Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики

СИСТЕМНЕ ПРОГРАМУВАННЯ

Синтез і Аналіз Мовних Процесорів

Віктор Волохов
Нікіта Скибицький

Київ — 2019

Зміст

Розділ 1

Лексичний аналіз

1.1 Мови програмування та мовні процесори

1.1.1 Мови програмування

При вивченні мов програмування, як правило, виділяють три аспекти:

- Прагматичний;
- Семантичний;
- Синтаксичний.

Прагматичний аспект

Прагматичний аспект (прагматика мови програмування) визначає клас задач, на розв'язування яких орієнтується мова програмування. Як правило, прагматичний аспект менш формалізований у порівнянні з семантичним та синтаксичним аспектами.

За класом задач на розв'язування яких орієнтуються мови програмування їх можна поділити передусім на

- процедурні;
- непроцедурні.

Процедурні мови програмування орієнтовані перш за все на опис (визначення) алгоритмів, тобто по суті використовуються для побудови процедур обробки даних. До таких мов ми відносимо всім відомі мови програмування, такі як Pascal, Fortran, C та ін.

Непроцедурні мови програмування на відміну від процедурних неявно визначають процедури обробки даних. Частіше всього такі мови використовуються для побудови завдань на обробку даних. При цьому, за допомогою інструкцій непроцедурної мови програмування визначається що необхідно зробити з даними і явно не визначається як (з використанням яких алгоритмів) необхідно розв'язати задачу. До непроцедурних мов програмування ми відносимо командні мови операційних систем, мови управління в пакетах прикладних програм та ін.

Як процедурні, так і непроцедурні мови програмування можуть орієнтуватися як на декілька класів задач, так і конкретну предметну область. У першому випадку ми будемо говорити про *універсальні* мови програмування (Pascal, Fortran, C), в другому — про *спеціалізовані* мови програмування (Snobol, Lisp).

Семантичний аспект

Семантичний аспект (семантика мови програмування) визначається шляхом конкретизації базових функцій обробки даних, набору конструкцій управління та методами побудови більш "складних" програм на основі "простих".

Наприклад, визначивши як базовий тип даних "рядок" ми повинні запропонувати "традиційний" набір функцій обробки таких даних: порівняння рядків, виділення частини рядка, конкатенацію рядків та ін.

Семантика мови програмування має бути визначена формально, бо інакше у подальшому неможливо буде побудувати відповідний мовний процесор. Станом на сьогодні існують два основних напрямки визначення семантики мов програмування:

- методи денотаційної семантики;
- методи операційної семантики.

Методи *денотаційної семантики* базуються на відповідних алгебрах, методи *операційної семантики* базуються на синтаксичних структурах програм.

Синтаксичний аспект

Синтаксичний аспект (синтаксис мови програмування) визначає набір синтаксичних конструкцій мови програмування, які використовуються

для нотації (запису) семантичних одиниць в програмі. Про синтаксис мови програмування можна сказати як про форму, яка є суть похідною від семантики. Для визначення (опису) синтаксису мови програмування використовуються як механізми, що орієнтовані на синтез, так і механізми, орієнтовані на аналіз.

Задачі аналізу та синтезу синтаксичних структур програм — це дуальні задачі. Їх конкретизацію ми будемо розглядати в наступних розділах.

Виходячи з вищенасказаного, щоб побудувати мову програмування потрібно:

- визначити клас (класи) задач, на розв'язок яких орієнтована мова програмування;
- виділити базові типи даних та функції їх обробки, вказати конструкції управління в програмах. Побудувати механізми конструювання більш складних програм та структур даних на основі більш простих одиниць;
- визначити синтаксис мови програмування.

1.1.2 Мовні процесори

Мовні процесори реалізують мови програмування. Точніше, мовний процесор призначений для обробки програм відповідної мови програмування. З точки зору прагматики, мовні процесори діляться на

- транслятори;
- інтерпретатори.

Мовний процесор типу транслятор (транслятор) — це програмний комплекс, котрий на вході отримує текст програми на вхідній мові, а на виході видає версію програми на вихідній мові, що називається об'єктною мовою. В більшості випадків як об'єктна мова виступає мова команд деякої обчислювальної машини. Серед трансляторів можна виділити дві програмні системи:

- компілятори — транслятори з мов програмування високого рівня;
- асемблери — транслятори машинно-орієнтованих мов програмування.

Мовний процесор типу інтерпретатор (інтерпретатор) — це програмний комплекс, котрий на вході отримує текст програми на вхідній мові та вхідні дані, які в подальшому обробляються програмою, а на виході видає результати обчислень (вихідні дані).

Оскільки транслятори та інтерпретатори реалізують мови програмування, вони мають спільні риси: їх структура досить схожа, в основу їх реалізації покладено спільні теоретичні результати та практичні методи реалізації.

Структура транслятора

1. Вхідний текст програми
2. Лексичний аналіз
3. Синтаксичний аналіз
4. Семантичний аналіз
5. Оптимізація проміжного коду
6. Генерація коду
7. Вихідний (об'єктний) код

Призначення основних компонентів транслятора

1. *Лексичний аналізатор.*

   Вхід: вхідний текст (послідовність літер) програми.

   Вихід: послідовність лексем програми.

   Лексема — це ланцюжок літер, що має певний зміст. Всі лексеми мови програмування (їх кількість, як правило, нескінчена) можна розбити на скінчену множину класів. Для більшості мов програмування актуальні наступні класи лексем:

   - зарезервовані слова;
   - ідентифікатори;
   - числові константи (цілі та дійсні числа);
   - літерні константи;
   - рядкові константи;

- коди операцій;
- коментарі. Безпосередньо не несуть інформації щодо структури програми. В подальшому не використовуються, тобто не передаються синтаксичному аналізатору.
- дужки та інші елементи програми.

2. *Синтаксичний аналізатор.*

   Вхід: послідовність лексем програми.

   Вихід:

   - "Так" + синтаксична структура (синтаксичний терм) програми,
   - "Ні" + синтаксичні помилки в програмі.

3. *Семантичний аналізатор.*

   Вхід: Синтаксичний терм програми.

   Вихід:

   - "Так" + семантична структура (семантичний терм) програми,
   - "Ні" + семантичні помилки в програмі.

4. *Оптимізація проміжного коду.*

   Вхід: семантичний терм програми.

   Вихід: оптимізований семантичний терм програми.

   Оптимізація — це еквівалентне перетворення програми на основі певних критеріїв. Серед критеріїв оптимізації можна виділити:

   - оптимізацію по пам'яті;
   - оптимізацію по швидкості виконання.

   В залежності від підходів по оптимізації програми можна розглядати такі методи оптимізації:

   - машинно-залежні;
   - машинно-незалежні.

На відміну від машинно-незалежних методів машинно-залежні методи оптимізації враховують архітектурні особливості ЕОМ, наприклад, наявність апаратного стека, наявність вільних регістрів, тощо.

5. *Генерація об'єктного коду.*

   Вхід: семантичний терм програми.

   Вихід: результуючий (об'єктний) код програми.

1.1.3 Контрольні запитання

1. Які три аспекти як правило виділяють при вивчення мов програмування?
2. Які два поділи мов програмування в залежності від орієнтації на розв'язання тих чи інших класів задач вам відомі?
3. Які традиційні функції обробки типу даних "рядок" вам відомі?
4. Які два класи задач пов'язаних з синтаксичними структурами програм вам відомі?
5. Які два типи мовних процесорів вам відомі?
6. Опишіть структуру транслятора.
7. Що таке лексема?
8. Які два поділи оптимізації ви знаєте?

1.2 Лексичний аналіз та скінченні автомати

1.2.1 Лексичний аналіз в мовних процесорах

Призначення: перетворення вхідного тексту програми з формату зовнішнього представлення в машинно-орієнтований формат — послідовність лексем.

Нагадаємо, що *лексема* — це ланцюжок літер елементарний об'єкт програми, що несе певний семантичний зміст. В подальшому кожну лексему будемо представляти як пару $\langle$клас лексеми, ім'я лексеми$\rangle$.

В більшості мов програмування для визначення класів лексем достатньо скінчених автоматів.

1.2.2 Скінчені автомати

Недетермінований скінчений автомат — це п'ятірка $M = \langle Q, \Sigma, \delta, q_0, F\rangle$, де

- $Q = \{q_0, q_1, \ldots, q_{n-1}\}$ — скінчена множина станів автомата;
- $\Sigma = \{a_1, a_2, \ldots, a_m\}$ — скінчена множина вхідних символів (вхідний *алфавіт*);
- $q_0 \in Q$ — *початковий* стан автомата;
- δ — відображення множини $Q \times \Sigma$ в множину 2^Q. Відображення δ як правило називають *функцією переходів*;
- $F \subset Q$ — множина заключних станів. Елементи з F називають *заключними* або *фінальними* станами.

Якщо M — скінчений автомат, то пара $(q, w) \in Q \times \Sigma^\star$ називається *конфігурацією* автомата M. Оскільки скінчений автомат — це дискретний пристрій, він працює по тактам. *Такт* скінченого автомата M задається бінарним відношенням $\models$, яке визначається на конфігураціях:

$$(q_1, aw) \models (q_2, w) \quad \text{if} \quad q_2 \in \delta(q_1, a), \quad \forall w \in \Sigma^\star.$$

Мова яку розпізнає скінченний автомат

Скінченний автомат M *розпізнає (допускає)* ланцюжок w, якщо

$$\exists q \in F: \quad (q_0, w) \models^\star (q, \varepsilon),$$

де $\models^\star$ — рефлексивно-транзитивне замикання бінарного відношення $\models$.

Мова, яку допускає автомат M (розпізнає автомат M)

$$L(M) = \{w \mid w \in \Sigma^\star, \exists q \in F : (q_0, w) \models^\star (q, \varepsilon)\}.$$

Способи визначення функції переходів

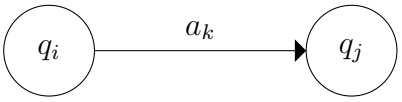
На практиці, при визначенні скінченого автомата M, використовують декілька способів визначення функції δ, наприклад:

- це табличне визначення δ;
- діаграма проходів скінченого автомата.

Табличне визначення функції δ — це таблиця $M(q_i, a_j)$, де $a_j \in \Sigma, q_i \in Q$, тобто

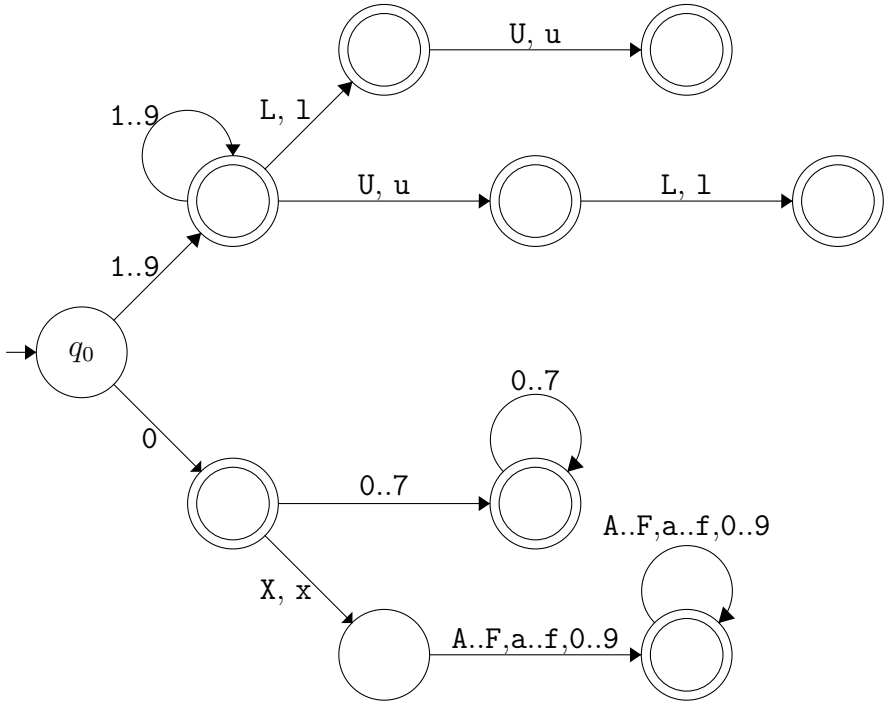
$$M(q_i, a_j) = \{q_k \mid q_k \in \delta(q_i, a_j)\}.$$

Діаграма переходів скінченого автомата M — це невпорядкований граф $G(V, P)$, де V — множина вершин графа, а P — множина орієнтованих дуг, причому з вершини q_i у вершину q_j веде дуга позначена a_k, коли $q_j \in \delta(q_i, a_k)$. На діаграмі переходів скінченого автомата це позначається так:



В подальшому, на діаграмі переходів скінченого автомата M елементи з множини заключних станів будемо позначити так:

Приклад. Побудуємо діаграму переходів скінченого автомата M, який розпізнає множину цілочислових констант мови C.



