



ТЕОРІЯ ФОРМАЛЬНИХ МОВ

Лекція 3

Регулярні граматики та регулярні вирази

Регулярні правила · розпізнавачі і складність · алгоритм згортки · синтаксис regex · застосування

ПЛАН ЛЕКЦІЇ

Що ми розглянемо сьогодні

1

Ієрархія Хомського, розпізнавачі

Автомат і складність розпізнавання для кожного типу

3

Розпізнавач і алгоритм згортки

Побудова розбору «знизу вгору» за таблицею

5

Синтаксис регулярних виразів

Квантифікатори, класи символів

7

Складність і застосування

$O(n)$ розпізнавання, лексери, валідація, `grep/regex`

2

Регулярні правила та застереження

Право-, ліволінійні граматики; чому їх не можна змішувати

4

Регулярні операції та вирази

Об'єднання, конкатенація, зірка Кліні; формальне визначення

6

Еквівалентність і приклади мов

Коли два вирази описують одну мову

8

Приклади опису мов

Від найпростіших до складніших шаблонів

Нагадування з лекції 2

Регулярні граматики (тип 3) - найвужчий, найпростіший клас в ієрархії Хомського.

$$\text{Тип 3} \subset \text{Тип 2} \subset \text{Тип 1} \subset \text{Тип 0}$$

Права частина кожного правила містить не більше одного нетермінала, і він стоїть на фіксованому краю ланцюжка, тобто усі правила мають вигляд $A \rightarrow aB$ або $A \rightarrow a$.

Кожному типу - свій автомат і своя складність

Вибір типу граматики визначає не лише виразальну силу мови, а і складність алгоритму, що перевіряє належність слова цій мові.

Тип	Розпізнавач	Пам'ять	Складність розбору
0 - загальна	Машина Тьюринга	необмежена	може не зупинитись
1 - контекстно-залежна	Лінійно обмежений автомат	лінійно обмежена	експоненційна
2 - контекстно-вільна	МП-автомат (зі стеком)	стек (необмежений)	поліноміальна
3 - регулярна	Скінченний автомат	$O(1)$ - лише поточний стан	$O(n)$ - лінійна

Саме тому регулярні мови - «робочий кінь» лексичного аналізу: перевірка належності слова відбувається за лінійний час і сталу пам'ять.

Праволінійна та ліволінійна форми

ПРАВОЛІНІЙНА

$$A \rightarrow aV, A \rightarrow a, A \rightarrow \varepsilon$$

Нетермінал у правій частині завжди стоїть останнім (крайнім праворуч) символом. Слово будується зліва направо.

$$S \rightarrow aS \mid b$$

ЛІВОЛІНІЙНА

$$A \rightarrow Va, A \rightarrow a, A \rightarrow \varepsilon$$

Нетермінал завжди стоїть першим (крайнім ліворуч) символом. Слово будується справа наліво.

$$S \rightarrow Sa \mid b$$

Теорема

Клас мов, породжених праволінійними граматиками, збігається з класом мов, породжених ліволінійними граматиками і обидва задають клас регулярних мов. Змішувати праволінійні й ліволінійні правила в одній граматичі не можна — граматика має бути послідовно одного типу.

Одна мова, дві граматики

Мова $L = \{ a^n b : n \geq 0 \}$ - довільна кількість «а», за якою йде рівно одне «b».

ПРАВОЛІНІЙНА: $S \rightarrow aS \mid b$

Виведення «aaab»: $S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaab$

ЛІВОЛІНІЙНА: $S \rightarrow Sa \mid b$

Виведення «aaab»: $S \Rightarrow Sa \Rightarrow Saa \Rightarrow Saaa \Rightarrow ba aa$

На практиці частіше використовують праволінійні граматики, вони природно відповідають скінченним автоматам, що читають слово зліва направо.

Не можна змішувати ліво- і праволінійні правила

ЯКЩО В ОДНІЙ ГРАМАТИЦІ ОДНОЧАСНО Є:

$A \rightarrow aB$ і $C \rightarrow Dd$

(перше правило праволінійне, друге - ліволінійне) - така граMATика в загальному випадку НЕ є регулярною.

Чому це важливо:

- Кожен нетермінал у праволінійній граматиці природно відповідає стану скінченного автомата, що читає слово зліва направо.
- Ліволінійне правило вимагало б «читати» слово в протилежному напрямку - сумісний автомат довелося б будувати інакше.
- Обидві форми окремо еквівалентні класу регулярних мов, але їхнє довільне змішування цю гарантію руйнує.

