося використовувати символ порожнього ланцюжка, записуючи таке правило у вигляді $E \rightarrow \$$.

Щоб встановити правила побудови ланцюжків, породжуваних граматикою, введемо наступні поняття.

Визначення. Нехай $\tau \rightarrow \gamma$ – правило граматики Г і $\alpha \rightarrow \chi' \tau \chi''$ – ланцюжок символів, причому $\chi', \chi'' \in (V_т \cup V_A)^*$. Тоді ланцюжок $\beta \rightarrow \chi' \gamma \chi''$ може бути отриманий з ланцюжка α шляхом застосування правила $\tau \rightarrow \gamma$. У цьому випадку говорять, що ланцюжок β безпосередньо виведений з ланцюжка α і позначають $\alpha \Rightarrow \beta$.

Визначення. Якщо задана сукупність ланцюжків $\Omega = (\varpi_0, \varpi_1,...,\varpi_n)$, за якої існує послідовність безпосередніх виводів:

$$\varpi_0 \Rightarrow \varpi_1, \varpi_1 \Rightarrow \varpi_2, \ldots, \varpi_{n-1} \Rightarrow \varpi_n$$

то таку послідовність називають виведення ϖ_n з ϖ_0 у граматиці Г і позначають

$$\varpi_0 \Rightarrow^* \varpi_n.$$

Визначення. Множина кінцевих ланцюжків термінального алфавіту $V_т$ граматики Г, виведених з початкового символу *I*, називається мовою, породжуваною граматикою Г, і позначається $L(\Gamma)$.

$$L(\Gamma) = \{\varpi \in V_т^* \mid <I> \Rightarrow^* \varpi \}.$$

1.2 Приклади, що ілюструють первинні поняття

Розглянемо кілька прикладів, що ілюструють введені поняття.

1. Задана граматика $\Gamma_{1.0}$ і потрібно визначити мову, породжувану цією граматикою:

$$\Gamma_{1.0}: \quad V_т = \{a, b, c\},\ V_A = \{I>\},\ R = \{I> \to abc\}.$$

Схема граматики містить одне правило, тому $\Gamma_{1.0}$ породжує мову з одного слова

$$L(\Gamma_{1.0}) = \{abc\}.$$

2. Задана граматика $\Gamma_{1.1}$. Потрібно визначити мову, породжувану цією граматикою:

$$\Gamma_{1.1}: \ V_т = \{a, b, c, d\},\ V_A = \{I, B, C\}$$
$$R = \{\ I \to aB$$
$$B \to Cd$$
$$B \to dc$$
$$C \to \$\}.$$

Побудуємо всі виводи в цій граматиці:

$I \Rightarrow aB \Rightarrow aCd \Rightarrow ad,\ I \Rightarrow aB \Rightarrow adc.$

Отже, мова $L(\Gamma_{1.1}) = \{adc, ad\}$.

3. Задана граматика $\Gamma_{1.2}$. Потрібно визначити мову, породжувану цією граматикою:

$$\Gamma_{1.2}: V_A = \{I, A\},\ V_т = \{0, 1\},$$
$$R = \{I \to 0A1$$
$$0A \to 00A1$$
$$A \to \$\}.$$

Розглянемо кілька виводів за допомогою правил граматики $\Gamma_{1.2.}$ Застосовуючи перше і третє правила, одержуємо:

$$I \Rightarrow 0A1 \Rightarrow 01.$$

Застосовуючи два рази перше правило і третє, маємо

$$I \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 0011.$$

У загальному випадку, застосовуючи K разів перше правило, одержимо в результаті ланцюжок, що містить K нулів і K одиниць.

Отже, мова, породжувана граматикою $\Gamma_{1.2}$, містить всякі ланцюжки, в яких число нулів дорівнює числу одиниць.

4. Задана граматика $\Gamma_{1.3}$. Потрібно визначити мову, породжувану цією граматикою .

$$\Gamma_{1.3}: V_т = \{a, b\}, V_A = \{I, A\},$$

$$R = \{ I \to aA$$

$$A \to bA\}.$$

Спроба побудови виведення в цій граматиці приводить нас до ланцюжка:

$$I \Rightarrow aA \Rightarrow abA \Rightarrow abbA \Rightarrow \ldots ,$$

який виявляється нескінченним. Іншими словами, $\Gamma_{1.3}$ породжує порожню мову.

1.3 Порожня мова

Визначення. Якщо мова, породжувана граматикою Г, не містить жодного кінцевого ланцюжка (кінцевого слова), то вона називається порожньою.

Твердження. Для того, щоб мова $L(\Gamma)$ не була порожньою, у множині R повинне бути хоча б одне правило вигляду $r = \chi \to \psi$, де $\psi \in V_т^*$ і повинно існувати виведення $I \Rightarrow^* \chi$.

1.4 Типи формальних мов і граматик

У теорії формальних мов виділяються 4 типи граматик, яким відповідають 4 типи мов. Такий розподіл зветься «граматична структура Хомського». Ці граматики виділяються шляхом накладення обмежень на правила граматики [4-7].

1.4.1 Граматики типу 0

Граматики типу 0, що називаються граматиками загального вигляду, не мають ніяких обмежень на правила породження. Будь-яке правило

$$r = \eta \to \psi$$

може бути побудоване з використанням довільних ланцюжків η, $\psi \in (V_т \cup V_a)^*$. Наприклад, $CCLW \to WT\ xSAb \to xrtHD$.

1.4.2 Граматики типу 1

Граматики типу 1, що називаються також контекстно-залежними граматиками, не допускають використання будь-яких правил. Правила виведення в таких граматиках повинні мати вигляд:

$$\chi_1 A \chi_2 \to \chi_1 w\ \chi_2,$$

де χ_1, χ_2 – ланцюжки, можливо порожні, з множини $(V_т \cup V_a)^*$, символ $A \in V_a$ і ланцюжок $w \in (V_т \cup V_a)^*$. Ланцюжки χ_1 і χ_2 залишаються незмінними при застосуванні правила, тому їх називають контекстом (відповідно лівим і правим), а граматику – контекстно-залежною.

Граматики типу 1 значно зручніші на практиці, ніж граматики типу 0, оскільки в лівій частині правила заміняється завжди один нетермінальний символ, який можна зв'язати з деяким синтаксичним поняттям, у той час як у граматиці типу 0 можна заміняти відразу кілька символів, у тому числі і термінальних.

Наприклад, граматика:

$$\Gamma_{1.4}: \quad V_т = \{a, b, c, d\},\ V_A = \{I, A, B\}$$

$$R = \{\ I \to aAI,$$

$$AI \to AAI$$

$$AA \to ABA$$

$$A \to b$$

$$bBA \to bcdA$$

$$bI \to ba\ \}$$

є контекстно-залежною, оскільки друге і шосте правила мають непорожній лівий контекст, а третє і п'яте правила містять обидва контексти. Виведення у такій граматиці може мати вигляд:

$$I \Rightarrow aAI \Rightarrow aAAI \Rightarrow abAI \Rightarrow abbI \Rightarrow abba.$$

1.4.3 Граматики типу 2

Граматики типу 2 називають контекстно-вільними (КВ) граматиками, або безконтекстними граматиками.

Правила виведення таких граматик мають вигляд:

$$A \to \alpha,$$

де $A \in V_a$ і $\alpha \in (V_т \cup V_a)^*$.

Очевидно, що ці правила виходять із правил граматики типу 1 за умови $\chi_1 = \chi_2 = \$$. Оскільки контекстні умови відсутні, то правила КВ- граматик виходять простіші, ніж правила граматик типу 1. Саме такі граматики використовують для опису мов програмування. Прикладом КВ-граматики може служити наступна граматика $\Gamma_{1.5}$:

$$V_т = \{a, b\}, V_A = \{I\},$$
$$R = \{ I \to aIa$$
$$I \to bIb$$
$$I \to a\text{a}$$
$$I \to bb\}.$$

Ця граматика породжує мови, що складаються з ланцюжків, кожний з яких у свою чергу складається з двох частин, ланцюжка $\beta \in V_т^*$ і дзеркального відображення цього ланцюжка β'.

$$L(\Gamma_5) = \{ \beta\beta' \mid \beta \in V_т^+\},$$

де $V_т^+$ – це множина $V_т^*$ без порожнього ланцюжка.

За допомогою правил цієї граматики може бути побудований, наприклад, ланцюжок:

$$I \Rightarrow aIa \Rightarrow abIba \Rightarrow abaIaba \Rightarrow ababbaba.$$

1.4.4 Граматики типу 3

Граматики типу 3 називають автоматними граматиками (А- граматиками). Правила виведення в таких граматиках мають вигляд:

$$A \to a, \text{ або } A \to aB, \text{ або } A \to B\ a,$$

де $a \in V_т$, $A, B \in V_A$, причому граматика може мати тільки правила вигляду $A \to aB$ – правосторонні правила, або тільки вигляду $A \to Ba$ – лівосторонні правила. Прикладами автоматних граматик можуть служити правостороння граматика $\Gamma_{1.6}$ і лівостороння граматика $\Gamma_{1.7.}$

$\Gamma_{1.6}$:

$$V_т = \{a, b\},\ V_A = \{I, A, Z\},$$

$$R = \{\ I \to aI$$

$$I \to aA$$

$$A \to bA$$

$$A \to aZ$$

$$A \to bZ$$

$$Z \to \$\ \}.$$

$\Gamma_{1.7}$:

$$V_т = \{a, b\},\ V_A = \{I, A, Z\},$$

$$R = \{\ I \to Ab$$

$$A \to Ab$$

$$A \to Za$$

$$Z \to Za$$

$$Z \to \$\ \}.$$

Ці граматики є еквівалентними і породжують мову

$$L(\Gamma_7) = \{a...ab...b \mid n,m\ \ 0\}.$$

Між множинами мов різних типів існує відношення включення:

$$\{ L_{\text{типу } 3} \} \in \{ L_{\text{типу } 2} \} \in \{ L_{\text{типу } 1} \} \in \{ L_{\text{типу } 0} \}.$$

Доведено, що існують мови типу 0, що не є мовами типу 1, мови типу 2, що не є мовами типу 1, і мови типу 3, які не є мовами типу 2.

З огляду на те, що найбільш практичне застосування знаходять граматики типу 2 і типу 3, подальший виклад присвячується розгляду саме цих типів граматик [8-9].

1.5 Виведення у КВ- граматиках і правила побудови дерева виведення

Формальні граматики дозволяють задавати мови, що являють собою множину ланцюжків, побудованих за визначеними правилами. Використовуваний спосіб задання дозволяє будь-як побудувати ланцюжок, що належить мові. Щоб зробити процес побудови виведення більш наглядним, його зображують у вигляді графа, точніше, у вигляді дерева, яке називають синтаксичним деревом, або деревом виведення. З огляду на те, що виведення будь-якого ланцюжка, який належить мові, породжуваній заданою граматикою, повинен починатися з початкового символу, правила побудови дерева можна сформулювати так:

1) Як початок чи вершину кореня дерева візьмемо вершину, яку позначимо початковим символом граматики *I*; ця вершина утворить нульовий ярус дерева;

2) Якщо при виводі ланцюжка на черговому кроці використовується правило граматики $A \to \alpha$ і вершина, позначена нетерміналом *A*, розташована на ярусі з номером *k*-1, то до побудованого дерева потрібно додати стільки вершин, скільки міститься символів у ланцюжку α, розташувати ці вершини на ярусі *k* , позначити їх символами ланцюжка α і з'єднати ці вершини дугами з вершиною *A*. Результатом виведення є множина

кінцевих вузлів – листів, що виписуються при обході дерева ліворуч – униз – праворуч – нагору. Розглянемо, наприклад, граматику $\Gamma_{1.8}$:

$\Gamma_{1.8}$:

$V_т = \{a, b\}$, $V_A = \{I\}$,

$R = \{I \to aIb,$

$I \to ab\ \}$,

яка породжує мову $L(\Gamma_{1.8}) = \{aa...abb...b\}$, де a і b повторюються однакову кількість разів.

Виведення ланцюжка за допомогою правил цієї граматики має наступний вигляд (рис. 1.1.):

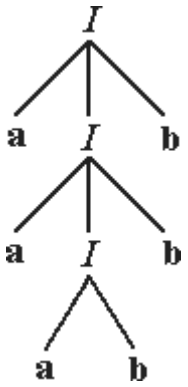


Рисунок 1.1 – Виведення ланцюжка *aaabbb*

1.6 Синтаксичний розбір

Виведення ланцюжка за допомогою правил граматики може бути заданий не тільки у вигляді синтаксичного дерева [4,5,9]. Якщо пронумерувати правила граматики, то послідовність номерів використовуваних правил також задає виведення.

Визначення. Послідовність номерів правил граматики Г, застосування яких дозволяє побудувати виведення розглянутого ланцюжка σ з початкового символу граматики, називається синтаксичним розбором σ.

Наприклад, у граматиці $\Gamma_{1.9}$:

$$V_т = \{ i, +, *, (,) \}, V_A = \{E, T, P\},$$

$R = \{$ 1. $E \to E + T$ 5. $P \to (E)$

2. $E \to T$ 6. $P \to i$ },

3. $T \to T * P$

4. $T \to P$

правила якої пронумеровані, виведення

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow T * P + T \Rightarrow P * P + T \Rightarrow i * P + T \Rightarrow i * i + T \Rightarrow i * i + P \Rightarrow$$
$$i * i + i$$

має синтаксичний розбір [1, 2, 3, 4, 6, 6, 4, 6].

Якщо в процесі побудови виведення з'являються проміжні ланцюжки, що містять кілька нетермінальних символів, то можна продовжувати виведення, заміняючи кожний з ланцюжків. Таким чином, ті самі правила можуть бути використані при виводі ланцюжка у будь-якому порядку.

Наприклад, виведення ланцюжка $i + i$ в граматиці $\Gamma_{1..9}$ може бути отриманий десятьма різними способами.

1.7 Ліве та праве виведення

Серед різного виведення найбільший інтерес становлять наступні два типи виведення.

Визначення. Якщо при побудові виведення ланцюжка α при кожнім застосуванні правила заміняється найлівіший нетермінальний символ, то таке виведення називається лівим, або лівостороннім виведенням α. Якщо при побудові виведення α, завжди заміняється найправіший нетермінальний символ проміжного ланцюжка, то виведення називається правим, або правостороннім виведенням α.

Наприклад вище наведене виведення ланцюжка $i * i + i$ в граматиці $\Gamma_{1.9}$ є лівостороннім виведенням. Слід зазначити, що різному виведенню

ланцюжка $i+i$ в граматиці $\Gamma_{1..9}$ відповідає те ж саме синтаксичне дерево. Аналогічна ситуація має місце і при виводі ланцюжка $i * i + i$.

1.8 Неоднозначні й еквівалентні граматики

Існують граматики, в яких той самий ланцюжок може бути отриманий за допомогою різних виводів [4,9]. Наприклад, у граматиці $\Gamma_{1..10}$ ланцюжок *abc* може бути отриманий за допомогою різного виведення, і йому відповідають два різних синтаксичних дерева.

$\Gamma_{1.10}$:

$$V_т = \{a, b, c, d\},\ V_A = \{I, A, B\},$$

$$R = \{\ I \to AB,$$

$$A \to a,$$

$$A \to ac,$$

$$B \to b,$$

$$B \to cb\}.$$

Перше виведення цього ланцюжка має вигляд :

1) $I \Rightarrow AB \Rightarrow Ab \Rightarrow acb$,

а другий можна одержати так :

2) $I \Rightarrow AB \Rightarrow Acb \Rightarrow acb$.

Цим виводам відповідають різні синтаксичні дерева і розбори (рис. 2.2):

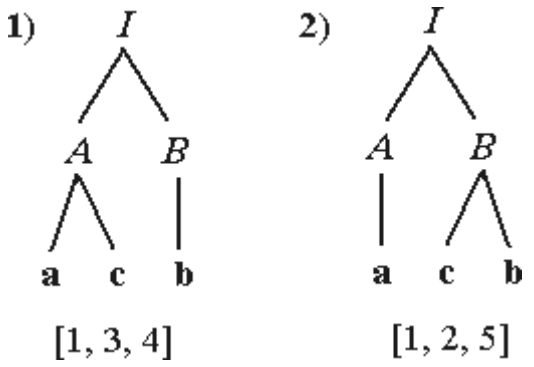


Рисунок 1.2 – Синтаксичні дерева і розбори виразу *acb*

Наступна граматика також допускає побудову одного і того ж ланцюжка за допомогою різного виведення, що має різні синтаксичні дерева.

$\Gamma_{1.11}$:

$V_т = \{0, +\}$, $V_A = \{I\}$,

$R = \{ I \to 0,$

$I \to I + 0,$

$I \to 0 + I \}.$

Два виводи цієї граматики, що породжують однакові ланцюжки, мають вигляд:

1) $I \Rightarrow I + 0 \Rightarrow I + 0 + 0 \Rightarrow 0 + 0 + 0,$

2) $I \Rightarrow 0 + I \Rightarrow 0 + 0 + I \Rightarrow 0 + 0 + 0,$

а синтаксичні дерева, що відповідають цим виводам, можна зобразити так (рис. 1.3):

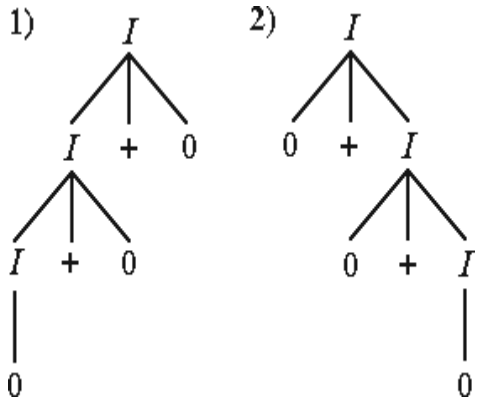


Рисунок 1.3 – Синтаксичні дерева виведення виразу: 0+0+0

Розглянута властивість граматик називається неоднозначністю. Вона може бути визначена в такий спосіб.

Визначення. Ланцюжок мови $L(\Gamma)$ називається неоднозначним, якщо для його виведення існує більш ніж одне синтаксичне дерево. Якщо граматика Г породжує неоднозначний ланцюжок, то вона називається неоднозначною.

Неоднозначність може існувати не тільки в штучних мовах. Добре відомо, що в природних мовах можуть бути пропозиції, що допускають неоднозначне написання. Наприклад, "Пальто забруднило вікно". У цій фразі не ясно, що є підметом, а що доповненням. Іншим прикладом служить англійська фраза: "*They are flying planes*", що може бути зрозуміла подвійно: “Вони пілотують літак” або “Це літаки, що летять”.

Властивість неоднозначності є вкрай небажаною для штучних мов, оскільки вона не дозволяє однозначно відновити дерево виведення за заданим ланцюжком мови.

У загальному випадку можна зробити такий висновок:

1) кожному ланцюжку, виведеному в граматиці, може відповідати одне або кілька синтаксичних дерев;

2) кожному синтаксичному дереву можуть відповідати різні виведення;

3) кожному синтаксичному дереву відповідають єдиний правий і єдиний лівий виводи.

Крім того, варто підкреслити, що та сама мова може бути отримана за допомогою різних граматик.

Визначення. Дві граматики – Γ_1 і Γ_2 – називаються еквівалентними, якщо вони породжують ту саму мову, тобто $L(\Gamma_1) = L(\Gamma_2)$.

1.9 Способи задання схем граматик

Схема граматики містить правила виведення, що визначають синтаксис мови, або, іншими словами, можливі компоненти і конструкції ланцюжків породжуваної мови. Для задання правил використовуються різні форми опису: символічна, форма Наура-Бекуса, ітераційна форма і синтаксичні діаграми.

У роботах, пов'язаних з розглядом загальних властивостей граматик, звичайно застосовують символічну форму завдання правил. Ця форма була розглянута в попередньому пункті. Вона передбачає використання як елементів нетермінального словника окремих символів і стрілки як роздільника правої і лівої частин правила.

1.9.1 Форма Наура-Бекуса

При описі синтаксису конкретних мов програмування доводиться визначати велику кількість нетермінальних символів, і символічна форма запису втрачає свою наглядність. У цьому випадку застосовують форму Наура-Бекуса (ФНБ), що припускає використання комбінацій слів природної мови, узятих у кутові дужки, як нетермінальних символів, а як роздільник – спеціальний знак, що складається з двох двокрапок і знаку дорівнює. Наприклад, якщо правила $L \to L$ і $L \to E$ записані в символічній формі і символ L відповідає синтаксичному поняттю "список", а символ E – "елемент списку", то їх можна подати у формі Наура-Бекуса так:

<список>::= <елемент списку><список>

<список>::=< елемент списку>.

Щоб скоротити опис схеми граматики, у ФБН дозволяється поєднувати правила з однаковою лівою частиною в одне правило, права частина якого повинна включати праві частини поєднуваних правил, розділені вертикальною рискою. Використовуючи об'єднання правил, для розглянутого прикладу одержуємо

<список>::=<елемент списку><список>|<елемент списку>.

1.9.2 Ітераційна форма

Для одержання більш компактних описів синтаксису застосовують ітераційну форму опису. Така форма припускає введення спеціальної операції, що називається ітерацією і позначається парою фігурних дужок із

зірочкою. Ітерація виду $\{a\}^*$ визначається як множина, що включає ланцюжки будь-якої довжини, побудовані з використанням символу a і порожній ланцюжок.

$$\{a\}^* = \{\$, a, aa, aaa, aaaa,...\}.$$

Використовуючи ітерацію для опису множини ланцюжків, що задаються символічними правилами, для списку одержуємо:

$$L \to E\ \{E\}^*.$$

Наприклад, опис множини ланцюжків, кожний з яких повинен починатися знаком # і може складатися з довільного числа букв x і y, може бути поданий в ітераційній формі так:

$$I \to \#\{x \mid y\}^*.$$

В ітераційних формах опису поряд з ітераційними дужками часто застосовують квадратні дужки для вказівки того, що ланцюжок , укладений в них, може бути опущений. За допомогою таких дужок правила

$$A \to xAyBz \text{ і } A \to xBz$$

можуть бути записані так:

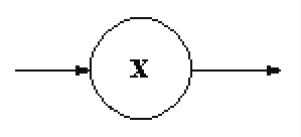
$$A \to x \mid Ay \mid Bz.$$

1.9.3 Синтаксичні діаграми

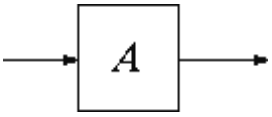
Для того щоб поліпшити зорове сприйняття і полегшити розуміння складних синтаксичних описів, застосовують подання правил граматики у вигляді синтаксичних діаграм. Правила побудови таких діаграм можна сформулювати в наступному вигляді:

1) кожному правилу вигляду $A \to a_1 \mid a_2 \mid ... \mid a_k$ ставиться у відповідність діаграма, структура якої визначається правою частиною правила;

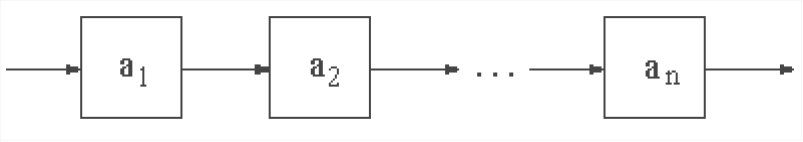
2) кожна поява термінального символу x у ланцюжку a_i зображується на діаграмі дугою, позначеною цим символом x, взятим в коло



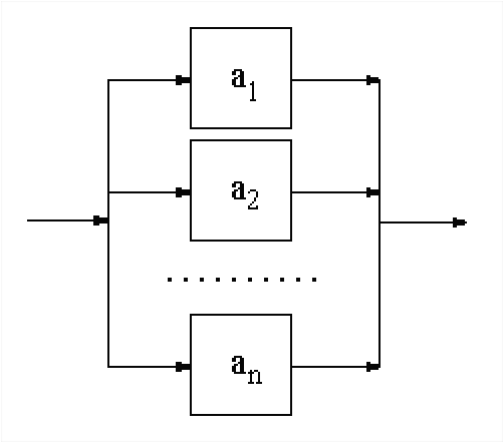
3) кожній появі нетермінального символу A у ланцюжку a_i ставиться у відповідність на діаграмі дуга, позначена символом, взятим у квадрат



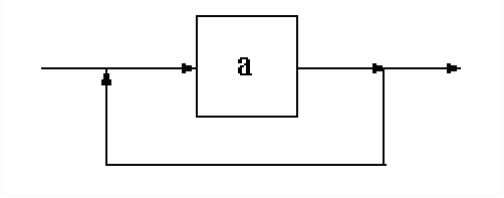
4) правило, що породжує, що має вигляд $A \to a_1 a_2 ... a_n$ і зображується на діаграмі в такий спосіб:



5) правило, що породжує, має вигляд $A \to a_1 \mid a_2 \mid ... \mid a_n$ і зображується на діаграмі так



6) якщо правило, що породжує, задано у вигляді ітерації $A \to \{a\}^*$, то йому відповідає діаграма



Число синтаксичних діаграм, які можна побудувати для заданої схеми граматики, визначається числом правил. Щоб скоротити число діаграм, їх поєднують, заміняючи нетермінальні символи, що входять у діаграму, побудованими для них діаграмами.