Зауваження. Цей автомат неповний, на два нижні праві вузли потрібно довісити "UL"-частину яка висить на вузлі "1..9".

З побудованого прикладу видно, що приведений автомат не повністю визначений.

Детерміновані скінченні автомати

Скінчений автомат M називається *детермінованим*, якщо $\delta(a_i, a_k)$ містить не більше одного стану для любого $q_i \in Q$ та $a_k \in \Sigma$.

Теорема. *Для довільного недетермінованого скінченого автомата M можна побудувати еквівалентний йому детермінований скінчений автомат M', такий що $L(M) = L(M')$.*

Доведення: Нехай M — недетермінований скінчений автомат

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle.$$

Детермінований автомат $M' = \langle Q', \Sigma, \delta', q_0', F \rangle$ побудуємо таким чином:

1. $Q' = 2^Q$, тобто імена станів автомата M' — це підмножини множини Q.

2. $q'_0 = \{q_0\} \in 2^Q = Q'$.

3. F' складається з усіх таких підмножин $S \in 2^Q = Q'$, що $S \cap F \neq \varnothing$.

4. $\delta'(S, a) \models \{q \mid q \in \delta(q_i, a), q_i \in S\}$.

Доводимо індукцією по i, що $(S, w) \models^i (S', \varepsilon)$, тоді і тільки тоді, коли $S' = \{q \mid \exists q_i \in S : (q_i, w) \models^i (q, \varepsilon)\}$.

Зокрема, $(\{q_0\}, w) \models^\star (S', \varepsilon)$, для деякого $S' \in F'$, тоді і тільки тоді, коли $\exists q \in F : (q_0, w) \models^\star (q, \varepsilon)$.

Таким чином, $L(M) = L(M')$.

Побудований нами автомат M має дві властивості: він детермінований та повністю визначений. До того ж кількість станів цього автомата $2^n - 1$.

1.2.3 Контрольні запитання

1. У чому призначення лексичного аналізу?

2. Що таке недетермінований скінчений автомат?

3. Яку мову розпізнає скінченний автомат?

4. Які два способи визначення функції переходів ви знаєте?

5. Спробуйте "зламати" вищенаведений автомат для цілочислових констант мови C (зверніть увагу на зауваження).

6. Що таке детермінований скінчений автомат?

7. Сформулюйте і доведіть теорему про детермінізацію скінченного автомата.

8. Нехай функція переходів δ не однозначна, але у той же час набуває не багато різних значень на одному наборі аргументів, наприклад не більше двох, тобто $|M(q, a)| \leq 2$ для довільних $q \in Q$ і $a \in \Sigma$. Чи можна тоді отримати кращу оцінку зверху на кількість станів еквівалентного детермінованого автомату ніж $2^n - 1$?

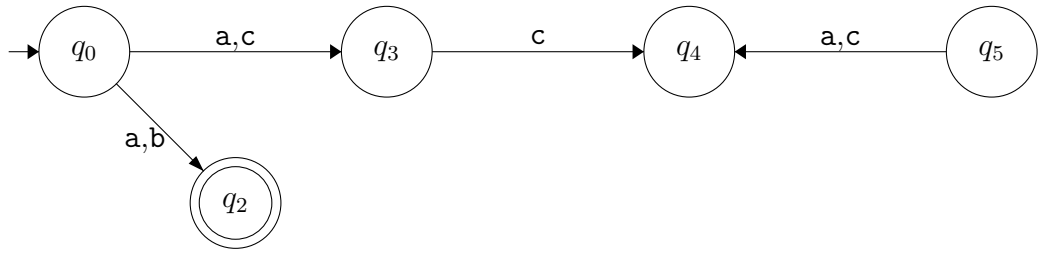
1.3 Мінімізація детермінованих скінчених автоматів

1.3.1 Мінімізація детермінованих скінчених автоматів

В подальшому при програмуванні скінчених автоматів важливо мати справу з так званими "мінімальними автоматами". *Мінімальним* для даного скінченого автомата називається еквівалентний йому автомат з мінімальною кількістю станів.

Нагадаємо, що два автомати називаються *еквівалентними* якщо вони розпізнають одну мову.

Те, що скінчені автомати можна мінімізувати покажемо на наступному прикладі:



Навіть при поверхневому аналізі діаграми переходів наведеного скінченого автомата видно, що вершини q_3, q_4 та q_5 є "зайвими", тобто при їх вилученні новий автомат буде еквівалентний початковому. З наведеного вище прикладу видно, що для отриманого детермінованого скінченого автомата можна запропонувати еквівалентний йому автомат з меншою кількістю станів, тобто мінімізувати скінчений автомат. Очевидно що серед зайвих станів цього автомата є недосяжні та тупикові стани.

Недосяжні стани

Стан q скінченого автомата M називається *недосяжним*, якщо на діаграмі переходів скінченого автомата не існує шляху з q_0 в q.

Алгоритм [пошуку недосяжних станів]. Спочатку спробуємо побудувати множину досяжних станів. Якщо Q_m — множина досяжних станів скінченого автомата M, то $Q \setminus Q_m$ — множина недосяжних станів. Побудуємо послідовність множин $Q_0, Q_1, Q_2, \ldots$ таким чином, що:

1. $Q_0 = \{q_0\}$.

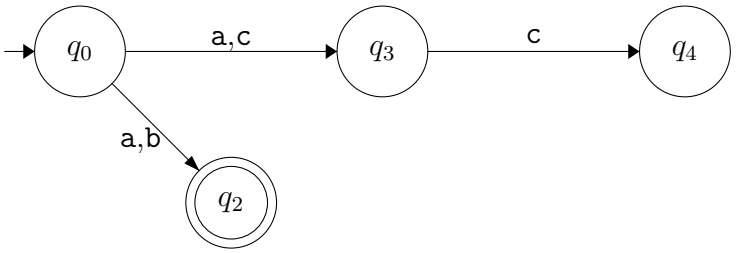
2. $Q_i = Q_{i-1} \cup \{q \mid \exists a \in \Sigma, q_j \in Q_{i-1} : q \in \delta(q_j, a)\}$.

3. $Q_m = Q_{m+1} = \ldots$.

Справді, очевидно, що кількість кроків скінчена, тому що послідовність Q_i монотонна ($Q_0 \subseteq Q_1 \subseteq Q_2 \subseteq \ldots$) та обмежена зверху: $Q_m \subseteq Q$.

Тоді Q_m — множина досяжних станів скінченого автомата, а $Q \setminus Q_m$ — множина недосяжних станів.

Вилучимо з діаграми переходів скінченого автомата M недосяжні вершини:



В новому автоматі функція δ визначається лише для досяжних станів. Побудований нами скінчений автомат з меншою кількістю станів буде еквівалентний початковому.

Тупикові стани

Стан q скінченого автомата M називається *тупиковим*, якщо на діаграмі переходів скінченого автомата не існує шляху з q в F.

Алгоритм [пошуку тупикових станів]. Спочатку спробуємо знайти нетупикові стани. Якщо S_m — множина нетупикових станів, то $Q \setminus S_m$ — множина тупикових станів. Побудуємо послідовність множин $S_0, S_1, S_2, \ldots$ таким чином, що:

1. $S_0 = F$.

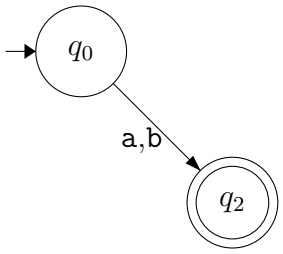
2. $S_i = S_{i-1} \cup \{q \mid \exists a \in \Sigma : \delta(q, a) \cap S_{i-1} \neq \varnothing\}$.

3. $S_m = S_{m+1} = \ldots$.

Очевидно, що кількість кроків скінчена, тому що послідовність S_i монотонна ($S_0 \subseteq S_1 \subseteq S_2 \subseteq \ldots$) та обмежена зверху — $S_m \subseteq Q$.

Тоді S_m — множина нетупикових станів скінченого автомата, а $Q \setminus S_m$ — множина тупикових станів.

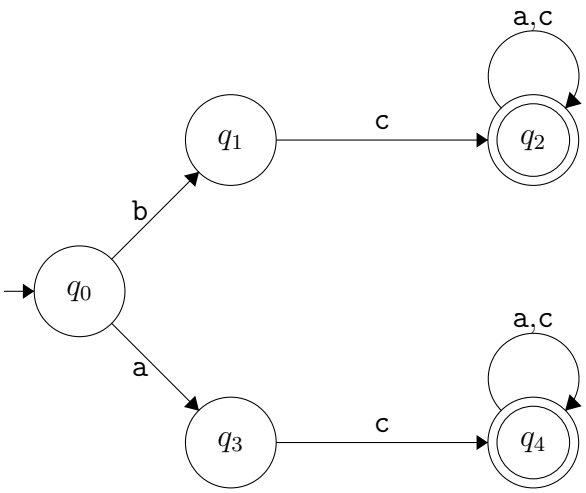
Вилучимо з діаграми переходів скінченого автомата M тупикові вершини:



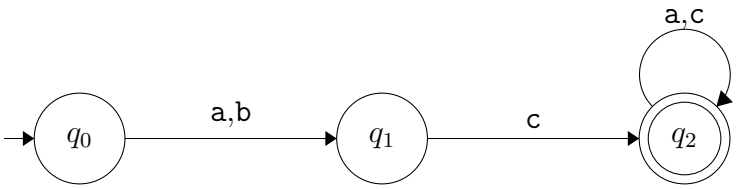
В новому автоматі функція δ визначається лише для нетупикових станів.

Еквівалентні стани

Автомат, у котрого відсутні недосяжні та тупикові стани, піддається подальшій мінімізації шляхом "склеювання" еквівалентних станів. Продемонструємо це на конкретному прикладі:



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



Ми досягли бажаного нам результату шляхом "склеювання" двох станів $q_1 \equiv q_3$ та $q_2 \equiv q_4$.

Два стани q_1 та q_2 скінченого автомата M називаються *еквівалентними* (позначається $q_1 \equiv q_2$), якщо множини слів, які розпізнає автомат, починаючи з q_1 та q_2, співпадають.

Нехай q_1 та q_2 — два різні стани скінченого автомата M, а $x \in \Sigma^\star$. Будемо говорити, що ланцюжок x *розрізняє* стани q_1 та q_2, якщо $(q_1, x) \models^\star (q_3, \varepsilon)$ та $(q_2, x) \models^\star (q_4, \varepsilon)$, причому рівно один зі станів q_3 і q_4 (не) належить множині заключних станів.

Стани q_1 та q_2 називаються *k-нерозрізнені*, якщо не існує ланцюжка x ($|x| \leq k$), що розрізняє стани q_1 та q_2.

Два стани q_1 та q_2 *нерозрізнені*, якщо вони k-нерозрізнені для довільного k.

Теорема. *Два стани q_1 та q_2 довільного скінченого автомата M з n станами нерозрізнені, якщо вони $(n-2)$-нерозрізнені.*

Доведення: На першому кроці розіб'ємо множину станів скінченого автомата на дві підмножини: F та $Q \setminus F$. На цій основі побудуємо відношення $\equiv^0$: $q_1 \equiv^0 q_2$, якщо обидва стани одночасно належать F або $Q \setminus F$.

Побудуємо відношення $\equiv^k$: $q_1 \equiv^k q_2$, якщо $q_1 \equiv^{k-1} q_2$ та $\delta(q_1, a) \equiv^{k-1} \delta(q_2, a)$ для всіх $a \in \Sigma$.

Очевидно, кожна побудована множина містить не більше $(n-1)$ елементи.

Таким чином, можна отримати не більше $(n-2)$ уточнення відношення $\equiv^0$.

Відношення $\equiv^{n-2}$ визначає класи еквівалентних станів автомата M.

Алгоритм

Алгоритм [побудови мінімального скінченого автомата].

1. Побудувати скінчений автомат без тупикових станів.

2. Побудувати скінчений автомат без недосяжних станів.