Які слова може породжувати граматика?

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

$T = \{a, b\}$

$P = \{ S \rightarrow bS,$

$S \rightarrow aA,$

$S \rightarrow \varepsilon,$

$A \rightarrow bA,$

$A \rightarrow aS \}$

$SS \Rightarrow bS \Rightarrow bbS \Rightarrow bb$

$S \Rightarrow aA \Rightarrow aaS \Rightarrow aa$

$S \Rightarrow aA \Rightarrow abA \Rightarrow abaS \Rightarrow aba$

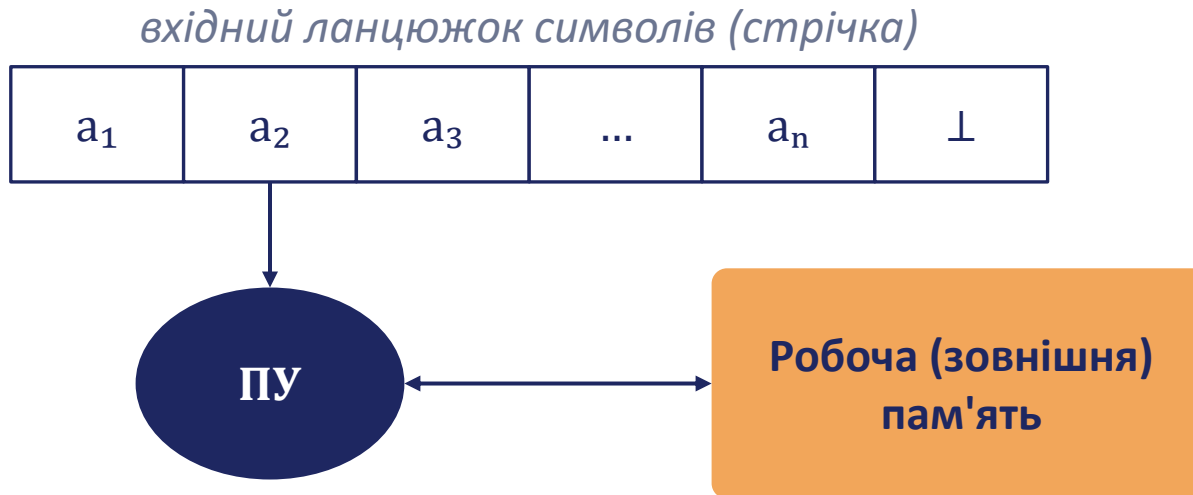
$S \Rightarrow aA \Rightarrow aaS \Rightarrow aabS \Rightarrow aabbS \Rightarrow aabb$

$S \Rightarrow bS \Rightarrow baA \Rightarrow baaS \Rightarrow baabS \Rightarrow baab$

$S \Rightarrow aA \Rightarrow abA \Rightarrow abbA \Rightarrow abbaS \Rightarrow abba$

Загальна схема автомата-розпізнавача

Розпізнавач - автомат, що визначає належність ланцюжка символів заданій мові.
Складається з трьох частин:



Стрічка + зчитуюча голівка - вхідний ланцюжок; ПУ (пристрій управління) - скінченна множина станів; зовнішня пам'ять - від відсутньої (СА) до необмеженої (машина Тьюринга).

Компілятор виступає розпізнавачем щодо вихідного тексту програми; автор програми - «генератором» ланцюжка цієї мови.

Згортка для ліволінійної грамматики

Р: $S \rightarrow C\perp$; $C \rightarrow Ab \mid Ba$; $A \rightarrow a \mid Ca$; $B \rightarrow b \mid Cb$ ($\perp$ - маркер кінця ланцюжка)

	a	b	$\perp$
C	A	B	S
A	–	C	–
B	C	–	–
S	–	–	–

fold(стан, символ) = наступний стан

ПРИКЛАД: слово « $abab\perp$ »

1) перший символ $a \rightarrow$ базове правило $A \rightarrow a \Rightarrow$ стан A

2) $(A, b) \rightarrow C$ (fold(A,b)=C)

3) $(C, a) \rightarrow A$ (fold(C,a)=A)

4) $(A, b) \rightarrow C$ (fold(A,b)=C)

5) $(C, \perp) \rightarrow S$ (fold(C, $\perp$)=S)

Досягли S $\rightarrow$ ланцюжок ПРИЙМАЄТЬСЯ

Ідея: зчитаний символ «згортаємо» в нетермінал за таблицею - це побудова дерева розбору «знизу вгору» (від листків до кореня), крок за кроком.