Правила 3–6 передбачають, що для ланцюжка a_1 на об'єднаній діаграмі можуть бути використані діаграми, побудовані для цих ланцюжків. Як приклад розглянемо наступну граматику з початковим символом A :

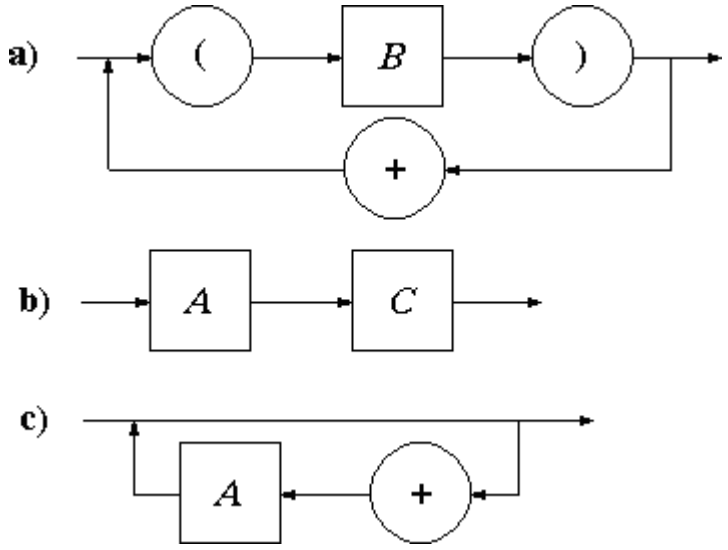
$$\Gamma_{1.12}: V_т = \{ x, +, (,) \}, V_A = \{A, B, C\},$$

$$R = \{A \to x \mid (B),$$

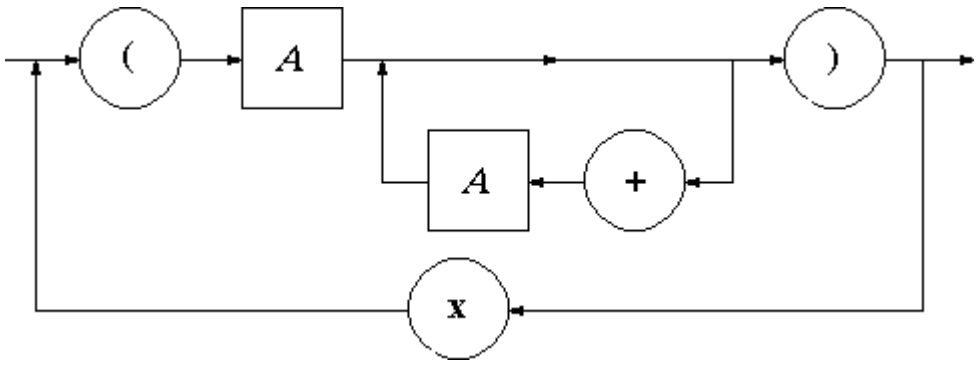
$$B \to AC,$$

$$C \to \{+A\}^*\}.$$

Синтаксичні діаграми для такої граматики мають такий вигляд:



Замінюючи нетермінальні символи, що відповідають діаграмам, одержуємо об'єднану діаграму в наступному вигляді



1.10 Побудова граматик і граматики, що описують основні конструкції мов програмування

Задача побудови формальних граматик полягає в тому, що для заданої у вигляді опису природною мовою множини кінцевих ланцюжків, потрібно побудувати формальну граматику, що породжує цю множину ланцюжків. З огляду на те, що термінальний словник граматики повинен включати всі символи, які використовуються для побудови ланцюжків, що входять у задану множину, результатом розв’язку задачі повинні стати нетермінальний словник і схема граматики.

Побудова цих об'єктів є дуже складною, оскільки вона повинна виконуватися неформально і вимагає уявного охоплення всіляких варіантів побудови ланцюжків заданої множини і синтезу правил їх побудови. Побудова ускладнюється ще і тим, що вона, як і всяка інша задача синтезу, має багато рішень.

1.10.1 Рекомендації з побудови граматик

Основою створення правил граматики є спосіб виділення структури заданої множини ланцюжків. Цей спосіб передбачає розчленування ланцюжків, що входять у задану множину, на їх частини таким чином, щоб виявити повторювані частини ланцюжків і частини, що входять у всі ланцюжки в незмінному вигляді. Таке розчленування на частини являє собою виявлення структури ланцюжків заданої множини.

Для кожного виявленого елемента структури введемо позначення. Множина таких позначень складає основу словника нетермінальних символів деякої граматики. Наступним кроком побудови є виявлення послідовностей, у яких елементи структури можуть входити в задані ланцюжки. Такі послідовності є основою для побудови правил граматики. Щоб показати, яким чином структура ланцюжків відображається в правилах граматики, розглянемо наступні приклади.

1. Ланцюжку, що складається з заданих символів *abc*, відповідає правило

$$I \to abc.$$

2. Ланцюжку, що починається з заданого символу *a*, відповідає правило

$$I \to aA.$$

3. Ланцюжку, що закінчується заданим символом *a*, відповідає правило

$$I \to Aa.$$

4. Ланцюжку, що починається і закінчується заданими символами *a*, *b*, відповідає правило

$$I \to aAb.$$

5. Ланцюжку, що містить у середині символ *a*, відповідає правило

$$I \to AaB.$$

6. Ланцюжку заданої довжини l =2 відповідають правила:

$$A \to aB \text{ і } B \to a.$$

7. Ланцюжку, що складається з повторюваних символів *a*, відповідають правила

$$A \to aA \text{ і } A \to a.$$

8. Ланцюжку, що складається із символів, що чергуються, *a* і *b*, відповідають правила:

$$A \to aB \text{ і } B \to bA.$$

1.10.2 Опис списків

Як перші приклади розглянемо побудову граматик для послідовностей символів і послідовностей символів з роздільниками. Такі послідовності часто називають списками.

1. Позначимо елемент послідовності a. Найпростіша послідовність може складатися з одного елемента a. Всі інші послідовності можуть бути отримані шляхом приписування до вже побудованої послідовності ще одного елемента. Якщо позначити побудовану частину послідовності нетермінальним символом R, а послідовність символом L, то одержимо правила граматики у вигляді:

$$\Gamma_{1.13}: \quad L \to aR,$$
$$R \to aR,$$
$$R \to \$,$$

2. У попередній задачі передбачалося, що список L повинен містити хоча б один елемент. Якщо ж допустити, що множина ланцюжків, породжених правилами граматики, може включати порожній символ, то до побудованих правил потрібно додати ще одне правило $L \to \$$. У цьому випадку набір правил має вигляд :

$$\Gamma_{1.14}: \quad L \to aR,$$
$$R \to aR,$$
$$R \to \$,$$
$$L \to \$.$$

3. Розглянемо побудову списку, між елементами якого повинні стояти роздільники. Виберемо як роздільник кому. Найпростіший список, як і в попередньому випадку, складається з одного елемента, а побудова списку з декількох елементів може бути виконана приписуванням до вже побудованої частини списку роздільника з елементом списку. Правила, що відповідають цій побудові, мають вигляд:

$$\Gamma_{1.15}: \quad L \to aR,$$
$$R \to \text{ , } aR,$$
$$R \to \$.$$

4. Якщо список з роздільниками може бути порожнім, то наведений вище набір правил потрібно доповнити ще одним правилом з порожньою правою частиною. У результаті одержимо:

$$\Gamma_{1..16}: \quad L \to aR,$$
$$R \to \text{ , } aR,$$
$$R \to \$,$$
$$L \to \$.$$

У загальному випадку, якщо описана множина ланцюжків, що являють собою деяку мову і потрібно побудувати граматику, яка породжує цю множину ланцюжків, то слід робити так:

1) виписати кілька прикладів із заданої множини ланцюжків;

2) проаналізувати структуру ланцюжків, виділяючи початок, кінець, що повторюються, або символи з групи символів;

3) ввести позначення для складних структур, що складаються з груп символів; такі позначення є нетермінальними символами шуканої граматики;

4) побудувати правила для кожної з виділених структур, використовуючи для завдання повторюваних структур рекурсивні правила;

5) об’єднати всі правила;

6) перевірити за допомогою виведення можливість одержання ланцюжків з різною структурою.

1.10.3 Приклад побудови граматик

Застосування наведених рекомендацій розглянемо на наступному прикладі. Потрібно побудувати граматику для мови L ,термінальний словник

якого V_m = {*, |}, а ланцюжки, що утворюють мову, мають наступну структуру:

а) кожен ланцюжок починається буквою * і закінчується двома буквами **.

б) між початком і кінцем ланцюжків можуть бути або непорожня послідовність паличок, або кілька таких послідовностей, розділених символами *.

1. Спочатку побудуємо кілька ланцюжків заданої мови, що можуть бути подані в наступному вигляді:

* |||**,

* |*|*|**,

* ||*||||*|||||** ,

* |||*|*||*||||||** .

2. Розглядаючи наведені ланцюжки, можна виділити наступні їх структурні компоненти:

початок ланцюжка (символ *);

кінець ланцюжка (символи **);

непорожня група паличок;

послідовність груп паличок, розділених зірочками.

3. Позначимо групу паличок символом *A*, а послідовність груп паличок символом *B*.

4. Виділені структури можна розглядати як списки. Так послідовність паличок являє собою список без роздільників, елементом якого є паличка. Правила граматики, що задають такий список, мають такий вигляд:

$$A \to | \; B,$$

$$B \to | \; B,$$

$$B \to \$.$$

Послідовність груп паличок, розділених зірочкою, являє собою список з роздільником. Елементом такого списку є група паличок *A*, а роздільником – зірочка. Правила граматики, що задає такий список, можна записати так:

$$C \to AE,$$

$$E \to *AE,$$

$$E \to \$.$$

З огляду на те, що кожен ланцюжок мови повинен мати початок і кінець, і , вибираючи як початковий символ граматики *I*, одержуємо правило, що визначає загальний вигляд ланцюжка:

$$I \to *C**.$$

5. Поєднуючи побудовані правила, остаточно одержимо схему шуканої граматики у вигляді:

$$\Gamma_{1.17}: R = \{ I \to *C**,$$

$$C \to AE,$$

$$E \to *AE,$$

$$E \to \$,$$

$$A \to | B,$$

$$B \to | B,$$

$$B \to \$ \}$$

6. За допомогою правил побудованої граматики можна отримати, наприклад, ланцюжок

$$I \Rightarrow *C** \Rightarrow *AE** \Rightarrow *A*AE** \Rightarrow$$

$$*A*A*AE \Rightarrow *A*A*A** \Rightarrow$$

$$*A*A* | B** \Rightarrow *A*A* | ** \Rightarrow$$

$$*A* | B* | ** \Rightarrow *A* | * | ** \Rightarrow$$

$$* | B* | * | ** \Rightarrow * | * | * | **.$$

Побудоване виведення ілюструє можливість породження ланцюжків заданої мови за допомогою побудованої граматики.

Однією з основних областей застосування формальних граматик є опис мов програмування. З огляду на широке використання подібних описів у літературі і їх важливе значення для створення компіляторів розглянемо побудову граматик для основних конструкцій мов програмування. Щоб скоротити розміри граматик, на розглянуті конструкції накладаються деякі, іноді істотні, обмеження.

1.10.4 Граматики, що описують цілі числа без знаку і ідентифікатори

Цілі числа являють собою послідовність цифр, тому їх можна розглядати як списки, елементами яких є цифри. Використовуючи як аналог граматику, що задає список без роздільників, одержимо схему граматики для цілих чисел у вигляді:

$$\Gamma_{1.18}: N \to DR,$$
$$R \to DR,$$
$$R \to \$,$$
$$D \to 0 \mid 1 \mid ... \mid 9.$$

Структуру ідентифікатора можна подати у вигляді двох компонентів: початку й основної частини. Початком може бути кожна з букв, а основна частина являє собою список без роздільників, елементами якого можуть бути або букви, або цифри. Використовуючи виділені компоненти, одержимо схему граматики такого вигляду:

$$\Gamma_{1.19}: R = \{ I \to CA,$$
$$A \to CA|DA,$$
$$A \to C \mid D,$$
$$A \to \$,$$
$$D \to 0 \mid 1 \mid ... \mid 9,$$
$$C \to a \mid d \mid c \mid ... \mid z \}.$$

Якщо накласти обмеження на довжину ідентифікатора, наприклад допустити використання ідентифікаторів, що складаються тільки з трьох символів, то схема граматики виходить простіше.

$$\Gamma_{1.20}: R = \{ I \to CA1,$$
$$A1 \to CB,$$
$$A1 \to DB,$$
$$B \to C,$$
$$B \to D,$$
$$D \to 0 \mid 1 \mid ... \mid 9,$$
$$C \to a \mid d \mid c \mid ... \mid z \}.$$

1.10.5 Граматики для арифметичних виразів

Домовимося розглядати арифметичні вирази, що використовують тільки знаки додавання і множення. Спочатку побудуємо граматику для виразів без дужок. Такі вирази являють собою ланцюжки, які можна розглядати як списки з роздільниками, в яких роль роздільників виконують знаки операцій. Відповідно до цієї аналогії одержуємо схему граматики, в якій символ *I* позначає ідентифікатор, визначення якого наведено вище.

$$\Gamma_{1.21}: R = \{ H \to IR,$$
$$R \to WIR,$$
$$R \to \$,$$
$$W \to +,$$
$$W \to * \}.$$

Щоб побудувати граматику, яка допускає використання в арифметичних виразах дужок без вкладеності, подамо структуру таких виразів у вигляді списку з роздільниками, елементами якого є вираз без дужок, або вираз, узятий в дужки. Роздільниками цього списку є знаки операцій. Такій структурі відповідає наступна схема граматики:

$$\Gamma_{1.22}: \quad R = \{ G \to HQ,$$
$$G \to (H)Q,$$
$$Q \to WG,$$
$$Q \to \$ \}.$$

Для побудови граматики арифметичних виразів, що допускають застосування вкладених дужок, варто передбачити можливість використання як елемента списку не тільки виразу без дужок, але і виразу, в якому можуть бути використані дужки. Схема граматики, що враховує цю можливість, може бути записана у вигляді:

$$\Gamma_{1.23}: R = \{ E \to GP,$$
$$E \to (E)P,$$
$$P \to WE,$$
$$P \to \$ \}.$$

1.10.6 Граматика для описів

Нехай потрібно побудувати граматику для опису цілих і дійсних змінних. Опис змінних визначеного типу повинен починатися покажчиком типу '*real*' або '*int*'. У повному тексті опису змінних визначеного типу вони можуть повторюватися. Наприклад, повний опис може включати три різних описи змінних цілого типу. Повний опис повинен закінчуватися крапкою. Як роздільник описів змінних різних типів приймемо крапку з комою, а як роздільник змінних одного типу – кому. Структуру повного опису можна подати у вигляді двох вкладених списків з роздільниками. Внутрішній список, розглянутий як елемент зовнішнього списку, являє собою опис змінних одного типу. Він має заголовок у вигляді покажчика типу, за яким випливає послідовність ідентифікаторів, розділених комами. Зовнішній список використовує як роздільник крапку з комою. Схема граматики розглянутого вигляду може бути записана так:

$\Gamma_{1.24}$: $R = \{ Z \to A_2,$

$A_2 \to B_1C_1,$

$C_1 \to ;B_1C_1,$

$C_1 \to \$,$

$B_1 \to float\ L,$

$B_1 \to int\ L,$

$L \to iK,$

$K \to ,iK,$

$K \to \$ \}.$

1.10.7 Граматика, що задає послідовність операторів присвоєння

Припустимо, що в правій частині оператора присвоювання можуть бути використані тільки вирази без дужок, описані граматикою $\Gamma_{1.21.}$ Як роздільник операторів у послідовності приймемо крапку з комою. З огляду на те, що послідовність операторів відповідає списку з роздільниками у вигляді крапки з комою і, використовуючи визначену раніше конструкцію ідентифікатора, одержимо шукану схему граматики у вигляді:

$\Gamma_{1.25}$: $R = \{ U \to A_3,$

$A_3 \to SR_1$

$R_1 \to ;SR_1,$

$R_1 \to \$,$

$S \to iB_2,$

$B_2 \to = H_1,$

$H_1 \to iR_2,$

$R_2 \to + iR_2,$

$R_2 \to * i R_2,$

$R_2 \to \$\}.$

1.10.8 Граматики, що описують умовні оператори й оператори циклу

Припустимо, що розглядаються умовні оператори, аналогічні використовуваним у мові С++. Як умову в таких операторах дозволяється використовувати відносини, що складаються з двох ідентифікаторів, з'єднаних знаками (наприклад: >, <, =, !=). Приклад граматики, що задає умовний оператор, може бути зображена так:

$$\Gamma_{1.26}: \quad R = \{ \; V \to if \; (\mathrm{O}),$$
$$\mathrm{O} \to \; S\, Z\, S,$$
$$Z \to < \mid > \mid = \mid \; != .$$

У цій граматиці ідентифікатор S визначається схемою граматики $\Gamma_{1.19.}$ Розглянемо опис операторів циклу з передумовою та з постумовою. Для оператору циклу з передумовою з використанням визначених раніше нетермінальних символів граматика має вигляд:

$$\Gamma_{1.27}: \quad R = \{ \; W \to while \; (0)\}.$$

У цій граматиці О визначається схемою граматики $\Gamma_{1.26.}$

Для оператору циклу з постумовою з використанням визначених раніше нетермінальних символів граматика має вигляд:

$$\Gamma_{1.28}: \quad R = \{ \; W1 \to do$$
$$\ldots$$
$$while \; (0)\},$$

де О визначається схемою граматики $\Gamma_{1.26.}$

1.11 Приведені граматики

З чотирьох типів граматик контекстно-вільні граматики (КВ) є найбільш важливими з погляду додатків до мов програмування і компіляції. За допомогою КВ-граматики можна визначити велику частину структури мови програмування. При побудові граматик, що задають конструкції мов програмування, часто доводиться вдаватися до їх перетворення, щоб породжувана мова набула потрібної структури. Тому спочатку розглянемо

трохи досить простих, але важливих перетворень КВ-граматик [10-12]. Перший вид перетворення пов'язаний з видаленням із граматики зайвих символів. Зайві символи в граматиці можуть виявитися в наступних випадках:

а) якщо символ не може бути отриманий при виводі;

б) якщо із символу не може бути отриманий кінцевий термінальний ланцюжок (виходить нескінченний ланцюжок або немає правил, що приводять до термінального ланцюжка).

Спочатку розглянемо алгоритм виявлення символів, з яких не можна вивести кінцеві ланцюжки.

1.12 Визначення непродуктивних символів

Символ $\boldsymbol{X} \in \boldsymbol{V_A}$ називається **непродуктивним**, якщо з нього не може бути виведений кінцевий термінальний ланцюжок.

Розглядаючи правила граматики, можна зробити висновок, що коли всі символи правої частини є продуктивними, то продуктивним є і символ, що стоїть в лівій частині. Останнє твердження дозволяє написати процедуру виявлення непродуктивних символів у наступному вигляді:

1. Скласти список нетерміналів, для яких знайдеться хоча б одне правило, права частина якого не містить нетермінали.

2. Якщо знайдене таке правило, і всі нетермінали, які стоять у його правій частині вже занесені в список, то додати в список нетермінал, що стоїть в його лівій частині.

3. Якщо на кроці 2 список більше не поповнюється, то отримано список усіх продуктивних нетерміналів граматики, а всі нетермінали, які не потрапили в нього, є непродуктивними.

Приклад. Маємо граматику $\Gamma_{1.29}$:

$$R = \{I \to aIa,$$

$$I \to bAd,$$

$$I \to c,$$

$$A \to cBd,$$

$$A \to aAd,$$

$$B \to dAf \ \},$$

знаходимо, що тут непродуктивними є символи A і *B*. Після виключення правил, що містять непродуктивні символи, одержуємо граматику:

$$R' = \{I \to a\ I\ a,$$

$$I \to c\}.$$

1.13 Визначення недосяжних символів

Символ $\boldsymbol{X} \in \boldsymbol{V_A}$ називається **недосяжним** у КВ-граматиці Г, якщо x не з'являється в жодному виведеному ланцюжку.

Розглядаючи правила граматики, можна помітити, що якщо нетермінал у лівій частині правила є досяжний, то і всі символи правої частини є досяжними. Ця властивість правил є основою процедури виявлення недосяжних символів, які можна записати так:

1. Створити одноелементний список, що складається з початкового символу граматики I.

2. Якщо знайдене правило, ліва частина якого вже є в списку, то включити до списку всі символи, які містяться в його правій частині.

3. Якщо на кроці 2 нові нетермінали в список більше не додаються, то отримано список усіх досяжних нетерміналів, а нетермінали, що не потрапили в список, є недосяжними.

Приклад. Маємо граматику $\Gamma_{1.30}$:

$$R = \{\ \ I \to aIb,$$

$$I \to c,$$

$$A \to bI,$$

$$A \to a \ \}.$$

Знаходимо, що A є недосяжним символом.

КВ-граматика називається **приведеною**, якщо вона не містить зайвих марних (непродуктивних і недосяжних) символів.

1.14 Виключення ліворекурсивних правил

Ряд граматик потребують виключення ліворекурсивних та ланцюгових правил.

Правило вигляду $A \to \alpha\ A$, де $A \in V_A$, $\alpha \in (V_т \cup V_A)^*$, називається **праворекурсивним**, а правило вигляду $A \to A\alpha$ – **ліворекурсивним**.

Для кожної КВ-граматики Г, що містить ліворекурсивні правила, можна побудувати еквівалентну граматику Г′, що не містять ліворекурсивних правил.

Щоб показати техніку перетворення, розглянемо приклад.

Приклад. Маємо граматику $Г_{1.31}$:

$$R=\{E \to E + T \mid T,$$

$$T \to T*F \mid F,$$

$$F \to (E) \mid a\}.$$

Правило $E \to E + T \mid T$ перетворимо в правила $E \to T \mid TE'$ і $E' \to +T \mid +TE$, а правила $T \to T * F \mid F$ перетворимо в правила $T \to F \mid FT'$ і $T' \to * F \mid * FT'$. В результаті одержимо еквівалентну граматику **Г,** що містить правила R':

$E \to T,$ $\quad$ $T \to FT',$

$E \to TE',$ $\quad$ $T' \to *F,$

$E' \to + T,$ $\quad$ $T' \to * FT',$

$E' \to + TE',$ $\quad$ $F \to a,$

$T \to F$, $F \to (E)$.

1.15 Виключення ланцюгових правил

Правило граматики виду $A \to B$, де $A,B \in V_A$, називається **ланцюговим.** Для КВ-граматики Г, яка містить ланцюгові правила, можна побудувати еквівалентну їй граматику Г′, яка не містить ланцюгових правил.

Ідея доказу полягає в наступному. Якщо схема граматики має вигляд

$$R = \{...,A \to B,...,B \to C, \ldots , C \to aX \},$$

то така граматика еквівалентна граматиці зі схемою

$$R' = \{...,A \to aX,...\},$$

оскільки виведення у граматиці зі схемою R ланцюжка aX :

$$A \Rightarrow B \Rightarrow C \Rightarrow ax$$

може бути отриманий у граматиці зі схемою R' за допомогою правила $A \to aX$.

Контрольні запитання

1. Дайте визначення формальної граматики.
2. Які символи граматики називаються термінальними, а які – не термінальними?
3. Що називається мовою, яка породжується граматикою?
4. Які існують типи граматик?
5. Яка граматика називається контекстно-вільною граматикою?
6. Які граматики називаються неоднозначними?
7. Які існують способи завдання схем граматик?
8. Що являє собою ітераційна форма?
9. Які існують рекомендації стосовно побудови правил граматики?
10. Побудувати правила граматики, що описують прототипи функцій.
11. Яка граматика називається приведеною?

12. Які символи граматики називаються непродуктивними?

13. Які символи граматики називаються недосяжними?

14. Які символи граматики називаються зайвими?

15. Які правила називаються ліворекурсивними, а які праворекурсивними?

16. Як можна виключити рекурсію?

2 СКІНЧЕННІ АВТОМАТИ

КВ-граматика визначає структуру ланцюжків і дозволяє будувати ланцюжки визначеної мови. Скінченні автомати дозволяють вирішувати для контекстно-вільних мов задачу розпізнавання, яка полягає в тому, що за заданим ланцюжком необхідно визначити, чи належить він заданій мові [9, 13].

2.1 Скінченні автомати

Скінченний автомат (*finite-state machine*) – це абстракція, що використовується для опису шляху зміни стану об'єкта в залежності від поточного стану та інформації отриманої ззовні. Його особливістю є скінченність множини станів автомату.

Скінченні автомати (СА) можуть розв'язувати велику кількість задач, серед яких автоматизація проектування електронних приладів, проектування комунікаційних протоколів, синтаксичний аналіз та інші інженерні застосування. В біології і дослідженнях штучного інтелекту, автомати або їх ієрархії іноді використовуються для описання неврологічних систем, в лінгвістиці – для описання граматики природних мов.

Існує дві різних групи автоматів: розпізнавачі і перетворювачі.

Результатом роботи розпізнавача є відповідь так, або ні на питання прийняті автоматом вхідні дані чи ні. Як правило на вхід подаються символи (літери). Автомат також може бути описаний як такий, що визначає мову, яка містить всі слова розпізнавані цим автоматом. Вихідними даними для такого автомату є множина термінальних символів мови. За визначенням, мова є регулярною, якщо існує деякий СА, який розпізнає її.

Перетворювачі створюють вихід, що базується на даному вході і/або на станах з використанням дій. Вони використовуються для керування і в галузі

математичної лінгвістики. Тут вирізняють два типи: автомат Мілі та автомат Мура.

На практиці часто використовується суміш моделей.

