6.2 Граматики Хомського. Ієрархія граматик

Граматики, що породжують (граматики Н. Хомського)[1]

Граматики, що породжують – це простий і потужний механізм, який дозволяє задавати великий клас мов, які містять нескінченну множину ланцюжків. За допомогою граматик, що породжують, ми зможемо, зокрема, визначити мови L_3, L_4 і L_5, для задання яких ми раніше обмежувалися словесними формулюваннями. Граматики, що породжують, використовуються і при описі синтаксису мов програмування.

Граматикою Хомського (граматикою, що породжує) називається четвірка:

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

де T – множина ***термінальних*** (основних) ***символів*** – основний алфавіт. Елементами множини T є символи, з яких в остаточному підсумку і складаються ланцюжки мови, породжуваної даною граматикою. Термінальний (від лат. terminus – межа, кінець) і означає "кінцевий". T – це не що інше, як алфавіт мови, породжуваної граматикою. Надалі, якщо не оговорено особливо, термінальні символи, або просто термінали, будуть позначатися малими буквами латинського алфавіту: a,b,c і т.д.;

N – множина нетермінальних (допоміжних) символів – допоміжний алфавіт. Нетермінальні символи, по-іншому нетермінали, – це поняття граматики (мови), які використовуються при її описі. Нетермінали будемо позначати заголовними латинськими буквами: A, B, C,D,E і т.д.;

P – скінчена множина правил виводу (продукцій). Кожне правило виводу множини P має вигляд:

$$\alpha \to \beta,$$

[1] Авра́м Ноа́м Чо́мскі (також передається як Хо́мський, англ. Avram Noam Chomsky; нар. 7 грудня, 1928, Філадельфія, Пенсильванія) – американський лінгвіст, філософ та політичний активіст, аналітик, літератор, професор мовознавства Масачусетського технологічного інституту (MTI) у відставці. Чомскі добре відомий академічній та науковій спільноті як один із засновників сучасної лінгвістики та визначна постать в аналітичній філософії.

де α і β – ланцюжки термінальних і нетермінальних символів. Ланцюжок α не повинен бути порожнім, ланцюжок β може бути порожнім: $\alpha \in (T \cup N)^+$; $\beta \in (T \cup N)^*$. Правило $\alpha \to \beta$ визначає можливість підстановки β замість α в процесі виводу (породження) ланцюжків мови;

S ($S \in N$) – початковий символ граматики – один з множини нетермінальних символів, початковий (стартовий) нетермінал. Початковий нетермінал – це поняття, що відповідає правильному реченню мови. Наприклад, початковий нетермінал граматики виразів позначає "вираз", а початковий нетермінал граматики мови Паскаль – "програма".

Приклади граматик. Породження речень мови

Приклад 1. Розглянемо граматику

$$G_1 = (\{a, b\},\ \{S\},\ \{S \to aSb,\ S \to \varepsilon\},\ S).$$

Тут всі елементи четвірки записані явно. Множина термінальних символів $T = \{a, b\}$; множина нетерміналів містить один елемент: $N = \{S\}$, а множина правил – два: $P = \{S \to aSb,\ S \to \varepsilon\}$; роль початкового нетермінала виконує S.

Граматика може використовуватися для породження (виводу) ланцюжків – речень мови. Процес породження починається з початкового нетермінала, у нашому прикладі це S. Якщо серед правил є таке, у лівій частині якого записаний ланцюжок S, то початковий нетермінал може бути замінений правою частиною кожного з таких правил. Обидва правила граматики G_1 містять у лівій частині S. Застосуємо підстановку, задану першим правилом, замінивши S на aSb:

$$S \Rightarrow aSb. \quad (1)$$

До ланцюжка, що вийшов, aSb знову, якщо вдасться, можна застосувати одне із правил граматики. Якщо в ланцюжку є підланцюжок, що збігається в лівій частині хоча б одного із правил, то цей підланцюжок можна замінити правою частиною кожного з таких

правил. У ланцюжка aSb є підланцюжок S, що збігається в лівій частині обох правил граматики G_1. Ми вправі застосувати кожне із цих правил. Використаємо знову правило (1) для продовження виводу:

$$S \underset{(1)}{\Rightarrow} aSb \underset{(1)}{\Rightarrow} aaSbb.$$

Тепер до ланцюжка, що вийшов, застосуємо правило (2) ($S \to \varepsilon$), замінивши S порожнім ланцюжком. Одержимо таку послідовність підстановок (саму букву ε в останньому ланцюжку записувати, звичайно, не потрібно):

$$S \underset{(1)}{\Rightarrow} aSb \underset{(1)}{\Rightarrow} aaSbb \underset{(2)}{\Rightarrow} aabb.$$

Очевидно, що до ланцюжка, що вийшов, жодне із правил граматики G_1 більше не застосовується. Процес породження завершений. Неважко помітити, що за допомогою граматики G_1 можна породити будь-який ланцюжок мови $L_1 = \{a^n b^n \mid n \geq 0\}$, застосувавши до початкового нетермінала правило (1) ($S \to aSb$) n разів, а потім один раз правило (2). У той же час, граматика G_1 не породжує ні одного ланцюжка термінальных символів, не приналежних мові L_1. Тобто множина термінальних ланцюжків, породжуваних граматикою G_1, збігається з мовою L_1. Інакше кажучи, граматика G_1 породжує мову L_1:

$$L(G_1) = L_1.$$

Звичайно при записі граматики не записують четвірку її елементів явно. При дотриманні угод про позначення терміналів і нетерміналів досить записати тільки правила. Правила з однаковою лівою частиною можна об’єднати, відокремлюючи альтернативні праві частини вертикальною рискою. Першим записується правило, що містить у лівій частині початковий нетермінал. З врахуванням цього граматика G_1 може бути записана так:

$$G_1\text{: } S \to aSb \mid \varepsilon.$$

Якщо (v, w) належить множині P (v *і* w – ланцюжки з термінальних і нетермінальних символів), то серед правил граматики G існує правило вигляду $v \to w$. Квадратні дужки, круглі дужки і знак альтернативи (|) дозволяють компактніше задавати продукції. Нижче наведено таблицю, в кожному рядку якої розташовані еквівалентні правила граматики.

$v \to w_1$ і $v \to w_2$	$v \to w_1 \mid w_2$
$v \to w_1 w w_2$ і $v \to w_1 w_2$	$v \to w_1[w]w_2$
$v \to w_1 u_1 w_2$ і $v \to w_1 u_2 w_2$	$v_{1\to} w_1(u_1 \mid u_2)w_2$.

Виведення слова за правилами граматики відбувається шляхом поступової заміни нетерміналів на термінальні символи. Заміна нетермінала з лівої частини продукції на її праву частину називається розгортанням нетермінала, а зворотна заміна – згортанням правої частини. Існують дві стратегії аналізу, основані на згортаннях та розгортаннях нетерміналів.