3. Знайти множини еквівалентних станів та побудувати найменший (мінімальний) автомат.

1.3.2 Контрольні запитання

1. Які автомати називаються еквівалентними?

2. Який стан автомату називається недосяжним?

3. Опишіть алгоритм пошуку недосяжних станів і доведіть його збіжність. Бонус: оцініть складність цього алгоритму за часом і пам'яттю.

4. Який стан автомату називається тупиковим?

5. Опишіть алгоритм пошуку тупикових станів і доведіть його збіжність. Бонус: оцініть складність цього алгоритму за часом і пам'яттю.

6. Які стани називаються еквівалентними?

7. Опишіть алгоритм пошуку еквівалентних станів і доведіть його збіжність. Бонус: оцініть складність цього алгоритму за часом і пам'яттю.

8. Опишіть алгоритм мінімізації детермінованого скінченного автомату. Бонус: виведіть з попередніх оцінок складність цього алгоритму за часом і пам'яттю.

1.4 Скінченно-автоматні мови і праволінійні граматики

1.4.1 Скінченно-автоматні мови

Ознайомившись з деякими результатами теорії скінчених автоматів, спробуємо з'ясувати, які мови (множини слів) є скінчено-автоматними.

Базові мови

Твердження: Скінчено автоматними є наступні множини:

1. порожня словарна множина — $\varnothing$;
2. словарна множина, що складається з одного ε-слова — $\{\varepsilon\}$;
3. множина $\{a\}$, $a \in \Sigma$.

Доведення: в кожному випадку нам доведеться конструктивно побудувати відповідний скінчений автомат:

1. Довільний скінчений автомат з пустою множиною заключних станів (а мінімальний — з пустою множиною станів) допускає $\varnothing$;
2. Розглянемо автомат $M = \langle\{q_0\}, \Sigma, q_0, \delta, \{q_0\}\rangle$, у якому δ не визначено ні для яких $a \in \Sigma$. Тоді $L(M) = \{\varepsilon\}$.
3. Розглянемо автомат $M = \langle\{q_0, q_1\}, \Sigma, q_0, \delta, \{q_1\}\rangle$, у якому функція δ визначена лише для пари (q_0, a), а саме: $\delta(q_0, a) = \{q_1\}$. Тоді $L(M) = \{a\}$.

Операції над мовами

Твердження: Якщо $M_1 = \langle Q_1, \Sigma, q_0^1, \delta_1, F_1\rangle$ та $M_2 = \langle Q_2, \Sigma, q_0^2, \delta_2, F_2\rangle$, що визначають відповідно мови $L(M_1)$ та $L(M_2)$, то скінченно-автоматними мовами будуть:

1. $L(M_1) \cup L(M_2) = \{w \mid q \in L(M_1) \text{ or } q \in L(M_2)\}$;
2. $L(M_1) \cdot L(M_2) = \{w = xy \mid x \in L(M_1), y \in L(M_2)\}$;
3. $L(M_1)^\star = \{\varepsilon\} \cup L(M_1) \cup L(M_1)^2 \cup L(M_1)^3 \cup \ldots$.

Доведення: в кожному випадку нам доведеться конструктивно побудувати відповідний скінчений автомат:

1. Побудуємо автомат $M = \langle Q, \Sigma, q_0, \delta, F\rangle$ такий, що $L(M) = L(M_1) \cup L(M_2)$:

   - $Q = Q_1 \cup Q_2 \cup \{q_0\}$, де q_0 — новий стан ($q_0 \notin Q_1 \cup Q_2$);
   - Функцію δ визначимо таким чином:

$$\delta(q,a) = \begin{cases} \delta_1(q,a), & q \in Q_1, \\ \delta_2(q,a), & q \in Q_2, \\ \delta_1(q_0^1,a) \cup \delta_2(q_0^2,a), & q = q_0. \end{cases}$$

   - Множина заключних станів:

$$F = \begin{cases} F_1 \cup F_2, & \text{if } \varepsilon \notin L_1 \cup L_2, \\ F_1 \cup F_2 \cup \{q_0\}, & \text{otherwise.} \end{cases}$$

   Побудований таким чином автомат взагалі кажучи недетермінований.

   Індукцією по i показуємо, що $(q_0, w) \models^i (q, \varepsilon)$ тоді і тільки тоді, коли $(q_0^1, w) \models^i (q, \varepsilon), q \in F_1$ або $(q_0^2, w) \models^i (q, \varepsilon), q \in F_2$.

2. Побудуємо автомат $M = \langle Q, \Sigma, q_0, \delta, F\rangle$ такий, що $L(M) = L(M_1) \cdot L(M_2)$:

   - $Q = Q_1 \cup Q_2$;
   - $q_0 = q_0^1$;
   - Функцію δ визначимо таким чином:

$$\delta(q,a) = \begin{cases} \delta_1(q,a), & q \in Q_1 \setminus F_1, \\ \delta_2(q,a), & q \in Q_2, \\ \delta_1(q,a) \cup \delta_2(q_0^2,a), & q \in F_1. \end{cases}$$

   - Множина заключних станів:

$$F = \begin{cases} F_2, & \text{if } \varepsilon \notin L_2, \\ F_1 \cup F_2, & \text{otherwise.} \end{cases}$$

3. Побудуємо автомат $M = \langle Q, \Sigma, q_0, \delta, F\rangle$ такий, що $L(M) = L(M_1)^\star$:

   - $Q = Q_1 \cup \{q_0\}$, де q_0 — новий стан ($q_0 \notin Q_1$);

- Функцію δ визначимо таким чином:

$$\delta(q,a) = \begin{cases} \delta_1(q,a), & q \in Q_1 \setminus F_1, \\ \delta_1(q_0^1,a), & q = q_0, \\ \delta_1(q,a) \cup \delta_1(q_0^1,a), & q \in F_1. \end{cases}$$

- Множина заключних станів $F = F_1 \cup \{q_0\}$.

1.4.2 Скінченні автомати та праволінійні граматики

Породжуюча граматика G — це четвірка

$$G = \langle N, \Sigma, P, S \rangle ,$$

де:

- N — скінченна множина — допоміжний алфавіт (нетермінали);
- Σ — скінченна множина — основний алфавіт (термінали);
- P — скінченна множина правил вигляду

$$\alpha \mapsto \beta, \quad \alpha \in (N \cup \Sigma)^\star \times N \times (N \cup \Sigma)^\star , \quad \beta \in (N \cup \Sigma) .$$

- S — виділений нетермінал (аксіома).

Класифікація граматик Хомського

В залежності від структури правил граматики діляться на чотири типи:

- Тип 0: граматики загального вигляду, коли правила не мають обмежень, тобто

$$\alpha \mapsto \beta, \quad \alpha \in (N \cup \Sigma)^\star \times N \times (N \cup \Sigma)^\star , \quad \beta \in (N \cup \Sigma) .$$

- Тип 1: граматики, що не укорочуються, коли обмеження на правила мінімальні, а саме:

$$\alpha \mapsto \beta, \quad \alpha \in (N \cup \Sigma)^\star \times N \times (N \cup \Sigma)^\star , \quad \beta \in (N \cup \Sigma) , \quad |\alpha| \leq |\beta|.$$

- Тип 2: контекстно-вільні граматики, коли правила в схемі P мають вигляд:

$$A_i \mapsto \beta, \quad A_i \in N, \quad \beta \in (N \cup \Sigma)^\star .$$

- Тип 3: скінченно-автоматні граматики, коли правила в схемі P мають вигляд:
$$A_i \mapsto wA_j, \quad A_i \mapsto w, \quad Ai \mapsto A_j w,$$
де $A_i, A_j \in N$, $w \in \Sigma^\star$.

В класі скінченно-автоматних граматик виділимо так звані *праволінійні граматики* — це граматики, які в схемі P мають правила вигляду:

$$A_i \mapsto wA_j, \quad A_i \mapsto w,$$

де $A_i, A_j \in N$, $w \in \Sigma^\star$.

Нескладно довести, що клас праволінійних граматик співпадає з класом граматик типу 3.

Мова породжена граматикою

Ланцюжок w_1 *безпосередньо виводиться* з ланцюжка w (позначається $w \Rightarrow w_1$), якщо $w = x\alpha y$, $w_1 = x\beta y$ та в схемі P граматики G є правило виду $\alpha \mapsto \beta$. Оскільки поняття "безпосередньо виводиться" розглядається на парах ланцюжків, то в подальшому символ $\Rightarrow$ буде трактуватися як бінарне вдіношення.

Ланцюжок w_1 *виводиться* з ланцюжка w (позначається $w \Rightarrow^\star w_1$), якщо існує скінчена послідовність виду $w \Rightarrow w'_1 \Rightarrow w'_2 \Rightarrow \ldots \Rightarrow w'_n \Rightarrow w_1$. Або кажуть, що бінарне відношення $\Rightarrow^\star$ — це рефлексивно-транзитивне замикання бінарного відношення $\Rightarrow$.

Мова, яку породжує граматика G (позначається $L(G)$) — це множина термінальних ланцюжків:

$$L(G) = \{w \mid S \Rightarrow^\star w, w \in \Sigma^\star\}.$$

Праволінійна граматика $\sim$ скінченний автомат

Теорема. *Клас мов, що породжуються праволінійними граматиками, співпадає з класом мов, які розпізнаються скінченими автоматами.*

Доведення. Спочатку покажемо, що для довільної праволінійної граматики G можна побудувати скінчений автомат M, такий що $L(M) = L(G)$.

Розглянемо правила праволінійної граматики. Вони бувають двох типів: $A_i \mapsto wA_j$, і $A_i \mapsto w$.

На основі правил граматики G побудуємо схему P_1 нової граматики, яка буде еквівалентною початковій, а саме:

- правила виду $A_i \mapsto a_1 a_2 \dots a_p A_j$ замінимо послідовністю правил

$$\begin{aligned} A_i &\mapsto a_1 B_1, \\ B_1 &\mapsto a_2 B_2, \\ &\dots \\ B_{p-1} &\mapsto a_p A_j. \end{aligned}$$

- правила виду $A_i \mapsto a_1 a_2 \dots a_p$ замінимо послідовністю правил

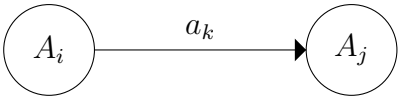
$$\begin{aligned} A_i &\mapsto a_1 B_1, \\ B_1 &\mapsto a_2 B_2, \\ &\dots \\ B_{p-1} &\mapsto a_p B_p, \\ B_p &\mapsto \varepsilon. \end{aligned}$$

де $B_1, B_2, \dots$ — це нові нетермінали граматики G_1.

Очевидно, що граматика G_1 ьуде еквівалентна граматиці G.

Далі, на основі граматики G_1 побудуємо скінчений автомат M, таким чином:

- як імена станів автомата візьмемо нетермінали граматики G_1;
- початковий стан автомата позначається аксіомою S;
- функція δ визначається діаграмою переходів, яка будується на основі правил вигляду $A_i \mapsto a_k A_j$:



- множина F заключних станів скінченого автомата визначається так: $F = \{A_i \mid A_i \mapsto \varepsilon\}$.

Індукцією по довжині вхідного слова покажемо, що якщо $S \Rightarrow^{n+1} w$, то $(q_0, w) \models^n (q, \varepsilon)$:

- База: $i = 0$: $S \Rightarrow \varepsilon$, тоді $(q_0, \varepsilon) \models^0 (q_0, \varepsilon)$.
- Перехід: нехай $|w| = i + 1$, тобто $w = aw_1$. Тоді $S \Rightarrow aA_p \Rightarrow^i aw_1$ та $(q_0, aw_1) \models (q_i, w_1) \models^{i-1} (q, \varepsilon)$, де $q \in F$.

Доведення навпаки є очевидним.

1.4.3 Контрольні запитання

1. Які три базові скінченно-автоматні мови ви знаєте?
2. Доведіть, що мови з попереднього питання справді є скінченно-автоматними.
3. Які три операції над скінченно-автоматними мовами ви знаєте?
4. Доведіть, що результати операцій з попереднього питання справді є скінченно-автоматними.
5. Що таке породжуюча граматика?
6. Які чотири типи граматик за Хомським ви знаєте?
7. Що таке праволінійна граматика?
8. Дайте визначення безпосереднього виведення, виведення, породженої граматикою мови.
9. Доведіть, що скінченний автомат це майже праволінійна граматика.