Автомати можуть бути детермінованими та недермінованими. В детермінованих автоматах, кожен стан має лише один перехід для кожного входу, в недетермінованих існує декілька переходів. Існують алгоритми трансформації будь-якого недермінованого автомату в більш складний детермінований автомат з однаковою функціональністю.

2.2 Детермінований скінченний автомат перетворювач

Детермінований скінченний автомат [10, 13] перетворювач є шісткою, тобто описується трьома кінцевими множинами, начальним станом і двома функціями: $A = \{X, Y, S, s_0, \delta, \lambda\}$,

де X – множина вхідних сигналів або вхідна абетка,

Y – множина вихідних сигналів або вихідна абетка,

S – множина станів або абетка станів,

δ – функція переходів, $s(t + 1) = \delta(s(t), x(t))$,

λ – функція виходів, $y(t) = \lambda(s(t), x(t))$.

$s_o \in S$, початковий стан автомату.

Функція переходів δ автомата A задає відображення $(X \times S) \rightarrow S$. Функція виходів λ автомата A задає або відображення $(X \times S) \rightarrow Y$, або відображення: $S \rightarrow Y$.

Функція переходів δ показує, що автомат А, знаходячись у деякому стані $s_i \in S$, під час появи вхідного сигналу $x_j \in X$ переходить в деякий стан $s_p \in S$. Це записується виразом $s_p = \delta(s_i, x_j)$. Функція виходів λ, яка задає відображення: $(X \times S) \rightarrow Y$, показує, що автомат A, знаходячись в деякому стані $s_i \in S$, під час появи вхідного сигналу $x_j \in X$, виробляє вихідний сигнал $y_k \in Y$. Це записується виразом $y_k = \lambda(s_i, x_j)$.

В автоматі Мілі функція виходів λ задає відображення $(X \times S) \rightarrow Y$. В автоматі Мура функція виходів λ задає відображення $S \rightarrow Y$. Довільний скінченний автомат Мілі або Мура має один вхідний і один вихідний канали.

Виділяють повністю визначені та частково визначені автомати. **Повністю визначеним** називається скінченний автомат, у якого функції переходів і виходів визначені для всіх пар (x_j, s_k).

Частково визначеним називається скінченний автомат, у якого функція переходів, або функція виходів, або обидві ці функції визначені не для всіх пар (x_j, s_k).

Вважається, що скінченний автомат функціонує в дискретному автоматному часі $t = 0, 1, 2, ...$, а переходи із стану в стан відбуваються миттєво. В кожний момент t дискретного часу автомат знаходиться в деякому стані $s(t)$ із множини станів S.

Якщо автомат ініціальний, то в початковий момент часу t, який дорівнює нулю, він завжди знаходиться в початковому стані $s(t) = s(0) = s_o$. В момент часу t, знаходячись в стані $s(t)$, автомат здатний сприйняти на вході букву вхідного алфавіту $x(t) \in X$. У відповідності до функції виходів λ він видасть у той же момент часу t букву вихідного алфавіту $y(t)$, а у відповідності до функції переходів δ перейде в наступний стан $s(t + 1)$. Згідно з визначенням скінченного автомата, автомат Мілі в цьому випадку характеризується системою рівнянь (6.1):

$$y(t) = \lambda[s(t), x(t)],$$
$$s(t + 1) = \delta\,[s(t), x(t)], \qquad (2.1)$$

а автомат Мура – системою рівнянь (6.2):

$$y(t) = \lambda[s(t)],$$
$$s(t + 1) = \delta\,[s(t), x(t)]. \qquad (2.2)$$

Якщо на вхід ініціального скінченного автомата Мілі або Мура, встановленого в початковий стан s_o, подавати буква за буквою деяку послідовність букв вхідного алфавіту x_0, x_1, ..., x_n – вхідне слово, то на виході автомата будуть послідовно з'являтись букви вихідного алфавіту y_0, y_1, ..., y_n – вихідне слово. Таким чином, на рівні абстрактної теорії функціонування цифрового автомата є перетворенням вхідних слів у вихідні слова.

Поняття стану у визначенні скінченного автомата введене у зв'язку з тим, що більшість реальних процесів, якими управляють реально побудовані цифрові автомати, вимагають для свого правильного функціонування попереднього розвитку процесу в часі. Вихідний сигнал, який видає автомат у даний момент часу, визначається не тільки вхідною дією на автомат, але і станом, в якому автомат в цей момент часу знаходився. Наприклад, під час побудови автомата з оплати послуг мобільного зв'язку треба мати інформацію як про номер абонента, так і про суму внесеної оплати. Ця інформація забезпечується наявністю різних станів у скінченного автомата. Кінцеві цифрові автомати, які відповідають уведеному визначенню скінченного автомата, називають також абстрактними автоматами з пам'яттю, оскільки існування в автоматі множини різних його станів можливе тільки за наявності пам'яті в автомата.

Ряд процесів, якими управляють реальні автомати, не потребують для свого правильного функціонування знання попереднього розвитку процесу в часі. В автоматах, які управляють такими процесами, вихідний сигнал визначається тільки вхідною дією на автомат. Такі автомати називаються абстрактними автоматами з тривіальною пам'яттю або комбінаційними автоматами. Найпростішим комбінаційним автоматом можна вважати схемну реалізацію будь-якої булевої функції.

2.3 Способи задання скінченних автоматів перетворювачів

Алфавіти входів, станів і виходів автоматів задаються як звичайні множини, наприклад, переліченням їх елементів. Функції переходів і виходів можуть бути задані у вигляді матриці, графічно та аналітично. Тому будь-який скінченний автомат може бути заданий трьома способами: **у вигляді матриці, графічно та аналітично.**

Під час використання **матричного** способу автомат визначається або двома таблицями – таблицею переходів і таблицею виходів, або повною матрицею. Часто таблиці переходів і виходів об'єднуються в одну таблицю, яка називається основною таблицею скінченного автомата. Таблиця переходів задає відображення $(X \times S) \to S$, тобто визначає функцію переходів автомата. Таблиця виходів, залежно від типу автомата, який розглядається, задає відображення $(X \times S) \to Y$, або $S \to Y$, тобто визначає функцію виходів автомата.

Таблиця переходів довільного повністю визначеного скінченного автомата будується таким чином (табл. 2.1). Стовпці таблиці позначаються буквами вхідного алфавіту автомата x_{jt}, а рядки таблиці – буквами алфавіту станів автомата $s_{i(t\text{-}1)}$. У клітинці таблиці переходів, що знаходиться на перетині стовпця, позначеного вхідним сигналом x_{jt}, і рядка, позначеного станом $s_{i(t\text{-}1)}$, вказується стан, який є результатом переходу автомата із стану $s_{i(t\text{-}1)}$ під впливом вхідного сигналу x_{jt}, що визначається виразом $s_{kt} = \delta\,(s_{i(t\text{-}1)}, x_{jt})$.

Таблиця 2.1 – Таблиця переходів автоматів Мілі та Мура

Стани	Входи x_{1t}	x_{2t}
$s_{0(t\text{-}1)}$	s_{0t}	s_{1t}
$s_{1(t\text{-}1)}$	s_{1t}	s_{2t}
$s_{2(t\text{-}1)}$	s_{2t}	s_{0t}

Таблиці виходів автоматів Мілі та Мура розрізняються. Таблиця виходів повністю визначеного автомата Мілі будується таким чином (табл. 2.2). Ідентифікація стовпчиків і рядків, а також формат таблиці відповідають таблиці переходів повністю визначеного автомата. Але в клітинці таблиці виходів, що знаходиться на перетині стовпця, позначеного вхідним сигналом x_{jt}, і рядка, позначеного станом $s_{i(t\text{-}1)}$, вказується вихідний сигнал y_{kt}, який автомат Мілі формує знаходячись у стані $s_{i(t\text{-}1)}$ за наявності вхідного сигналу x_{jt}, що визначається виразом $y_{kt} = \lambda\,(s_{i(t\text{-}1)}, x_{jt})$.

Таблиця 2.2 – Таблиця виходів автомата Мілі

Стани	Виходи	
	x_{1t}	x_{2t}
$s_{0(t\text{-}1)}$	y_{0t}	y_{0t}
$s_{1(t\text{-}1)}$	y_{0t}	y_{0t}
$s_{2(t\text{-}1)}$	y_{0t}	y_{1t}

Таблиця виходів повністю визначеного автомата Мура (табл.6.3) будується так: кожному стану автомата приписується свій вихідний сигнал.

Таблиця 2.3 – Таблиця виходів автомата Мура

Стани	Виходи
$s_{0(t\text{-}1)}$	y_{0t}
$s_{1(t\text{-}1)}$	y_{0t}
$s_{2(t\text{-}1)}$	y_{1t}

Якщо автомат частково визначений, то в деяких клітинках його таблиці виходів може стояти дефіс, яким позначається відсутність вихідного сигналу. При цьому дефіс обов’язково ставиться в тих клітинках таблиці виходів, які відповідають таким же клітинкам з дефісом в таблиці переходів автомата Мілі. Останнє пов’язане з тим, що, якщо в частково визначеному автоматі Мілі є пара $s_{i(t\text{-}1)} \in S$ та $x_{jt} \in X$ і така, що перехід із стану $s_{i(t\text{-}1)}$ під дією вхідного

сигналу x_{jt} невизначений, то невизначеним є і значення вихідного сигналу на такому (неіснуючому) переході.

Таблиці переходів і виходів автомата часто поєднуються в одну таблицю, яка називається **основною таблицею** автомата. Якщо об'єднати в одну таблицю табл. 2.1 і табл. 2.2, то отримаємо основну таблицю автомата Мілі.

Таблиця 2.4 – Основна таблиця автомата Мілі

Стани	Входи	
	x_{1t}	x_{2t}
$s_{0(t\text{-}1)}$	s_{0t}/y_{0t}	s_{1t}/y_{0t}
$s_{1(t\text{-}1)}$	s_{1t}/y_{0t}	s_{2t}/y_{0t}
$s_{2(t\text{-}1)}$	s_{2t}/y_{0t}	s_{0t}/y_{1t}

Окрім розглянутих вище таблиць переходів і виходів, довільний скінченний автомат може бути описаний **повною матрицею** або матрицею з'єднань (табл. 6.5). Такий опис – один із способів матричного задання скінченних автоматів. Повна матриця довільного скінченного автомата є квадратною і має стільки стовпців (рядків), скільки різних станів має автомат, що розглядається. Кожний стовпчик (рядок) матриці з'єднань позначається буквою стану автомата. В клітинці повної матриці, яка знаходиться на перетині стовпця, позначеного буквою стану s_{jt}, і рядка, позначеного буквою стану $s_{i(t\text{-}1)}$ автомата, записується буква вхідного сигналу x_{pt} (або диз'юнкція вхідних сигналів), під дією якого (або яких) відбувається перехід автомата із стану $s_{i(t\text{-}1)}$ в стан s_{jt}. Якщо матрицею з'єднань задається скінченний автомат Мілі, то поряд з буквою вхідного сигналу x_{pt} через косу лінію вказується буква вихідного сигналу y_{kt} який автомат Мілі формує, здійснюючи перехід $s_{jt} = \delta(s_{i(t\text{-}1)}, x_{jt})$. Якщо матрицею з'єднань задається скінченний автомат Мура, то вихідними сигналами позначаються стани автомата, які ідентифікують рядки повної матриці.

Таблиця 2.5 – Повна матриця автомата Мілі

Стани автомата $s_{i(t\text{-}1)}$	Стани автомата s_{jt}		
	s_{0t}	s_{1t}	s_{2t}
$s_{0(t\text{-}1)}$	x_{1t}/y_{0t}	x_{1t}/y_{1t}	–
$s_{1(t\text{-}1)}$	–	x_{1t}/y_{0t}	x_{2t}/y_{0t}
$s_{2(t\text{-}1)}$	x_{2t}/y_{1t}	–	x_{1t}/y_{0t}

При графічному способі задання кінцеві автомати подаються орієнтованими графами: стани автомата відображаються вершинами графа, а переходи між станами – дугами між відповідними вершинами. При цьому кожній дузі графа приписується деяка буква x_i вхідного алфавіту автомата, яка вказує, що даний перехід автомата відбувається тільки при появі вхідного сигналу x_i, а кожній вершині графа – буква відповідного стану автомата. Якщо графом відображається автомат Мілі, то вихідні сигнали автомата проставляються на дугах графа (у відповідності до таблиці виходів автомата) поряд з буквою вхідного сигналу. Якщо графом відображається автомат Мура, то вихідні сигнали автомата проставляються біля вершин графа (у відповідності до таблиці виходів автомата). На рис. 2.1 подана граф-схема автомата Мілі, заданого табл. 2.5.

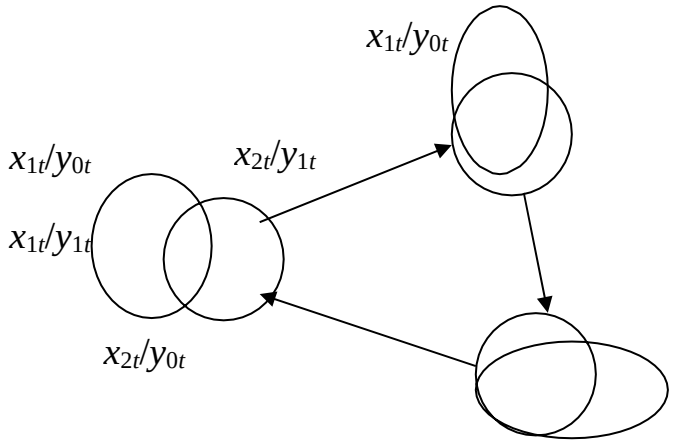


Рисунок 2.1 – Граф-схема автомата Мілі

При аналітичному способі задання кінцеві автомати подаються четвіркою об’єктів $\{X, S, Y, F\}$, де F задає для кожного стану $s_i \in S$ автомата

відображення $(X \times S) \rightarrow (S \times Y)$. Іншими словами, при аналітичному способі задання для кожного стану s_i автомата вказується відображення Fs_i, яке є множиною всіх трійок s_j, x_k, y_p, причому таких, що під дією вхідного сигналу x_k автомат здійснює перехід із стану s_i в стан s_j,, формуючи при цьому вихідний сигнал y_p. Загальне визначення скінченного автомата рівнозначне опису функцій δ і λ у відповідності до виразів: $s_j = \delta(s_i, x_k)$; $y_p = \lambda(s_i, x_k)$. Відображення Fs_i записується так:

$$Fs_i = \{s_i(x_k/y_p), s_m(x_e/y_r), \ldots\}.$$

2.4 Приклади синтезу скінченних автоматів перетворювачів

Кінцеві автомати перетворювачі найшли широке використання при виділенні слів (лексем) в реченні. Лексема (лексична одиниця мови) – це структурна одиниця мови, яка складається з елементарних символів мови і не містить у своєму складі інших структурних одиниць мови. Лексемами мов природного спілкування є слова, лексемами мов програмування є ідентифікатори, константи, ключові слова мови, знаки операцій і т. п. Склад можливих лексем кожної конкретної мови програмування визначається синтаксисом цієї мови.

Виділення меж лексем становить певну проблему. Адже у вхідному тексті програми лексеми не обмежені жодними спеціальними символами. Для більшості вхідних мов межа лексеми визначається за заданими термінальними символами. Ці символи – це пробіли, знаки операцій, а також роздільники (коми, крапки з комою і т. п.). Набір таких термінальних символів може змінюватись залежно від синтаксису вхідної мови. Крім того, важливо враховувати, що роздільник також може бути лексемою.

У КА ці дії можна відобразити порівняно просто – достатньо мати можливість з кожним переходом на графі автомата (або у функції переходу автомата) зв'язати виконання деякої довільної функції $f(x, s)$, де x – поточний

стан автомата, *s* – поточний вхідний символ. Функція може виконувати довільні дії, доступні сканеру, у тому числі працювати зі сховищами даних, які є в компіляторі (функція може бути порожньою – не виконувати жодних дій). Таку функцію, якщо вона є, зазвичай записують на графі переходів СА під дугами, що з'єднують стани СА.

Розглянемо приклад роботи СА.

2.4.1 Приклад 1

Маємо англійський текст, який містить букви *a, b, c, …, z* і пробіл. Підрахувати кількість слів, які починаються на букви *un* і закінчуються на букву *d*. Синтезувати скінченний автомат.

Визначимо множину вхідних станів (вхідну абетку) *X*. Через те що на вхід поступають 27 різних сигналів, виберемо з них значущі. Для визначення роботи автомата мають значення букви *u, n, d* і пробіл. Решту букв, які надходять на вхід, можна закодувати одним станом, оскільки реакція на них буде однакова. Закодуємо вхідні стани: x_0 – поява букви *u*, x_1 – поява букви *n*, x_2 – поява букви *d*, x_3 – поява пробілу, x_4 – поява будь-якої іншої букви. Таким чином, множина вхідних станів $X = \{x_0, x_1, x_2, x_3, x_4\}$. Визначимо вихідні стани $Y = \{y_0, y_1\}$, де y_0 – не рахувати, y_1 – рахувати. Задамо внутрішні стани: s_0 – початковий стан, s_1 – поява букви *u*, s_2 – поява букв *un*, s_3 – поява букв *un…d*. Якщо автомат знаходився в стані s_0 або s_1 і на вхід поступила буква з множини x_4, то автомат переходить в стан s_4 – очікування появи пробілу.

Оскільки кожний стан автомата неможливо зв’язати з вихідним сигналом, то маємо автомат Мілі.

Будуємо основну таблицю скінченного автомата (табл. 2.6).

Таблиця 2.6 – Основна таблиця скінченного автомата для прикладу 1

Стани	Входи				
	x_{0t}	x_{1t}	x_{2t}	x_{3t}	x_{4t}
$s_{0(t\text{-}1)}$	s_{1t} / y_{0t}	s_{4t} / y_{0t}	s_{4t} / y_{0t}	s_{0t} / y_{0t}	s_{4t} / y_{0t}
$s_{1(t\text{-}1)}$	s_{4t} / y_{0t}	s_{2t} / y_{0t}	s_{4t} / y_{0t}	s_{0t} / y_{0t}	s_{4t} / y_{0t}
$s_{2(t\text{-}1)}$	s_{2t} / y_{0t}	s_{2t} / y_{0t}	s_{3t} / y_{0t}	s_{0t} / y_{0t}	s_{2t} / y_{0t}
$s_{3(t\text{-}1)}$	s_{2t} / y_{0t}	s_{2t} / y_{0t}	s_{3t} / y_{0t}	s_{0t} / y_{1t}	s_{2t} / y_{0t}
$s_{4(t\text{-}1)}$	s_{4t} / y_{0t}	s_{4t} / y_{0t}	s_{4t} / y_{0t}	s_{0t} / y_{0t}	s_{4t} / y_{0t}

Розглянемо роботу автомата. Якщо автомат знаходився в стані $s_{0(t\text{-}1)}$ і на його вхід була подана буква *u* (стан x_0), то він перейде в стан s_{1t}. Поява на вході будь-якої іншої букви переведе його в стан s_{4t}. Якщо автомат знаходився в стані $s_{1(t\text{-}1)}$ і на його вхід була подана буква *n* (стан x_1), то він перейде в стан s_{2t}. Поява на вході будь-якої іншої букви переведе його в стан s_{4t}. Якщо автомат знаходився в стані $s_{2(t\text{-}1)}$ і на його вхід була подана буква *d* (стан x_2), то він перейде в стан s_{3t}. Поява на вході будь-якої іншої букви залишить його в стані s_2, оскільки слово буде починатись на *un*. Якщо автомат знаходився в стані $s_{3(t\text{-}1)}$ і на його вхід був поданий пробіл (стан x_3), то він перейде в стан s_{0t} і на виході з’явиться одиничний сигнал. Поява на вході будь-якої іншої букви переведе його в стан s_2, оскільки слово все також буде починатись на *un*. Якщо автомат знаходився в стані $s_{4(t\text{-}1)}$, то поява на вході будь-якої букви залишить його в цьому стані. Тільки поява пробілу переведе автомат у стан s_{0t}.

2.4.2 Приклад 2

Розглянемо виділення лексем у виразі описання масиву чисел штучної мови програмування, який містить змінну (у вигляді послідовності літер англійського алфавіту та цифр за умови, що першим символом може бути літера, знак присвоювання «=», дужки «[,]», десяткові цифри та кома «,». Допускається також, що вираз може містити пробіл. Прикладом виразу може бути такий рядок: *ident* =[1, 2, 34].

Для спрощення ситуації будемо вважати, що допустимі літери є тільки рядковими. При виділенні лексем необхідно врахувати такі ситуації. Межею лексеми можуть бути знак присвоювання «=», дужки «[,]», кома «,» та пробіл, при цьому всі перераховані символи, окрім пробілу, також є лексемами.

Визначимо X – множину вхідних станів сканера. Для визначення роботи автомата немає значення яка англійська літера або яка цифра надійшла, тому їх можливо згрупувати в два різних вхідних стани. Появу всіх інших символів, окрім символів присвоювання «=», дужки «[,]» та коми «,», пробілу будемо вважати забороненими.

Закодуємо вхідні стани: x_1 – поява пробілу, x_2 – поява будь-якої англійської літери, x_3 – поява цифри, x_4 – знаку присвоювання «=», x_5 – поява дужки «[», x_6 – поява коми «,», x_7 – поява дужки «]», x_8– поява забороненого символу. Таким чином, множина вхідних станів $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$. Визначимо вихідні стани $Y = \{y_0, y_1, y_2, y_4,\}$, де y_0 – лексема не виділена, y_1 – виділена одна лексема, y_2 – виділено дві лексеми (при появі на вході знаків =,], [, , » які одночасно є межею між лексемами і лексемами), y_3 – помилка зчитування; y_4 – виділена одна лексема і помилка. Визначимо внутрішні стани: s_0 – очікування наступної лексеми, s_1 – зчитування ідентифікатора, s_2 – зчитування числа.

Розглянемо роботу автомата (табл.2.7). Якщо автомат знаходився в стані $s_0(t\text{-}1)$ і на його вхід було подано символ (стан x_2), то він виділить його як складову лексеми ідентифікатора, перейде в стан s_{1t} і видасть на виході стан y_{0t} . Якщо автомат знаходився в стані $s_{0(t\text{-}1)}$ і на його вхід була подана цифра (стан x_3),то він перейде в стан s_{2t}. Поява на вході іншої цифри залишить його в стані s_{2t}, оскільки проходить процес визначення числа. Якщо автомат знаходився в стані $s_{2(t\text{-}1)}$ і на його вхід був поданий пробіл (стан x_1), то він перейде в початковий стан s_{0t}, а на виході буде сигнал y_{1t} , тобто буде виділена лексема: число. Якщо автомат знаходився в стані $s_{2(t\text{-}1)}$ і на його вхід було

подано кому (стан x_6), то він перейде в початковий стан s_0t, а на виході буде сигнал y_{2t}, тобто буде виділено дві лексеми: число та кома.

Таблиця 2.7 – Основна таблиця скінченного автомата для прикладу 2

Внутрішні стани	Вхідні стани							
	x_{1t}	x_{2t}	x_{3t}	x_{4t}	x_{5t}	x_{6t}	x_{7t}	x_{8t}
$S_{0\ t-1}$	S_{0t}/y_{0t}	S_{1t}/y_{0t}	S_{2t}/y_{0t}	S_{0t}/y_{1t}	S_{0t}/y_{1t}	S_{0t}/y_{1t}	S_{0t}/y_{1t}	S_{0t}/y_{3t}
$S_{1\ t-1}$	S_{0t}/y_{1t}	S_{1t}/y_{0t}	S_{1t}/y_{0t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{4t}
$S_{2\ t-1}$	S_{0t}/y_{1t}	S_{0t}/y_{1t}	S_{2t}/y_{0t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{4t}

2.4.3 Приклад 3

Розглянемо роботу СА, який на виході має булеве рішення (так, ні). Підкидається монетка, і робляться замітки при кожному другому підряд випадінні герба і кожному другому не обов'язково підряд випадінні цифри. Необхідно синтезувати скінченний автомат.

Визначимо множину вхідних станів (вхідну абетку) X = $\{x_0, x_1\}$, де x_0 – герб, x_1 – цифра. Визначимо множину вихідних станів $Y = \{y_0, y_1\}$, де y_0 – не робити замітку, y_1 – робити замітку. Виберемо внутрішні стани автомата. У відповідності до умови треба запам'ятати випадіння як герба, так і цифри на попередньому кроці. Оскільки автомат також має враховувати кожне друге не обов'язково підряд випадіння цифри, то треба ввести стан, який буде зберігати інформацію щодо випадіння підряд цифри і герба. Назвемо такий стан «цифра-герб». Таким чином, визначимо такі внутрішні стани: s_0 – початковий стан, s_1 – випадання герба, s_2 – випадання цифри, s_3 – випадання підряд цифри і герба.

Нехай в результаті підкидання монетки, отримали таку комбінацію вхідних даних (вхідна абетка): г г ц ц г ц ц ц г ц.

Розглянемо реакцію автомата на дану послідовність у вигляді таблиці-рядка (табл. 2.8).