Приклад 2. Розглянемо граматику G_2 (цифри праворуч – номери правил)

$S \to aSBc$ (1)

$S \to abc$ (2)

$cB \to Bc$ (3)

$bB \to bb$ (4)

$S \to \varepsilon$ (5)

Проведемо вивід ланцюжків з початкового нетермінала граматики G_2. Отже:

$$S \Rightarrow aSBc \Rightarrow aabcBc \Rightarrow aabBcc \Rightarrow aabbcc.$$
$$(1) \quad (2) \quad (3) \quad (4)$$

Ще одна серія підстановок:

$$S \Rightarrow aSBc \Rightarrow aaSBcBc \Rightarrow aaabcBcBc \Rightarrow aaabBccBc \Rightarrow aaabBcBcc \Rightarrow$$
$$(1) \quad (1) \quad (2) \quad (3) \quad (3) \quad (3)$$

$$aaabBBccc \Rightarrow aaabbBccc \Rightarrow aaabbbccc$$
$$(4) \quad (4) \quad (4)$$

Можна переконатися, що граматика G_2 породжує ланцюжки терміналів вигляду $a^n b^n c^n$ і ніякі інші. Кількість повторень символів у

результуючому ланцюжку визначається тим, на якому кроці застосовується правило (2). Наявність правила (5) дозволяє одержати порожній ланцюжок, якщо застосувати це правило першим, у той час як спроба використання цього правила на наступних кроках не дозволить вивести ланцюжок терміналів. Граматика G_2 породжує множину термінальних ланцюжків, що збігає з мовою $L_2 = \{a^n b^n c^n \mid n \geq 0\}$:

$$L(G_2) = L_2.$$

Приклад 3. Граматика, що породжує мову правильних дужкових виразів (мова Діка).

G_3: $S \to (S)$ (1)

$S \to SS$ (2)

$S \to \varepsilon$ (3)

Неважко зрозуміти логіку побудови правил цієї граматики. Зміст першого правила такий: взявши в дужки правильний дужковий вираз S, ми знову одержимо правильний дужковий вираз. Друге правило означає, що два правильних дужкових вирази, записані одне за іншим, дають новий правильний вираз. Нарешті, за правилом (3) порожній ланцюжок вважається правильним виразом. Якби ми вирішили, що не слід дозволяти порожні вирази, правило (3) можна було б замінити на $S \to ( )$.

Приклад 4. Граматика простих арифметичних виразів. Єдиним нетерміналом цієї граматики (він же початковий) буде «Вираз»:

G_4: Вираз → Вираз + Вираз | Вираз - Вираз | Вираз * Вираз | Вираз / Вираз | a | b | c | (Вираз).

Така граматика породжує ланцюжки терміналів, що є правильними арифметичними виразами. Символи *a*, *b* і *c* у таких виразах позначають операнди, а "+", "-", "*", і "/" – знаки операцій. Дозволяються круглі дужки (у тому числі вкладені). Приклади правильних виразів: (a+b)/(b*c), (a). Всі операції двомістні, унарні

плюс і мінус не передбачені, тому, наприклад, ланцюжок *a* не належить мові, що породжується цією граматикою.

Вираз – одне з основних понять мов програмування. В подальшому граматикам виразів буде приділена чимала увага. Щоб запис цих граматик був коротшим, замінимо нетермінал «Вираз» на «E» (від expression – вираз), повернувшись тим самим до прийнятої раніше угоди про іменування нетерміналів. Тоді граматика, що позначена G_5, запишеться в такий спосіб:

G_5: E→E + E | E- E | E* E | E/E | a | b | c | (E).

Очевидно, що вона породжує ту ж мову, що і граматика G_4. А саме:

$$L(G_5) = L_5.$$

Граматики можна класифікувати за виглядом їх правил.

Нехай $G = (N, T, P, S)$. Введемо далі наступні позначення:

- a,b, …,0,1 – термінали;
- A, B, C, D, S – не термінали;
- S, A – початкові не термінали;
- α, β – ланцюжки, що містять термінали і нетермінали.

Класифікація, наведена нижче, називається ієрархією за Хомським.

Граматика G називається ***праволінійною (ліволінійною)***, якщо $A \to x, A \to xB$, де $A, B \in N, x \in T^*$ (якщо $A \to x, A \to Bx$, де $A, B \in N, x \in T^*$).

Приклад. $G = (\{S\}, \{0,1\}, \{S \to 0S \mid 1S \mid e\}, S)$ – праволінійна граматика задає мову ланцюжків з нулів та одиниць. Правила граматики тут представляють три продукції, записані з використанням знаку альтернативи (|).

Виведемо один із ланцюжків граматики, використавши для розгортання нетермінала S послідовно перше, друге, перше, перше і третє правила:

S=>0S=> 01S =>010S => 0100S =>0100

Граматика G називається ***контекстно-вільною (безконтекстною)***, якщо кожне правило з P має вигляд: $A \to \alpha, A \in N, \alpha \in (T \bigcup N)^*$.

Приклад. G = ({ A, B},{ x, +, [,] }, P, A)

P: A→x

A→[B]

B→A

B→B+A

Мова L(G) складається з усіх виразів, побудованих з операнда x, операції + та квадратних дужок: x, [x], [x+x], [[x]], [x+x +x], ...

Знайдемо один вираз, який належить мові граматики (нетермінал, який використовується для розгортання, будемо підкреслювати):

A=>[B]=> [B+A] =>[A+A] =>[x+A] =>[x+x]

Аналіз ланцюжка, що відтворює такі розгортання від початкового символу до термінального слова, називається низхідним, або аналізом "від верху до низу". При цьому кожен раз розгортається самий лівий нетермінал.

Якщо вибирати інші правила, то виведеться інший ланцюжок.

Всі ланцюжки, які належать мові граматики, виводяться з початкового нетермінала і складаються лише з термінальних символів. Отже, ланцюжок [x+x] належить мові L(G).

Для того, щоб показати, що деякий ланцюжок w належить мові L(G), потрібно знати послідовність правил, застосування яких веде від початкового нетермінала до ланцюжка w. Для розв'язання цієї задачі треба аналізувати ланцюжок w і фактично будувати дерево виводу (розбору).

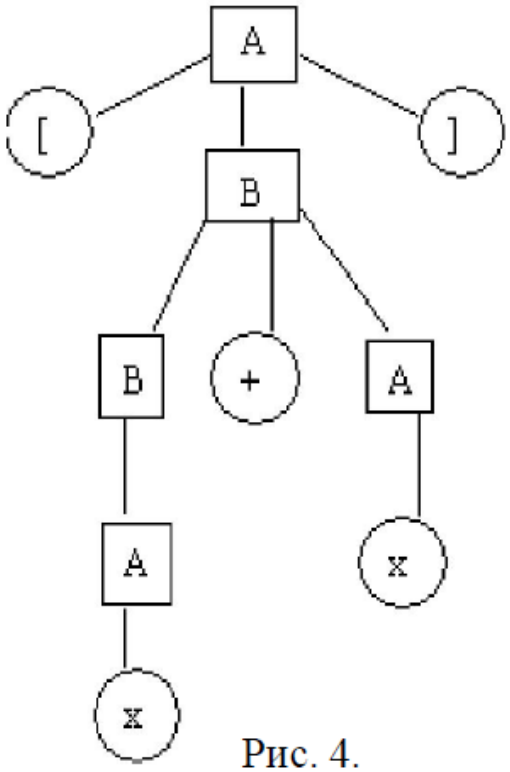


Рис. 4.

Деревом виводу (розбору) граматики G=(N,T,P,S) називається дерево (граф), яке відповідає деякому ланцюжку виводу та задовольняє наступні умови:

- кожна вершина дерева позначається символом граматики $A \in N \cup T$;
- коренем дерева є вершина, позначена головним нетерміналом S;
- листи дерева (кінцеві вершини) є вершини, позначені термінальними символами граматики або символом порожнього ланцюжка;
- якщо деякий вузол дерева позначений нетермінальним символом $A \in N$, а пов'язані з ним вузли – символами $b_1 b_2 \ldots b_n$, $b_i \in N \cup T$, $i=1,\ldots,n$, то в граматиці G=(N,T,P,S) існує правило $A \to b_1 b_2 \ldots b_n \in P$.