Об'єднання, конкатенація, зірка Кліні

ОБ'ЄДНАННЯ

$L_1 \cup L_2$

$$\{a, ab\} \cup \{b, ba\} = \{a, ab, b, ba\}$$

КОНКАТЕНАЦІЯ

$L_1 L_2$

$$\{a, ab\} \cdot \{b, ba\} = \{ab, aba, abb, abba\}$$

ЗІРКА КЛІНІ

L^*

$$\{a\}^* = \{\epsilon, a, aa, aaa, \dots\}$$

Клас регулярних мов будується індуктивно з $\emptyset$, $\{\epsilon\}$ та $\{a\}$ за допомогою цих трьох операцій

Які слова може породжувати граматика?

Нехай:

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

Тоді зірка Кліні дозволяє брати **a або b** і складати з них слова довільної довжини.

Отримаємо:

$$L^*=\{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \dots\}$$

Тобто фактично:

$\{a,b\}^*$ - це **множина всіх слів над алфавітом $\{a,b\}$.**

Рекурсивне визначення

- 1 $\emptyset$ є регулярним виразом
- 2 ϵ є регулярним виразом
- 3 кожен символ $a \in \Sigma$ є регулярним виразом
- 4 якщо r, s - регулярні вирази, то $r|s, rs, r^*$ - також регулярні вирази

$$L(a|b) = \{a, b\} \quad L(ab) = \{ab\} \quad L(a^*) = \{\epsilon, a, aa, aaa, \dots\}$$

Праволінійна граматика $S \rightarrow aS \mid b$ відповідає регулярному виразу a^*b .

Квантифікатори та класи символів

*

0 або більше

+

1 або більше

?

0 або 1

{n,m}

від n до m

[a-z]

мала латинська літера

\d

цифра

\w

буква, цифра або «_»

\s

Пробельний символ

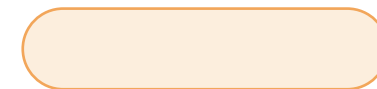
[^0-9]

будь-який символ, крім цифри

()

групування / захоплення

Приклад: ідентифікатор мови програмування - `[a-zA-Z_][a-zA-Z0-9_]*`; ціле число - `[0-9]+`



Приклади регулярних виразів

Email:

```
^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$
```

Номер телефону (міжнародний формат):

```
^\+?[1-9]\d{1,14}$
```

Ідентифікатор змінної(C#/Python/Java):

```
^[a-zA-Z_][a-zA-Z0-9_]*$
```

$r \equiv s$, якщо $L(r) = L(s)$

ЕКВІВАЛЕНТНІ

$a|a \equiv a$

Обидва описують мову $\{a\}$.

НЕЕКВІВАЛЕНТНІ

$(ab)^* \not\equiv (ab)(ab)^*$

Перший допускає ϵ , другий - ні (перша «ab» обов'язкова).

Еквівалентність не залежить від зовнішньої схожості запису - лише від того, чи описують два вирази одну й ту саму мову.

Мова ідентифікаторів у програмуванні

Мова ідентифікаторів у програмуванні:

Рядки, які починаються з однієї з букв a або b, за якою може йти довільна кількість цифр 0 або 1.

Приклади допустимих слів: a, b, a0, b1, a0110, b1101

1. Формалізація через Регулярний Вираз

Регулярний вираз описує структуру мови як єдину компактну формулу:

$$R = (a \mid b)(0 \mid 1)^*$$

$(a \mid b)$ - вибір першого символу: або a, або b.

$(0 \mid 1)^*$ - замикання Кліні: будь-яка послідовність (включаючи порожній рядок) з 0 та 1.

2. Формалізація через Регулярну Граматику $G = (N, T, P, S)$, де:

Термінали: $T = \{a, b, 0, 1\}$

Нетермінали: $N = \{S, A\}$ (де S - початковий стан, а A - стан зчитування цифр)

Правила виводу P:

$S \rightarrow aA \mid bA \mid a \mid b$ $A \rightarrow 0A \mid 1A \mid 0 \mid 1$

Лема Ардена

1 Лема Ардена — це важливий математичний інструмент у теорії формальних мов та автоматів, який використовується для розв'язання рівнянь відносно регулярних виразів і мов.