Таблиця 2.8 – Таблиця-рядок для прикладу 1

Стани		Вхідні символи (вхідна абетка)									
		г	г	ц	ц	г	ц	ц	ц	г	ц
Вхідні	x_i	x_0	x_0	x_1	x_1	x_0	x_1	x_1	x_1	x_0	x_1
Внутрішні	s_i	s_1	s_0	s_2	s_0	s_1	s_2	s_0	s_2	s_3	s_0
Вихідні	y_i	y_0	y_1	y_0	y_1	y_0	y_0	y_1	y_0	y_0	y_1

Якщо в послідовності випадають два герби або дві цифри підряд, то автомат переходить в початковий стан, а на виході з'являється сигнал y_1. Якщо після випадіння герба випадає цифра, то автомат переходить у стан s_2 – цифра. Випадіння герба на попередньому кроці нас більше не цікавить. Зовсім інша ситуація відбувається у випадку випадіння цифри, а потім – герба. Інформація про це має бути збережена у вигляді стану s_3 – «цифра-герб». Якщо після цього стану випаде герб, то автомат збереже стан цифри – s_2. Якщо після цього випаде цифра, то автомат перейде в стан s_0. І в першому, і в другому випадку на виході з'явиться вихідний стан y_1.

Керуючись викладеними вище міркуваннями, побудуємо основну таблицю скінченного автомата для прикладу 1 (табл. 2.9).

Таблиця 2.9 – Основна таблиця скінченного автомата для прикладу 3

Стани	Входи	
	x_{1t}	x_{2t}
$s_{0(t\text{-}1)}$	s_{1t} / y_{0t}	s_{2t} / y_{0t}
$s_{1(t\text{-}1)}$	s_{0t} / y_{1t}	s_{2t} / y_{0t}
$s_{2(t\text{-}1)}$	$s_{3\,t} / y_{0t}$	s_{0t} / y_{1t}
$s_{3(t\text{-}1)}$	s_2 / y_{1t}	s_{0t} / y_{1t}

Оскільки кожному стану автомата неможливо приписати вихідний сигнал, то маємо автомат Мілі.

Побудуємо повну матрицю скінченного автомата (табл. 2.10). Дана матриця являється частково-визначеною таблицею. Відповідно до табл. 6.6 із стану $s_{0(t\text{-}1)}$ можна потрапити тільки в стан s_{1t} або s_{2t}. Тому в клітинки на перетині стовпчиків s_{0t} і s_{3t} та рядка $s_{0(t\text{-}1)}$ вписуємо дефіс. У стан s_{1t}

потрапляємо, подаючи на вхід $x_{0t.}$ На виході при цьому буде сигнал y_{0t}. Отже, на перетині рядка $s_{0(t\text{-}1)}$ і стовпчика s_{1t} маємо – $x_{0t} / y_{0t.}$ Аналогічно заповнюємо решту клітинок таблиці і будуємо граф-схему автомата (рис. 2.2).

Таблиця 2.10 – Повна матриця скінченного автомата для прикладу 1

Стани автомата $s_{i(t\text{-}1)}$	Стани автомата s_{it}			
	s_{0t}	s_{1t}	s_{2t}	s_{3t}
$s_{0(t\text{-}1)}$	–	x_{0t} / y_{0t}	x_{1t} / y_{0t}	–
$s_{1(t\text{-}1)}$	x_{0t} / y_{1t}	–	x_{1t} / y_{0t}	–
$s_{2(t\text{-}1)}$	x_{1t} / y_{1t}	–	–	x_{0t} / y_{0t}
$s_{3(t\text{-}1)}$	x_{1t} / y_{1t}	–	x_{0t} / y_{1t}	–

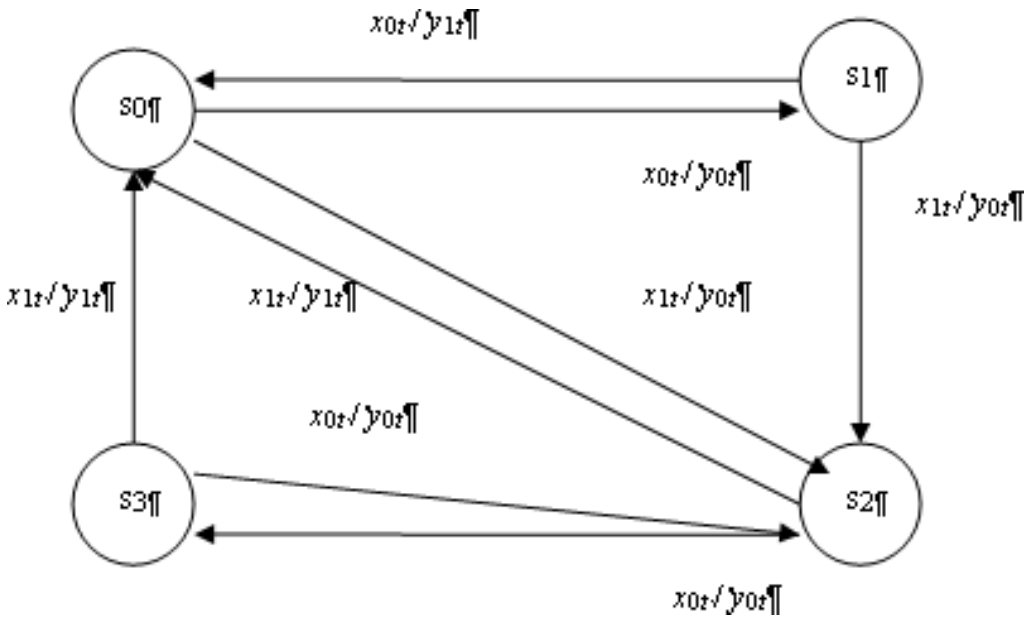


Рисунок 2 .2 – Граф-схема автомата для прикладу 1

2.5 Детермінований скінченний автомат розпізнавач

Детермінований скінченний автомат розпізнавач [14-16] є п’ятіркою, тобто описується трьома кінцевими множинами, начальним станом і двома функціями: $A = \{X, Y, S , s_0 , \delta, \lambda\}$,

де X – множина вхідних сигналів або вхідна абетка,

Y – множина вихідних сигналів або вихідна абетка,

S – множина станів або абетка станів,

δ – функція переходів, $s(t + 1) = \delta(s(t), x(t))$,

$so \in S$, початковий стан автомату.

2.6 Магазинні автомати

У роботах, пов'язаних з формальними мовами і граматиками, використовується модель скінченного автомату розпізнавача, яка називається магазинний **автомат.** Магазинний автомат (рис. 2.1) складається з вхідної стрічки, **пристрою керування** і допоміжної стрічки, названої **стеком**. **Вхідна стрічка** розділяється на клітини (позиції), у кожній з який може бути записаний символ вхідного алфавіту. При цьому передбачається, що у вільних клітинах вхідної стрічки розташовані порожні символи ε. Допоміжна стрічка також розділена на клітини, у яких можуть розташовуватися символи магазинного алфавіту. Початок допоміжної стрічки називається **дном** магазина. Зв'язок пристрою керування зі стрічками здійснюється двома **головками**, що можуть переміщатися уздовж стрічок.

Головка вхідної стрічки може переміщатися тільки в один бік – вправо чи залишатися на місці. Вона може виконувати тільки операцію читання. Головка допоміжної стрічки здатна виконувати як читання, так і запис, але ці операції пов'язані з переміщенням головки певним чином:

– при записі головка попередньо зрушується на одну позицію вверх, а потім символ заноситься на стрічку;

– при читанні символ, що знаходиться під головкою зчитується зі стрічки, а потім головка зрушується на одну позицію вниз, таким чином головка завжди встановлена проти останнього записаного символу. Позицію, що знаходиться в розглянутий момент часу під головкою, називають **вершиною** магазина.

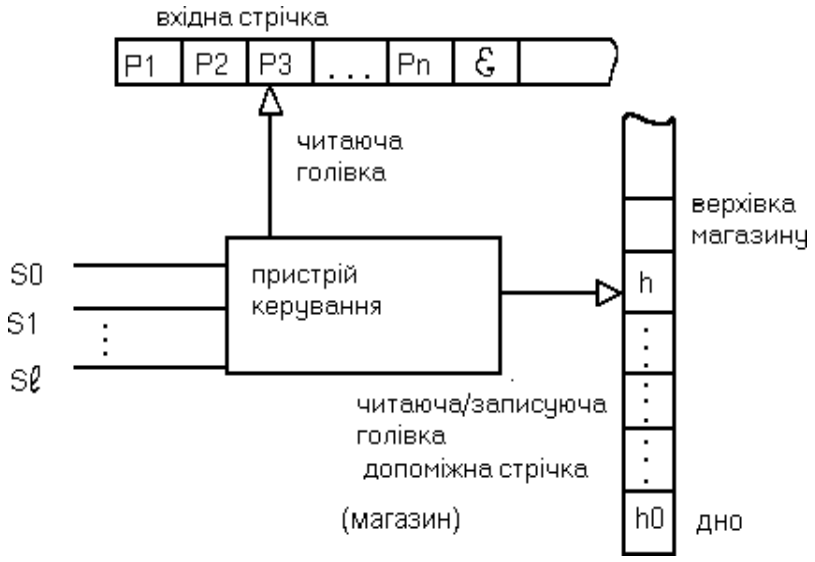


Рис 2.3 – Магазинний автомат

Магазинний автомат М визначається наступною сукупністю семи об'єктів:

$$M=\{P\ ,\ S\ ,\ s_o\ ,\ f\ ,\ F\ ,\ H\ ,\ h_o\ \},$$

де P – вхідний алфавіт, S – алфавіт станів, s_o – початковий стан, $s_o \in S$, F – множина скінченних станів , F є підмножиною S, H – алфавіт магазинних символів, записуваних на допоміжну стрічку, h_o – маркер дна, він завжди записується на дно магазина , $h_o \in H$, f – функція переходів.

Робота МА зводиться до зміни конфігурацій. Конфігурацією автомата називають трійку об'єктів (S, α, γ), де S – поточний стан автомата, α – вхідний ланцюжок, найлівіший символ a знаходиться під головкою, γ – це магазинний ланцюжок, найверхній символ якої h.

Якщо a =\$, то вважається, що вхідний ланцюжок прочитаний. Якщо γ= \$, то магазин порожній.

Робота автомата може бути подана як зміна конфігурацій. Один такт роботи автомата полягає у визначенні нової конфігурації по заданій. Це записується так:

$$(s, a\alpha , \gamma\, b) \text{ |-- } (s', \alpha , \gamma\beta .)$$

При цьому передбачається, що автомат читає символ *a*, що знаходиться під головкою, і символ *b*, що знаходиться у вершині магазина, визначає новий стан *s*' і записує ланцюжок β у магазин замість символу *b*. Якщо β =$, то верхній символ виявляється вилученим з магазину. Така зміна конфігурацій можлива, якщо функція *f*(*s*, *a*, *b*) визначена і їй належить значення (*s*', β). Описаний такт роботи виконується з переміщенням головки.

Для опису роботи автомата також буде потрібен інший вид такту, що припускає зміну станів і магазина без переміщення вхідної голівки. Якщо функція $f^*(s, a, b)$ визначена і їй належить значення (*s*', β), то такт визначає зміну конфігурацій так:

$$(s, a\alpha , \gamma\, bh) \text{ |-- } (s', a\alpha , \gamma\beta).$$

У загальному виді дії, що задаються функцією переходів і виконуються автоматом, демонструє наступна форма запису:

f(*s*, <вхідний символ>, <магазинний символ>)=(s_1, < ланцюжок, що заноситься>).

При цьому варто мати на увазі, що при виконанні такту спочатку зчитується символ з вершини, а потім на його місце заноситься новий символ чи ланцюжок.

Початковою конфігурацією називається конфігурація (s_o, α , h_o) , де s_o – початковий стан і h_o – маркер дна магазина, а **заключною** – конфігурація (*s*, $, $) , де *s* належить до множини кінцевих станів *F***.**

Для позначення послідовності змінюючих одна одну конфігурацій Домовимося використовувати знак ├*. У такий спосіб послідовність

$$(s_1, \alpha_1, \gamma_1) \vdash (s_2, \alpha_2, \gamma_2) \vdash \ldots \vdash (s_n, \alpha_n, \gamma_n)$$

записується в скороченому вигляді так:

$$(s_1,\alpha_1, \gamma_1) \vdash^* (s_n, \alpha_n,\gamma_n).$$

Контрольні запитання

1. Що таке автомат перетворювач? Як він задається?

2. Синтезувати скінченний автомат перетворювач для розливу кави. Автомат приймає купюри 1, 2, 5 гривень. Кава коштує 5 гривень. Автомат може видавати решту.

3. На вхід пристрою подаються цифри 0,1,2. Автомат видає на вихід одиничний сигнал, якщо накопичена сума вхідних сигналів дорівнює або більше 3.Синтезувати абстрактний автомат.

4. Маємо український текст. Підрахувати кількість слів, які починаються на букву «п» та закінчуються на «ов».

5. Синтезуйте абстрактний автомат для продажу проїзних квитків в метрополітені. Проїзд коштує 3 гривні. Автомат приймає монети по 50 копійок, а також 1 або 2 гривні . Автомат може видавати решту.

6. Що таке магазинний автомат? Яка його структура?

7. Яка мова вважається допустимою для магазинного автомата?

8. Який автомат називається детермінованим, а який недетермінованим?

3 СПАДНІ РОЗПІЗНАВАЧІ

Моделювання роботи недетермінованих **магазинних розпізнавачів** зв'язано з пошуком послідовності переходів з початкового в одне з кінцевих станів. Пошук складається з окремих кроків, кожний з яких може закінчитися невдало і призвести до повернення у вихідний стан для виконання наступного кроку. Такий пошук з поверненням пов'язаний зі значними витратами часу, тому на практиці використовують більш економічні детерміновані розпізнавачі, що працюють без повернень. Ці розпізнавачі допускають тільки обмежені класи КВ-мов, що відображають усі синтаксичні риси мов програмування.

Розпізнавачі можна розділити на дві категорій: **спадні (низхідні)** і **висхідні** [1,6]. Кожна категорія характеризується порядком, в якому розташовуються правила в дереві виведення. **Спадні розпізнавачі** обробляють правила зверху вниз, верхні правила раніш нижніх, у той час як висхідні аналізатори використовують нижні правила раніш тих, що розташовано вище.

У загальному випадку граматика відноситься до класу $LL(K)$ граматик, якщо для неї можна побудувати спадний детермінований розпізнавач, що враховує K вхідних символів, розташованих праворуч від поточної вхідної позиції.

Назва *LL* відбулася від слова *Left*, оскільки аналізатор переглядає вхідний ланцюжок зліва-направо, і слова *Leftmost*, оскільки він виявляє появу правила по одному чи групі символів, що утворять лівий край ланцюжка. На практиці найбільше застосування має клас $LL(1)$ граматик, для яких детермінований розпізнавач працює по одному вхідному символу, розташованому в поточній позиції. До класу $LL(1)$ граматик відносяться розділені та слабкорозділені граматики.

3.1 Розділені граматики

Контекстно-вільна граматика, яка не містить правил, що анулюють, називається розділеною, або простою, якщо виконуються наступні дві умови:

1. Права частина кожного правила починається термінальним символом;

2. Якщо два правила мають однакові ліві частини, то праві частини цих правил повинні починатися різними термінальними символами.

Важливою властивістю розділених граматик є те, що для кожної з них можна побудувати **детермінований спадний розпізнавач.**

3.2 Побудова детермінованого спадного розпізнавача

Побудова розпізнавача передбачає протиставлення кожному правилу граматики команди розпізнавача. Відповідно до загального способу побудови розпізнавача кожному правилу розділеної граматики, що мають вигляд

$$A \to a\,\alpha\,,$$

де α – ланцюжок символів повного словника і a належить термінальному словнику, потрібно поставити у відповідність команду

$$f(s_0, a, A) = (s_0, \alpha'),$$

яка визначає такт роботи розпізнавача зі зрушенням вхідної головки і у якій α' являє собою дзеркальне відображення ланцюжка α.

Крім того, варто врахувати, що термінальні символи можуть бути розташовані в правих частинах правил не тільки на самій лівій позиції. Для таких терміналів необхідно побудувати команди вигляду:

$$f(s_0, b, b) = (s_0, \$).$$

Для переходу в заключний стан додамо правило

$$f(s_0, \$, h_0) = (s_1, \$),$$

а як початкову конфігурацію розпізнавача приймемо звичайний вираз:

$$(s_0, \alpha, h_0 I),$$

де I – початковий символ граматики, а α – заданий вхідний ланцюжок.

Застосовуючи наведені вище правила, побудуємо розпізнавач для розділеної граматики $\Gamma_{3.1}$:

$$R = \{I \to abB,$$
$$I \to bBbI,$$
$$B \to a,$$
$$B \to bB\},$$
$$P = \{ a , b \},\ H = \{ a , b , I , B , h_0 \}, S = \{s_0\}, F = \{s_0\}.$$

Побудуємо команди розпізнавача:

$$f(s_0 , a , I) = (s_0 , Bb)$$
$$f(s_0 , a , B) = (s_0 , \$)$$
$$f(s_0 , b , I) = (s_0 , I\ b\ B$$
$$f(s_0 , b , B) = (s_0 , B)$$
$$f(s_0 , b , b) = (s_0 , \$)$$
$$f(s_0 , \varepsilon , h_0) = (s_0 , \$).$$

Роботу побудованого автомата покажемо на прикладі аналізу ланцюжка *bbabab*: $(s_0 , bbababa , h_0I) \vdash (s_0 , bababa , h_0IbB) \vdash (s_0 , ababa , h_0IbB) \vdash (s_0 , baba , h_0Ib) \vdash (s_0 , aba , h_0I) \vdash (s_0 , ba , h_0Bb) \vdash (s_0 , a , h_0B) \vdash (s_0 , \$, h_0) \vdash (s_0 , \$, \$)$.

Наведена послідовність конфігурацій показує, що в кожній конфігурації може бути застосована єдина команда розпізнавача, тому такий розпізнавач називається **детермінованим**.

3.2.1 Множина ВИБІР(μ)

Інший клас граматик, що породжують детерміновані мови, називається **слабкорозділеними граматиками**. Цей клас відрізняється від класу розділених граматик тим, що він допускає використання правил, які анулюють, у схемі граматики. Однак, не всяка розділена граматика з правилами, що анулюють, відноситься до класу слабкорозділених граматик. Щоб сформулювати визначення, яке дозволяє пізнавати слабкорозделені і

LL(1) граматики, нам будуть потрібні нові поняття: множина ВИБІР(μ), функції ПЕРШ(μ) і СЛІД(А), μ – ланцюжок із термінальних та не термінальних символів, $A - \in V_A$ [6].

Множина ВИБІР(μ) будується для кожного правила і включає ті термінальні символи, з появою яких під читаючою голівкою розпізнавач повинний застосовувати це правило.

Для визначення множини ВИБІР(μ) використовуються функції ПЕРШ(μ) і СЛІД(А). Аргументом функції ПЕРШ(μ) може бути будь-який ланцюжок повного словника μ, а значенням функції ПЕРШ(μ) є множина термінальних символів, що можуть стояти на першому місці в ланцюжках, виведених з ланцюжка μ.

3.2.2 Побудова функції ПЕРШ(μ)

Значення функції ПЕРШ(μ) можна визначити користуючись наступними правилами:

1) Якщо ланцюжок μ починається термінальним символом і має вигляд $\mu \rightarrow b\mu'$, то функція ПЕРШ(μ) визначається як:

$$\text{ПЕРШ}(\mu \rightarrow b\mu') = \{b\}.$$

2) Якщо ланцюжок μ є порожнім ланцюжком, $\mu \rightarrow \$$, то функція ПЕРШ(μ) визначається як:

$$\text{ПЕРШ}(\mu \rightarrow \$) = \$.$$

3) Якщо ланцюжок μ починається нетермінальним символом ***B*** і має вигляд $\mu \rightarrow B\mu'$, а в схемі граматики є *n* правил, у будь-якій частині яких знаходиться символ *B*:

$$B \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n ,$$

і, якщо не існує виведення $B \rightarrow \$$, то функція **ПЕРШ**$(\mu \rightarrow B\mu')$ є об'єднанням множин:

$$\text{ПЕРШ}(\mu \rightarrow B\mu') = \text{ПЕРШ}(\alpha_1) \cup \text{ПЕРШ}(\alpha_2) \cup \ldots \cup \text{ПЕРШ}(\alpha_n).$$

4) Якщо ланцюжок μ починається нетермінальним символом і має

вигляд, μ→Bμ’ , а в схему граматики входять *n* правил вигляду:

$$B \to \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_n ,$$

і *B* є нетерміналом, що анулює, тобто існує *B*→**$**, то функція **ПЕРШ**(μ→*B*μ') є об'єднанням множин:

ПЕРШ(μ→*B*μ'))= ПЕРШ(μ') ПЕРШ(α_1) ∪ ПЕРШ (α_2) ∪...∪ПЕРШ(α_n).

Як приклад виконаємо обчислення функції ПЕРШ для правил наступної граматики $\Gamma_{3.2}$:

1. $A \to BCc$,
2. $A \to gDB$,
3. $B \to \$$,
4. $B \to bCDE$,
5. $C \to DaB$,
6. $C \to ca$,
7. $D \to \$$,
8. $D \to dD$,
9. $E \to gAf$,
10. $E \to c$

Спочатку знайдемо значення функції для правих частин правил (2), (4), (6), (8), (9) , (10) , що починаються термінальними символами:

ПЕРШ(*gDB*) = {*g*}

ПЕРШ(*bCDE*) = {*b*}

ПЕРШ(*ca*) = {*c*}

ПЕРШ(*dD*) = {*d*}

ПЕРШ(*gAf*) {*g*}

ПЕРШ(*c*) = {*c*}

Потім обчислимо функцію для правил (5) і (6) :

ПЕРШ (*C*) = ПЕРШ (*DaB*) ∪ ПЕРШ (*ca*).

З огляду на те, що *D* є нетерміналом, що анулює, одержуємо:

ПЕРШ(*C*) = ПЕРШ(*aB*) ∪ ПЕРШ(*D*) ∪{*c*} = {*a*}∪{*d*}∪{*c*}={*a, d, c*}.

При обчисленні функції для правил (1) і (2) також необхідно врахувати те, що *B* є терміналом, що анулює, тому маємо:

ПЕРШ(*A*) = ПЕРШ(*BCc*) ∪ ПЕРШ(*gDB*) =

ПЕРШ(*Cc*) ∪ ПЕРШ(*B*) ∪ ПЕРШ(*gDB*) = {*a,d,c*} ∪ {*b*} ∪{*g*} = {*a,b,c,d,g*}.

3.2.3 Побудова функції СЛІД(А)

Аргументом функції **СЛІД**(А) є нетермінальний символ, наприклад *A*, а значення функції **СЛІД**(*A*) є множина терміналів, що можуть виходить безпосередньо за нетерміналом *B* у ланцюжках, виведених з початкового символу граматики.

Обчислення значення функції СЛІД(*B*) повинне виконуватися за наступними правилами.

1) Якщо в схемі граматики є правила вигляду

$$X_1 \rightarrow \mu_1 B\, \alpha_1,\ X_2 \rightarrow \mu_2 B\, \alpha_2,\ \ldots,\ X_n \rightarrow \mu_n B\, \alpha_n,$$

і не існує правил $\alpha_i \rightarrow \$$, то

$$\text{СЛІД}(B) = \text{ПЕРШ}(\alpha_1) \cup \text{ПЕРШ}(\rightarrow\alpha_2) \cup \ldots \cup \text{ПЕРШ}(\alpha_n).$$

2) Якщо ж серед приведених вище правил існує хоча б один ланцюжок $\alpha_i \rightarrow \$$, (наприклад, нехай $\alpha_i \rightarrow \$$, то функція обчислюється так

$$\text{СЛІД}(B) = \text{СЛІД}(X_1) \cup \text{ПЕРШ}(\alpha_2) \cup \ldots \cup \text{ПЕРШ}(\alpha_n).$$

Виконаємо обчислення функції СЛІД(А) для нетерміналів граматики $\Gamma_{3..2}$ Спочатку визначимо функцію для нетермінала *A*, що зустрічається в правій частині правила (9).

$$\text{СЛІД}(A) = \text{ПЕРШ}(f) = \{f\}.$$

Нетермінал *C* входить у праві частини правил (1) і (4), з огляду на те, що нетермінал *D* є анулюючим, одержуємо:

$$\text{СЛІД}(C) = \text{ПЕРШ}(D) \cup \text{ПЕРШ}(E) \cup \text{ПЕРШ}(c) = \{c,d,g\}.$$

Нетермінал *B* входить у праві частини правил (1), (2), (5), тому маємо

$$\text{СЛІД}(B) = \text{ПЕРШ}(Cc) \cup \text{СЛІД}(A) \cup \text{СЛІД}(C).$$

Підставляючи в отриманий вираз значення функцій, що входять у праву частину, одержуємо

$$\text{СЛІД}(B) = \{\ a, c, d, \} \cup \{\ f\ \} \cup \{\ c, d, g, \} = \{\ a, c, d, g, f\ \}.$$

Для нетермінала *D*, що входить у правила (2), (4) , (5) і (8), врахувати те, що нетермінал *B* є анулюючим, одержуємо:

СЛІД(D) =ПЕРШ(B) ∪ СЛІД(A) ∪ ПЕРШ(E) ∪ ПЕРШ(aB),

З огляду на те, що, для нетермінала E, який входить у правило (4)

$$\text{СЛІД}(E) = \text{СЛІД}(B) = \{a,d,c,g,f\},$$

остаточно маємо

$$\text{СЛІД}(D) = \text{ПЕРШ}(B) \cup \text{СЛІД}(A) \cup \text{ПЕРШ}(E) \cup \{a\} =$$

$$= \{b\} \cup \{f\} \cup \{c, g\} \cup \{a\} = \{a, b, c, g, f\}.$$

3.2.4 Побудова множини ВИБІР(μ)

Множину **ВИБІР**(μ), яка буде потрібна нам для побудови функції переходів магазинних автоматів, можна визначити за допомогою функцій ПЕРШ(μ) і СЛІД(А) у такий спосіб:

1) Якщо правило граматики має вигляд $B \to \alpha$ і α не є анулюючим ланцюжком, іншими словами не існує виведення $\alpha \to \$$, то

$$\text{ВИБІР}(B \to \alpha) = \text{ПЕРШ}(\alpha).$$

2) Для правил граматики, що анулюють, вигляду $B \to \$$ множина **ВИБІР**(μ) визначається так

$$\text{ВИБІР}(B \to \$) = \text{СЛІД}(B).$$

3) Якщо правило граматики має вид $B \to \mu$ і μ є ланцюжком, що анулює, то

$$\text{ВИБІР}(B \to \mu) = \text{ПЕРШ}(\mu) \cup \text{СЛІД}(B).$$

Для розглянутої граматики множина **ВИБІР**(μ) для кожного з правил, побудованих описаним вище способом, має вигляд:

$$\text{ВИБІР}(A \to BCc) = \text{ПЕРШ}(BCc) = \{a,b,c,d\},$$

$$\text{ВИБІР}(A \to gDB) = \text{ПЕРШ}(gDB) = \{g\},$$

$$\text{ВИБІР}(B \to \$) = \text{ПЕРШ}(\$) \cup \text{СЛІД}(B) = \{a,c,d,g,f\},$$

$$\text{ВИБІР}(B \to bCDE) = \text{ПЕРШ}(bCDE) = \{b\},$$

$$\text{ВИБІР}(C \to DaB) = \text{ПЕРШ}(DaB) = \{a,d\},$$

$$\text{ВИБІР}(C \to ca) = \text{ПЕРШ}(ca) = \{\text{c}\},$$

ВИБІР($D \rightarrow \$$) = ПЕРШ(\$)$\cup$ СЛІД(D) = {a,b,c,g,f},

ВИБІР($D \rightarrow dD$) = ПЕРШ(dD) = {d},

ВИБІР($E \rightarrow gAf$) = ПЕРШ(gAf) = {g},

ВИБІР($E \rightarrow c$) = ПЕРШ(c) = {c}.