З означення дерева виводу видно, що такі дерева завжди можна побудувати для контекстно-вільних граматик і для праволінійних (ліволінийних) граматик.

Контекстно-вільна граматика називається ***однозначною***, якщо для кожного ланцюжка, заданого цією граматикою існує єдине дерево виводу. Отже, існують неоднозначні граматики.

Граматика G називається ***контекстно-залежною*** або нескоротною, якщо кожне правило з P має вигляд: $\alpha \to \beta, \alpha, \beta \in (T \cup N)^*$, $|\alpha| \le |\beta|$.

Зазначимо, що контекстно-залежні граматики не допускають правила вигляду $A \to e$.

Приклад. $G = (\{B, C, S\}, \{a, b, c\}, P, S)$.

P:

$S \to aSBC \mid abC$

$CB \to BC$

$bB \to bb$

$bC \to bc$

$cC \to cc$

Можна показати, що мову цієї граматики можна подати наступним чином $L(G) = \{a^n b^n c^n \mid n \geq 1\}$.

Виведемо один з ланцюжків мови:

$S \Rightarrow aSBC \Rightarrow aabCBC \Rightarrow aabBCC \Rightarrow aabbCC \Rightarrow$

$\Rightarrow aabbcC \Rightarrow aabbcc.$

Граматика, яка не підлягає жодним обмеженням з наведених вище, називається ***необмеженою*** або ***граматикою загального вигляду***.

Праволінійна граматика G = (T, N, P, S) називається ***регулярною (автоматною)***, якщо

- кожне правило із P, за виключенням S→e, має вигляд A→aB або A→a, де A,B∈N, a∈T;
- якщо S→e належить P, то S не зустрічається в правих частинах правил.

Як бачимо з прикладів, продукції граматики дуже схожі на БНФ як за формою, так і за змістом – лише замість знака "::=" вживається "→". Проте в лівій частині їх продукцій може бути не один нетермінал, а цілий ланцюжок, у якому обов'язково є нетермінал. За рахунок такого узагальнення граматики виявляються більш потужним засобом завдання мов, ніж системи БНФ, тобто існують мови, які задаються граматиками, але не задаються БНФ.

Узагальнимо тепер спосіб, у який граматика $G = (T, N, P, S)$ задає мову (фактично, застосування цього способу ми бачили у прикладах):

- на множині $(T \cup N)^*$ означається ***відношення безпосередньої виводимості***, позначене знаком $\Rightarrow_G$ (або $\Rightarrow$, коли відомо, якою саме є G). Ланцюжок w безпосередньо виводиться з ланцюжка v ($v \Rightarrow_G w$), якщо $v = x_1\, u\, x_2$, $w = x_1\, y\, x_2$, $u \to y \in P$. При цьому кажуть, що w безпосередньо виводиться *з* v ***застосуванням продукції*** $u \to y$
- на множині $(T \cup N)^*$ означається ***відношення виводимості***, позначене $\Rightarrow^*_G$ (або $\Rightarrow^*$, коли відомо, якою саме є G). Ланцюжок w безпосередньо виводиться з ланцюжка v ($v \Rightarrow^* w$), якщо $v=w$ або існує послідовність $w_1, w_2, \ldots, w_n$ слів, де $n \geq 1$, така, що $v \Rightarrow w_1$, $w_1 \Rightarrow w_2, \ldots, w_{n-1} \Rightarrow w_n$, $w_n = w$. Ця послідовність називається ***виведенням*** w_n із v, а n – ***довжиною виведення***. Інколи довжину виведення вказують точно: $v \Rightarrow^n w_1$.
- якщо $S \Rightarrow_G^* w$, то послідовність $S \Rightarrow \ldots \Rightarrow w$ називається ***виведенням слова w у граматиці G***, а слово w – ***вивідним***.
- вивідні слова в алфавіті T називаються ***породжуваними***, а множина їх усіх – ***мовою***, що задається (породжується) граматикою G : $L(G) = \{w \mid w \in T^* \text{ та } S \Rightarrow^* w\}$.
- вивідний ланцюжок граматики G , що не містить нетермінальних символів називається ***термінальним ланцюжком***, що породжується граматикою G , а мова, що породжується граматикою G – це множина всіх термінальних ланцюжків, що породжуються граматикою G .
- граматики називаються ***еквівалентними***, якщо задають ту саму мову.
- ланцюжок $\alpha \in (T \cup N)^*$, який виводиться з початкового нетермінала S, називається ***сентенціальною формою*** в граматиці $G=(T,N,P,S)$.
- виведення ланцюжка $\beta \in T^*$ з $S \in N$ у контекстно-вільній граматиці $G=(T, N, P, S)$ називається ***лівим (лівостороннім)***, якщо в цьому виводі кожна чергова сентенціальна форма виводиться з попередньою заміною самого лівого нетермінала.
- виведення ланцюжка $\beta \in T^*$ з $S \in N$ у контекстно-вільній граматиці $G=(T, N, P, S)$ називається ***правим (правостороннім)***, якщо в цьому виводі кожна чергова сентенціальна форма утворюється з попередньою заміною самого правого нетермінала.

У граматиці для одного і того ж ланцюжка може бути кілька виведень, еквівалентних у тому сенсі, що в них у тих самих місцях застосовуються ті самі правила виводі, але в різному порядку. Наприклад, для ланцюжка a+b+a у граматиці G = ({a, b, +}, {S, T}, {S→ T | T+S; T → a | b}, S) можна побудувати такі виведення:

(1) $\underline{S}\Rightarrow T+\underline{S}\Rightarrow T+T+\underline{S}\Rightarrow \underline{T}+T+T\Rightarrow a+\underline{T}+T\Rightarrow a+b+\underline{T}\Rightarrow a+b+a$
(2) $\underline{S}\Rightarrow \underline{T}+S\Rightarrow a+\underline{S}\Rightarrow a+\underline{T}+S\Rightarrow a+b+\underline{S}\Rightarrow a+b+\underline{T}\Rightarrow a+b+a$
(3) $\underline{S}\Rightarrow T+\underline{S}\Rightarrow T+T+\underline{S}\Rightarrow T+T+\underline{T}\Rightarrow T+\underline{T}+a\Rightarrow \underline{T}+b+a\Rightarrow a+b+a$

Тут (2) – лівостороннє виведення, (3) – правостороннє, а (1) не є ні лівостороннім, ні правостороннім, але всі ці виведення є еквівалентними в зазначеному вище сенсі.

Ієрархія граматик Н. Хомського

Розпізнавання структури ланцюжка мови відбувається, звичайно, при відновленні виведення цього ланцюжка з початкового символу граматики. Таке відновлення називається синтаксичним аналізом (або парсингом, від англ. – parse), воно являє собою перетворення вхідного ланцюжка в послідовність кроків його виведення. У трансляторах парсинг повинен бути виконаний автоматично відповідно до деякого алгоритму. Але чи існує такий загальний алгоритм синтаксичного аналізу, відповідний для будь-яких граматик Хомського, і якщо існує, то яка його складність?

Очевидно, що якщо загальний алгоритм синтаксичного аналізу взагалі існує, він може бути виконаний машиною Тьюринга.