1.5 Регулярні множини і регулярні вирази

1.5.1 Регулярні множини

Нехай Σ — скінчений алфавіт. *Регулярна множина* в алфавіті Σ визначається рекурсивно:

1. $\varnothing$ — пуста множина — це регулярна множина в алфавіті Σ;
2. $\{\varepsilon\}$ — пусте слово — регулярна множина в алфавіті Σ;
3. $\{a\}$ — однолітерна множина — регулярна множина в алфавіті Σ;
4. Якщо P та Q — регулярні множини, то такими є наступні множини:
   - $P \cup Q$ (операція об'єднання);
   - $P \times Q$ (операція конкатенації);
   - $P^\star = \{\varepsilon\} \cup P \cup P^2 \cup \ldots$ (операція ітерації).
5. Ніякі інші множини, окрім побудованих на основі 1–4 не є регулярними множинами.

Таким чином, регулярні множини можна побудувати з базових елементів 1–3 шляхом скінченого застосування операцій об'єднання, конкатенації та ітерації.

1.5.2 Регулярні вирази

Регулярні вирази позначають регулярні множини таким чином, що:

1. 0 позначає регулярну множину $\varnothing$;
2. ε позначає регулярну множину $\{\varepsilon\}$;
3. a позначає регулярну множину $\{a\}$;
4. Якщо p та q позначають відповідно регулярні множини P та Q, то
   - $p + q$ позначає регулярну множину $P \cup Q$;
   - $p \cdot q$ позначає регулярну множину $P \times Q$;
   - $p^\star$ позначає регулярну множину $P^\star$.
5. Ніякі інші вирази, окрім побудованих на основі 1–4 не є регулярними виразами.

Алгебра регулярних виразів

Оскільки ми почали вести мову про вирази, нам зручніше перейти до поняття алгебри регулярних виразів. Для кожної алгебри одним з важливих питань є питання еквівалентних перетворень, які виконуються на основі тотожностей у цій алгебрі. Сформулюємо основні тотожності алгебри регулярних виразів:

1. $a + b + c = a + (b + c)$ (ліва асоціативність додавання);
2. $a + b + c = (a + b) + c$ (права асоціативність додавання);
3. $a + 0 = 0 + a = a$ (0 — нейтральний елемент за додаванням);
4. $a \cdot b \cdot c = a \cdot (b \cdot c)$ (ліва асоціативність множення);
5. $a \cdot b \cdot c = (a \cdot b) \cdot c$ (права асоціативність множення);
6. $a + b = b + a$ (комутативність додавання);
7. $a \cdot \varepsilon = \varepsilon \cdot a = a$ (ε — нейтральний елемент за множенням);
8. $a \cdot 0 = 0 \cdot a = 0$ (0 — нульовий елемент за множенням);
9. $a \cdot (b + c) = a \cdot b + a \cdot c$ (ліва дистрибутивність множення відносно додавання);
10. $(a + b) \cdot c = a \cdot c + b \cdot c$ (права дистрибутивність множення відносно додавання).

У алгебри регулярних виразів є і некласичні властивості:

1. $a + a = a$;
2. $p + p^\star = p^\star$;
3. $0^\star = \varepsilon$;
4. $\varepsilon^\star = \varepsilon$.

Лінійні рівняння

За аналогією з класичними алгебрами розглянемо лінійне рівняння в алгебрі регулярних виразів: $X = a \cdot X + b$, де a, b — регулярні вирази.

Взагалі кажучи таке рівняння (в залежності від a та b) може мати безліч розв'язків.

Серед всіх розв'язків рівняння з регулярними коефіцієнтами виберемо найменший розв'язок $X = a^\star \cdot b$, який назвемо *найменша нерухома точка.*

Щоб перевірити, що $a^\star \cdot b$ справді розв'язок рівняння в алгебрі регулярних виразів, підставимо його в початкове рівняння та перевіримо тотожність виразів на основі системи тотожних перетворень:

$$a^\star b = aa^\star b = (aa^\star + \varepsilon)b = (a(\varepsilon + a + a^2 + \ldots) + \varepsilon)b = (\varepsilon a + a^2 + \ldots)b = a^\star b.$$

Системи рівнянь

В алгебрі регулярних виразів також розглядають і системи лінійних рівнянь з регулярними коефіцієнтами:

$$\begin{cases} X_1 = a_{11}X_1 + a_{12}X_2 + \ldots + a_{1n}X_n + b_1, \\ X_2 = a_{21}X_1 + a_{22}X_2 + \ldots + a_{2n}X_n + b_2, \\ \quad \ldots \\ X_n = a_{n1}X_1 + a_{n2}X_2 + \ldots + a_{nn}X_n + b_n. \end{cases}$$

Метод розв'язування системи лінійних рівнянь з регулярними коефіцієнтами нагадує метод виключення Гауса.

1. Для $i = \overline{1..n}$, використавши систему тотожних перетворень, записати i-е рівняння у вигляді: $X_i = aX_i + b$, де a — регулярний вираз в алфавіті Σ, а b — регулярний вираз виду

   $$\beta_0 + \beta_{i+1}X_{i+1} + \beta_{i+2}X_{i+2} + \ldots + \beta_n X_n,$$

   де β_k $(k = 0, \overline{i+1..n})$ — регулярні коефіцієнти. Далі, в правих частинах рівнянь зі змінними $X_{i+1}, X_{i+2}, \ldots, X_n$ в лівій частині рівняння підставити замість X_i значення $a^\star b$.

2. Для $i = \overline{n..1}$ розв'язати i-е рівняння яке зараз має вигляд $X_i = aX_i + b$, де a, b — регулярні вирази в алфавіті Σ, і підставити його розв'язок $a^\star b$ у коефіцієнти рівнянь зі змінними $X_{i-1}, X_{i-2}, \ldots, X_1$.

Приклад. Розв'язати систему 2×2:

$$\begin{cases} X_1 = a_{11}X_1 + a_{12}X_2 + b_1, \\ X_2 = a_{21}X_1 + a_{22}X_2 + b_2. \end{cases}$$

Розв'язок:

1. З першого рівняння

$$X_1 = a_{11}^{\star}(a_{12}X_2 + b_1).$$

2. Підставляємо у друге рівняння, воно набуває вигляду

$$X_2 = a_{21}(a_{11}^{\star}(a_{12}X_2 + b_1)) + a_{22}X_2 + b_2.$$

Або, після спрощень:

$$X_2 = (a_{21}a_{11}^{\star}a_{12} + a_{22})X_2 + (a_{21}a_{11}^{\star}b_1 + b_2).$$

Звідки знаходимо:

$$X_2 = (a_{21}a_{11}^{\star}a_{12} + a_{22})^{\star}(a_{21}a_{11}^{\star}b_1 + b_2).$$

3. Підставляємо у вираз для X_1:

$$X_1 = a_{11}^{\star}(a_{12}(a_{21}a_{11}^{\star}a_{12} + a_{22})^{\star}(a_{21}a_{11}^{\star}b_1 + b_2) + b_1).$$

Тут можна розкрити дужки, але зараз у цьому немає потреби.

1.5.3 Контрольні запитання

1. Яка множина називається регулярною (в алфавіті Σ)?
2. Як регулярні вирази позначаються регулярні множини?
3. Які класичні тотожності алгебри регулярних виразів ви знаєте?
4. Які не класичні тотожності алгебри регулярних виразів ви знаєте?
5. Який розв'язок лінійного рівняння $X = a \cdot X + b$ називається найменшою нерухомою точкою?
6. Доведіть, що згаданий у попередньому рівняння вираз справді є розв'язком.

7. Сформулюйте алгоритм Гауса розв'язування систем лінійних регулярних рівнянь.

8. Розв'яжіть систему 3×3:

$$\begin{cases} X_1 = aX_1 + bX_2 + c, \\ X_2 = cX_1 + aX_2 + bX_3, \\ X_3 = b + cX_2 + aX_3. \end{cases}$$

1.6 ПОЛІЗ, регулярні вирази, і автомати

1.6.1 Польський інверсний запис для регулярних виразів

Польський інверсний запис (ПОЛІЗ) для регулярних виразів будується на основі початкового регулярного виразу на основі наступних правил:

1. Порядок операндів в початковому виразі і в перетвореному виразі співпадають.

2. Операції в перетвореному виразу йдуть з урахуванням пріоритету безпосередньо за операндами.

Наприклад, ПОЛІЗ для виразу $(a^\star+b)^\star c$ має такий вигляд: $a, \star, b, +, \star, c, \cdot$.

В цьому прикладі в стандартному записі регулярного виразу бінарна операція конкатенація $\cdot$ природньо опущена, але в ПОЛІЗ потрібно завжди цю операцію явно вказувати.

Важливою характеристикою ПОЛІЗ є відсутність дужок в запису виразу, тобто його можна опрацьовувати лінійно.

Алгоритм

Для перетворення виразу в ПОЛІЗ необхідно з кожною операцією зв'язати деяке число, яке будемо називати "пріоритет" (0 — найвищий пріоритет присвоїмо дужці `'('`). Наведемо псевдокод алгоритму:

```
while lexem <- прочитати поточну лексему:
    if lexem is операнд:
        занести її в поле результату

    if lexem = '(':
        занести її в стек

    if lexem is код операції:
        while (пріоритет операції на вершині стека >= \
         пріоритет поточної операції):
            елемент з вершини стека перенести в поле результату
        поточну лексему занести в стек

    if lexem = ')':
```

```
        while код операції на вершині стеку != '(':
            елемент з вершини стека перенести в поле результату
        дужку '(' зняти з вершини стека

всі елементи із стека перенести в поле результату
```

1.6.2 Інтерпретація ПОЛІЗ регулярного виразу

Результат інтерпретації ПОЗІЗ — це скінченний автомат M, який розпізнає (сприймає) множину ланцюжків, котрі позначає регулярний вираз.

Алгоритм

Наведемо псевдокод алгоритму:

```
while lexem <- прочитати поточну лексему:
    if lexem is операнд ai:
        M: L(M) = {ai\}

    if lexem = '+':
        M1, M2 <- автомати з вершини стеку
        M: L(M) = L(M1) \cup L(M2)

    if lexem = '\times':
        M1, M2 <- автомати з вершини стеку
        M: L(M) = L(M2) \times L(M1)

    if lexem = '\star':
        M1 <- автомат з вершини стеку
        M: L(M)= L(M1)^\star

    M занести в стек

вершину стека перенести в поле результату
```

Якщо досягли кінця регулярного виразу, то на вершині стека знаходиться автомат M, який розпізнає множину слів (ланцюжків), які позначає регулярний вираз.

1.6.3 Контрольні запитання

1. Що таке ПОЛІЗ?

2. Чи можна в ПОЛІЗ опускати операції які природнім чином опускаються у класичному записі?

3. Яка основна характеристика ПОЛІЗ і яку обчислювальну перевагу вона пропонує?

4. Сформулюйте алгоритм перетворення регулярного виразу у ПОЛІЗ та оцініть його складність.

5. Що є результатом інтерпретації ПОЛІЗ регулярного виразу?

6. Сформулюйте алгоритм інтерпретації ПОЛІЗ регулярного виразу.

7. Оцініть складність попереднього алгоритму через складності операцій побудови автоматів.

8. Для регулярного виразу $(a^\star + b)^\star \cdot c$ побудуйте скінчений автомат, який розпізнає множину ланцюжків, що позначаються цим виразом.

Розділ 2

Синтаксичний аналіз

2.7 Синтаксичний аналіз в мовних процесорах

2.7.1 Синтаксичний аналіз

Для визначення синтаксичної компоненти мови програмування використовують контекстно-вільні граматики (КС-граматики). На відміну від скінченно-автоматних граматик потужність класу КС-граматик достатня, щоб визначити майже всі так звані синтаксичні властивості мов програмування. Якщо цього недостатньо, то розглядають деякі спрощення у граматиках типу 2 або параметричні КС-граматики.

Звичайно, із синтаксичною компонентою мови програмування пов'язана семантична компонента. Тоді, якщо ми говоримо про семантику мови програмування, ми вимагаємо семантичної однозначності для кожної вірно написаної програми. За аналогією з семантикою, при описі синтаксичної компоненти мови програмування необхідно користуватися однозначними граматиками.

Граматика G називається *неоднозначною*, якщо існує декілька варіантів виводу ω в G ($\omega \in L(G)$).