3.3 Слабкорозділені граматики

Використовуючи введені поняття, можна дати визначення слабкорозділеної граматики.

КВ-граматика називається **слабкорозділеною**, якщо виконуються наступні три умови:

– права частина кожного правила являє собою або порожній ланцюжок \$, або починається з термінального символу;

– якщо два правила мають однакові ліві частини, то праві частини правил повинні починатися різними символами;

– множина ВИБІР, побудована для правил з однаковою лівою частиною, не містить однакових елементів.

3.4 *LL*(1)–граматики

Розділені і слабкорозделені граматики є підкласом граматик більш загального виду [5,9], що називаються *LL*(1) граматиками. Граматика називається *LL*(1) граматикою за умов:

– права частина кожного правила являє собою або порожній ланцюжок \$, або починається з термінального або нетермінального символу;

– якщо два правила мають однакові ліві частини, то праві частини правил повинні починатися різними символами;

– множина ВИБІР, побудована для правил з однаковою лівою частиною, не містить однакових елементів.

3.5 Побудова магазинного автомата

Для граматик, що задовольняють умовам *LL*(1) граматик, справедливе наступне твердження: для кожної *LL*(1) граматики можна побудувати детермінований магазинний автомат М, що допускає мова, породжувана даною граматикою

$$L\ (\Gamma) = L\ (\text{M}).$$

Задача побудови магазинного автомата для заданої *LL*(1) граматики визначається в такий спосіб.

Задано граматику $\Gamma = \{V_\text{T}, V_\text{A}, I, R\}$, і потрібно визначити об'єкти, що визначають автомат $\text{M} = \{ P , S , s_0 , F , H , h_0 , f \}$.

З огляду на те, що автомат повинен допускати ланцюжки мови, породжуваної заданою граматикою, приймемо $P = V_\text{т}$. Щоб спростити опис автомата, припустимо, що він має один стан s_0, що є і початковим і заключним, тобто – $S = \{s\}$ і $F = \{s\}$.

Як магазинний алфавіт приймемо наступне об'єднання

$$H = \{V_\text{т} \cup V_\text{A} \cup h_0\}.$$

Побудову функції переходів виконаємо з використанням множин **ВИБІР**(μ) правил заданої граматики:

1) Для кожного правила граматики, що починається термінальним символом вигляду $A \rightarrow b\ \alpha$, побудуємо команду автомата

$$f (s , b , A) = (s , \alpha') ,$$

де α' є дзеркальним відображенням ланцюжка α.

2) Для кожного правила граматики, що починається нетермінальним символом вигляду $A \rightarrow B\alpha$ побудуємо команду автомата

$$f^* (s , x , A) = (s , B\alpha')$$

де f^* – команда автомата без зрушення вхідної головки, а α' є дзеркальним відображенням ланцюжка α**,** x – елемент множини **ВИБІР**(μ) для поточного правила. Кількість команд, які необхідно побудувати для заданого правила,

визначається числом елементів множини **ВИБІР**(μ), визначеної для цього правила. Команди магазинного автомата, що працюють без зрушення вхідної головки, виконують заміну нетермінала, що відповідає лівій частині правила, ланцюжком, що відповідає правій частини цього правила. У цьому випадку зіставлення термінального символу, одержуваного при виводі, і чергового символу вхідного ланцюжка не робиться, тому для таких команд зрушення вхідної голівки не потрібне.

3) Для кожного правила граматики, що анулює, вигляду $A \rightarrow \$$ побудуємо команди автомата:

$$f^*(s, x, A) = (s, \$)$$

для кожного елемента x з множини **ВИБІР**($A \rightarrow \$$). Кількість таких команд визначається числом елементів множини ВИБІР.

4) Для кожного термінального символу $\boldsymbol{b}$, розташованого в середині або на кінці правих частин правил граматики, побудуємо команду:

$$f(s_0, b, b) = (s, \$).$$

5) Для переходу в заключний стан побудуємо команду:

$$f^*(s, \$, h_0) = (s, \$, \$).$$

Як початкову конфігурацію автомата домовимося використовувати наступний вираз з заданим вхідним ланцюжком μ

$$(s, \mu, h_0 I).$$

3.6 Приклади побудови спадного розпізнавача

3.6.1 Приклад побудови розпізнавача для граматики $Г_{3.3}$

Побудувати спадний розпізнавач для оператора присвоювання арифметичного виразу. Арифметичний вираз містить ідентифікатори: i, :=, (,), ; . Кількість вкладених дужок не обмежена.

Маємо наступні приклади ланцюжків:

$i=(i+i)+(i+(i+i+i))$;

i=*i*+(*i*+*i*+*i*+*i*);

i=(*i*+*i*+*i*+*i*+*i*)+*i*+(*i*+*i*+*i*);

Введемо наступні нетермінальні символи (рис. 3.1): I – початковий символ граматики; *S* – список, де у якості елемента списку виступає змінна *i*, знак + виступає у якості роздільника. Нетермінал *R* будемо використовувати для рекурсії елементів списку. Список *S* може бути обмежений дужками.

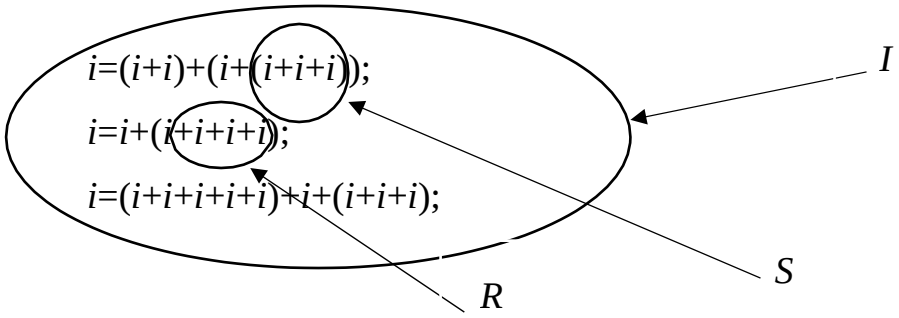


Рисунок 3.1 – Визначення нетермінальних символів для граматики $Г_{3.3}$

1. Будуємо правила граматики і отримуємо граматику $Г_{3.3}$:

1) $I \rightarrow i=S$;

2) $S \rightarrow iR$

3) $S \rightarrow (S)R$

4) $R \rightarrow +S$

5) $R \rightarrow$ \$.

2. Визначаємо, чи містить наша граматика нетвірні символи: відшукуємо правило, в правій частині якого розташовано термінал або \$.

$\{R\}$ – продуктивний символ. Відшукуємо правила, праві символи яких чи лівий символ вже занесені в список. Якщо так, то додаємо в список нетерміналів:

$\{R\}$

$\{R,S\}$

$$\{R,S,I\}$$

У список потрапили всі нетермінали, отже непродуктивних символів немає.

3. Шукаємо недосяжні символи.

Заносимо в список початковий символ граматики $\{I\}$.

Шукаємо правило, ліва частина якого вже занесена в список. Додаємо нетермінали правої частини:

$$\{I, S\}$$

$$\{I, S,R\}$$

У список потрапили всі нетермінали, отже недосяжних символів немає.

4. Дана граматика містить правила, що анулюють, та правила, права частина котрих починається з нетермінала, отже вона не може бути розділеною та слабкорозділеною граматикою. Визначимо належність даної граматики до $LL(1)$ – граматики.

Аналізуючи складені нами правила, дійдемо висновку, що ліворекурсивних правил немає.

Побудуємо функцію ВИБІР:

1. ВИБІР($I\rightarrow$i=S;)= ПЕРШ($I\rightarrow$i=S;)= {i},
2. ВИБІР($S\rightarrow iR$)={i},
3. ВИБІР($R\rightarrow$+S)={+},
4. ВИБІР($S\rightarrow$(S)R)={(},
5. ВИБІР($R\rightarrow$\$)=СЛІД(R)$\cup${\$}=СЛІД(S}={;,)}.

Дана граматика є $LL(1)$ – граматикою, так як множина ВИБІР для правил, що починаються з однакових терміналів, не містить однакових символів.

6. Будуємо команди розпізнавача:

1) $f(s, i, I)=(s, ;S:=)$
2) $f(s, i, S)=(s, R)$
3) $f(s, +, R)=(s, S)$
4) $f(s, (, S)=(s ,R)S)$
5) $f^*(s, ;, R)=(s, \$)$
6) $f^*(s,), R)=(s, \$)$
7) $f(s ;, ;)=(s, \$)$
8) $f(s,),))=(s, \$)$
9) $f(s, =, =)=(s, \$)$
10) $f(s, \$, h_0)=(s, \$)$

7. Далі відповідно до побудованих правил розпізнаємо заданий ланцюжок:

$(s, i=i+(i+i); , h_0 I)$ ├ 1
$(s, =i+(i+i);, h_0;S=)$ ├ 9
$(s, i+(i+i); , h_0;S)$ ├ 2
$(s, +(i+i);, h_0;R)$ ├ 3
$(s, (i+i);, h_0;S)$ ├ 4
$(s, i+i);, h_0;R)S)$ ├ 2
$(s, +i);, h_0;R)R)$ ├ 3
$(s, i);, h_0;R)S)$ ├ 2
$(s,);, h_0;R)R)$ ├ 6
$(s,);\ h_0;R))$ ├9
$(s, ;, h_0;R)$ ├ 5
$(s, ; , h_0;)$ ├ 7
$(s, \$, h_0)$

Ланцюжок розпізнано.

3.6.2 Приклад побудови розпізнавача для граматики $Г_{3.4}$

Побудувати правила граматики для нижченаведених прикладів ланцюжків коду програми. Перевірити граматику на наявність непродуктивних та недосяжних символів**.** Побудувати спадний розпізнавач.

Приклади ланцюжків:

{*a*++; *b*--; --a; *a*++;}

{ *b*--; *a*++; }

{*a*--; *b*--; *a*++; ++a;}

Проаналізувавши приклади ланцюжків(рис. 3.2), можливо помітити наступне. В прикладах має місце список змінних a та *b* з операцією інкремент «++» та декремент «--», причому операція може бути розташована до або після змінних (преінкремент або постінкремент). Позначимо змінні a та *b* нетермінальним символом A, а операції інкремент «++» та декремент «--» –

символом B. Тоді I – початковий символ граматики, *S* – це елемент списку. Символ *R* будемо використовувати для рекурсії.

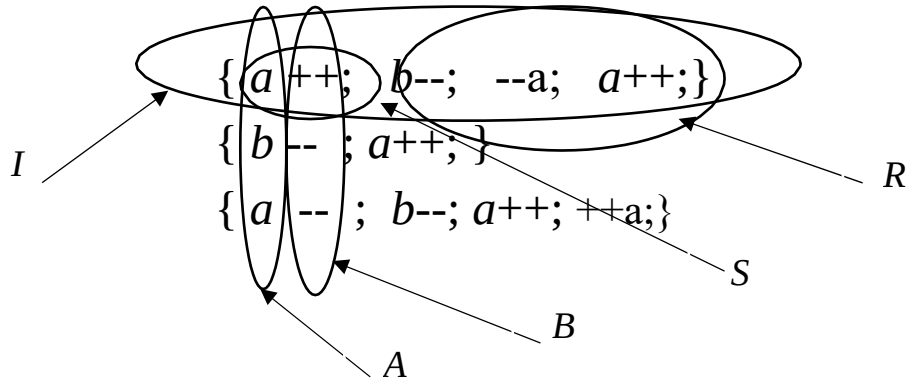


Рисунок 3.2 – Визначення нетермінальних символів для граматики $\Gamma_{3.4}$

Отримаємо такі правила граматики

$\Gamma_{3.4}$: $R = \{1.\ I \rightarrow \{SR\}$

$2.\ R \rightarrow SR|\$$

$3.\ S \rightarrow AB;|BA;$

$4.\ A \rightarrow a|b$

$5.\ B \rightarrow ++|--\ \}.$

Перевіримо правила граматики для фрагменту коду програми {*a*++; --*b*:} за допомогою виведення:

$I \rightarrow \{SR\} \Rightarrow \{AB;R\} \Rightarrow \{aB;R\} \Rightarrow \{a{+}{+};R\} \Rightarrow \{a{+}{+};SR\} \Rightarrow \{a{+}{+};BA;R\} \Rightarrow \{a{+}{+};\ {-}{-}A;R\ \} \Rightarrow \{a{+}{+};\ {-}{-}b;R\ \} \Rightarrow \{a{+}{+};\ {-}{-}b;\ \}.$

Синтаксичний розбір для даного прикладу має такий вигляд:

[1, 3.1, 4.1, 5.1, 2.2, 3.2, 5.2, 4.2, 2.2]

Перевіримо граматику на наявність непродуктивних символів:

1) *S,A,B*
2) *S,A,B,R*
3) *S,A,B,R,I*

Непродуктивних символів немає.

Перевіримо граматику на наявність недосяжних символів.

Пошук недосяжних символів:

1) *I*

2) I,S,R

3) I,S,R,A,B

Недосяжних символів немає.

Для знаходження функцій ПЕРШ(μ), СЛІД(А) та ВИБІР(μ) ДомовимосяДомовимося у якості аргументу функцій використовувати номер правила.

Знайдемо функцію ПЕРШ(μ):

ПЕРШ(1)={{}

ПЕРШ(2.1)= ПЕРШ(*S*)= ПЕРШ(*A*) *U* ПЕРШ(*B*)={+,-,*a*,*b*}

ПЕРШ(2.2)={$}

ПЕРШ(3.1)= ПЕРШ(*A*)={*a*,*b*}

ПЕРШ(3.2)= ПЕРШ(*B*)={+,-}

ПЕРШ(4.1)={*a*}

ПЕРШ(4.2)={*b*}

ПЕРШ(5.1)={+}

ПЕРШ(5.2)={-}

Знайдемо функцію СЛІД(А) :

СЛІД(*S*)={+,-,*a*,*b*,}}

СЛІД(*R*)={}}

СЛІД(*A*)={+,-,;}

СЛІД(*B*)={;, *a*, *b*}

Знайдемо множину ВИБІР(μ):

ВИБІР(1)={{}

ВИБІР (2.1)= Перш(*S*)={+,-,*a*,*b*}

ВИБІР (2.2)= СЛІД(*R*)={}}

ВИБІР (3.1)= Перш(*A*)={*a*,*b*}

ВИБІР (3.2)= Перш(*B*)={+,- }

ВИБІР (4.1)={*a*}

ВИБІР (4.2)={*b*}

ВИБІР (5.1)={+}

ВИБІР (5.2)={-}

Елементи множини ВИБІР(μ) для правил з однаковим лівим нетермінальним символом не перетинаються (правила 2.1 та 2.2, 3.1 та 3.2, 4.1 та 4.2, 5.1 та 5.2). Отже, дана граматика є *LL*(1) граматикою.

Будуємо функцій переходів магазинного автомату:

1. $f(s, \{ , I) = (s, \}RS)$
2. $f(s, \} , \}) = (s, \$)$
3. $f^*(s, + , R) = (s, RS)$
4. $f^*(s, - , R) = (s, RS)$
5. $f^*(s, a , R) = (s, RS)$
6. $f^*(s, b , R) = (s, RS)$
7. $f^*(s, \} , R) = (s, \$)$
8. $f^*(s, b , S) = (s, ;BA)$
9. $f^*(s, a , S) = (s, ;BA)$
10. $f(s, ; , ;) = (s, \$)$
11. $f^*(s, -, S) = (s, ;AB)$
12. $f^*(s, + , S) = (s, ;AB)$
13. $f(s, a , A) = (s, \$)$
14. $f(s, b , A) = (S, \$)$
15. $f(s, + , B) = (s, +)$
16. $f(s, + , +) = (s, \$)$
17. $f(s, - , B) = (s, -)$

18. $f(s, -, -) = (s, \$)$

19. $f^*(s, \$, h_0) = (s, \$)$.

Виконаємо розпізнавання ланцюжка: {*a*++;--*b*;}.

(*s*, {*a*++;--*b*;}, h_0I) ├ 1

(*s*, *a*++;--*b*;}, h_0}*RS*) ├ 9

(*s*, *a*++;--*b*;}, h_0}*R*;*BA*) ├ 13

(*s*, ++;--*b*;}, h_0}*R*;*B*) ├ 15

(*s*, +;--*b*;}, *h*0}*R*;+) ├ 16

(*s*,;--*b*;}, h_0}*R*;) ├ 10

(*s*, --*b*;}, h_0}*R*) ├ 4

(*s*, --*b*;}, h_0}*RS*) ├ 11

(*s*, --*b*;}, h_0}*R*;*AB*) ├ 17

(*s*, -*b*;}, h_0}*R*;*A*-) ├ 18

(*s*, *b*;}, *h*0}*R*;*A*) ├ 14

(*s*,;}, h_0}*R*;) ├ 10

(*s*, }, h_0}*R*) ├ 7

(*s*, }, h_0}) ├ 2

(*s*, $, h_0) ├ 19

(*s*, $, $)

Ланцюжок розпізнано.

3.6.3 Приклад побудови розпізнавача для граматики $\Gamma_{3.5}$

Побудувати правила граматики для нижченаведених прикладів ланцюжків коду програми. Перевірити граматику на наявність непродуктивних та недосяжних символів. Побудувати спадний розпізнавач.

Приклади ланцюжків:

int **a*[3]={}, **bc[2]* = {3, 58};

int **d*[3] = {1, 111, 1111};

Після аналізу ланцюжків отримаємо граматику $\Gamma_{3.5}$ з наступними правилами:

1. $I \rightarrow int\ BP;$
2. $P \rightarrow ,BP \mid \$$
3. $B \rightarrow *CT[DM]=\{KL\}$
4. $K \rightarrow DM|\$$
5. $D \rightarrow 1 \mid ...|9$
6. $M \rightarrow DM|0M|\$$
7. $L \rightarrow ,KL|\$$
8. $C \rightarrow a| \ldots | z$
9. $T \rightarrow CT|\$$

Дана граматика містить анулюючі правила, права частина правил починається з термінальних та нетермінальних символів, отже граматика не може бути розділеною та слаборозділеною граматикою. Перевіримо належність граматики до класу $LL(1)$ граматики. Знайдемо функції ПЕРШ(μ), СЛІД(А) і множину ВИБІР(μ).

Для спрощення, як аргумент будемо вказувати номер правила.

Знайдемо функцію ПЕРШ(μ) :

ПЕРШ (1) = {*int*}

ПЕРШ (2.1) = {,}

ПЕРШ (2.2) = {$}

ПЕРШ (3) = { * }

ПЕРШ (4.1) = ПЕРШ (*D*) = {1-9}

ПЕРШ (4.2) = {$}

ПЕРШ (5.1) = { 1 }

…

ПЕРШ (5.9) = { 9 }

ПЕРШ (6.1) = ПЕРШ (*D*) = {1-9}

ПЕРШ (6.2) = {0}

ПЕРШ (6.3) = {$}

ПЕРШ (7.1) = {,}

ПЕРШ (7.2) = {$}

ПЕРШ (8.1) = {*a*}

…

ПЕРШ (8.26) = {*z*}

ПЕРШ (9.1) = ПЕРШ (*C*) = {*a*-*z*}

ПЕРШ (9.2) = {$}

Знайдемо функцію СЛІД(А):

СЛІД (*B*) = ПЕРШ (*P*) ∪ {;} = {, , ;}

СЛІД (*P*) = {;}

СЛІД (*K*) = ПЕРШ (*L*) ∪ { } }= {, , } }

СЛІД (*D*) = ПЕРШ (*M*) ∪ {*]*} *v* ПЕРШ (*D*) ∪ СЛІД (*K*)= {1-9, 0, *]*, , , } }

СЛІД (*M*) = СЛІД (*K*) ∪ {*]*} = ПЕРШ (*L*) ∪ {*]*} = {, , }, *]* }

СЛІД (*L*) = { } }

СЛІД (*C*) = ПЕРШ (*T*) = ПЕРШ (*C*) *v* {*[*} = {*a*-*z*, *[* }

СЛІД (*T*) = { *[* }

Знайдемо множину ВИБІР(μ) :

ВИБІР (1) = {*int*}

ВИБІР (2.1) = {,}

ВИБІР (2.2) = СЛІД (*P*) = {;}

ВИБІР (3) = { * }

ВИБІР (4.1) = {1-9}

ВИБІР (4.2) = СЛІД (*K*) = ПЕРШ(*L*) ∪ { } }= {, , } }

ВИБІР (5.1) = { 1 }

ВИБІР (5.9) = { 9 }

ВИБІР (6.1) = ПЕРШ (*D*) = {1-9}

ВИБІР (6.2) = {0}

ВИБІР (6.3) = СЛІД (*K*) ∪ {*]*} = ПЕРШ(*L*) *v* {*]*} = {, , }, *]* }

ВИБІР (7.1) = {,}

ВИБІР (7.2) = { } }

ВИБІР (8.1) = {*a*}

…

ВИБІР (8.26) = {*z*}

ВИБІР (9.1) = ПЕРШ (*C*) = {*a-z*}

ВИБІР (9.2) = { *[* }

Елементи множини ВИБІР(μ) для правил з однаковим лівим нетермінальним символом не перетинаються. Отже, дана граматика є *LL*(1) граматикою.