Найпростіший такий алгоритм, який може бути реалізований на машині Тьюринга, складається просто з недетермінованої перевірки можливих підстановок правих частин продукцій замість підланцюжків проміжного ланцюжка виведення, які збігаються з лівими частинами продукцій. Доведено, що нічого кращого для загальної проблеми розпізнавання синтаксичної структури ланцюжка придумати не можливо. Однак, для такого алгоритму ми не можемо знати при довільній вхідний стрічці машини Тьюринга, чи зупиниться ця машина (або в заключному стані, або через неможливість застосування продукції), і не можемо оцінити час роботи алгоритму.

Таким чином, виявляється, що в загальному випадку завдання синтаксичного аналізу не може бути ефективно вирішено в класі моделі породжуючої граматики Хомського.

На цьому можна було б завершити розгляд цього підходу до трансляції мов, вважаючи його невдалим і непридатним на практиці, оскільки з цього факту, здавалося б, випливає неможливість побудови загальної теорії і необхідність розробки свого індивідуального алгоритму синтаксичного аналізу для кожного конкретного випадку. Однак, як це часто буває в науці, проблеми, які важко або навіть неможливо вирішити в загальній постановці, ефективно вирішуються для частинних постановок. Більш того, саме частинні постановки цих проблем, як виявляється, покривають переважну більшість виникаючих на практиці важливих випадків.

Саме така ситуація виявляється в теорії формальних мов і граматик. Найважливішу роль в методах трансляції грають окремі випадки, підкласи загальних граматик Хомського, і, зокрема, контекстно-вільні граматики, правила яких значно простіші, ніж правила граматик Хомського загального вигляду.

Очевидно, що саме форма правил граматики (а не їх кількість) має істотне значення для алгоритмів синтаксичного аналізу. Хомський визначив чотири типи граматик в залежності від того, який вигляд мають ліва та права частини правил

виду $\alpha \to \beta$. Хомський ввів чотири різних типи мов, породжуваних цими граматиками, і для кожного сімейства згодом були знайдені класи абстрактних перетворювачів (автоматів), які можуть здійснити відповідне перетворення, а саме, за заданим ланцюжком α і граматиці G з цього класу побудувати вивід цього ланцюжка в G. Чим більше обмежень накладається на вигляд правил в цьому підкласі граматик, тим ефективнішим виявляється алгоритм синтаксичного аналізу для цього підкласу.

Н. Хомський запропонував поділ граматик, що породжують, на чотири типи, залежно від виду їх правил.

Тип 0. Граматики загального вигляду (або **необмежені**). На вигляд їх правил не накладається яких-небудь обмежень. Будь-яка граматика, що породжує, є граматикою типу 0. Правила мають вигляд:

$$\alpha \to \beta,$$

де α і β – ланцюжки терміналів і нетерміналів. Ланцюжок α не повинен бути порожнім.

Тип 1. Контекстно-залежні (або **нескоротні**) **граматики.**

Граматика G = {T,N,P,S} називається ***нескоротною***, якщо права частина кожного правила з P не коротша лівої частини (тобто для будь-якого правила $\alpha \to \beta \in P$ виконується нерівність $|\alpha| \le |\beta|$). У вигляді виключення у нескоротній граматиці допускається наявність правила $S \to \varepsilon$, за умови, що S не зустрічається в правих частинах правил.

В деякій літературі тип 1 визначають за допомогою так званих контекстно-залежних граматик.

Граматика G = {T,N,P,S} називається ***контекстно-залежною*** (КЗ), якщо кожне правило з P має вигляд $\alpha \to \beta$, де $\alpha = \xi_1 A \xi_2$, $\beta = \xi_1 \gamma \xi_2$, $A \in N$, $\gamma \in (T \cup N)^+$, $\xi_1, \xi_2 \in (T \cup N)^*$. У вигляді виключення в КЗ-граматиці допускається наявність правила з порожньою правою частиною $S \to \varepsilon$, за умови, що S не зустрічається в правих частинах правил.

Ланцюжок ξ_1 називають лівим контекстом, ξ_2 – правим контекстом. Мова, що породжується контекстно-залежною граматикою, називається контекстно-залежною мовою.

Твердження. Нехай L – формальна мова. Наступні твердження є еквівалентними:

1) існує контекстно-залежна граматика G_1, така що $L = L(G_1)$;

2) існує нескоротна граматика G_2, така що $L = L(G_2)$.

З даного твердження випливає, що мова, яка породжується нескоротною граматикою, є контекстно-залежною. Таким чином, нескоротні та контекстно-залежні граматики визначають один і той же клас мов.

Тип 2. Контекстно-вільні граматики (КВ-граматики). Їх правила мають вигляд:

$$A \to \gamma,$$

де A – нетермінал; γ – ланцюжок терміналів і нетерміналів. Характерна риса – у лівій частині правил завжди один нетермінальний символ.

Зауважимо, що у КВ-граматиках допустимі правила з порожніми правими частинами.

Тип 3. Регулярні (або **автоматні) граматики.**

Граматика G = {T,N,P,S} називається ***праволінійною***, якщо кожне правило з P має вигляд $A \to wB \mid w$, де $w \in T^*$, $A,B \in N$.

Граматика G = {T,N,P,S} називається ***ліволінійною***, якщо кожне правило з P має вигляд $A \to Bw \mid w$, де $w \in T^*$, $A,B \in N$.

Праволінійні та ліволінійні граматики називають ***регулярними***.

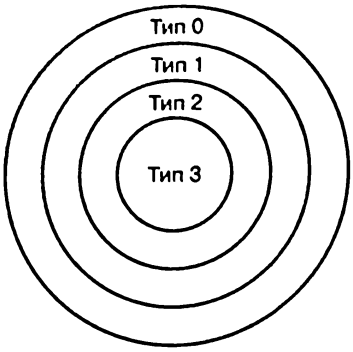
Автоматною граматикою називається праволінійна (ліволінійна) граматика, така, що кожне правило з непорожньою правою частиною має вигляд: $A \to wB \mid w$ (для ліволінійних, відповідно, $A \to Bw \mid w$), де $w \in T$, $A,B \in N$.

Автоматна граматика є більш простою формою регулярної граматики. Існує алгоритм, що дозволяє за регулярною (право- або ліволінійною) граматикою побудувати відповідну автоматну граматику. Таким чином, будь-яка регулярна мова може бути породжена автоматною граматикою. Крім того, існує алгоритм, що дозволяє усунути з регулярної (автоматної) граматики все ε-правила (крім $S \to \varepsilon$ в разі, якщо порожній ланцюжок належить мові; при цьому S не зустрічатиметься в правих частинах правил).

Ієрархія Хомського є включаючою. Як можна бачити, граматики типу 1 є частинним випадком граматик типу 0, граматики типу 2 – окремим випадком контекстно-залежних граматик, автоматні – окремим випадком контекстно-вільних. Тобто, граматика типу 3 є і граматикою типу 2, і типу 1, і типу 0.

Таким чином, маємо наступну ієрархію типів мов:

$$\text{Тип 3 (Регулярні)} \subset \text{Тип 2 (КВ)} \subset \text{Тип 1 (КЗ)} \subset \text{Тип 0}.$$



Мови, породжувані граматиками типу 0-3, називаються відповідно мовами без обмежень (або рекурсивно-перелічувальними), контекстно-залежними, контекстно-вільними і автоматними (або регулярними) мовами. Але контекстно-вільною мовою називається мова, для якої існує породжуюча її контекстно-вільна, але не автоматна граматика. Такий же підхід застосовується до контекстно-залежних мов і мов без обмежень.