Приклад. Розглянемо таку граматику $G = \langle N, \Sigma, P, S\rangle$ з двома правилами у схемі P: $S \Rightarrow S + S$, і $S \Rightarrow a$. Покажемо, що для ланцюжка $\omega = a + a + a$ існує щонайменше два варіанти виводу:

1. $S \Rightarrow S + S \Rightarrow S + S + S \Rightarrow a + S + S \Rightarrow a + a + S \Rightarrow a + a + a$.

2. $S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S + S \Rightarrow a + a + S \Rightarrow a + a + a$.

Стратегії виведення

В теорії граматик розглядається декілька стратегій виведення ланцюжка ω в G. Визначимо дві стратегії які будуть використані в подальшому.

Лівостороння стретегія виводу ланцюжка ω в G — це послідовність кроків безпосереднього виводу, при якій на кожному кроці до увагі береться перший зліва направо нетермінал.

Правостороння стратегія виводу ω в G протилежна лівосторонній стратегії.

З виводом ω в G пов'язане синтаксичне дерево, яке визначає синтаксичну структуру програми.

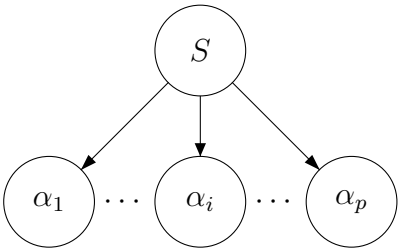
Синтаксичні дерева

Синтаксичне дерево виведення ω в G — це впорядковане дерево, корінь котрого позначено аксіомою, в проміжних вершинах знаходяться нетермінали, а на кроні — елементи з $\Sigma \cup \{\varepsilon\}$. Побудова синтаксичного дерева виведення ω в G виконується покроково з урахуванням стратегії виводу ω в G.

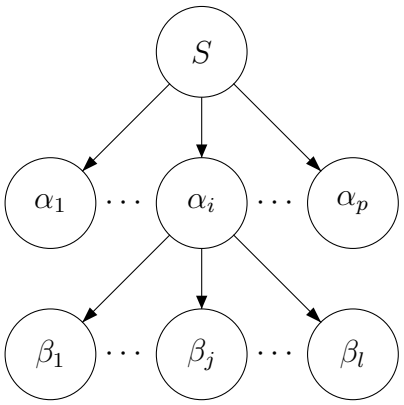
Алгоритм [побудови синтаксичного дерева ланцюжка ω в граматиці G урахуванням лівосторонньої стратегії виводу].

1. Будуємо корінь дерева та позначимо його аксіомою S.

2. В схемі P граматики G візьмемо правило виду $S \Rightarrow \alpha_1\alpha_2 \dots \alpha_p$, де $\alpha_i \in N \cup \Sigma \cup \{\varepsilon\}$ і побудуємо дерево висоти 1:



3. На кроні дерева, побудованого на попередньому кроці, візьмемо перший зліва направо нетермінал. Нехай це буде α_i. Тоді в схемі P виберемо правило виду $\alpha_i \Rightarrow \beta_1\beta_2 \dots \beta_l$, де $\beta_i \in N \cup \Sigma \cup \{\varepsilon\}$ і побудуємо наступне дерево:



Цей крок виконується доки на кроні дерева є елементи з N.

Зауважимо очевидні факти, що випливають з побудови синтаксичного дерева:

- крона дерева, зображеного на попередньому малюнку наступна:

$$\alpha_1\alpha_2\ldots\alpha_{i-1}\beta_1\beta_2\ldots\beta_l\alpha_{i+1}\ldots\alpha_p;$$

- ланцюжок $\alpha_1\alpha_2\ldots\alpha_{i-1} \in \Sigma^\star$ з крони — термінальний ланцюжок;

- для однозначної граматики G існує лише одне синтаксичне дерево виводу ω в G.

Власне аналіз

Будемо говорити, що ланцюжок $\omega \in \Sigma^\star$, побудований на основі граматики G ($\omega \in L(G)$) *проаналізований*, якщо відоме одне з його дерев виводу.

Зафіксуємо послідовність номерів правил, які були використані під час побудови синтаксичного дерева виводу ω в G з урахуванням стратегії виводу.

Лівостороннім аналізом π ланцюжка $\omega \in L(G)$ будемо називати послідовність номерів правил, які були використані при лівосторонньому виводі ω в G.

Приклад: Для граматики $G = \langle N, \Sigma, P, S\rangle$ зі схемою P:

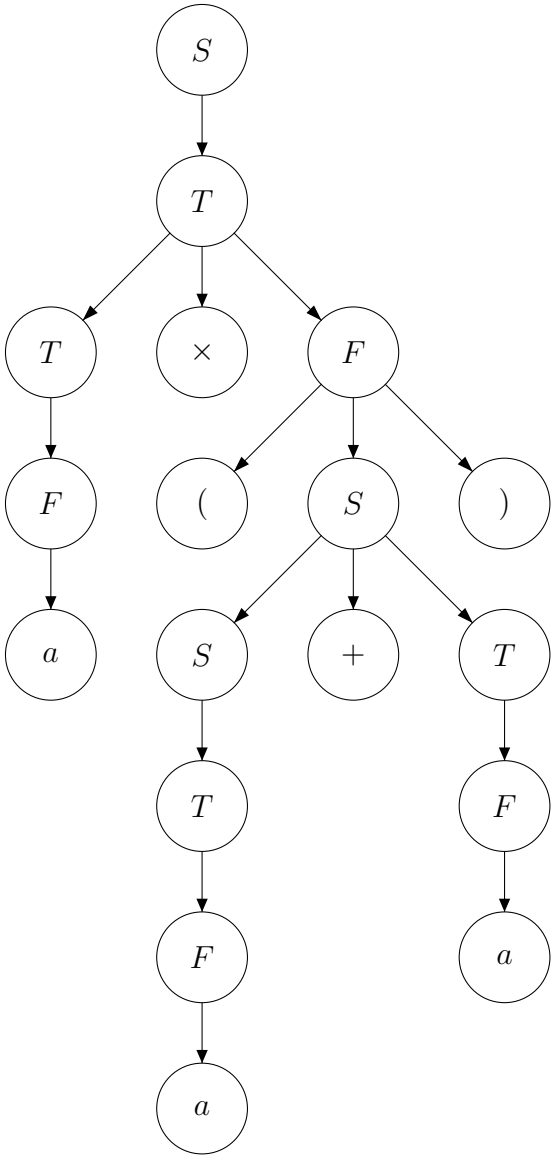
$$S \Rightarrow S + T \tag{2.1}$$
$$S \Rightarrow T \tag{2.2}$$
$$T \Rightarrow T \times F \tag{2.3}$$
$$T \Rightarrow F \tag{2.4}$$
$$F \Rightarrow (S) \tag{2.5}$$
$$F \Rightarrow a \tag{2.6}$$

і для ланцюжка $\omega = a \times (a + a)$ побудуємо лівосторонній аналіз π:

Виведення має вигляд:

$$S \Rightarrow T \Rightarrow T \times F \Rightarrow F \times F \Rightarrow a \times F \Rightarrow a \times (S) \Rightarrow a \times (S + T) \Rightarrow$$
$$\Rightarrow a \times (T + T) \Rightarrow a \times (F + T) \Rightarrow a \times (a + T) \Rightarrow a \times (a + F) \Rightarrow a \times (a + a).$$

З наведеного вище виводу ланцюжка $\omega \in L(G)$ лівосторонній аналіз π буде: $\pi = (2, 3, 4, 6, 5, 1, 2, 4, 6, 4, 6)$, а синтаксичне дерево виводу $\omega = a \times (a + a)$ наступне:



Синтез дерева за аналізом

Нехай π — лівосторонній аналіз ланцюжка $\omega \in L(G)$. Знаючи π досить легко побудувати (відтворити) синтаксичне дерево. Відтворення (синтез) синтаксичного дерева можна виконати, скориставшись однією з стратегій синтаксичного аналізу:

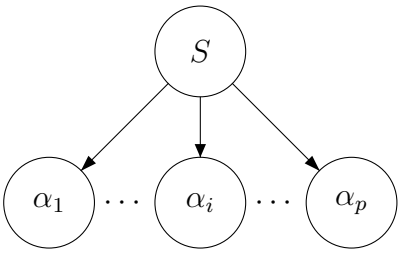
- стратегія “зверху донизу”;
- стратегія “знизу догори”.

Стратегія синтаксичного аналізу *"зверху донизу"* — це побудова синтаксичного дерева крок за кроком починаючи від кореня до крони.

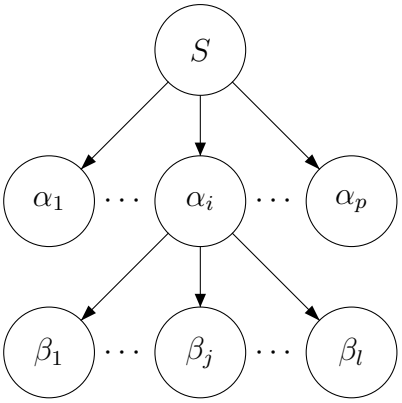
Алгоритм [синтезу синтаксичного дерева на основі лівостороннього аналізу π ланцюжка $\omega \in L(G)$].

1. Побудуємо корінь дерева та позначимо його аксіомою S. Тоді, якщо $\pi = (p_1, p_2, \ldots, p_m)$, то

2. Побудуємо дерево висоти один, взявши зі схеми P правило з номером p_1 виду $S \Rightarrow \alpha_1\alpha_2 \ldots \alpha_p$:



3. На кроні дерева, отриманого на попередньому кроку, візьмемо перший зліва направо нетермінал (нехай це буде нетермінал α_i) та правило з номером p_j вигляду: $\alpha_i \Rightarrow \beta_1\beta_2 \ldots \beta_l$ та побудуємо нове дерево:



Даний пункт виконувати доти, доки не переглянемо всі елементи з π.

Проблеми стратегії "зверху донизу"

Сформулюємо декілька проблему для стратегії аналізу "зверху донизу":

У загальному випадку у класі КС-граматик існує проблема неоднозначності (недетермінізму) виводу $\omega \in L(G)$. Як приклад можемо розглянути граматику з "циклами". Це така граматика, у якої в схемі P існує така послідовність правил за участю нетермінала A_i, що: $A_i \Rightarrow A_j$ і $A_j \Rightarrow A_i$, де A_j — будь-який нетермінал граматики G.

Як наслідок, граматики з ліворекурсивним нетерміналом для стратегії аналізу "зверху донизу" недопустимі.

Зауважимо, що існують підкласи класу КС-граматик, які природно забезпечують стратегію аналізу "зверху донизу". Один з таких підкласів — це $LL(k)$-граматики, які забезпечують синтаксичний аналіз ланцюжка $\omega \in L(G)$ за час $O(n)$, де $n = |\omega|$, та при цьому аналіз є однозначним.

2.7.2 Контрольні запитання

1. Які граматики називаються однозначними?
2. Які дві стратегії виведення ви знаєте?
3. Що таке синтаксичне дерево виведення?
4. Що таке лівосторонній аналіз ланцюжка?
5. Що таке синтез дерева за аналізом?
6. Які дві стратегії синтезу дерева за аналізом ви знаєте?
7. Що таке граматика з циклами і які проблеми вона створює для стратегії "згори донизу"?
8. Який підклас КС-граматик забезпечує стратегію аналізу "зверху донизу"?

2.8 Магазинні автомати

2.8.1 Магазинні автомати

Магазинний автомат M — це сімка $M = \langle Q, \Sigma, \Gamma, q_0, j_0, \sigma, F \rangle$, де:

- $Q = \{q_0, q_1, \ldots, q_{m-1}\}$ — множина станів магазинного автомату;
- $\Sigma = \{a_1, a_2, \ldots, a_n\}$ — основний алфавіт;
- $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_k\}$ — допоміжний алфавіт (алфавіт магазина);
- $q_0 \in Q$ — початковий стан магазинного автомату;
- $j_0 \in \Gamma^\star$ — початковий вміст магазину;
- $\sigma : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^\star}$.
- $F \subseteq Q$ — множина заключних станів автомата M.