Будуємо функцій переходів магазинного автомату (розпізнавача):

1. *f*(*s*, *i*, *I*) = (*s*, ;*PBtn*)
2. *f*(*s*, *n*, *n*) = (*s*, $)
3. *f*(*s*, *t*, *t*) = (*s*, $)
4. *f*(*s*, ;, ;) = (*s*, $)
5. *f*(*s*, ,, *P*) = (*s*, *PB*)
6. *f**(*s*, ;, *P*) = (*s*, $)
7. *f*(*s*, *, *B*) = (*s*, }*LK*{*=]MD[TC*)
8. *f*(*s*, [, [) = (*s*, $)
9. *f*(*s*,],]) = (*s*, $)
10. *f*(*s*, =, =) = (*s*, $)
11. *f*(*s*, {, {) = (*s*, $)

12. $f(s, \}, \}) = (s, \$)$
13. $f^*(s, 2, K) = (s, MD)$
14. $f^*(s, 4, K) = (s, MD)$
15. $f^*(s, 3, K) = (s, MD)$
16. $f^*(s, 9, K) = (s, MD)$
17. $f^*(s, 7, K) = (s, MD)$
18. $f^*(s, 8, K) = (s, MD)$
19. $f^*(s, 5, K) = (s, MD)$
20. $f^*(s, 1, K) = (s, MD)$
21. $f^*(s, 6, K) = (s, MD)$
22. $f^*(s, \textbf{,}, K) = (s, \$)$
23. $f^*(s, \}, K) = (s, \$)$
24. $f(s, 1, D) = (s, \$)$
25. $f(s, 2, D) = (s, \$)$
26. $f(s, 3, D) = (s, \$)$
27. $f(s, 4, D) = (s, \$)$
28. $f(s, 5, D) = (s, \$)$
29. $f(s, 6, D) = (s, \$)$
30. $f(s, 7, D) = (s, \$)$
31. $f(s, 8, D) = (s, \$)$
32. $f(s, 9, D) = (s, \$)$
33. $f^*(s, 2, M) = (s, MD)$
34. $f^*(s, 4, M) = (s, MD)$
35. $f^*(s, 3, M) = (s, MD)$
36. $f^*(s, 9, M) = (s, MD)$
37. $f^*(s, 7, M) = (s, MD)$
38. $f^*(s, 8, M) = (s, MD)$
39. $f^*(s, 5, M) = (s, MD)$
40. $f^*(s, 1, M) = (s, MD)$

41. $f^*(s, 6, M) = (s, MD)$
42. $f(s, 0, M) = (s, M)$
43. $f^*(s, ,, M) = (s, \$)$
44. $f^*(s,], M) = (s, \$)$
45. $f^*(s, \}, M) = (s, \$)$
46. $f(s, ,, L) = (s, LK)$
47. $f^*(s, \}, L) = (s, \$)$
48. $f(s, a, C) = (s, \$)$
49. $f(s, b, C) = (s, \$)$
50. $f(s, c, C) = (s, \$)$
51. $f(s, d, C) = (s, \$)$
52. $f(s, e, C) = (s, \$)$
53. $f(s, f, C) = (s, \$)$
54. $f(s, g, C) = (s, \$)$
55. $f(s, h, C) = (s, \$)$
56. $f(s, i, C) = (s, \$)$
57. $f(s, j, C) = (s, \$)$
58. $f(s, k, C) = (s, \$)$
59. $f(s, l, C) = (s, \$)$
60. $f(s, m, C) = (s, \$)$
61. $f(s, n, C) = (s, \$)$
62. $f(s, o, C) = (s, \$)$
63. $f(s, p, C) = (s, \$)$
64. $f(s, q, C) = (s, \$)$
65. $f(s, r, C) = (s, \$)$
66. $f(s, s, C) = (s, \$)$
67. $f(s, t, C) = (s, \$)$
68. $f(s, u, C) = (s, \$)$
69. $f(s, v, C) = (s, \$)$

70. $f(s, w, C) = (s, \$)$
71. $f(s, x, C) = (s, \$)$
72. $f(s, y, C) = (s, \$)$
73. $f(s, z, C) = (s, \$)$
74. $f^*(s, a, T) = (s, TC)$
75. $f^*(s, j, T) = (s, TC)$
76. $f^*(s, k, T) = (s, TC)$
77. $f^*(s, u, T) = (s, TC)$
78. $f^*(s, s, T) = (s, TC)$
79. $f^*(s, q, T) = (s, TC)$
80. $f^*(s, v, T) = (s, TC)$
81. $f^*(s, f, T) = (s, TC)$
82. $f^*(s, c, T) = (s, TC)$
83. $f^*(s, y, T) = (s, TC)$
84. $f^*(s, w, T) = (s, TC)$
85. $f^*(s, g, T) = (s, TC)$
86. $f^*(s, r, T) = (s, TC)$
87. $f^*(s, p, T) = (s, TC)$
88. $f^*(s, n, T) = (s, TC)$
89. $f^*(s, z, T) = (s, TC)$
90. $f^*(s, l, T) = (s, TC)$
91. $f^*(s, b, T) = (s, TC)$
92. $f^*(s, h, T) = (s, TC)$
93. $f^*(s, x, T) = (s, TC)$
94. $f^*(s, o, T) = (s, TC)$
95. $f^*(s, d, T) = (s, TC)$
96. $f^*(s, e, T) = (s, TC)$
97. $f^*(s, i, T) = (s, TC)$
98. $f^*(s, t, T) = (s, TC)$

99. $f^*(s, m, T) = (s, TC)$

100. $f^*(s, [, T) = (s, \$)$

$f(s, \$, h_0) =(s,\$)$

Виконаємо розпізнавання ланцюжка: *int *a[3]={}, *b[2]* = {3,58};

(s, *int *a[3]={}, *b[2]* = {3,58}; , h_0I) ├1

(s, *nt *a[3]={}, *b[2]* = {3,58}; , h_0I) ├2

(s, *t *a[3]={}, *b[2]* = {3,58}; , h_0I) ├3

(s, **a[3]={}, *b[2]* = {3, 58};, h_0;*PB*) ├7

(s, *a[3]={}, *b[2]* = {3, 58};, h_0;*P*}*LK*{*=]MD[TC*) ├ 48

(s, *[3]={}, *b[2]* = {3, 58};, h_0;*P*}*LK*{*=]MD[T*) ├ 100

(s, *[3]={}, *b[2]* = {3, 58};, h_0;*P*}*LK*{*=]MD[*) ├ 8

(s, *3]={}, *b[2]* = {3, 58};, h_0;*P*}*LK*{*=]MD*) ├ 26

(s, *]={}, *b[2]* = {3, 58};, h_0;*P*}*LK*{*=]M*) ├ 43

(s, *]={}, *b[2]* = {3, 58};, h_0;*P*}*LK*{*=]*) ├ 9

(s, *={}, *b[2]* = {3, 58};, h_0;*P*}*LK*{=) ├ 10

(s, {}, **b[2]* = {3, 58};, h_0;*P*}*LK*{) ├ 11

(s, }, **b[2]* = {3, 58};, h_0;*P*}*LK*) ├ 23

(s, }, **b[2]* = {3, 58};, h_0;*P*}*L*) ├ 47

(s, }, **b[2]* = {3, 58};, h_0;*P*}) ├ 12

(s, , **b[2]* = {3, 58};, h_0;*P*) ├ 5

(s, **b[2]* = {3, 58};, h_0;*PB*) ├ 7

(s, *b[2]* = {3, 58};, h_0;*P*}*LK*{*=]MD[TC*) ├ 49

(s, *[2]* = {3, 58};, h_0;*P*}*LK*{*=]MD[T*) ├ 100

(s, *[2]* = {3, 58};, h_0;*P*}*LK*{*=]MD[*) ├ 8

(s, *2]* = {3, 58};, h_0;*P*}*LK*{*=]MD*) ├ 25

(s, *]* = {3, 58};, h_0;*P*}*LK*{*=]M*) ├ 44

(s, *]* = {3, 58};, *h*0;*P*}*LK*{*=]*) ├ 9

(s, = {3, 58};, h_0;*P*}*LK*{=) ├ 10

(s, {3, 58};, h_0;P}LK{) ├ 11

(s, 3, 58};, h_0;P}LK) ├ 15

(s, 3, 58};, h_0;P}LMD) ├ 26

(s, , 58};, h_0;P}LM) ├ 44

(s, , 58};, h_0;P}L) ├ 46

(s, 58};, h_0;P}LK) ├ 19

(s, 58};, h_0;P}LMD) ├ 28

(s, 8};, h_0;P}LM) ├ 38

(s, 8};, h_0;P}LMD) ├ 31

(s, };, h_0;P}LM) ├ 45

(s, };, h_0;P}L) ├ 47

(s, };, h_0;P}) ├ 12

(s, ;, h_0;P) ├ 6

(s, ;, h_0;) ├ 4

(s, $, h_0) ├ 101

(s,$,$)

Ланцюжок розпізнано.

Контрольні запитання

1. Як працює спадний розпізнавач?
2. Яка граматика називається розділеною або простою?
3. Які типи команд розпізнавача можуть бути зіставлені з кожним правилом граматики?
4. Яка граматика називається слабкорозділеною?
5. Як будується функція ПЕРШ(μ)?
6. Як будується функція СЛІД(А)?
7. Як будується множина ВИБІР(μ)?
8. Які граматики називаються $LL(R)$ – граматиками?

9. Як будуються команди розпізнавача для $LL(R)$ – граматик?

10. Побудувати спадний розпізнавач для фрагменту програми, яка описує список змінних цілого і символьного типів.

4 ВИСХІДНІ *LR*(*k*) – РОЗПІЗНАВАЧІ

4.1 *LR*(*k*) – граматики

Детерміновані висхідні розпізнавачі, так само як і спадні, можуть бути побудовані не для всякої КВ-граматики, а тільки для визначених підкласів таких граматик. Найбільш широким підкласом КВ-граматик є ***LR*(*k*)-граматики**. Ці граматики забезпечують розпізнавання ланцюжка при перегляді зліва направо, про що свідчить буква *L* (*Left*) у назві граматики, і дозволяють виконати правобічну згортку, що показує буква *R* (*Right*) у назві. Параметр *k* свідчить про те, що для визначення правила граматики, яке потрібно застосувати для згортання ланцюжка, буде потрібно переглянути не більш *k* ще не прочитаних символів вхідного ланцюжка.

У загальному випадку алгоритми побудови розпізнавачів для *LR*(*k*)-граматик виявляються досить складними і трудомісткими, тому на практиці найчастіше використовують підкласи *LR*(*k*)-граматик: *LR*(0) або ***SLR*(1)**-прості (*Simple*), *LR*(1)-граматики, що дозволяють відносно просто виконувати побудову висхідних розпізнавачів [4]. При цьому для кожного підкласу *LR*(*k*)-граматик використовується свій алгоритм побудови. Якщо задана КВ-граматика, то визначити її приналежність до одного з підкласів граматик *LR*(*k*) можна тільки шляхом аналізу можливості побудови для неї за допомогою визначеного алгоритму детермінованого розпізнавача. З огляду на останню обставину домовимося називати розпізнавачі по підкласах відповідних граматик: *LR*(0)-розпізнавач або *SLR*(1)-розпізнавач.

Перш ніж перейти до розгляду правил побудови висхідних розпізнавачів, визначимо кілька допоміжних понять. Домовимося називати символи повного словника граматики **граматичними символами**. Кожен граматичний символ може входити в різні правила граматики і, більш того, з'являтися в тому самому правилі кілька разів. При цьому положення символу в правилі граматики може показувати, яку дію потрібно виконати –

перенос чи згортку, а також, які граматичні символи можуть за ним випливати. Ця обставина дозволяє зв'язати позицію граматичного символу в правилі граматики з поняттям стану розпізнавача. Для зручності подальших міркувань визначимо поняття **граматичного входження.**

Визначення. Граматичне входження символу граматики задається номером правила і номером позиції, що вказує місце символу в правій частині правила, вважаючи, що найлівіший символ правої частини правила є першим.

Домовимося позначати граматичні входження символів, що входять у праву частину правила тільки один раз, за допомогою одного індексу, що дорівнює номеру правила. Домовимось також, що кожна граматика містить граматичне входження I_0, назване **початковим входженням**. Це входження задається початковим символом граматики.

4.2 Побудова таблиць розпізнавача. Алгоритм роботи розпізнавача

Для граматичних входжень визначимо дві функції ВПЕРШ(Y) і ВПІСЛЯ(Y). Функція ВПЕРШ(Y) за аналогією з функцією ПЕРШ(Y) визначає множину граматичних входжень, що можуть стояти на першому місці в ланцюжках, виведених з Y. Ця множина будується в такий спосіб: у нього входить символ Y і всі символи, що починають проміжні ланцюжки, виведені з Y без застосування правил, що анулюють.

Як приклад побудуємо функції ВПЕРШ(Y) для символів наступної граматики, що не містить правил, що анулюють:

$$\Gamma_{4.1}:\ I_0 \to a_1 I_{12} I_{13} b_1$$

$$I \to c_2$$

Ці функції мають наступний вигляд:

$$\text{ВПЕРШ}(a_1) = \{a_1\},$$

$$\text{ВПЕРШ}(I_{12}) = \{I_{12}, a_1, c_2\},$$

$$\text{ВПЕРШ}(I_{13}) = \{I_{13}, a_1, c_2\},$$
$$\text{ВПЕРШ}(b_1) = \{b_1\},$$
$$\text{ВПЕРШ}(C_2) = \{c_2\},$$
$$\text{ВПЕРШ}(I_0) = \{I_0, a_1, c_2\}.$$

Функція ВПІСЛЯ(Y) є аналогом функції СЛІД(Y). Вона визначає множину граматичних входжень, що можуть зустрічатися безпосередньо після Y у ланцюжках, виведених з початкового символу граматики. Правило визначення функції ВПІСЛЯ (Y) можна записати так: якщо в правій частині деякого правила після Y безпосередньо випливає Z, то

$$\text{ВПІСЛЯ}(Y) = \text{ВПЕРШ}(Z).$$

При побудові розпізнавачів необхідно враховувати наявність маркера дна, тому, забігаючи вперед, сформулюємо ще одне правило обчислення цієї функції

$$\text{ВПІСЛЯ}(h_0) = \text{ВПЕРШ}(I_0),$$

де I_0 – початкове входження.

Для граматики $\Gamma_{4.1}$ функції ВПІСЛЯ (Y) мають вигляд:

$$\text{ВПІСЛЯ}(a_1) = \{I_{12}, a_1, c_2\},$$
$$\text{ВПІСЛЯ}(I_{12}) = \{I_{13}, a_1, c_2\},$$
$$\text{ВПІСЛЯ}(I_{13}) = \{b_1\},$$
$$\text{ВПІСЛЯ}(b_1) = \{\$\},$$
$$\text{ВПІСЛЯ}(C_2) = \{\$\},$$
$$\text{ВПІСЛЯ}(I_0) = \{\$\},$$
$$\text{ВПІСЛЯ}(h_0) = \{I_0, a_1, c_2\}.$$

Функція ВПІСЛЯ(Y) визначає граматичні входження, що можуть випливати після входження Y у виведених ланцюжках, що згортаються. Наприклад, якщо черговим граматичним входженням є символ a_1 і за ним повинен випливати граматичний символ I, то за значенням функції ВПІСЛЯ (a_1) можна визначити, що символу I у цьому випадку відповідає входження I_{12}. Таким чином, при згортанні за допомогою функції ВПІСЛЯ(Y)

можна визначити, якому граматичному входженню відповідає черговий граматичний символ ланцюжка, що згортається. Якщо взяти послідовність граматичних символів, то користуючись функціями ВПІСЛЯ(Y) їй можна поставити у відповідність послідовність граматичних входжень. У цьому випадку зручно розглядати граматичні входження, як стани скінченного автомата, а граматичні символи, як вхідні дані. Зміну станів цього автомата можна представити у вигляді таблиці переходів, що будується в такий спосіб. Кожному граматичному входженню виділимо один рядок таблиці, а кожному граматичному символу – один стовпець. Клітини таблиці заповнюються елементами функцій ВПІСЛЯ(Y) таким чином, що елемент $X_k \in$ ВПІСЛЯ(Y_j) заноситься в клітинку, що знаходиться на перетині рядка Y_j і стовпця, позначеного граматичним символом X.

Таблиця переходів розпізнавача (табл. 4.1), побудована для розглянутої граматики, має такий вигляд

Таблиця 4.1 – Таблиця переходів

Граматичні входження	Граматичні символи			
	a	b	c	I
a_1	a_1		c_2	I_{12}
I_{12}	a_1		c_2	I_{13}
I_{13}		b_1		
b_1				
C_2				
h_0	a_1		c_2	I_0
I_0				

Ця таблиця задає функцію: $f(\ B_{ij}\ ,\ X\ ) = X_{kl}$, що для поточного граматичного входження B_{ij} і чергового символу граматики X визначає граматичне входження чергового символу X_{kl}.

Слід зазначити, що при побудові таблиці переходів у клітинах таблиці можуть виявитися по кілька граматичних входжень відповідних символів. Така таблиця є недетермінованою, і її потрібно перетворити в детерміновану за допомогою прийомів, використаних для перетворення таблиць скінченних автоматів. У результаті можна одержати таблицю, в якій рядки відзначені множинами граматичних входжень.

Побудована таблиця переходів описує зміну станів розпізнавача, але вона ніяк не відбиває, в яких випадках розпізнавач повинен виконувати дії **переносу** прочитаних символів у магазин чи **згортання**. Якщо для збереження проміжних ланцюжків, отриманих у процесі згортання, використовувати магазин, то таблиця переходів може бути використана для визначення граматичних входжень, записуваних у магазин. Для опису порядку дій магазинного розпізнавача побудуємо ще одну таблицю, яку назвемо керуючою таблицею. У цій таблиці позначимо дію переносу символів із вхідного ланцюжка в магазин символом П (перенос), а дії, пов'язані зі згортанням ланцюжків, що відповідають правим частинам правил, позначимо символом З (№), де № – номер використаного правила. Для позначення дій, що здійснюють передачу на вихід результатів роботи розпізнавача, домовимося використовувати початкові букви слів “допустити” (Д) і “відкинути” (В). Позначимо рядки таблиці граматичними входженнями, а стовпці – термінальними символами граматики і символом кінця ланцюжка $\perp_к$.

Підставою для заповнення таблиці є наступні два положення.

1. Операція згортання повинна виконуватися незалежно від вхідного символу завжди, якщо у вершині магазина знаходиться саме праве

граматичне входження деякого правила. Для таких граматичних входжень значення функції ВПІСЛЯ(Y) є порожньою множиною.

2. Якщо у вершині магазина знаходиться граматичне входження, що не є найправішим входженням якого-небудь правила, то варто виконати перенос чергового символу вхідного ланцюжка в магазин.

Процес розпізнавання закінчується успішно при виявленні символу $\perp_к$ на вході і символу $I_о$ в магазині. В інших випадках, що залишилися, вхідний ланцюжок повинен бути відкинутий. Таким чином , одержуємо керуючу табл. 4.2.

Ця таблиця задає функцію дій $f(B_{i\,j}, x)$ (- { Д, В, З (1), З (2), ..., З (N) }, де N – число правил заданої граматики, що визначає, яку дію повинен виконати розпізнавач, що знаходиться в стані $B_{i\ j}$ залежно від вхідного символу x. Для розглянутого прикладу операції згортання визначаються таким чином: З (1) = { $a_1 I_{12} I_1\ b_1 \rightarrow I_0$ }, З (2) = { $c_2 \rightarrow I_0$ }.

Таблиця 4.2 – Керуюча таблиця

Граматичні входження	Термінальні символи			
	a	b	c	$\perp_к$
a_1	П	П	П	В
I_{12}	П	П	П	В
I_{13}	П	П	П	В
b_1	З(1)	З(1)	З(1)	З(1)
c_2	З(2)	З(2)	З(2)	З(2)
h_0	П	П	П	В
I_0	В	В	В	Д

Керуюча таблиця розпізнавача, і таблиця переходів цілком задають роботу розпізнавача.

Алгоритм роботи, що використовує таблицю станів і керуючу таблицю, можна описати так:

1. Прочитати черговий символ вхідного ланцюжка x.

2. Прочитати символ стану, що знаходиться у вершині магазина $Ук_1$.

3. Прочитати значення елемента керуючої таблиці, що знаходиться в рядку Y_{K1} і стовпці x.

4. Якщо прочитане значення є В чи Д, то роботу варто закінчити, оскільки результат отриманий.

5. Якщо прочитане значення визначає операцію Перенос, то прочитати в таблиці станів елемент $Z_{i\,j}$, що знаходиться в рядку Y_{K1} і стовпці X. Записати символ $Z_{i\,j}$ у магазин.

6. Якщо прочитане значення визначає операцію З в нетерміналі Z, то прочитати в таблиці переходів елемент $Z_{i\ j}$, що знаходиться в стовпці Z і рядку, який відповідає верхньому символу магазина, що не приймає участі у згортанні.

Використовуючи описаний алгоритм, роботу розпізнавача, можна подати у вигляді послідовності конфігурацій (табл. 4.3).

Таблиця 4.3 – Приклад роботи розпізнавача для граматики $Г_{4.1.}$

Вхід	Магазин	Дія
$aaccbcb\perp$	h_0	П
$accbcb\perp$	h_0a_1	П
$ccbcb\perp$	$h_0a_1a_1$	П
$cbcb\perp$	$h_0a_1a_1c_2$	З(2)
$cbcb\perp$	$h_0a_1a_1I_{12}$	П
$bcb\perp$	$h_0a_1a_2I_{12}c_2$	З(2)
$bcb\perp$	$h_0a_1a_2I_{12}I_{13}$	П

Продовження табл. 4.3

$cb\perp$	$h_0a_1a_2I_{12}I_{13}b_1$	З(1)
$cb\perp$	$h_0a_1I_{12}$	П
$b\perp$	$h_0a_1I_{12}c_2$	З(2)
$\perp$	$h_0a_1I_{12}\mathrm{I}_{13}b_1$	З(1)
$\perp$	h_0I_0	Д

У загальному випадку процедуру побудови висхідного розпізнавача за заданою граматкою, що не містить правил, які анулюють, можна описати в такий спосіб:

1. Визначити для даної граматики функції ВПЕРШ(Y) і ВПІСЛЯ(Y).

2. Побудувати детерміновану таблицю переходів , що має по одному стовпцю для кожного граматичного символу і по одному рядку для кожного граматичного входження і маркера дна. Елемент у рядку R_j і стовпці С повинен містити всі такі граматичні входження C_K, що $C_K \in$ ВПІСЛЯ(R_j).

3. Якщо таблиця, побудована на кроці 2, виходить недермінованою, то потрібно перетворити цю таблицю в детерміновану, розглядаючи її як недетерміновану таблицю переходів скінченного автомата з початковим станом h_o.

4. Стани, отримані на кроці 3 (крім стану, що відповідає порожній множині), варто використовувати як магазинні символи. Отримана таблиця переходів може містити переходи в порожню множину. Такі елементи варто розуміти як заборонені і розглядати переходи в них як помилки.

5. Керуючу таблицю заповнюють рядок за рядком відповідно до множини граматичних входжень, що позначають рядки, в такий спосіб:

5.1. Якщо рядок позначений початковим входженням I_0, то в стовпець, що відповідає маркеру кінця рядка $\perp_к$, заноситься операція Д, а в усі інші рядки – операція В.

5.2. Якщо рядок позначений маркером дна h_0, або якщо всі граматичні входження, що входять у множину, яка позначає рядок, не є найправішими у своїх правилах, то в стовпець, позначений кінцевим маркером рядка, заноситься операція В, а у всі інші стовпці – операція П.

5.3. Якщо рядок позначений граматичним входженням, що є найправішим входженням у правилі з номером k, то у всі елементи рядка розмістити операцію З(k).

5.4. Якщо множина, що позначає рядок після перетворення НКА, містить початкове входження і хоча б одне входження, відмінне від початкового, але не містить жодного найправішого входження, то в стовпець, позначений символом кінця рядка, потрібно помістити операцію Д, а в інші стовпці – П.

Наведена процедура забезпечує побудову розпізнавача, тільки у разі, якщо задана граматика належить підкласу $LR(0)$, оскільки дії в кожному рядку керуючої таблиці однакові, тобто не залежать від вхідного символу. Якщо ж у процесі побудови виявляється, що хоча б один з пунктів виконати неможливо, то це означає, що для заданої граматики неможливо побудувати $LR(0)$-розпізнавач і що вона не є $LR(0)$-граматикою.

4.3 Приклад побудови $LR(0)$-розпізнавача для граматики $\Gamma_{4.2}$

Як ілюстрацію застосування описаної процедури розглянемо побудову розпізнавача для наступної граматики $\mathbf{\Gamma_{4.2}}$:

1. $E \rightarrow E_1 + T_1$
2. $E \rightarrow T_2$
3. $T \rightarrow (E_3)$
4. $T \rightarrow i$

Функції ВПЕРШ і ВПІСЛЯ для цієї граматики мають вид:

ВПЕРШ(E_1)={E_1,T_2,(,i},

ВПЕРШ(T_1)={T_1,(,i},

ВПЕРШ(T_2)={T_2,(,i},

ВПЕРШ(+) = {+},

ВПЕРШ(i) = {i},

ВПЕРШ(() = {(},

ВПЕРШ()) = {)},

ВПЕРШ(E_3)={E_3,E_1,T_2,(,i},

ВПІСЛЯ(E_1) = {+},

ВПІСЛЯ(T_1) ={$},

ВПІСЛЯ(T_2) = {$},

ВПІСЛЯ(+) = {T_1,(,i},

ВПІСЛЯ(i) = {$},

ВПІСЛЯ(()={E_1,E_3,T_2,(,i}

ВПІСЛЯ()) = {$},

ВПІСЛЯ(E_0) = {$},

ВПІСЛЯ(h_0)={E_0,E_1,T_2,(,i},

ВПІСЛЯ(E_3) = {)}.

Таблиця переходів, побудована по функціях ВПІСЛЯ, зображується так:

Таблиця 4.4 – Таблиця переходів

Граматичні входження	Граматичні символи					
	E	*T*	+	(	)	*i*
E_0						
E_1			+			
T_1						
T_2						
+		T_1		(		*i*
i						

Продовження табл. 4.4

(	E_1 E_3	T_2		(		i
)						
h_0	E_1 E_0	T_2		(		i
E_3					)	

Отримана таблиця переходів є недермінованою. Після перетворення таблиці, позначаючи множину станів (E_0 ,E_1) =E_x і (E_1,E_3) =E_y і вважаючи, що початковим станом є h_0, одержуємо наступну табл. 4.5:

Таблиця 4.5 – Детермінована таблиця переходів

Граматичні входження	Граматичні символи					
	E	T	+	(	)	i
E_x			+			
E_y			+		)	
T_1						
T_2						
+		T_1		(		i
(	E_y	T_2		(		i
)						
h_0	E_x	T_2		(		i
i						

Побудуємо керуючу таблицю розпізнавача (табл. 4.6).