Розпізнавачі для різних класів граматик

Під розпізнавачем будемо розуміти узагальнений алгоритм, що дозволяє визначити деяку множину (в нашому випадку – мову) і використовує в своїй роботі такі компоненти: вхідну стрічку, керуючий пристрій з кінцевої пам'яттю і додаткову робочу пам'ять. Як приклади розпізнавачів можна назвати машину Тьюринга, скінченні та магазинні автомати.

Виявляється, кожному класу граматик з ієрархії Хомського відповідає клас розпізнавачів, що визначає той же клас мов:

- мова L регулярна тоді і тільки тоді, коли вона визначається (одностороннім детермінованим) кінцевим автоматом;
- мова L контекстно-вільна тоді і тільки тоді, коли вона визначається (одностороннім недетермінованим) автоматом з магазинною пам'яттю;
- мова L контекстно-залежна тоді і тільки тоді, коли вона визначається (двостороннім недетермінованим) автоматом з магазинною пам'яттю;
- мова L рекурсивно-перелічувальна тоді і тільки тоді, коли вона визначається машиною Тьюринга.

Розпізнавач:

Деякий алгоритм, що дозволяє визначити мову (L) і використовує:

1) вхідну стрічку;

2) керуючий пристрій з скінченою пам'яттю;

3) доповнювач до робочої пам'яті (машина Тьюринга, кінцеві автомати, магазинний автомат).

Якщо розпізнавач зупиняється в заключному стані, то ланцюжок належить мові (див. табл. 1).

Таблиця 1

Тип	Мови за Хомським	Розпізнавач
3	Регулярна мова	Скінчений автомат (односторонній детермінований)
2	Контекстно-вільна мова	Автомат з магазинною пам’яттю (односторонній недетермінований)
1	Контекстно-залежна мова	Лінійно-обмежений автомат (двосторонній недетермінований)
0	Рекурсивно-перелічувальна мова	Машина Тьюринга

СПОСОБИ ВИЗНАЧЕННЯ ФОРМАЛЬНИХ МОВ (ПРОДОВЖЕННЯ)

Відомі різні способи опису мов. Скінченну мову можна описати простим перерахуванням її ланцюжків. Оскільки формальна мова може бути і нескінченною, потрібні механізми, що дозволяють скінченним чином представляти нескінченні мови. Можна виділити два основних підходи для такого представлення: механізм розпізнавання і механізм породження (генерації).

Основний спосіб реалізації *механізму породження* – використання граматик, що породжують, які також називають граматиками Хомського. Граматики, що породжують, було розглянуто минулої лекції.

Механізм розпізнавання (*розпізнавач*), по суті, є процедурою спеціального виду, яка за заданим ланцюжком визначає, чи належить він мові. Якщо належить, то процедура зупиняється з відповіддю «так», тобто допускає ланцюжок; інакше – зупиняється з відповіддю «ні» або зациклюється. Мова, яка визначається розпізнавачем – це безліч всіх ланцюжків, які він допускає.

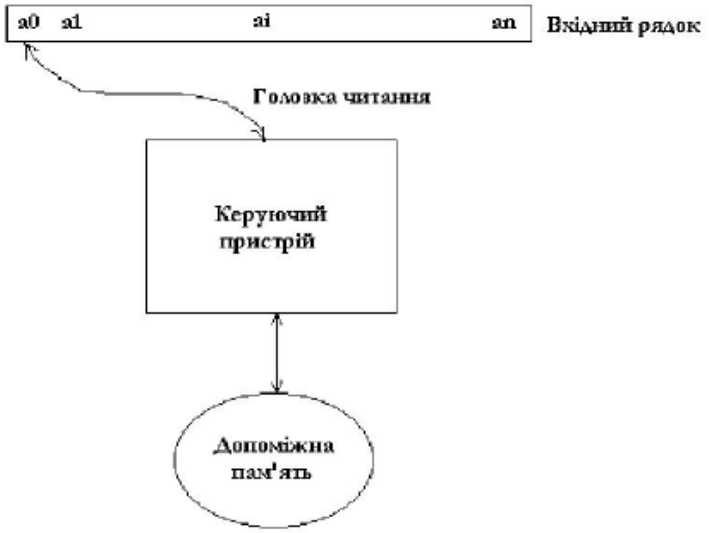
Приклади розпізнавачів:

- Машина Тьюринга (МТ). Мова, яку можна задати за допомогою МТ, називається рекурсивно-перелічуваною. Замість МТ можна використовувати еквівалентні алгоритмічні схеми: нормальний алгоритм Маркова (НАМ), машину Поста та ін.
- Лінійно-обмежений автомат (ЛОА). Являє собою МТ, в якій стрічка не нескінченна, а обмежена довжиною вхідного слова. Визначає контекстно-залежні мови.
- Автомат з магазинної (зовнішньої) пам'яттю (МП-автомат). На відміну від ЛОА, головка не може змінювати вхідне слово і не може зрушуватися вліво; є додаткова нескінченна пам'ять (магазин, або стек). Визначає контекстно-вільні мови.
- Скінченний автомат (СА). Відрізняється від МП-автомата відсутністю магазину. Визначає регулярні мови.

Розпізнавачі

Розпізнавач – це схематизований алгоритм, який визначає деяку множину ланцюжків. Він складається з трьох частин:

- вхідна стрічка;
- керуючий пристрій із скінченною пам’яттю;
- робоча (допоміжна) пам’ять.



Вхідна стрічка – це послідовність комірок, в якій можуть знаходитись символи алфавіту. У кожен

момент часу головка читання переглядає одну комірку, За один крок головка може зсунутись на одну комірку вправо, вліво або ж залишитися на місці. Розпізнавач, який ніколи не зсуває головку вліво називається ***одностороннім***. Як правило вважається, що вхідна головка читає, тобто під час роботи розпізнається якийсь символ.

Пам'ять розпізнавача – сховище інформації або даних. Внутрішній алфавіт пам'яті Q скінченний і інформація в пам'яті побудована тільки із символів даного алфавіту. Описати пам'ять (її вміст і структуру) можна скінченними засобами, а об'єм пам'яті може бути як завгодно великим. Важливим прикладом допоміжної пам'яті є магазинна пам'ять. Магазинна пам'ять містить ланцюжок символів $z_1 z_2 ... z_n, z_i \in Q$, а z_1- верхній символ магазину. Поведінка допоміжної пам'яті характеризується за допомогою двох функцій: функція доступу і функція перетворення пам'яті.

Функція доступу до пам'яті відображає множину можливих станів (конфігурацій) у можливу множину інформаційних символів. Функція доступу до пам'яті у випадку магазинної пам'яті визначається таким чином: $f(z_1, z_2, ..., z_n) = z_1$.

Функція перетворення пам'яті – це правило, яке описує зміну пам'яті. Тип допоміжної пам'яті визначає назву розпізнавача. Якщо пам'ять влаштована, як магазин, то розпізнавач називається розпізнавачем із магазинною пам'яттю.

Ядром розпізнавача є керуючий пристрій із скінченною пам'яттю, під яким можна розуміти програму, що керує роботою розпізнавача. Керуючий пристрій являє собою скінченну множину станів. Він керує рухом головки читання. Розпізнавач працює дискретно, виконуючи послідовно такти. На початку такту зчитується поточний символ і за допомогою функції доступу досліджується пам'ять. Вхідний символ, інформація із пам'яті і поточний стан керуючого пристрою визначають дії під час такту:

- вхідна головка рухається вліво, вправо або залишається на місці;
- в пам'яті розміщується деяка інформація і змінюється стан керуючого пристрою.