Поточний стан магазинного автомата M описується конфігурацією. *Конфігурація* магазинного автомата M — це трійка (q, ω, j), де $q \in Q$, $\omega \in \Sigma^\star$, $j \in \Gamma^\star$. Серед конфігурацій магазинного автомата M виділимо дві:

- *початкова конфігурація* (q_0, ω, j_0), де $q_0 \in Q$, ω — вхідне слово ($\omega \in \Sigma^\star$), $j_0 \in \Gamma^\star$;
- *заключна конфігурація* $(q_f, \varepsilon, \varepsilon)$, $q_f \in F$. В загальній теорії магазинних автоматів іноді як заключну конфігурацію розглядають (q_f, ε, j_f), де (q_f, j_f) — фіксована пара. Можна довести, що визначення заключної конфігурації виду $(q_f, \varepsilon, \varepsilon)$ не зменшує потужності класу магазинних автоматів.

Такт роботи (позначається $\models$) магазинного автомата M — це перехід від однієї конфігурації до іншої, а точніше:

$$(q_1, a\omega, \gamma_1 j) \models (q_2, \omega, \gamma_2 j) \quad \text{if} \quad (q_2, \gamma_2) \in \sigma(q_1, a, \gamma_1)$$

Робота магазинного автомата M (позначається $\models^\star$) — це послідовність тактів роботи, а точніше: $(q_1, \omega_1, j_1) \models^\star (q_2, \omega_2, j_2)$ тоді і тільки тоді, коли

$$(q_1, \omega_1, j_1) \models (q_1^1, \omega_1^1, j_1^1) \models (q_1^2, \omega_1^2, j_1^2) \models \ldots \models (q_1^n, \omega_1^n, j_1^n) \models (q_2, \omega_2, j_2).$$

Операції $\models$ та $\models^\star$ можна трактувати як бінарні відношення на відповідних кортежах. Тоді робота магазинного автомата M — це рефлексивно-транзитивне замикання бінарного відношення $\models$.

Мова магазинного автомату

Мова, яку розпізнає магазинний автомат M — позначається $L(M)$ — це множина слів $\omega \in \Sigma^\star$, які задовольняють умові:

$$L(M) = \{\omega | \exists q_f \in F : (q_0, \omega, j_0) \models^\star (q_f, \varepsilon, \varepsilon)\}.$$

Зафіксуємо наступні результати теорії магазинних автоматів:

1. Не існує алгоритму перетворення недетермінованого магазинного автомата у еквівалентний йому детермінований магазинний автомат.

2. Існує алгоритм, який вирішує проблему порожньої множини $L(M)$ для конкретного магазинного автомата.

3. Існує алгоритм, який за час, пропорційний $O(n^3)$ перевіряє, чи належить $\omega \in \Sigma^\star$ мові, яку розпізнає магазинний автомат M.

4. Клас мов, які розпізнаються магазинними автоматами, співпадає з класом мов, що породжуються КС-граматиками.

На основі сформульованих вище результатів для лівосторонньої стратегії виводу $\omega \in \Sigma^\star$ в G запропонуємо наступне твердження: для довільної КС-граматики G можна побудувати магазинний автомат M такий, що $L(G) = L(M)$. При цьому автомат буде моделювати лівосторонню стратегію виводу ω в G.

Магазинний автомат за КС-граматикою

Нехай $G = \langle N, \Sigma, P, S\rangle$ — КС-граматика. Побудуємо відповідний магазинний автомат $M = \langle Q, \Sigma, \Gamma, q_0, j_0, \sigma, F\rangle$:

- $Q = \{q_0\}$ — множину станів автомата складає один стан q_0;
- $\Gamma = N \cup \Sigma$ — допоміжний алфавіт;
- $j_0 = S$ — початковий вміст магазина;
- функцію σ визначимо так:
  - якщо $A \mapsto \omega_1 \mid \omega_2 \mid \ldots \mid \omega_p$ належить P, то

$$\sigma(q_0, \varepsilon, A) = \{(q_0, \omega_1), (q_0, \omega_2), \ldots, (q_0, \omega_p)\}.$$

– також поповнимо множину значень функції σ наступними значеннями:

$$\sigma(q_0, a_i, a_i) = \{(q_0, \varepsilon)\}, \quad a_i \in \Sigma.$$

Для слова $\omega \in \Sigma^\star$, $|\omega| = n$ покажемо, якщо ми за m кроків безпосереднього виводу $S \Rightarrow^m \omega$, то відповідний автомат за $(m+n)$ кроків допустить ω. Зробимо перший крок безпосереднього виведення $S \Rightarrow x_1 x_2 \dots x_k$ тоді магазинний автомат з початкової конфігурації (q_0, ω, S) перейде в наступну конфігурацію $(q_0, \omega, x_1 x_2 \dots x_k)$. Далі розглянемо наступні ситуації:

- коли x_1 — термінал a_1 (тобто $\omega = a_1\omega_1$), тоді МП-автомат виконає наступний такт: $(q_0, a_1\omega_1, a_1 x_2 \dots x_k) \models (q_0, \omega_1, x_2 \dots x_k)$;
- коли x_1 — нетермінал, тоді в схемі P граматики G виберемо правило виду $x_1 \mapsto y_1 y_2 \dots y_l$, зробимо наступний крок безпосереднього виведення: $S \Rightarrow y_1 \dots y_l x_2 \dots x_k$. При таких умовах автомат перейде в наступну конфігурацію:

$$(q_0, \omega, x_1 x_2 \dots x_k) \models (q_0, \omega, y_1 y_2 \dots y_l x_2 \dots x_k).$$

Очевидно, якщо слово ω виводиться за m кроків, то МП-автомат зробить $m + |\omega|$ кроків та розпізнає ω. Таким чином, $L(G) = L(M)$.

2.8.2 Контрольні запитання

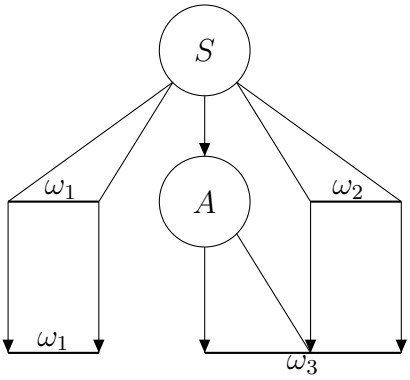
1. Що таке магазинний автомат?
2. Що таке конфігурація магазинного автомату?
3. Які конфігурації магазинного автомату називаються початковою і заключною?
4. Що таке такт роботи і робота магазинного автомату?
5. Яку мову розпізнає магазинний автомат?
6. Що таке проблема порожньої множини $L(M)$?
7. Як за КС-граматикою G побудувати магазинний автомат M такий, що $L(G) = L(M)$?
8. Яку стратегію виведення ω в G реалізує побудований у попередньому питанні автомат, і яка обчислювальна складність алгоритму розпізнавання слова ω?

2.9 Синтаксичний аналіз без повернення назад

Проблеми загальних граматик

При виведенні слова ω в G на кожному кроці безпосереднього виведення, коли ми беремо до уваги виділений нами нетермінал (в залежності від стратегії виведення), виникає питання, яку альтернативу для A_i використати. З точки зору практики, нас цікавить така стратегія виведення ω в граматиці G, коли кожний наступний крок безпосереднього виведення наближав би нас до мети. Ця стратегія дасть можливість виконати виведення ω в G за час $O(n)$, де $n = |\omega|$.

Зрозуміло, що не маючи інформації про структуру ω, досягнути вибраної нами мети в більшості випадків неможливо. Але ж тримати інформацію про все слово ω також недопустимо. З точки зору практики, отримати потрібний результат розумно при наявності локальної інформації, наприклад, k поточних вхідних лексем програми (k — наперед фіксоване число) достатньо для організації виведення ω в G за час $O(n)$. З точки зору синтаксичного аналізу слова ω мова ведеться про наступну ситуацію:



Зафіксуємо стратегію виведення: далі будемо розглядати лише лівосторонню стратегію виведення ω в G. Тоді:

- $S \Rightarrow^{\star} \omega_1 A \omega_2$ (A — перший зліва направо нетермінал);
- ω_1 — термінальна частина слова ω, яку вже виведено (проаналізована частина слова);
- результат ω_3, який потрібно ще вивести, виводиться зі слова $A\omega_2$;

- щоб зробити вірний крок виведення (без повернення назад) нам достатньо k поточних вхідних символів з непроаналізованої частини програми ω_3.

Сформульовані умови забезпечує клас $LL(k)$-граматик.

2.9.1 $LL(k)$-граматики

КС-граматика $G = \langle N, \Sigma, P, S\rangle$ називається $LL(k)$-*граматикою* для деякого фіксованого k, якщо для двох лівосторонніх виведень вигляду:

1. $S \Rightarrow^\star \omega_1 A\omega_2 \Rightarrow \omega_1\alpha\omega_2 \Rightarrow^\star \omega_1 x$;
2. $S \Rightarrow^\star \omega_1 A\omega_2 \Rightarrow \omega_1\beta\omega_2 \Rightarrow^\star \omega_1 y$;

з $\mathrm{First}_k(x) = \mathrm{First}_k(y)$ випливає, що $\alpha = \beta$, де $A \mapsto \alpha \mid \beta$, а

$$\mathrm{First}_k(\alpha) = \{\omega \mid \alpha \Rightarrow^\star \omega x, |\omega| = k\} \cup \{\omega \mid \alpha \Rightarrow^\star \omega, |\omega| < k\}\,.$$

Неформально, граматика G буде $LL(k)$-граматикою, якщо для слова

$$\omega_1 A\omega_2 \in (N \cup \Sigma)^\star$$

достатньо k перших символів (за умови, що вони існують) решти непроаналізованого слова щоб визначити, що з $A\omega_2$ існує не більше однієї альтернативи виведення слова, що починається з ω та продовжується наступними k термінальними символами.

Сформулюємо основні твердження стосовно класу $LL(k)$-граматик:

1. Не існує алгоритма, який перевіряє належність КС-граматики класу $LL(k)$-граматик.
2. Для кожного конкретного k існує алгоритм, який перевіряє, чи є задана граматика $LL(k)$-граматикою.
3. Якщо граматика є $LL(k)$-граматикою, то вона є $LL(k{+}p)$-граматикою, $(p \geq 1)$.
4. Клас $LL(k)$-граматик — це підклас КС-граматик, який не покриває його.

Продемонструємо на **прикладі** справедливість останнього твердження. Розглянемо граматику G з наступною схемою P: $S \mapsto Sa \mid b$.

Мова, яку породжує наведена вище граматика $L(G) = \{ba^i, i = 0, 1, \ldots\}$. Візьмемо виведення наступного слова $S \Rightarrow^{i+1} ba^i$. За визначенням $LL(k)$-граматики якщо покласти $A = S$, $\omega_2 = a^i$, $\alpha = Sa$, $\beta = b$, то маємо отримати

$$\mathrm{First}_k\left(Saa^i\right) \cap \mathrm{First}_k\left(ba^i\right) = \varnothing.$$

Втім, для $i \geq k$ маємо:

$$\mathrm{First}_k\left(Saa^i\right) = \mathrm{First}_k\left(ba^i\right) = \left\{ba^{k-1}\right\}.$$

Таким чином, КС-граматика G не може бути $LL(k)$-граматикою для жодного k.

Як наслідок, КС-граматика G, яка має ліворекурсивний нетермінал A (нетермінал A називається *ліворекурсивним*, якщо в граматиці G існує вивід виду $A \Rightarrow^\star A\omega$), не може бути $LL(k)$-граматикою.

З практичної точки зору в більшості випадків ми будемо користуватися $LL(1)$-граматиками. У класі $LL(1)$-граматик існує один цікавий підклас — це розподілені $LL(1)$-граматики.

$LL(1)$-граматика називаються *розподіленою*, якщо вона задовольняє наступним умовам:

- у схемі P граматики відсутні ε-правила (правила вигляду $A \mapsto \varepsilon$);
- для нетермінала A праві частини A-правила починаються різними терміналами.

$\mathbf{First}_k$

Зауважимо, що $\mathrm{First}_k(\omega_1\omega_2) = \mathrm{First}_k(\omega_1) \oplus_k \mathrm{First}_k(\omega_2)$, де $\oplus_k$ — бінарна операція над словарними множинами (мовами) визначена наступним чином:

$$L_1 \oplus_k L_2 = \{\omega \mid \omega\omega_1 = xy, |\omega| = k\} \cup \{\omega \mid \omega = xy, |\omega| < k\}, \quad x \in L_1, \quad y \in L_2.$$

Звідси маємо наступний тривіальний висновок: якщо $\omega = \alpha_1\alpha_2 \ldots \alpha_p$, де $\alpha_i \in (N \cup \Sigma)$, то

$$\mathrm{First}_k(\omega) = \mathrm{First}_k(\alpha_1) \oplus_k \mathrm{First}_k(\alpha_2) \oplus_k \ldots \oplus_k \mathrm{First}_k(\alpha_p)$$

Для подальшого аналізу визначення $LL(k)$-граматики розглянемо алгоритм обчислення функції $\mathrm{First}_k(\alpha)$, $\alpha \in (N \cup \Sigma)$.