Таблиця 4.6 – Керуюча таблиця

Граматичні входження	Термінальні символи				
	$\perp_к$	+	(	)	*i*
E_x	*D*	П	П	П	П
E_y	В	П	П	П	П
T_1	З (1)	З (1)	З (1)	З (1)	З (1)
T_2	З (2)	З (2)	З (2)	З (2)	З (2)
+	В	П	П	П	П
i	З (4)	З (4)	З (4)	З (4)	З (4)
(	В	П	П	П	П
)	З (3)	З (3)	З (3)	З (3)	З (3)
h_0	В	П	П	П	П

Побудований розпізнавач є еквівалентним недетермінованому розпізнавачу, але ці розпізнавачі мають різні стани. Отже, їм повинні відповідати еквівалентні, але неоднакові граматики. Таке розходження повинне відбитися в операціях згортання. В розглянутому випадку операція З(1) повинна враховувати, що недетермінованому розпізнавачу відповідає граматика з правилами $E \rightarrow E_x + T$ і $E \rightarrow E_y + T$. Приклад роботи розпізнавача для граматики $Г_{4.2}$ наведено в табл. 4.7

Таблиця 4.7 – Приклад роботи розпізнавача для граматики $Г_{4.2}$

Вхід	Магазин	Дія	Вхід	Магазин	Дія
1	2	3	4	5	6
((*i*+*i*)+*i*)+i	h_0	П	+ *i*)+i	h_0 (*E_y*	П
(*i*+*i*)+*i*)+i	h_0 (	П	*i*)+i	h_0 (*E_y*+	П
i+*i*)+*i*)+i	h_0 ((	П	)+i	h_0 (*E_y*+i	З(4)

Продовження табл. 4.7

1	2	3	4	5	6
+*i*)+*i*)+i	h_0((i	З(4)	)+i	h_0(E_y+T_1	З(1)
+*i*)+*i*)+i	h_0((T_2	З(2)	)+i	h_0(E_y	П
+*i*)+*i*)+i	h_0((E_y	П	+i	h_0(E_y)	З(3)
i)+*i*)+i	h_0((E_y +	П	+i	h_0 T_2	З(2)
)+*i*)+i	h_0((E_y +i	З(4)	*i*	h_0 E_y	П
)+*i*)+i	h_0((E_y + T_1	З(1)	⊥к	h_0 E_x+	П
)+*i*)+i	h_0((E_y	П	⊥к	h_0 E_x+i	З(4)
+*i*)+i	h_0((E_y)	З(3)	⊥к	h_0 E_x+T_1	З(1)
+ *i*)+i	h_0((T_2	З(2)	⊥к	h_0E_0	Д

4.4 Побудова *SLR*(1)- розпізнавача

Розглянемо ще один приклад побудови розпізнавача для наступної граматики $\Gamma_{4.3}$, що не містить правил, які анулюють:

1. $I \rightarrow t_1A_1$

2. $A \rightarrow A_2,B_2$

3. $A \rightarrow B_3$

4. $B \rightarrow a$

5. $B \rightarrow b$

Після побудови функцій ВПЕРШ(*Y*) і ВПІСЛЯ (*Y*) для даної граматики, одержуємо таблицю переходів (табл. 4.7).

Це – недетермінована таблиця, тому введемо позначення A_x= (A_1,A_2) і перетворимо її до детермінованого виду. У результаті маємо табл. 4.8.

Таблиця 4.8 – Таблиця переходів

Граматичні входження	Граматичні символи						
	t	,	*a*	*b*	*I*	*A*	*B*
h_0	*T*				I_0		
I_0							
A_1							
t			*a*	*B*		A_1 A_2	B_3
A_2		,					
,			*a*	*b*			B_2
B_2							
B_3							
a							
b							

Таблиця 4.9 – Детермінована таблиця переходів

Граматичні входження	Граматичні символи						
	t	,	*a*	*b*	*I*	*A*	*B*
h_0	*t*				I_0		
I_0							
A_x		,					
t			*a*	*b*		A_x	B_3
,			*a*	*b*			B_2
B_2							
B_3							
a							
b							

Продовжуючи побудову відповідно до описаної процедури, ми зіштовхнемося з протиріччям при заповненні рядків керуючої таблиці для рядка, позначеного символом A_x. Оскільки A_x містить граматичне входження A_1, що є найправішим символом правила 1, то потрібно було б записати в розглянутий рядок таблиці дій операцію З(1). Однак, у множині A_x міститься також граматичне входження A_2, що не є найправішим символом правила.

Для такого символу даний рядок таблиці дій потрібно заповнити операцією П. Виявлене протиріччя показує, що граматика $\mathbf{\Gamma_{4.3}}$ не є *LR*(0)-граматикою і що побудувати *LR*(0)-розпізнавач за допомогою описаної процедури неможливо.

Спробуємо тепер перевірити, чи не можна побудувати для заданої граматики *SLR*(1)-розпізнавач. Побудова такого розпізнавача відрізняється від вище описаного тим, що заповнення керуючої дії виконується не цілими рядками, а кожен елемент рядка будується окремо, і за цих умов враховуються тільки ті символи, що можуть зустрічатися за символом, що розглядається.

У нашому прикладі при побудові рядка керуючої таблиці для символу A_x у стовпець, позначений символом кома, запишемо операцію П, оскільки в таблиці переходів у рядку A_2 і стовпці “кома” знаходиться як граматичний символ. У стовпець, позначений символом кінця рядка $\perp_к$, занесемо операцію З(1), оскільки за символом A_1 може випливати тільки символ $\perp_к$. Останнє твердження витікає з того, що СЛІД(*I*) = {$\perp_к$=$}. З огляду на те, що

$$\text{СЛІД}(B) = \{, , \perp_к \}$$
$$\text{СЛІД}(A) = \{, , \perp_к \}$$

одержуємо керуючу таблицю для шуканого розпізнавача (табл. 4.9).

Ця таблиця разом з табл. 4.8 задає *SLR*(1)-розпізнавач, що працює згідно з тими же правилами, що і *LR*(0)-розпізнавач. Незаповнені клітини таблиці відповідають операції відкинути (В).

Таблиця 4.10 – Керуюча таблиця

Граматичні входження	Термінальні символи				
	t	,	*a*	*b*	$\perp_к$
h_0	П				
I_0					Д
A_x		П			З(1)
t			П	П	
,			П	П	
B_2		З(2)			З(2)
B_3		З(3)			З(3)
a		З(4)			З(4)
b		З(5)			З(5)

Процедура побудови *SLR*(1)-розпізнавача відрізняється від процедури побудови *LR*(0)-розпізнавача змістом тільки одного пункту 5, який можна описати так. Керуюча таблиця заповнюється поелементно для кожного рядка, позначеного маркером дна або множиною граматичних входжень *Q*, у такий спосіб:

5.1. Якщо *Q* містить початкове входження, то в стовпець, позначений маркером вхідного рядка, занести операцію Д.

5.2. Якщо *Q* містить граматичне входження $R_{i\ j}$, що не є найправішим входженням ніякого правила, і якщо елемент таблиці переходів для цього символу $R_{i\ j}$ і деякого символу граматики *S* не є порожнім, то в елемент керуючої таблиці відповідній парі ($R_{i\,j}$,*S*) заноситься операція П.

5.3. Якщо *Q* містить найправіше граматичне входження *p* ланцюжка α правила $A\rightarrow \alpha$ з номером *k*, то для кожного вхідного символу $x \in$ СЛІД(*A*) в елемент керуючої таблиці, що відповідає парі (*p*,*x*), заноситься операція З(*k*).

Якщо в результаті виконання вище описаного пункту 5 вдасться побудувати керуючу таблицю, то це означає, що задана граматика є граматикою *SLR*(1), а побудований автомат – *SLR*(1)-розпізнавачем.

Всі вище описані процедури побудови розпізнавачів застосовані до граматик, що не укорочують. Вони є також базою для побудови висхідних розпізнавачів для граматик, що укорочують. Однак ці процедури повинні бути доповнені правилами побудови, які враховують наявність правил, що анулюють.

4.5 Висхідні розпізнавачі для граматик із правилами, що анулюють

Перш ніж сформулювати правила побудови висхідних розпізнавачів для граматик із правилами, що анулюють, розглянемо приклад і спробуємо з'ясувати які додаткові дії повинен виконувати розпізнавач.

Нехай задана граматика, що задає арифметичні вирази без дужок із двома операціями:

$\mathbf{Г}_{4.4}$:

1. $I \rightarrow aR$

2. $R \rightarrow +aR$

3. $R \rightarrow -aR$

4. $R \rightarrow \$$

Четверте правило даної граматики має порожню праву частину, тому воно не повинно впливати на виконання перших чотирьох етапів процедури побудови *SLR*(1)-розпізнавача. Використовуючи цю процедуру, побудуємо таблицю переходів (табл.4.11) і керуючу таблицю (табл. 4.12) розпізнавача, яка не враховує наявності правила, що анулює.

Таблиця 4.11 – Таблиця переходів

Граматичні входження	Граматичні символи				
	a	+	-	R	I
I_0					
A_1		+	-	R_1	
R_1					
+	a_2				
A_2		+	-	R_2	
R_2					
-	a_3				
A_3		+	-	R_3	
R_3					
H_0	a_1				I_0

Таблиця 4.12 – Керуюча таблиця

Граматичні входження	Термінальні символи			
	a	+	-	$\perp$
I_0				Д
a_1		П	П	
R_1				З(1)
+	П			
a_2		П	П	
R_2				З(2)
-	П			
a_3		П	П	
R_3				З(3)
h_0	П			

При виводі четверте правило граматики дозволяє виключити нетермінал R з виведеного ланцюжка. Отже, при згортанні цьому правилу необхідно зіставити операцію запису символу R у магазин. Цю операцію позначимо З(4). Щоб визначити, в яких випадках повинна виконуватися ця операція,

необхідно вирішити, після яких символів у виведених ланцюжках може зустрічатися нетермінал R і які символи можуть випливати за ним. Множину символів x_1, x_2, …, x_k, за який може зустрічатися R, можна знайти, визначивши, в які множини ВПІСЛЯ(X_k) входить нетермінальний символ R, що анулює. Цю множину можна знайти по таблиці переходів (табл. 5.10) у такий спосіб.

Візьмемо стовпець, позначений символом R таблиці переходів і знайдемо всі рядки, в яких на перетинанні з цим стовпцем знаходяться не порожні елементи. Множина поміток цих рядків $\{a_1,a_2,a_3\}$ і є множиною граматичних входжень, за якими може слідувати R. Враховуючи, що за символом R можуть слідувати вхідні символи множини СЛІД(R) = $\{\perp_к\}$, знаходимо, що в елементи керуючої таблиці, які відповідають парам $(a_1, \perp_к)$, $(a_2, \perp_к)$ і $(a_3, \perp_к)$, потрібно вписати операцію З(4). У результаті одержуємо табл. 4.12, що з табл. 4.10 задає висхідний розпізнавач для заданої граматики.

Таблиця 4.13 – Керуюча таблиця

Граматичні входження	Термінальні символи			
	a	+	–	$\perp_к$
I_0				Д
a_1		П	П	З(4)
R_1				З(1)
+	П			
a_2		П	П	З(4)
R_2				З(2)
–	П			
a_3		П	П	З(4)
R_3				З(3)
h_0	П			

Послідовність конфігурацій, що описує роботу розпізнавача для вхідного ланцюжка $a + a - a \perp_к$ наведено в табл. 4.14.

Таблиця 4.14 – Приклад роботи розпізнавача

Вхід	Магазин	Дія
1. $a + a - a \perp_к$	h_0	П
2. $+ a - a \perp_к$	$h_0 a_1$	П
3. $a - a \perp_к$	$h_0 a_1 +$	П
4. $- a \perp_к$	$h_0 a_1 + a_2$	П
5. $a \perp_к$	$h_0 a_1 + a_2 -$	П
6. $\perp_к$	$h_0 a_1 + a_2 - a_3$	З(4)
7. $\perp_к$	$h_0 a_1 + a_2 + a_3 \ R_3$	З(3)
8. $\perp_к$	$h_0 a_1 + a_2 \ R_2$	З(2)
9. $\perp_к$	$h_0 a_1 \ R_1$	З(1)
10. $\perp_к$	$h_0 \ I_0$	Д

Розглянутий приклад показує, що в загальному випадку правила побудови висхідних *SLR*(1)-розпізнавачів необхідно доповнити ще одним п. 5.4, що повинен враховувати наявність правил, що анулюють, у заданій граматиці. Цей пункт процедури побудови запишемо в наступному вигляді:

5.4. Заповнення елементів керуючої таблиці для правила, що анулює, $A \rightarrow \$$ з номером k виконується таким чином:

Щоб знайти множину граматичних входжень, за яких може слідувати символ Y, виділимо в таблиці переходів стовпець, позначений символом Y. У цьому стовпці виділимо рядки, що мають непорожні елементи. Припустимо, що ці рядки відзначені символами $x_1, x_2, \ldots, x_m$. Знайдемо множину СЛІД(A) = $\{z_1, z_2, \ldots, z_n\}$. Це множина граматичних символів, що можуть випливати

за символом *Y*. Для кожної пари елементів (x_i,z_j) у відповідну клітину керуючої таблиці потрібно вписати операцію З(*k*).

Процедура побудови розпізнавача з п. 5.1, 5.2, 5.3 і 5.4 дозволяє одержати результат тільки в тому випадку, якщо задана граматика з правилами, що анулюють, є граматикою *SLR*(1). Якщо ж при побудові виявляються протиріччя, то це означає, що задана граматика не належить підкласу *SLR*(1) граматик, і для неї не можна побудувати *SLR*(1)-розпізнавач.

4.6 Приклад побудови *LR*(1) – розпізнавача для граматики $\Gamma_{4.5}$

Побудувати *LR*(1) розпізнавач для опису арифметичного виразу, який містить: ідентифікатор – *i*, =, +, (,), ;

Будуємо правила граматики і отримуємо граматику $\Gamma_{4.5}$:

$I \to i = A$; $C \to + A$

$A \to i\ C$ $C \to \$$

$A \to (\ A\ )\ C$

Граматика не містить недосяжних та зайвих символів, тому вона є приведеною.

Переписуємо правила, застосовуючи граматичні входження:

1. $I \to i_1 = A_1$;
2. $A \to i_2\ C_2$
3. $C \to + A_3$
4. $A \to (\ A_4\ )\ C_4$
5. $C \to \$$

Примітка. Якщо граматика містить правила, що анулюють, то для неї будується тільки *LR*(1)-розпізнавач.

При побудові висхідного розпізнавача використовуються функції ВПЕРШ(*Y*) і ВПІСЛЯ(*Y*). Функція ВПЕРШ(*Y*) визначає множину символів, які можуть стояти на першому місці в ланцюжках, виведених з *Y*. В нього входять сам *Y* і всі символи, виведені з *Y* без правил, що анулюють. Функція ВПЕРШ(*Y*) визначається і для початкового символу граматики.

Наприклад, функція ВПЕРШ(*Y*) для першого правила граматики буде мати вигляд:

ВПЕРШ(i_1)={i_1}

ВПЕРШ(=)={=}

ВПЕРШ(A_1)={A_1, i_2,(}

ВПЕРШ(;)={$}

Функція ВПІСЛЯ(*Y*) визначає множину символів, що можуть зустрічатися безпосередньо після *Y* у ланцюжках, виведених з початкового символу граматики. Якщо після *Y* випливає символ *Z*, то $\alpha \rightarrow \varphi YZ$, а ВПІСЛЯ(*Y*)=ВПЕРШ(*Z*).

Крім того тут визначається функція ВПІСЛЯ(*Y*) для h_0, що дорівнює ВПЕРШ(I_0), тобто ВПІСЛЯ(h_0)= ВПЕРШ(I_0).

Для заданої граматики будуємо функцію ВПІСЛЯ (*Y*):

ВПІСЛЯ (i_1) = {=};

ВПІСЛЯ (=) = { A_1 , i_2, (};

ВПІСЛЯ (A_1) = { ; };

ВПІСЛЯ (;)= { $ };

ВПІСЛЯ (i_2) = { C_2, + };

ВПІСЛЯ (C_2) = { $ };

ВПІСЛЯ (( )= { A_4, i_2, (};

ВПІСЛЯ ( )) = { C_4, + };

ВПІСЛЯ (A_4) = {) };

ВПІСЛЯ (C_4) = { $ };

ВПІСЛЯ (+) = { A_3, i_2, (};

ВПІСЛЯ (h_0) = { I_0, i_1 };

ВПІСЛЯ (I_0) = { $ };

Побудуємо функцію СЛІД для самих правих символів у всіх правилах:

СЛІД (A) = {), ;}; СЛІД (C) = {), ; }; СЛІД (I_0) = {⊥};

Використовуючи функцію ВПІСЛЯ, будують таблицю переходів, котра визначає зміну станів автомата. Таблиця переходів використовується для визначення граматичних входжень, записуваних у магазин. Таблиця будується в такий спосіб:

Кожному граматичному входженню відповідає рядок таблиці, кожному граматичному символу відповідає стовпець. Клітини таблиці заповнюються елементами функції ВПІСЛЯ(Y). Елемент $x_к$ приналежний множині функції ВПІСЛЯ(Y_j), заноситься в клітину, що знаходиться на перетині рядка x_j та стовпця x.

Наприклад, рядок граматичного входження **’i_1’ – ,** для нього функція ВПІСЛЯ містить таку множину: ВПІСЛЯ(i_1) = {=}. Отже, на перетині рядка ’i_1’ і стовпця ’=’ ставимо елемент із множини ВПІСЛЯ(i_1), тобто ’=’. І так з усіма рядками.

Наведемо таблицю переходів (табл. 4.15), побудовану для заданої граматики.

Треба відзначити, що при побудові таблиці переходів у клітинках може бути кілька граматичних входжень відповідних символів. Така таблиця є недетермінованою, і її треба перетворити в детерміновану за допомогою засобів, що використовуються для перетворення таблиць скінченних автоматів. У результаті одержимо таблицю, у якій рядки позначені множинами граматичних входжень.

Для опису порядку дій розпізнавача будується керуюча таблиця (табл. 5.15), в якій містяться рядки, що є граматичним входженням, і стовпці, які є термінальними символами (символами вхідного ланцюжка). У таблиці також присутній стовпець $⊥_к$, що позначає кінець вхідного рядка.

Таблиця 4.15 – Таблиця переходів

Граматичні входження	Граматичні символи								
	I	i	=	A	C	+	(	)	;
i_1			=						
=		i_2		A_1			(		
A_1									;
;									
i_2					C_2	+			
+		i_2		A_3			(		
C_2									
A_3									
(		i_2		A_4			(		
A_4								)	
)					C_3	+			
C_4									
h_0	I_0	i_1							
I_0									

Заповнюється таблиця таким чином:

– якщо рядок позначений символом I_0, то в стовпець, позначений маркером h_0, заносимо операцію Д;

– якщо рядок позначений граматичним входженням R_{ij}, який не є найправішим входженням ніякого правила і якщо елемент таблиці переходу для R_{ij} та вхідного символу S_i не є порожнім, то в керуючу таблицю, відповідно рядку R_{ij} і стовпцю S_i заноситься операція П;

– якщо рядок позначений найправішим граматичним входженням р ланцюжка α, і є правило $A \rightarrow \alpha$ з номером k то для кожного вхідного символу х, що належить множині СЛІД(А) в елемент керуючої таблиці (відповідний рядок р і стовпець х) заноситься операція згортки З(k);

– якщо граматика містить правила, що анулюють, A→$, з номером k , то необхідно відзначити рядки, для яких функція ВПІСЛЯ містить A. На перетинанні даних рядків і стовпців відповідних функції СЛІД(A) ставимо операцію згортки З(k).

Наведемо керуючу таблицю (табл. 4.15), побудовану для заданої граматики.

Алгоритм роботи використовує таблицю переходів і керуючу таблицю і працює таким чином:

– прочитати черговий символ вхідного ланцюжка x;

– прочитати символ стану, що знаходиться у вершині магазина y_{ij};

– прочитати значення елемента керуючої таблиці, що знаходиться в рядку y_{ij} і стовпці x;

– якщо прочитане значення є В або Д, то роботу автомата закінчити;

– якщо отримане значення визначене операцією П, то прочитати в таблиці переходів елемент, що знаходиться в рядку y_{ij} і стовпці x. Записати отриманий символ у магазин, перейти до п. 1;

– якщо отримане значення в керуючій таблиці визначено операцією З у нетермінал Z (згідно з к-м правилом), то необхідно в таблиці переходів прочитати елемент Z_{ij}, що знаходиться в стовпці Z і в рядку, який відповідає верхньому символу магазина, котрий не брав участі в згортанні. Записати Z_{ij} у магазин і перейти до п. 1.

Таблиця 4.16 – Керуюча таблиця

Граматичні входження	Термінальні символи						
	i	=	+	(	)	;	$\perp_к$
1	2	3	4	5	6	7	8
i_1		П					
=	П			П			

Продовження Табл.4.16

1	2	3	4	5	6	7	8
A_1						П	
;							З(1)
i_2			П		З(5)	З(5)	
C_2					З(2)	З(2)	
+	П			П			
A_3					З(3)	З(3)	
(	П			П			
A_4					П		
)			П		З(5)	З(5)	
C_4					З(4)	З(4)	
h_0	П						

Розпізнаємо ланцюжок i=((i+i+i)). Розпізнавання виразу наведено в табл. 4.17

Таблиця 4.17 – Розпізнавання ланцюжка i=((i+i+i))

Вхід	Магазин	Дія
1	2	3
i=((i+i+i)) $\perp_к$	h_0	П
=((i+i+i)); $\perp_к$	h_0 i_1	П
((i+i+i)); $\perp_к$	h_0 i_1=	П
(i+i+i)); $\perp_к$	h_0 i_1=(	П
i+i+i)); $\perp_к$	h_0 i_1=((	П
+i+i)); $\perp_к$	h_0 i$_1$=((i_2	П
i+i)); $\perp_к$	h_0 i_1=((i_2+	П
+i)); $\perp_к$	h_0 i_1=((i_2+i_2	П

Продовження табл. 4.17

1	2	3
i)); $\perp_к$	$h_0\ i_1$=((i_2+i_2+	П
)); $\perp_к$	$h_0\ i_1$=((i_2+i_2+i_2	З(5)
)); $\perp_к$	$h_0\ i_1$=((i_2+i_2+i_2C_2	З(2)
)); $\perp_к$	$h_0\ i_1$=((i_2+i_2+A_3	З(3)
)); $\perp_к$	$h_0\ i_1$=((i_2+i_2C_2	З(2)
)); $\perp_к$	$h_0\ i_1$=((i_2+$i_2\ C_2$	З(2)
)); $\perp_к$	$h_0\ i_1$=((i_2+A_3	З(3)
)); $\perp_к$	$h_0\ i_1$=((i_2C_2	З(2)
)); $\perp_к$	$h_0\ i_1$=$_1$((A_4	П
); $\perp_к$	$h_0\ i$=((A_4)	З(5)
); $\perp_к$	$h_0\ i_1$=((A_4)C_4	З(4)
); $\perp_к$	$h_0\ i_1$=(A_4	П
; $\perp_к$	$h_0\ i_1$=(A_4)	З(5)
); $\perp_к$	$h_0\ i_1$=(A_4)C_4	З(4)
); $\perp_к$	$h_0\ i_1$=A_1	П
$\perp_к$	$h\ i_1$=A_1;	З(1)
$\perp_к$	$h_0\ I_0$	Д

Контрольні запитання

1. Що лежить в основі побудови висхідних розпізнавачів?
2. Що таке граматичні символи і граматичні входження?
3. Як визначається функція ВПЕРШ?
4. Як визначається функція ВПІСЛЯ?
5. Як будується таблиця переходів розпізнавача?
6. Як будується керуюча таблиця розпізнавача?
7. Коли виконується операція переносу, а коли згортання?
8. Коли виконується операція “допустити”, а коли “відкинути”?
9. Чим відрізняється побудова $LR(0)$ від $LR(1)$ розпізнавача?
10. Побудувати висхідний розпізнавач, що розпізнає список змінних цілого типу.

5 ГРАМАТИКИ ПЕРЕДУВАННЯ

Ще одним розповсюдженим класом КВ-граматик, для яких можна побудувати детермінований висхідний розпізнавач, є граматики передування. Розпізнавач для граматик передування будується на основі алгоритму «перенос-згортання».

Даний метод зводиться до пошуку основи (найлівішої простої фрази u), що відповідно до правила $S \rightarrow u$ призводить до нетермінала S.

Принцип організації розпізнавача вхідних ланцюжків мови, заданого граматикою передування, ґрунтується на тому, що для кожної упорядкованої пари символів у граматиці встановлюється деяке відношення, яке називається відношенням передування. В процесі розбору вхідного ланцюжка розширений МП-автомат порівнює поточний символ вхідного ланцюжка з одним із символів, що знаходяться на верхівці стека автомата. У процесі порівняння перевіряється, яке з можливих відносин передування існує між цими двома символами. Залежно від знайденого відношення виконується операція або перенос, або згортання. За відсутності відносин передування між символами алгоритм сигналізує про помилку.

Задача полягає в тому, щоб мати можливість дійсним чином визначити відносини передування між символами граматики. Якщо це можливо, то граматика може бути віднесена до одного з класів граматик передування.

Існує кілька видів граматик передування. Вони розрізняються по тому, які відносини передування в них визначені і між якими типами символів (термінальними чи нетермінальними) можуть бути встановлені ці відносини. Крім того, можливі незначні модифікації функціонування самого алгоритму «зсув-згортання» у розпізнавачах для таких граматик (в основному на етапі вибору правила для виконання переносу, коли можливі неоднозначності) .

Виділяють наступні види граматик передування:

- простого передування;

• розширеного передування;

• слабкого передування;

• змішаної стратегії передування;

• операторного передування.