Поведінку розпізнавача зручно задавати у термінах конфігурацій. У кожен момент часу ***конфігурація*** містить інформацію про

- стан керуючого пристрою;
- містиме вхідної стрічки разом з положенням вхідної головки;

- містиме пам'яті.

Керуючий пристрій може бути детермінованим і недетермінованим. Якщо керуючий пристрій недетермінований, то для довільної конфігурації існує скінченна кількість наступних кроків. Будь-який з них розпізнавач може зробити.

Керуючий пристрій називається ***детермінованим***, якщо для кожної конфігурації існує не більше одного наступного детермінованого кроку.

Конфігурація називається ***початковою***, якщо керуючий пристрій знаходиться в заданому початковому стані, вхідна головка розглядає найлівіший символ, а пам'ять має початкове містиме.

Конфігурація називається ***заключною***, якщо керуючий пристрій знаходиться в одному із заключних станів, а вхідна головка оглядає правий кінець стрічки. Іноді вимагається, щоб в заключній конфігурації і пам'ять задовольняла певним умовам.

Кажуть, що розпізнавач ***допускає*** вхідний рядок ω, якщо починаючи із початкової конфігурації, розпізнавач може виконати послідовність кроків, що закінчиться заключною конфігурацією.

Якщо розпізнавач недетермінований, то він може виконувати різну послідовність кроків. Якщо одна з цих послідовностей закінчується заключною конфігурацією, то вхідний рядок ***допустимий***.

Мова, що дозволяється розпізнавачем – це множина вхідних ланцюжків, які він допускає.

Для кожного класу граматик із ієрархії Хомського існує свій клас розпізнавачів, які визначають ті самі класи мов, що і граматики, наприклад:

- мова L праволінійна тоді і тільки тоді, коли вона розпізнається одностороннім детермінованим скінченним розпізнавачем;
- мова L контекстно-вільна тоді і тільки тоді, коли вона визначається одностороннім недетермінованим розпізнавачем з магазинною пам'яттю;
- мова L контекстно-залежна тоді і тільки тоді, коли вона розпізнається двостороннім недетермінованим обмеженим розпізнавачем.

Скінченні автомати

Неформально скінченний автомат можна розуміти як систему, яка у кожен момент часу може знаходиться в одному з скінченної кількості заданих станів. Кожен такт автомата полягає в тому, що при перебуванні у будь-якому стані і зчитуванні одного з вхідних символів він переходить у визначений стан, причому символи, які зчитувались раніше не запам'ятовуються. Скінчений автомат – це односторонній скінченний розпізнавач без пам'яті.

Формально ***скінченним автоматом*** M називається п'ятірка позначень $M = (Q, \Sigma, \delta, q_0, F)$, де

Q - скінченна множина станів автомату $Q = \{q_0, q_1, \ldots, q_n\}$;

Σ - скінченна множина вхідних символів $\Sigma = \{a_1, a_2, \ldots, a_m\}$;

δ - відображення множини $Q \times \Sigma$ в $P(Q)$ (множину всіх підмножин Q), що визначає поведінку керуючого пристрою (функцію $\delta(q, a)$, де $q \in Q$, $a \in \Sigma$, називають функцією переходів);

$q_0 \in Q$ - початковий стан керуючого пристрою;

$F \subseteq Q$ - множина заключних станів.

Робота скінченного автомату полягає у виконанні послідовності кроків (тактів). Такт визначається поточним станом керуючого пристрою, який оглядається головкою читання. Зміна стану автомата – це зміна керуючого пристрою і зсув головки вправо на одну комірку.

Для того, щоб визначити поведінку скінченного автомата треба знати

- поточний стан керуючого пристрою;
- ланцюжок символів, що починаються з комірки, на який вказує головка читання.

Ці два параметри визначають миттєвий опис скінченного автомата, який називається ***конфігурацією (тактом)***.

Пара $(q,\omega) \in Q \times \Sigma^*$ називається ***конфігурацією*** автомата M.

Конфігурація (q_0,ω) називається ***початковою конфігурацією***, а (q,e), $q \in F$ - ***заключною***.

Кажуть, що автомат переходить з конфігурації $(q, a\omega)$, $a \in \Sigma$, в конфігурацію (q', ω), якщо $q' \in \delta(q,a)$ і позначають це так: $(q,a\omega) \mapsto (q',\omega)$.

Робота скінченого автомата – це послідовність конфігурацій. Нехай $K_0, K_1, ..., K_p$ - деякі конфігурації автомата. Якщо можна перейти $K_0 \mapsto K_1 \mapsto ... \mapsto K_p$, то це можна позначити так: $K_0 \overset{*}{\mapsto} K_p$ або $K_0 \overset{p}{\mapsto} K_p$.

Кажуть, що автомат M ***допускає ланцюжок*** ω, якщо $(q_0,\omega) \overset{p}{\mapsto} (q,e)$, де $q \in F$.

Мовою, що визначається (допускається) автоматом M, називається множина вхідних ланцюжків, що допускається цим автоматом.

$$L(M) = \{\omega \mid \omega \in \Sigma^*, (q_0,\omega\} \overset{*}{\mapsto} (q,e) \text{ для деякого } q \in F\}.$$

Стан p називається ***досяжним***, якщо існує ланцюжок ω такий, що $(q_0,\omega) \overset{*}{\mapsto} (p,e)$.

За даним скінченним автоматом можна знайти найменший еквівалентний йому автомат, якщо вилучити всі недосяжні стани, а потім склеїти еквівалентні стани.

Кажуть, що ланцюжок $x \in \Sigma^*$ ***розрізняє*** стани $q_1 \neq q_2$, $q_1 \in Q$, $q_2 \in Q$, якщо із стану q_1 по ланцюжку x можна перейти в q_3, а з

q_2 по x можна перейти в q_4, причому $q_3 \in F$, $q_4 \notin F$ або $q_3 \notin F$, $q_4 \in F$.

Якщо ланцюжок x такий, що $(q_1, x) \mapsto^k (q_3, e)$, $(q_2, x) \mapsto^k (q_4, e)$, то ланцюжок x довжиною k розрізняє ці стани.

Кажуть, що стани q_1, q_2 $k-$***не розрізняються:*** $q_1 \equiv^k q_2$, якщо не існує ланцюжка $x \in \Sigma^*, |x| \le k.$, який розрізняє ці стани.

Будемо говорити, що q_1, q_2 ***не розрізняються*** і писати $q_1 \equiv q_2$, якщо вони k -не розрізняються для будь-якого $k \in N$.

Стан q називається ***недосяжним***, якщо не існує вхідного ланцюжка $x \in \Sigma^* : (q_0, x) \mapsto^* (q, e)$.

Скінченний автомат M є ***недетермінованим***, якщо існують такі $a \in \Sigma, q \in Q$, що функція переходів δ є множиною, що складається більш, ніж з одного стану: $\delta(q,a) = \{q_1, q_2, ..., q_n\}$, $n > 1$.

Скінченний автомат M є ***детермінованим***, якщо для кожних $a \in \Sigma, q \in Q$ існує не більше одного стану, в який відбувається перехід.

Скінченний детермінований автомат називається ***повністю визначеним***, якщо $\delta(q,a)$ для кожних $a \in \Sigma, q \in Q$ містить рівно один стан.