Алгоритм пошуку $\mathbf{First}_k$

Очевидно, що якщо $\alpha_i \in \Sigma$, то $\mathrm{First}_k(\alpha_i) = \{\alpha_i\}$ при $k > 0$. Розглянемо алгоритм пошуку $\mathrm{First}_k(A_i)$, $A_i \in N$.

Алгоритм [пошуку $\mathbf{First}_k(A_i)$, $A_i \in N$]: визначимо значення функції $F_i(x)$ для кожного $x \in (N \cup \Sigma)$:

1. $F_i(a) = \{a\}$ для всіх $a \in \Sigma$, $i \geq 0$.

2. $F_0(A_i) = \big\{\omega \mid \omega \in \Sigma^{\star k} : A_i \mapsto \omega x, |\omega| = k\big\} \cup \big\{\omega \mid \omega \in \Sigma^{\star k} : A_i \mapsto \omega, |\omega| < k\big\}$.

3. $$F_n(A) = F_{n-1}(A_i) \cup \\ \cup \left\{\omega \mid \omega \in \Sigma^{\star k} : \omega \in F_{n-1}(\alpha_1) \oplus_k \ldots \oplus F_{n-1}(\alpha_p), A_i \mapsto \alpha_1 \ldots \alpha_p\right\}.$$

4. $F_m(A_i) = F_{m+1}(A_i) = \ldots$ для всіх $A_i \in N$.

Очевидно, що:

- послідовність $F_0(A_i) \subseteq F_1(A_i) \subseteq \ldots$ — монотонно зростаюча;
- $F_n(A_i) \subseteq \Sigma^{\star k}$ — послідовність обмежена зверху.

Тоді покладемо $\mathrm{First}_k(A_i) = F_m(A_i)$ для кожного $A_i \in N$.

Приклад: знайти множину $\mathrm{First}_k(A_i)$ для нетерміналів граматики з наступною схемою правил:

$$\begin{aligned} &S \mapsto BA, \\ &A \mapsto +BA \mid \varepsilon, \\ &B \mapsto DC, \\ &C \mapsto \times DC \mid \varepsilon, \\ &D \mapsto (S) \mid a. \end{aligned}$$

Нехай $k = 2$, тоді маємо наступну таблицю:

	S	A	B	C	D
F_0	$\varnothing$	$\{\varepsilon\}$	$\varnothing$	$\{\varepsilon\}$	$\{a\}$
F_1	$\varnothing$	$\{\varepsilon\}$	$\{a\}$	$\{\varepsilon, \times a\}$	$\{a\}$
F_2	$\{a\}$	$\{\varepsilon, +a\}$	$\{a, a\times\}$	$\{\varepsilon, \times a\}$	$\{a\}$
F_3	$\{a, a+, a\times\}$	$\{\varepsilon, +a\}$	$\{a, a\times\}$	$\{\varepsilon, \times a\}$	$\{a, (a\}$
F_4	$\{a, a+, a\times\}$	$\{\varepsilon, +a\}$	$\{a, a\times, (a\}$	$\{\varepsilon, \times a, \times(\}$	$\{a, (a\}$
F_5	$\{a, a+, a\times, (a\}$	$\{\varepsilon, +a, +(\}$	$\{a, a\times, (a\}$	$\{\varepsilon, \times a, \times(\}$	$\{a, (a\}$
F_6	$\{a, a+, a\times, (a\}$	$\{\varepsilon, +a, +(\}$	$\{a, a\times, (a\}$	$\{\varepsilon, \times a, \times(\}$	$\{a, (a, ((\}$
F_7	$\{a, a+, a\times, (a\}$	$\{\varepsilon, +a, +(\}$	$\{a, a\times, (a, ((\}$	$\{\varepsilon, \times a, \times(\}$	$\{a, (a, ((\}$
F_8	$\{a, a+, a\times, (a, ((\}$	$\{\varepsilon, +a, +(\}$	$\{a, a\times, (a, ((\}$	$\{\varepsilon, \times a, \times(\}$	$\{a, (a, ((\}$
F_9	$\{a, a+, a\times, (a, ((\}$	$\{\varepsilon, +a, +(\}$	$\{a, a\times, (a, ((\}$	$\{\varepsilon, \times a, \times(\}$	$\{a, (a, ((\}$

Скористаємося визначенням $\mathrm{First}_k(\alpha)$ сформулюємо необхідні й достатні умови, за яких КС-граматика буде $LL(k)$-граматикою: для довільного виводу в граматиці G вигляду $S \Rightarrow^\star \omega_1 A \omega_2$ та правила $A \mapsto \alpha \mid \beta$:

$$\mathrm{First}_k(\alpha\omega_2) \cap \mathrm{First}_k(\beta\omega_2) = \varnothing.$$

Вище сформульована умова для $LL(k)$-граматик може бути перефразована з урахуванням визначення множини First_k: для довільного виведення в граматиці G вигляду $S \Rightarrow^\star \omega_1 A \omega_2$ та правила $A \mapsto \alpha \mid \beta$:

$$\mathrm{First}_k(\alpha \cdot L) \cap \mathrm{First}_k(\beta \cdot L) = \varnothing, \quad L = \mathrm{First}_k(\omega_2).$$

Оскільки $L \subseteq \Sigma^{\star k}$, то остання умова є конструктивною умовою і може бути використана для перевірки, чи КС-граматика є $LL(k)$-граматикою для фіксованого k.

2.9.2 Сильні $LL(k)$-граматики

КС-граматика називається *сильною $LL(k)$-граматикою*, якщо для кожного правила вигляду $A \mapsto \alpha \mid \beta$ виконується умова:

$$\mathrm{First}_k(\alpha \cdot \mathrm{Follow}_k(A)) \cap \mathrm{First}_k(\beta \cdot \mathrm{Follow}_k(A)) = \varnothing,$$

де $\mathrm{Follow}_k(\alpha)$, $\alpha \in (N \cup \Sigma)^\star$ визначається так:

$$\mathrm{Follow}_k(\alpha) = \{\omega \mid S \Rightarrow^\star \omega_1 \alpha \omega_2, \omega \in \mathrm{First}_k(\omega_2)\}.$$

Неформально, відмінність сильних $LL(k)$-граматик від звичайних $LL(k)$-граматик полягає у тому, що наступне правило безпосереднього виведення, яке буде застосовано до A можна визначити абстраговано від уже

виведеної частини слова ω_1, розглядаючи тільки наступні k символів які потрібно отримати після A.

Операції First_k та Follow_k можна узагальнити для словарної множини L, тоді:

$$\mathrm{First}_k(L) = \{\omega \mid \exists \alpha_i \in L : \omega \in \mathrm{First}_k(\alpha_i)\}.$$
$$\mathrm{Follow}_k(L) = \{\omega \mid \exists \alpha_i \in L : S \Rightarrow^\star \omega_1 \alpha_i \omega_2, \omega \in \mathrm{First}_k(\omega_2)\}.$$

Без доведення зафіксуємо наступні твердження:

- кожна $LL(1)$-граматика є сильною $LL(1)$-граматикою;
- існують $LL(k)$-граматики ($k > 1$), які не є сильними $LL(k)$-граматиками.

Не всі граматики сильні

На **прикладі** продемонструємо останнє твердження. Нехай граматика G визначена наступними правилами: $S \mapsto aAaa \mid bAba$, $A \mapsto b \mid \varepsilon$.

Відповідні множини $\mathrm{First}_2(S) = \{ab, aa, bb\}$, $\mathrm{First}_2(A) = \{b, \varepsilon\}$, $\mathrm{Follow}_2(A) = \{aa, ba\}$, $\mathrm{Follow}_2(S) = \{\varepsilon\}$.

Перевіримо умову для сильної $LL(2)$-граматики:

1. виконаємо перевірку $LL(2)$-умови для правила $S \mapsto aAaa \mid bAba$:

$$\begin{aligned}&\mathrm{First}_2(aAaa \cdot \mathrm{Follow}_2(S)) \cap \mathrm{First}_2(bAba \cdot \mathrm{Follow}_2(S)) = \\ &= (\mathrm{First}_2(aAaa) \oplus_2 \mathrm{Follow}_2(S)) \cap (\mathrm{First}_2(bAba) \oplus_2 \mathrm{Follow}_2(S)) = \\ &\quad = (\{ab, aa\} \oplus_2 \{\varepsilon\}) \cap (\{bb\} \oplus_2 \{\varepsilon\}) = \{ab, aa\} \cap \{bb\} = \varnothing.\end{aligned}$$

2. виконаємо перевірку $LL(2)$-умови для правила $A \mapsto b \mid \varepsilon$:

$$\begin{aligned}&\mathrm{First}_2(b \cdot \mathrm{Follow}_2(A)) \cap \mathrm{First}_2(\varepsilon \cdot \mathrm{Follow}_2(A)) = \\ &\quad = \{ba, bb\} \cap \{aa, ba\} = \{ba\}.\end{aligned}$$

Висновок: вище наведена граматика не є сильною $LL(2)$-граматикою. Перевіримо цю ж граматику на властивість $LL(2)$-граматики. Тут ми маємо два різні варіанти виводу з S:

1. $S \Rightarrow^\star aAaa$: $\mathrm{First}_2(b \cdot aa) \cap \mathrm{First}_2(\varepsilon \cdot aa) = \{ba\} \cap \{aa\} = \varnothing$.
2. $S \Rightarrow^\star bAba$: $\mathrm{First}_2(b \cdot ba) \cap \mathrm{First}_2(\varepsilon \cdot ba) = \{bb\} \cap \{ba\} = \varnothing$.

Висновок: наведена вище граматика є $LL(2)$-граматикою.

Алгоритм пошуку $\mathbf{Follow}_k$

Алгоритм [обчислення $\mathbf{Follow}_k(A_i)$, $A_i \in N$]: будемо розглядати всілякі дерева, які можна побудувати, починаючи з аксіоми S:

1. $\sigma_0(S,S) = \{\varepsilon\}$. Очевидно, за 0 кроків ми виведемо S, після якої знаходиться ε. У інших випадках $\sigma_0(S,A_i)$ — невизначено, $A_i \in (N \setminus \{S\})$.

2. $\sigma_1(S,A_i) = \sigma_0(S,A_i) \cup \{\omega \mid S \mapsto \omega_1 A_i \omega_2, \omega \in \mathrm{First}_k(\omega_2)\}$. В інших випадках $\sigma_1(S,A_i)$ — невизначено.

3. $\sigma_n(S,A_i) = \sigma_{n-1}(S,A_i) \cup \{\omega \mid A_j \mapsto \omega_1 A_i \omega_2, \omega \in \mathrm{First}_k(\omega_2 \cdot \sigma_{n-1}(S,A_j))\}$. В інших випадках $\sigma_n(S,A_i)$ — невизначено.

Настане крок m, коли $\sigma_m(S,A_i) = \sigma_{m+1}(S,A_i) = \ldots$, $\forall A_i \in N$.

Тоді покладемо $\mathrm{Follow}_k(A_i) = \sigma_m(S,A_i)$, $\forall A_i \in N$.

Очевидно, що:

- послідовність $\sigma_0(S,A_i) \subseteq \sigma_1(S,A_i) \subseteq \ldots$ монотонно зростаюча;
- $\sigma_n(S,A_i) \subseteq \Sigma^{\star k}$ — послідовність обмежена зверху.

Разом ці умови гарантують збіжність послідовності $\{\sigma_n(S,A_i)\}$, а отже і алгоритму пошуку $\mathrm{Follow}_k(A_i)$.

ε-нетермінали

Нетермінал A_i КС-граматики G називається *ε-нетерміналом*, якщо $A_i \Rightarrow^\star \varepsilon$.

Алгоритм [пошуку ε-нетерміналів]:

1. $S_0 = \{A_i \mid A_i \mapsto \varepsilon\}$.
2. $S_1 = S_0 \cup \{A_i \mid A_i \mapsto \alpha_1\alpha_2\ldots\alpha_p, \alpha_j \in S_0, j = \overline{1..p}\}$.
3. $S_n = S_{n-1} \cup \{A_i \mid A_i \mapsto \alpha_1\alpha_2\ldots\alpha_p, \alpha_j \in S_{n-1}, j = \overline{1..p}\}$.
4. $S_m = S_{m+1} = \ldots$.