5.1 Граматики простого передування

Граматикою простого передування називають таку наведену КВ-граматику (без правил, що анулюють,) у якій:

1) для кожної упорядкованої пари термінальних і нетермінальних символів виконується не більш ніж одне з трьох відношень передування: =• (складає основу), <• (передує) ,• > (слідує) ;

2) різні правила в граматиці мають різні праві частини (тобто у граматиці не повинно бути двох різних правил з однією і тією ж правою частиною**).**

Відношення =• ,<• і •> називають відношеннями передування для символів. Відношення передування єдине для кожної упорядкованої пари символів. При цьому між якими-небудь двома символами може і не бути відношення передування. Це означає, що вони не можуть знаходитися поруч у жодному елементі розбору синтаксично правильного ланцюжка. Відношення передування залежать від порядку, в якому розміщуються символи, і в цьому змісті їх не можна плутати зі знаками математичних операцій – вони не мають ні властивості комутативності, ні властивості асоціативності. Наприклад, якщо відомо, що B_i •> B_j, то не обов'язково виконується B_i <• B_j (тому знаки передування іноді позначають спеціальною крапкою: =• , <• , •>).

Як і для багатьох інших класів граматик, для граматик простого передування не існує алгоритму, який би міг перетворити довільну КВ-

граматику в граматику простого передування (чи довести, що перетворення неможливе).

Метод передування заснований на тому факті, що відношення передування між двома сусідніми символами розпізнаваного рядка відповідають трьом наступним варіантам:

• $R <\bullet S$, якщо S – частина основи, а R -ні. (це відношення між символами можна назвати «передує»). При цьому символ S – перший символ правої частини правила;

• $R \bullet> S$, якщо R – частина основи, а S – ні (це відношення між символами можна назвати «слідує»). При цьому символ R – останній символ правої частини правила;

• $R =\bullet S$, якщо символи R і S належать одній **основі (**це відношення між символами можна назвати «складають основу»). При цьому існує правило, у правій частині якого підряд йдуть символи RS.

На підставі відношення передування будують матрицю передування. Матрицю передування граматики краще побудувати, спираючись безпосередньо на визначення відносин передування. Зручніше скористатися двома додатковими множинами – множиною крайніх лівих і множиною крайніх правих символів щодо нетермінальних символів граматики. Ці множини визначаються в такий спосіб:

• $L(A)$ – множина крайніх лівих символів щодо нетермінального символу А, які можуть бути виведені із символу А;

• $R(A)$ – множина крайніх правих символів щодо нетермінального символу А, що можуть бути виведені із символу А.

Множина $L(A)$ і $R(A)$ можуть бути побудовані за дуже простим алгоритмом:

1. Для кожного нетермінального символу А шукаємо всі правила, що містять А в лівій частини. В множину $L(A)$ включаємо найлівіший символ із

правої частини правил, а в множину $R(A)$ — самий крайній правий символ із правої частини. Переходимо до кроку п.2.

2. Для кожного нетермінального символу А: якщо множина $L(A)$ містить нетермінальні символи граматики A', A", ..., то її треба доповнити символами, що входять у відповідні множини $L(A')$, $L(A")$, ... і не входять в $L(A)$. Ту ж операцію треба виконати для $R(A)$.

3. Якщо на попередньому кроці хоча б одна множина $L(A)$ чи $R(A)$ для деякого символу граматики змінилася, то треба повернутися до п. 2, інакше побудова закінчена.

Після побудови множини $L(A)$ і $R(A)$ за правилами граматики створюється матриця передування. Рядки і стовпці матриці передування позначаються термінальними і нетермінальними символами. Матрицю передування доповнюють символами $\perp_п$ і $\perp_к$ (початок і кінець ланцюжка). У клітинки матриці на перетинанні відповідних стовпця і рядку містяться знаки відношенні. При цьому порожні клітки матриці свідчать про те, що між даними символами немає жодного відношення передування.

5.1.1 Алгоритм побудови матриці передування

Для будь-яких двох символів АВ, що розміщені поруч, справедливе наступне:

1) на перетинанні рядка А и стовпця В ставимо знак =•;

2) на перетинанні рядка А і стовпців, що належать множині $L(B)$ ставимо знак <•;

3) на перетинанні рядків, що належать множині R(А), і стовпців В, а також стовпців, що відповідають множині $L(B)$ ставимо знак •>;

4) у рядку $\perp_п$ і стовпцях, що належать множині $L(I)$ ставимо знак <•, тут I – початковий символ граматики;

5) у стовпці $\perp_к$ і рядках, що належать множині $R(I)$ ставимо знак **•>,** тут I – початковий символ граматики.

Матриця передування є основою для роботи розпізнавача мови, заданого граматикою простого передування.

5.1.2 Алгоритм «зсув-згортання» для граматики простого передування

Даний алгоритм виконується розширеним МП-автоматом з одним станом. Відносини передування служать для того, щоб визначити в процесі виконання алгоритму, яка дія: зсув чи згортання повинна виконуватися на кожному кроці алгоритму, і однозначно вибрати ланцюжок для згортання. Однозначний вибір правила при згортанні забезпечується за рахунок розходження правих частин усіх правил граматики. У початковому стані автомата головка, що зчитує, аналізує перший символ вхідного ланцюжка. Кінець ланцюжка помічається символом $\perp_к$ (кінець ланцюжка). У стеці МП-автомата знаходиться символ $\perp_п$ (початок ланцюжка). Символи $\perp_к$ і $\perp_п$ введені для зручності роботи алгоритму, в мову, задану вихідною граматикою, не входять.

Розбір вважається закінченим (алгоритм завершується), якщо головка автомата, що зчитує, знаходиться на символі $\perp_к$, а в стеці знаходяться початковий символ граматики I і символ $\perp_п$. Виконання алгоритму може бути перервано, якщо на одному з його кроків виникне помилка. Тоді вхідний ланцюжок не приймається.

Алгоритм складається з наступних кроків.

Крок 1. Помістити у верхівку стека символ $\perp_п$ головку, що зчитує, з початку вхідного ланцюжка символів.

Крок 2. Порівняти за допомогою відношення передування символ, що знаходиться на вершині стека, з поточним символом вхідного ланцюжка.

Крок 3. Якщо має місце відношення **<•** чи **=•** , то зробити зсув (перенос поточного символу з вхідного ланцюжка в стек і зсув головки, що зчитує, на один крок вправо) і повернутися до кроку 2. Інакше перейти до кроку 4.

Крок 4. Якщо має місце відношення **•>,** то зробити згортання. Для цього треба знайти на вершині стеку всі символи, зв'язані відношенням **=•** і видалити ці символи зі стеку. Потім вибрати з граматики правило, що має праву частину, яка збігається з основою, і помістити в стек ліву частину обраного правила (якщо символів, пов'язаних відношенням **=•**, на верхівці стека немає, то як основа використовується один, найверхній символ стеку). Якщо правило, що збігається з основою, знайти не вдалося, то необхідно перервати виконання алгоритму і повідомити про помилку, інакше, якщо розбір не закінчений, повернутися до кроку 2.

Крок 5. Якщо не встановлене жодне відношення передування між поточним символом вхідного ланцюжка і символом на верхівці стеку, то треба перервати виконання алгоритму і повідомити про помилку.

Помилка в процесі виконання алгоритму виникає, коли неможливо виконати черговий крок, наприклад, якщо не встановлене відношення передування між двома порівнюваними символами (на кроках 2 і 4) чи якщо не вдається знайти потрібне правило в граматиці (на кроці 4). Тоді виконання алгоритму негайно переривається.

5.2 Приклади побудови розпізнавача для граматики простого передування $\Gamma_{5.1}$

Дано граматику $\Gamma_{5.1}$:

$S \rightarrow TR$

$S \rightarrow T$

$R \rightarrow +T$

$R \rightarrow -T$

$R \rightarrow +TR$

$R \rightarrow -TR$

$T \rightarrow a$

$T \rightarrow b$

Побудуємо множину крайніх лівих і крайніх правих символів щодо нетермінальних символів граматики.

Крок 1

$L(S) = \{T\}$ $R(S) = \{R, T\}$

$L(R) = \{+,-\}$ $R(R) = \{R, T\}$

$L(\mathrm{T}) = \{ a, b\}$ $R(\mathrm{T}) = \{\mathrm{a}, b\}$

Крок 2

$L(S) = \{T, a, b\}$ $R(S) = \{R, T, a, b\}$

$L(R) = \{+,-\}$ $R(R) = \{R, T, a, b\}$

$L(\mathrm{T}) = \{ a, b\}$ $R(\mathrm{T}) = \{a, b\}$

Побудуємо матрицю передування – (табл. 5.1).

Таблиця 5.1– Матриця передування

Термінальні та нетермінальні символи	Термінальні та нетермінальні символи						
	+	-	*a*	*b*	*R*	*T*	$\perp_к$
+			<•	<•		=	
-			<•	<•		=	
a	•>	•>			•>		•>
b	•>	•>			•>		•>
R							• >
T	<•	<•			=		•>
$\perp_п$			<•	<•		<•	

Перевіримо роботу автомата на прикладі ланцюжка (табл. 5.2)

Таблиця 5.2 – Приклад розпізнавання ланцюжка

Вхідний ланцюжок	Стек	Команда
$a+b-a\perp_к$	$\perp_п$	П
$+b-a\perp_к$	$\perp_п$ a	З(7)
$+b-a\perp_к$	$\perp_п$ T	П
$b-a\perp_к$	$\perp_п$ $T+$	П
$-a\perp_к$	$\perp_п$ $T+b$	З(8)
$-a,\perp_к$	$\perp_п$ $T+T$	П
$a\perp_к$	$\perp_п$ $T+T-$	П
$\perp_к$	$\perp_п$ $T+T-a$	З(7)
$\perp_к$	$\perp_п$ $T+T-T$	З(4)
$\perp_к$	$\perp_п$ $T+TR$	З(5)
$\perp_к$	$\perp_п$ TR	З(1)
$\perp_к$	$\perp_п S$	Д

5.3 Граматика операторного передування

Операторною граматикою називається КВ-граматика без правил, що анулюють, у якій праві частини всіх правил не містять суміжних нетермінальних і для якої виконуються наступні умови:

– для кожної упорядкованої пари термінальних символів виконується не більш ніж одне з трьох відносин передування;

– різні правила мають різні праві частини і не мають правил, що анулюють;

– правила граматики не можуть містити двох суміжних нетермінальних символів у правій частині.

Принцип роботи розпізнавача подібний розпізнавачу граматики простого передування, але відношення передування перевіряються тільки

між термінальними символами. Для граматики даного виду на основі встановлених відношень передування також будується матриця передування, але вона містить тільки термінальні символи.

Для побудови матриці вводиться множина крайніх лівих $L^t(A)$ і крайніх правих $R^t(A)$ термінальних символів.

Для визначення множини $L^t(A)$ і $R^t(A)$ попередньо необхідно виконати побудову множини $L(A)$ і $R(A)$, як це було розглянуто раніше.

Далі для побудови $L^t(A)$ і $R^t(A)$ використовують наступний алгоритм.

Крок 1. Для кожного нетермінального символу А шукаємо всі правила, що містять А в лівій частини. В множину $L^t(A)$ включаємо найлівіший термінальний символ із правої частини правил, ігноруючи нетермінальні символи, у множину $R^t(A)$ – самий крайній правий термінальний символ із правої частини правил. Переходимо до кроку 2.

Крок 2. Для кожного нетермінального символу А: якщо множина $L(A)$ містить нетермінальні символи граматики A', A", ..., то її треба доповнити символами, що входять у відповідні множини $L^t(A')$, $L^t(A")$, ... і не входять в $L(A)$. Ту ж операцію треба виконати для $R(A)$ і $R^t(A)$.

Крок 3. Якщо на попередньому кроці хоча б одна множина $L^t(A)$ і $R^t(A)$ для деякого символу граматики змінилася, то треба повернутися до кроку 2, інакше побудова закінчена.

5.3.1 Алгоритм побудови керуючої таблиці

Алгоритм побудови керуючої таблиці є наступним:

– Якщо в правилах граматики підряд йдуть символи АВ, де А – термінал і В – нетермінал, то в рядку А і стовпцях, які належать множині $L^t(B)$ ставимо знак <•.

– Якщо в правилах граматики підряд йдуть символи АВ, де А – нетерминал і В – термінал, то в стовпці В і рядках, які належать множині R^t(А) ставимо знак •>.

– Якщо в правилах граматики йдуть термінали AB (можливо розділені нетерміналом C), то у рядку A і стовпці B ставимо знак **=.**

– У рядку $\perp_{\text{п}}$ і стовпцях, які, належать множині $L^t(I)$, де I – початковий символ граматики, ставимо знак <•.

– У стовпці $\perp_{\text{к}}$ і рядках, що належать множині $R^t(I)$, де I – початковий символ граматики, ставимо знак •>.

5.3.2 Алгоритм «зсув-згортання» для граматики операторного передування

Цей алгоритм у цілому схожий на алгоритм для граматик простого передування, що розглянутий вище. Він також виконується розширеним МП-автоматом і має ті ж умови завершення і виявлення помилок. Основна різниця полягає в тому, що при визначенні відносин передування цей алгоритм не бере до уваги нетермінальні символи, що знаходяться в стеці, і при порівнянні шукає найближчий до верхівки стеку термінальний символ. Однак після виконання порівняння і визначення границь основи при пошуку правила в граматиці, нетермінальні символи, безумовно, беруться до уваги.

Алгоритм складається з наступних кроків.

Крок 1. Помістити у верхівку стека символ $\perp_{\text{п}}$, головку, що зчитує, – у початок вхідного ланцюжка символів.

Крок 2. Порівняти за допомогою відношення передування термінальний символ, найближчий до вершини стеку (лівий символ відношення), з поточним символом вхідного ланцюжка. При цьому зі стеку треба вибрати найверхній термінальний символ, ігноруючи всі можливі нетермінальні символи.

Крок 3. Якщо має місце відношення **<•** чи =• , то зробити зсув (перенос поточного символу з вхідного ланцюжка в стек і зсув головки, що зчитує, на один крок управо) і повернутися до кроку 2. Інакше перейти до кроку 4.

Крок 4. Якщо має місце відношення **•>,** то зробити згортання. Для цього треба знайти на вершині стеку всі термінальні символи, пов'язані відношенням =• («основу»), а також усі нетермінальні символи, що сусідять з ними, (при визначенні відношення нетермінальні символи ігноруються). Якщо термінальних символів, пов'язаних відношенням =• , на верхівці стеку немає, то за основу використовується один, найверхній у стеці термінальний символ. Усі (і термінальні, і нетермінальні) символи, що складають основу, треба видалити зі стеку, а потім вибрати з граматики правило, що має праву частину, яка збігається з основою, і помістити в стек ліву частину обраного правила. Якщо правило, що збігається з основою, знайти не вдалося, то необхідно перервати виконання алгоритму і повідомити про помилку. Інакше, якщо розбір не закінчений, то повернутися до кроку 2.

Крок 5. Якщо не встановлене жодне відношення передування між поточним символом вхідного ланцюжка і найверхнім термінальним символом у стеці, то треба перервати виконання алгоритму і повідомити про помилку. Кінцева конфігурація даного МП-автомата збігається з конфігурацією при розпізнаванні ланцюжків граматик простого передування.

5.3.3 Приклад побудови розпізнавача для граматики операторного передування $\Gamma_{5.2}$

Задано граматику $\mathbf{\Gamma_{5.2}}$:

1. $E \rightarrow E+T$
2. $E \rightarrow T$
3. $T \rightarrow T*F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow i$

Для граматики $\Gamma_{5.2}$ побудувати висхідний розпізнавач.

Будуємо множину лівих і правих символів, а потім – множину лівих і правих термінальних символів:

$L(E)=\{E,T$	$R(E)=\{T\}$	$L^t(E)=\{+$	$R^t(E)=\{+\}$
$L(T)=\{T,F\}$	$R(T)=\{F\}$	$L^t(T)=\{*\}$	$R^t(T)=\{*$
$L(F)=\{(,i\ \}$	$R(F)=\{),i\}$	$L^t(F)=\{(,i\}$	$R^t(F)=\{),i\}$
$L(E)=\{E,T,F,(,i\}$	$R(E)=\{T,F,),i\}$	$L^t(E)=\{+,*,(,i\}$	$R^t(E)=\{+,*,),i\}$
$L(T)=\{T,F,(,i\}$	$R(T)=\{F,)i\}$	$L^t(T)=\{*,(,i\}$	$R^t(T)=\{*,),i\}$
$L(F)=\{(,i\}$	$R(F)=\{),i\}$	$L^t(F)=\{(,i\}$	$R^t(F)=\{),i\}$

Будуємо матрицю передування (табл. 5.3).

Таблиця 5.3 – Матриця передування

Термінальні символи	Термінальні символи					
	+	*	(	)	*i*	$\perp_к$
+	•>	<•	<•	•>	<•	•>
*	•>	•>	<•	•>	<•	•>
(	<•	<•	<•	=•	<•	
)	•>	•>		•>		•>
i	•>	•>		• >		
$\perp_п$	<•	<•	<•		<•	•>

Алгоритм розбору ланцюжків граматики операторного передування ігнорує нетермінальні символи. Тому можна перетворити вихідну граматику таким чином, щоб залишити в ній тільки один нетермінальний символ, наприклад символ S . Тоді отримаємо наступний вид правил:

1. $S \rightarrow S+S$

2. $S \rightarrow S$

3. $S \rightarrow S*S$

4. $S \rightarrow (S)$

5. $S \rightarrow i$

Перевіримо роботу автомата на прикладі вхідного ланцюжка (табл.5.4).

Таблиця 5.4 – Приклад роботи розпізнавання ланцюжка.

Вхідний ланцюжок	Стек	Команда
$i*(i+i)\perp_к$	$\perp_п$	П
$*(i+i)\perp_к$	$\perp_п i$	З(5)
$*(i+i)\perp_к$	$\perp_п S$	П
$(i+i)\perp_к$	$\perp_п S*$	П
$i+i)\perp_к$	$\perp_п S*($	П
$+i)\perp_к$	$\perp_п S*(i$	З(5)
$+i)\perp_к$	$\perp_п S*(S$	П
$i)\perp_к$	$\perp_п S*(S +$	П
$)\perp_к$	$\perp_п S*(S+ i$	З(5)
$)\perp_к$	$\perp_п S*(S+ S$	З(1)
$)\perp_к$	$\perp_п S*(S$	П
$\perp_к$	$\perp_п S*(S)$	З(4)
$\perp_к$	$\perp_п S*S$	З(3)
$\perp_к$	$\perp_п S$	Д

Ланцюжок розпізнано.

5.3.4 Приклад побудови розпізнавача для граматики операторного передування $\Gamma_{5.3}$

Побудувати висхідний розпізнавач для оператора виведення списку ідентифікаторів мови *Python*. Для спрощення у якості ідентифікатору використати ідентифікатор і. Будуємо граматику $\Gamma_{5.3}$ **:**

1. $I \rightarrow$ *>>>print A*
2. $A \rightarrow$ **(***B***)**
4. $B \rightarrow$ *i,B*
5. $B \rightarrow$ *i*

Знаходимо множину крайніх лівих та крайніх правих символів.

$L^t(I)=\{>>>\}$ $R^t(I)=\{print\}$

$L^t(A)=\{(\}$ $R^t(A)=\{)\}$

$L^t(B)=\{i\}$ $R^t(B)=\{,\ , i\}$

Будуємо матрицю передування (табл. 5.5).

Таблиця 5.5 – Матриця передування

Термінальні символи	Термінальні символи						
	>>>	*print*	(	*i*	)	,	$\perp_к$
>>>		=•					
print			<•				
(				<•	=•		
i					•>	=•	•>
)							
;							
,				<•	•>		•>
$\perp_п$	<•						

Перетворимо граматику, замінивши усі нетермінальні символи на один нетермінальний символ I:

1. $I \rightarrow$ <<<*print I*;
2. $I \rightarrow (I)$
3. $I \rightarrow i, i$
4. $I \rightarrow i$

Розглянемо приклад розпізнавання ланцюжка (табл. 5.6)

Таблиця 5.6 – Приклад розпізнавання ланцюжка

Вхідний ланцюжок	Стек	Команда
<<<*print* (*i*,*i*);$\perp_к$	$\perp_п$	П
print (*i*,*i*) $\perp_к$	$\perp_п$<<<	*G*
(*i*,*i*) $\perp_к$	$\perp_п$ <<<*print*	П
i,*i*)$\perp_к$	$\perp_п$ <<<*print* (	П
,*i*) $\perp_к$	$\perp_п$ <<<*print* (*i*	П
i) $\perp_к$	$\perp_п$ <<<*print* (*i*,	П
)$\perp_к$	$\perp_п$ <<<*print* (*i*,*i*	З(4)
) $\perp_к$	$\perp_п$ <<<*print* (*i*,*I*	З(3)
)$\perp_к$	$\perp_п$ <<<*print* (*I*	П
$\perp_к$	$\perp_п$ <<<*print* (*I*)	З(3)
$\perp_к$	$\perp_п$ <<<*print I*	З(1)
$\perp_к$	$\perp_п$ *I*	Д

Ланцюжок розпізнано.

Контрольні запитання

1. Що лежить в основі побудови граматик простого передування?

2. Які існують типи граматик простого передування?

3. Які відносини передування можуть бути виділеними для кожної пари термінальних і нетермінальних символів?

4. Як будується матриця передування для граматики простого передування?

5. Який принцип роботи алгоритму “зсув-згортання” для граматики простого передування?

6. Яка граматика називається граматикою операторного передування?

7. Як будується матриця передування для граматики операторного передування?

8. Чи може бути побудовано розпізнавач для нижченаведеної граматики:

1. $I \rightarrow A$B;

2. $A \rightarrow$ (i)

3. $B \rightarrow$, A

4. $B \rightarrow$ \$.

9. Що потрібно зробити, якщо не встановлене жодне відношення передування між поточним символом вхідного ланцюжка і символом на верхівці стека?

10. Коли виникає помилка в процесі виконання алгоритму роботи розпізнавача?

РЕКОМЕНДОВАНА ЛІТЕРАТУРИ

1. Спекторський І.Я., Статкевич В.М. Формальні мови та автомати . – Київ: КПІ ім. Ігоря Сікорьского, 2019. – 167 с.

2. Гавриленко С.Ю., Клименко А.М. Любченко Н.Ю. та ін. Теорія цифрових автоматів та формальних мов. – Харків, НТУ «ХПІ», 2010. – 176 с.

3. Гавриленко С.Ю., Клименко А.М. Носков. В.І Логіка дискретних автоматів: навч. посіб.–Х: НТУ «ХПІ», 2014.–129 с.

4. Захарія Л. М., Заяць М. М. Формальні мови та граматики: навч. посіб.– Львів, «Львівська політехніка», 2016.– 196 с.

5. Гаврилків В.М. Формальні мови та алгоритмічні моделі: навч. посіб. – Івано-Франківськ: «Сімик», 2012. – 172 с.

6. И. А. Волкова, А. А. Вылиток, Т. В. Руденко. Формальные грамматики и языки. Элементы теории трансляции: Учебное пособие – М.: МГУ, 2009 – 115 с.

7. Пентус А. Е. Контекстно-свободные языки.Сборник задач / А. Е. Пентус, М. Р. Пентус. – М. : РГГУ, 2016. – 75 с.

8. Пентус А. Е. Конечные автоматы и регулярные выражения. Сборник задач / А. Е. Пентус, М. Р. Пентус. – М.: МГУ, 2015. – 72 с.

9. Шостак, Е. В. Основы построения трансляторов языков программирования : учеб.- метод. пособие / Е. В. Шостак, И. М. Марина, Д. Е. Оношко. – Минск: БГУИР, 2019. – 66 с.

10. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий = Compilers: Principles, Techniques, and Tools. – 2-е изд. – М.: Вильямс, 2008.– 768 с.

11. Гавриков М.М. Теоретические основы разработки реализации языков программирования: учеб. пособие/– М.: КНОРУС, 2010. – 184 с.

12. Бенгфорт Б., Бильбро Р., Охеда Т. Прикладной анализ текстових данньх на Python. Машинное обучение и создание приложений обработки естественного языка. СПб.: Питер, 2019. – 368 с.

13. Сопронюк Т.М. Системне програмування. Частина І. Елементи теорії формальних мов: Навчальний посібник у двох частинах. –Чернівці: ЧНУ, 2008. – 84 с.

14. Кудрявцев В. Б., Алешин С. В., Подколзин А. С. Теория автоматов. 2-е изд., испр. и доп. – М.:Издательство Юрайт – 2019 – 320с.

15. МалявкоА.А. Формальные языки и компиляторы. – Новосиб.: НГТУ, 2014. – 431 с.

16. Кононюк А. Е.3 Дискретно-непрерывная математика. (Алгоритмы). – В 12-и кн. Кн. 10,Ч.4– К.: 2017. –788с.

Навчальне електронне видання

ГАВРИЛЕНКО Світлана Юріївна

ФОРМАЛЬНІ МОВИ, ГРАМАТИКИ ТА АВТОМАТИ

Навчальний посібник
для студентів вищих навчальних закладів, які навчаються за спеціальністю
123 «Комп'ютерна інженерія»

Роботу до видання рекомендував Заполовський М.Й

В авторській редакції

Рекомендовано Вченою Радою Національного технічного університету «Харківський політехнічний інститут», протокол № 2, від 26 лютого 2021 року

Самостійне електронне видання