Скінченний детермінований автомат M називається ***зведеним (мінімальним)***, якщо в Q немає недосяжних станів і немає двох станів, що не розрізняються.

Діаграмою переходів (графом переходів) автомата M називається навантажений граф, вершини якого навантажені іменами станів автомату, і в діаграмі присутнє ребро (p, q), якщо справджується таке співвідношення: $q \in \delta(p, a), a \in \Sigma$. Ребра навантажені всіма символами $a \in \Sigma$, по яких є перехід з p в q.

Тобто, діаграмою переходів скінченого автомата є граф, вершинами якого є стани автомата, а ребра графа визначаються функцією переходу автомата.

Приклад недетермінованого скінченного автомата

Побудуємо автомат в алфавіті $\Sigma = \{1,2,3\}$, який буде розпізнавати слова, в яких є хоча б одна двійка.

Наприклад, $w_1 = 12231 \in L(M)$, $w_2 = 1131131 \notin L(M)$.
Визначимо стани:

p - початковий стан: двійки ще не було

q - заключний стан: двійка вже була.

Отже, автомат має вигляд $M = (\{p,q\},\{1,2,3\},\delta,p,\{q\})$.

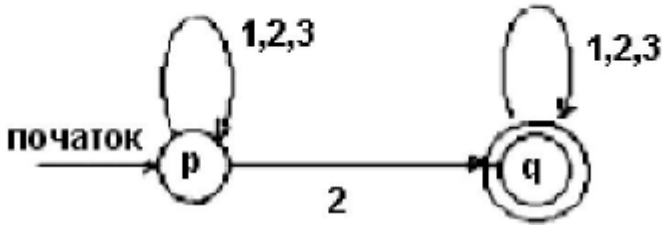
Визначимо функцію переходів δ:

$$\delta(p,1) = p, \delta(p,2) = \{p,q\}, \delta(p,3) = p,$$
$$\delta(q,1) = q, \delta(q,2) = q, \delta(q,3) = q.$$

Функцію δ прийнято зображати у вигляді ***таблиці переходів***:

Стани	Вхідні символи		
	1	2	3
p	$\{p\}$	$\{p,q\}$	$\{p\}$
(q)	$\{q\}$	$\{q\}$	$\{q\}$

Функцію δ можна зобразити у вигляді ***діаграми переходів***:



Заключний стан у таблиці обведений, а на діаграмі – знаходиться у подвійному кружечку.

Нехай у нас є ланцюжок 12231. Випишемо всі можливі послідовності конфігурацій:

$(p,12231) \mapsto (p,2231) \mapsto (p,231) \mapsto (p,31) \mapsto (p,1) \mapsto (p,\text{e})$
$(p,12231) \mapsto (p,2231) \mapsto (p,231) \mapsto (q,31) \mapsto (q,1) \mapsto (q,\text{e})$

$(p,12231) \mapsto (p,2231) \mapsto (q,231) \mapsto (q,31) \mapsto (q,1) \mapsto (q,\text{e})$

Оскільки серед останніх конфігурацій присутня заключна конфігурація (q,e), то існує послідовність, що перетворює початкову конфігурацію $(p,12231)$ в заключну. Це означає, що ланцюжок $12231 \in L(M)$.

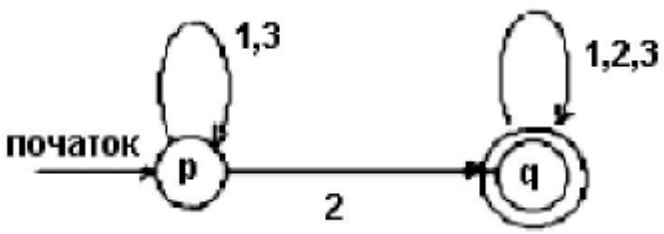
Приклад детермінованого скінченного автомата

Знов побудуємо автомат в алфавіті $\Sigma = \{1,2,3\}$, який буде розпізнавати слова, в яких є хоча б одна двійка. Але тепер він буде детермінованим.

Визначимо такі ж стани, як і раніше, але змінимо функцію переходів δ:

$$\delta(p,1) = p, \delta(p,2) = \{q\}, \delta(p,3) = p,$$
$$\delta(q,1) = q, \delta(q,2) = q, \delta(q,3) = q.$$

Стани	Вхідні символи		
	1	2	3
p	$\{p\}$	$\{q\}$	$\{p\}$
(q)	$\{q\}$	$\{q\}$	$\{q\}$



Отже, автомат має вигляд $M = (\{p,q\},\{1,2,3\},\delta,p,\{q\})$. Він повністю визначений і мінімальний.

Оскільки автомат детермінований, то для будь-якого ланцюжка існує єдина послідовність конфігурацій. Нехай у нас є ланцюжок 12231. Тоді послідовність конфігурацій має вигляд:

$$(p,12231) \mapsto (p,2231) \mapsto (q,231) \mapsto (q,31) \mapsto (q,1) \mapsto (q,\text{e})$$

Ця послідовність перетворює початкову конфігурацію в заключну. Це означає, що ланцюжок $12231 \in L(M)$.

Зазначимо, що працювати з детермінованим автоматом простіше, ніж з недермінованим, але, як правило, для складних мов його непросто визначити. Існують алгоритми перетворення недетермінованого автомата в детермінований, а потім – і у мінімальний. Вони розглянуті далі, а нижче наведено алгоритм моделювання роботи скінченного детермінованого автомату:

```
q = q0;                 // початковий стан автомата
a = GetChar();          // зчитування першого символу з вхідної стрічки
while (a != eof)        // поки не кінець стрічки
{
  // перехід автомату в інший стан по прочитаному символу
  q = Delta(q, a);      // функція переходів Delta керує роботою автомата
  a = GetChar();        // зчитування наступного символу з вхідної стрічки
}                       // стрічку прочитано
if (q is in F)          // автомат попав в заключний стан ?
       return "yes";    // ланцюжок належить мові автомата
else return "no";       // ланцюжок не належить мові автомата
```

Магазинні автомати

Магазинні автомати, відомі також як автомати з магазинною пам'яттю або як МП-автомати, формально визначаються у такий спосіб:

МП-автомат – це сімка позначень P = (Q, Σ, Г, δ, q_0, Z_0, F), де:

Q – скінченна множина станів;

Σ – скінченний вхідний алфавіт;

Г – скінченний алфавіт магазинних символів;