2 Вона дає готовий алгоритмічний розв'язок для рекурсивних алгебраїчних рівнянь вигляду

$$X = A X \cup B$$

(або в термінах регулярних виразів $X = AX + B$).

Якщо підставимо X сам у себе:

$$X = A(AX \cup B) \cup B = A^2 X \cup AB \cup B$$

$$X = A(A^2 X \cup AB \cup B) \cup B = A^3 X \cup A^2 B \cup AB \cup B$$

Лема Ардена

*Знайти регулярний вираз
для рівняння:*

$$X = 0X \cup 1$$

Розв'язок за лемою Ардена

Порівняємо з загальною формою $X = AX \cup B$:

$$A = 0$$

$$B = 1$$

Оскільки $\varepsilon \notin \{0\}$, лема Ардена дає єдиний розв'язок:

$$X = A^* B = 0^* 1$$

Результат. Слово починається з довільної кількості нулів 0, а закінчується одиницею

6

ПРИКЛАД

Приклад застосування леми Ардена

Граматика: $S \rightarrow aA \mid \varepsilon$, $A \rightarrow bS$

1

Рівняння

$$S = aA + \varepsilon, \quad A = bS$$

2

Підстановка A у перше рівняння

$$S = a(bS) + \varepsilon = abS + \varepsilon$$

3

Лема Ардена ($A_{\text{Арден}} = ab$, $B = \varepsilon$)

$$S = (ab)^*\varepsilon = (ab)^*$$

Від простого до складного

Усі слова над $\{a,b\}$

$(a|b)^*$

Починаються з a

$a(a|b)^*$

Закінчуються на b

$(a|b)^*b$

Містять підрядок ab

$(a|b)^*ab(a|b)^*$

Двійкові, закінч. на 01

$(0|1)^*01$

Натуральні числа без 0

$[1-9][0-9]^*$

Мова $a^n b^n$ НЕ може бути описана жодним регулярним виразом - немає «пам'яті» для підрахунку кількості символів.

Чому регулярні мови всюди в інженерії ПЗ

Розпізнавання слова довжини n : $O(n)$ за часом, $O(1)$ за додатковою пам'яттю

Лексичний аналіз

Розбиття коду на токени:
ідентифікатори, числа, ключові слова

`[a-zA-Z_][a-zA-Z0-9_]*`

Валідація даних

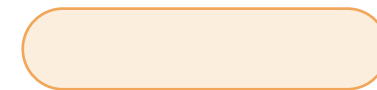
Перевірка формату email, телефону,
URL

`^[\\w.-]+@[\\w.-]+\\.\\w+$`

Пошук і обробка тексту

Утиліти `grep`/`sed`; `regex` у
Python/JavaScript

`grep -E "^[0-9]+$"`



Regular Expressions у мовах програмування

Python (re модуль)

```
import re
pattern = r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"
if re.match(pattern, email):
    print("Valid email")
```

C# (System.Text.RegularExpressions)

```
using System.Text.RegularExpressions;
string pattern = @"\d+";
MatchCollection matches = Regex.Matches(text, pattern);
```

re.search()

перший збіг у рядку

re.match()

збіг з початку рядка

re.fullmatch()

перевірка всього рядка

re.findall()

усі знайдені збіги

re.sub()

заміна фрагментів

re.split()

поділ рядка за шаблоном

```
import re
text = "У групі 25 студентів, з них 12 виконали роботу."
numbers = re.findall(r"\d+", text)
print(numbers)
# ['25', '12']
```

Ключові тези

- Регулярна граматики буває праволінійною або ліволінійною; змішувати обидві форми в одній граматиці не можна.
- Кожному типу граматики Хомського відповідає свій розпізнавач і своя складність: тип 3 розбирається за $O(n)$ часу й $O(1)$ пам'яті.
- Розпізнавач (стрічка + ПУ + пам'ять) - загальна модель; для ліволінійних граматик існує алгоритм розбору «знизу вгору» через таблицю згортки.
- Регулярні операції ($\cup$, конкатенація, $*$) та регулярний вираз (рекурсивне означення) - два еквівалентні способи описати ту саму мову.
- Квантифікатори ($*$, $+$, $?$, $\{n,m\}$) і класи символів ($[a-z]$, $\backslash d$, $\backslash w$) роблять регулярні вирази практичним інструментом.
- Лінійна складність розпізнавання - причина, чому регулярні мови лежать в основі лексичних аналізаторів, валідації даних і утиліт `grep/regex`.