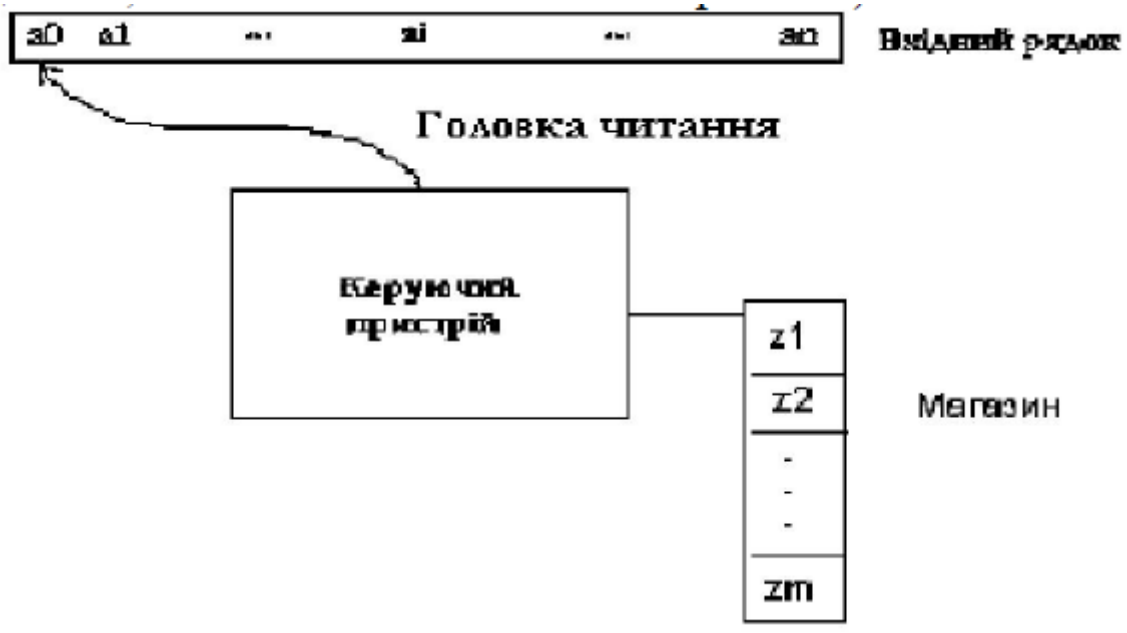
δ – функція переходу, відображення множини $Q \times (\Sigma \cup \{e\}) \times \Gamma$ у множину скінченних підмножин множини $Q \times \Gamma^*$;

$q_0 \in Q$ – початковий стан керуючого пристрою;

$Z_0 \in \Gamma$ – символ, що знаходиться в магазині у початковий момент (початковий символ);

$F \subseteq Q$ – множина заключних станів.

Конфігурацією МП-автомата P називається трійка позначень $(q,\omega,\alpha) \in Q \times \Sigma^* \times \Gamma^*$, де q – поточний стан керуючого пристрою, ω – невикористана частина вхідного ланцюжка (якщо $\omega=\varepsilon$, то вважається, що весь вхідний ланцюжок прочитаний), α – вміст магазина (самий лівий символ ланцюжка α вважається верхнім символом магазина; якщо $\alpha=\varepsilon$, те магазин вважається порожнім).



На кожному кроці роботи МП-автомат може або занести щось у магазин, або зняти якісь значення з його вершини. Відзначимо, що МП-автомат може продовжувати працювати у випадку закінчення вхідного ланцюжка, але не може продовжувати роботу у випадку спустошення магазина.

Клас мов, що розпізнаються МП-автоматами, у точності збігається з класом мов, які задаються контекстно-вільними граматиками.

МП-автомат P = (Q, Σ, Γ, δ, q_0, Z_0, F) називається ***детермінованим***, якщо для кожних $q \in Q$ і $Z \in \Gamma$ вірно одне з наступних тверджень:

- δ(q, a, Z) містить не більш одного елемента для кожного $a \in \Sigma$ і δ(q, ε, Z) = Ø;
- δ(q, a, Z) = Ø для всіх $a \in \Sigma$ і δ(q, ε, Z) містить не більш одного елемента.

Детерміновані МП-автомати описують тільки підмножину з класу контекстно-вільних мов – ця підмножина називається ***детермінованими контекстно-вільними мовами***. Цей клас мов називають також LR(k)-граматиками, тому що вони можуть бути однозначно розібрані шляхом перегляду ланцюжка зліва направо із загляданням уперед не більш, ніж на k символів.

Приклад магазинного автомата

Розглянемо магазинний автомат, що розпізнає мову $\{0^n1^n \mid n \geq 0\}$. Нехай P = ($\{q_0, q_1, q_2\}$, {0,1}, {Z,0}, δ, q_0, Z, $\{q_0\}$), де:

$\delta(q_0, 0, Z) = \{(q_1, 0Z)\}$
$\delta(q_1, 0, 0) = \{(q_1, 00)\}$
$\delta(q_1, 1, 0) = \{(q_2, \varepsilon)\}$
$\delta(q_2, 1, 0) = \{(q_2, \varepsilon)\}$
$\delta(q_2, \varepsilon, Z) = \{(q_0, \varepsilon)\}$

Робота автомата полягає в копіюванні в магазин початкових нулів із вхідного ланцюжка і наступному вилученню по одному нулю з магазина на кожну прочитану одиницю.

Регулярні множини і регулярні вирази

Нехай V- скінченний алфавіт. Рекурсивно ***регулярна множина*** в алфавіті V визначається так:

1) $\varnothing$ – регулярна множина в алфавіті V;
2) $\{e\}$ – регулярна множина в алфавіті V;
3) $\{a\}$, де $a \in V$ – регулярна множина в алфавіті V;
4) Якщо P, Q - регулярні множини в V, то $P \cup Q, PQ, P^*$ – регулярні множини;
5) Ніщо інше не є регулярною множиною в V.

Таким чином, множина в V є регулярною тоді і тільки тоді, коли вона є регулярною внаслідок або 1), або 2), або 3), або її можна одержати з них за допомогою операції об'єднання, конкатенації чи ітерації. Регулярні множини можна визначити через регулярні вирази, праволінійні граматики, детерміновані і недетерміновані скінченні автомати.

Регулярний вираз в алфавіті V визначається рекурсивно:

1) $\varnothing$ - регулярний вираз, що позначає регулярну множину $\varnothing$;
2) e - регулярний вираз , що позначає регулярну множину {e};
3) якщо a∈V, то a - регулярний вираз , що позначає регулярну множину {a};
4) якщо p,q - регулярні вирази, що позначають регулярні множини P,Q, то
   - (p+q) регулярний вираз, що позначає регулярну множину P∪Q;
   - (pq) регулярний вираз, що позначає регулярну множину PQ;
   - (p)* регулярний вираз, що позначає регулярну множину P* ;
5) ніщо інше не є регулярним виразом.

Для спрощення запису і для того, щоб в регулярних виразах писати менше дужок, введемо позначення ($p^+ = pp^*$) та пріоритети операцій : ітерація (*), конкатенація, об'єднання (+).

Операцію об'єднання будемо називати ще й операцією альтернативи і позначати вертикальною рискою (|).

Приклади регулярних виразів:

- 01 позначає множину {01}
- 0* позначає множину {0}*
- (0+1)* позначає множину {0,1}*
- (0+1)*011 позначає множину усіх ланцюжків, що складаються з нулів і одиниць і закінчуються ланцюжком 011
- (a+b)(a+b+0+1)* позначає множину усіх ланцюжків з множини {a,b,0,1}*, що починаються з a або b.

Для кожного регулярного виразу існує регулярна мова, а для довільної регулярної мови існують різні регулярні вирази. Будемо говорити, що ***регулярні вирази рівні***, якщо вони позначають однакові регулярні множини.

Нехай α, β, γ - регулярні вирази. Тоді справджуються ***тотожності*** над регулярними виразами:

1. $\alpha + \beta = \beta + \alpha$
2. $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$
3. $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$
4. $\alpha e = e\alpha = \alpha$
5. $\alpha^* = \alpha + \alpha^*$
6. $\alpha + \alpha = \alpha$
7. $\varnothing^* = e$
8. $\alpha(\beta\gamma) = (\alpha\beta)\gamma$
9. $(\alpha + \beta)\gamma = \alpha\gamma + \beta\gamma$
10. $\alpha \varnothing = \varnothing \alpha = \varnothing$
11. $(\alpha^*)^* = \alpha^*$
12. $\alpha + \varnothing = \alpha$