Тоді множина S_m — множина ε-нетерміналів.

Приклад. Для граматики G з схемою правил P знайдемо множину ε-нетерміналів:

$$
\begin{aligned}
&S \mapsto aBD \mid D \mid AC \mid b,\\
&A \mapsto SCB \mid SABC \mid CbD \mid \varepsilon,\\
&B \mapsto CA \mid d,\\
&C \mapsto ADC \mid a \mid \varepsilon,\\
&D \mapsto EaC \mid SC,\\
&E \mapsto BCS \mid a.
\end{aligned}
$$

$$
\begin{aligned}
&S_0 = \{A, C\},\\
&S_1 = \{A, C\} \cup \{B, S\},\\
&S_2 = \{A, B, C, S\} \cup \{D\},\\
&S_3 = \{A, B, C, S, D\} \cup \{E\},\\
&S_4 = \{A, B, C, S, D, E\} \cup \{E\}.
\end{aligned}
$$

Таким чином, множина ε-нетерміналів для наведеної вище граматики — $\{S, A, B, C, D, E\}$.

Ліва рекурсія

До того, як перевірити граматику на $LL(k)$-властивість необхідно перевірити її на наявність ліворекурсивних нетерміналів та спробувати уникнути лівої рекурсії.

Алгоритм [тестування нетермінала A_i на ліву рекурсію]: для кожного нетермінала A_i побудуємо наступну послідовність множин $S_0, S_1, \ldots$:

1. $S_0 = \{A_i \mid A_i \mapsto \omega_1 A_i \omega_2, \omega_1 \Rightarrow^\star \varepsilon\}$, починаємо з нетерміналу A_i.
2. $S_1 = S_0 \cup \{A_i \mid A_i \mapsto \omega_1 A_j \omega_2, \omega_1 \Rightarrow^\star \varepsilon, A_j \in S_0\}$.
3. $S_n = S_{n-1} \cup \{A_i \mid A_i \mapsto \omega_1 A_j \omega_2, \omega_1 \Rightarrow^\star \varepsilon, A_j \in S_{n-1}\}$.
4. $S_m = S_{m+1} = \ldots$.

Тоді якщо $A_i \in S_m$, то A_i — ліворекурсивний нетермінал.

Приклад. Для граматики G зі схемою правил P знайдемо множину ліворекурсивних нетерміналів:

$$\begin{aligned}
&S \mapsto AbS \mid AC,\\
&A \mapsto BD,\\
&B \mapsto BC \mid \varepsilon,\\
&C \mapsto Sa \mid \varepsilon,\\
&D \mapsto aB \mid BA.
\end{aligned}$$

Виконаємо процедуру тестування для кожного нетермінала окремо, наприклад, для нетермінала S:

$$\begin{aligned}
&S_0 = \{A\},\\
&S_1 = \{A, B, D\},\\
&S_2 = \{A, B, D, C\},\\
&S_3 = \{A, B, D, C, S\}.
\end{aligned}$$

Запропонуємо декілька прийомів, що дають можливість при побудові $LL(k)$-граматик уникнути лівої рекурсії. Розглянемо граматику зі схемою правил $S \mapsto Sa \mid b$, яка має ліворекурсивний нетермінал S. Замінимо схему правил новою схемою з трьома правилами $S \mapsto bS_1$, $S_1 \mapsto aS_1 \mid \varepsilon$.

Приклад: для граматики G з схемою правил P для кожного нетермінала знайдемо множину $\mathrm{Follow}_1(A)$ $(k = 1)$:

$$\begin{aligned}
&S \mapsto BA,\\
&A \mapsto +BA \mid \varepsilon,\\
&B \mapsto DC,\\
&C \mapsto \times DC \mid \varepsilon,\\
&D \mapsto (S) \mid a.
\end{aligned}$$

З прикладу, що наведено раніше множини $\mathrm{First}_1(A)$, будуть такими:

$$\begin{aligned}
&\mathrm{First}_1(S) = \mathrm{First}_1(B) = \mathrm{First}_1(D) = \{(, a\},\\
&\mathrm{First}_1(A) = \{+, \varepsilon\},\\
&\mathrm{First}_1(C) = \{\times, \varepsilon\}.
\end{aligned}$$

	S	A	B	C	D
δ_0	$\{\varepsilon\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
δ_1	$\{\varepsilon\}$	$\{\varepsilon\}$	$\{+, \varepsilon\}$	$\varnothing$	$\varnothing$
δ_2	$\{\varepsilon\}$	$\{\varepsilon\}$	$\{+, \varepsilon\}$	$\{+, \varepsilon\}$	$\varnothing$
δ_3	$\{\varepsilon\}$	$\{\varepsilon\}$	$\{+, \varepsilon\}$	$\{+, \varepsilon\}$	$\{\times, +, \varepsilon\}$
δ_4	$\{\varepsilon,)\}$	$\{\varepsilon\}$	$\{+, \varepsilon\}$	$\{+, \varepsilon\}$	$\{\times, +, \varepsilon\}$
δ_5	$\{\varepsilon,)\}$	$\{\varepsilon,)\}$	$\{+, \varepsilon\}$	$\{+, \varepsilon\}$	$\{\times, +, \varepsilon\}$
δ_6	$\{\varepsilon,)\}$	$\{\varepsilon,)\}$	$\{+, \varepsilon,)\}$	$\{+, \varepsilon,)\}$	$\{\times, +, \varepsilon,)\}$
δ_7	$\{\varepsilon,)\}$	$\{\varepsilon,)\}$	$\{+, \varepsilon,)\}$	$\{+, \varepsilon,)\}$	$\{\times, +, \varepsilon,)\}$

Таким чином, $\text{Follow}_1(S) = \{\varepsilon,)\}$, $\text{Follow}_1(A) = \{\varepsilon,)\}$, $\text{Follow}_1(B) = \{+, \varepsilon,)\}$, $\text{Follow}_1(C) = \{+, \varepsilon,)\}$, $\text{Follow}_1(D) = \{\times, +, \varepsilon,)\}$.

2.9.3 Контрольні запитання

1. Яка граматика називається $LL(k)$-граматика?
2. Чи кожна КС-граматика є $LL(k)$-граматикою для деякого k?
3. Яка $LL(1)$-граматика називається розподіленою?
4. Яку бінарну операцію над мовами позначає символ $\oplus_k$?
5. Яку мову (множину слів) позначає запис $\text{First}_k(\alpha)$?
6. Опишіть алгоритм пошуку First_k і доведіть його збіжність.
7. Яка $LL(k)$-граматика називається сильною?
8. Чи кожна $LL(k)$-граматика є сильною $LL(k)$-граматикою?
9. Яку мову (множину слів) позначає запис $\text{Follow}_k(\alpha)$?
10. Опишіть алгоритм пошуку Follow_k і доведіть його збіжність.
11. Який нетермінал $A_i \in N$ називається ε-нетерміналом?
12. Опишіть алгоритм перевірки нетерміналу $A_i \in N$ на ε-нетермінал і доведіть його збіжність.
13. Який нетермінал $A_i \in N$ називається ліворекурсивним?
14. Опишіть алгоритм перевірки нетерміналу $A_i \in N$ на ліву рекурсію і доведіть його збіжність.

2.10 Синтаксичний аналіз на $LL(1)$-граматиках

2.10.1 Синтаксичний аналіз на основі $LL(1)$-граматик

Згідно визначення $LL(1)$-граматики, граматика G буде $LL(1)$ граматикою тоді і тільки тоді, коли для кожного A-правила вигляду $A \mapsto \omega_1 \mid \omega_2 \mid \ldots \mid \omega_p$ виконуються умови

- $\mathrm{First}_1(\omega_i) \cap \mathrm{First}_1(\omega_j) = \varnothing$ для довільних $i \neq j$.
- якщо $\omega_i \Rightarrow^\star \varepsilon$ для якогось i, то $\mathrm{First}_1(\omega_j) \cap \mathrm{Follow}_1(A) = \varnothing$ для усіх $j \neq i$.

Таблиця $M(a, b)$ де $a \in (N \cup \Sigma \cup \{\varepsilon\})$, $b \in (\Sigma \cup \{\varepsilon\})$ *керування* $LL(1)$-синтаксичним аналізатором визначається наступним чином:

1. $M(A, b)$ — номер правила вигляду $A \mapsto \omega_i$ такого, що $\{b\} = \mathrm{First}_1(\omega_i \cdot \mathrm{Follow}_1(A))$.
2. $M(a, a)$ містить інструкцію `pop` для аналізатора яка позначає необхідність перенести символ a з пам'яті аналізатору у поле результату.
3. $M(\varepsilon, \varepsilon)$ містить інструкцію `accept` для аналізаторп яка позначає що опрацьоване слово необхідно допустити (повернути `true` абощо).
4. В інших випадках $M(a, b)$ невизначено, чи радше містить інструкцію `reject` для аналізатора яка позначає що опрацьоване слово необхідно недопустити (повернути `false` абощо).

Приклад

Розглянемо вже добре відому нам граматику зі схемою

$$
\begin{aligned}
&S \mapsto BA,\\
&A \mapsto +BA \mid \varepsilon,\\
&B \mapsto DC,\\
&C \mapsto \times DC \mid \varepsilon,\\
&D \mapsto (S) \mid a.
\end{aligned}
$$

і пронумеруємо її правила таким чином:

$$S \mapsto BA, \tag{2.1}$$
$$A \mapsto +BA, \tag{2.2}$$
$$A \mapsto \varepsilon, \tag{2.3}$$
$$B \mapsto DC, \tag{2.4}$$
$$C \mapsto \times DC, \tag{2.5}$$
$$C \mapsto \varepsilon, \tag{2.6}$$
$$D \mapsto (S), \tag{2.7}$$
$$D \mapsto a. \tag{2.8}$$

Нагадаємо що для цієї граматики $\mathrm{First}_1(S) = \mathrm{First}_1(B) = \mathrm{First}_1(D) = \{(, a\}$, $\mathrm{First}_1(A) = \{+, \varepsilon\}$, $\mathrm{First}_1(A) = \{\times, \varepsilon\}$, а також $\mathrm{Follow}_1(S) = \mathrm{Follow}_1(A) = \{\varepsilon,)\}$, $\mathrm{Follow}_1(B) = \mathrm{Follow}_1(C) = \{+, \varepsilon,)\}$, $\mathrm{Follow}_1(D) = \{+, \times, \varepsilon,)\}$.

Знайдемо множини $\mathrm{First}_1(\omega_i \cdot \mathrm{Follow}_1(A))$ як $\mathrm{First}_1(\omega_i) \oplus_1 \mathrm{Follow}_1(A))$ використовуючи результати минулої лекції.

При побудові таблиці $M(a, b)$ керування $LL(1)$-синтаксичним аналізатором достатньо лише побудувати першу її частину, тобто ту яка з $N \times (\Sigma \cup \{\varepsilon\})$, оскільки решта таблиці визначається стандартно:

	a	(	)	+	×	ε
S	1	1				
A			3	2		3
B	4	4				
C			6	6	5	6
D	8	7				

Алгоритм

Побудуємо $LL(1)$-синтаксичний аналізатор на основі таблиці керування $M(a, b)$:

1. Прочитаємо поточну лексему з вхідного файла, у стек магазинного автомата занесемо аксіому S.

2. Загальний крок роботи:

   - Якщо на вершині стека знаходиться нетермінал A_i, то активізувати рядок таблиці, позначений A_i. Елемент $M(A_i, \langle\text{поточна лексема}\rangle)$ визначає номер правила, права частина якого заміняє A_i на вершині стека.

- Якщо на вершині стека лексема $a_i = \langle$поточна лексема$\rangle$, то з вершини стека зняти a_i та прочитати нову поточну лексему.
- Якщо стек порожній та досягли кінця вхідного файла, то вхідна програма синтаксично вірна.
- В інших випадках — синтаксична помилка.

Майже $LL(1)$-граматики

У деяких випадках досить складно (а інколи й принципово неможливо побудувати $LL(1)$-граматику для реальної мови програмування. При цьому $LL(1)$-властивість задовольняється майже для всіх правил — лише декілька правил створюють конфлікт, але для цих правил задовольняється **сильна** $LL(2)$-властивість. Тоді таблиця $M(a, b)$ визначається в такий спосіб:

- $M(A, b) = \langle$номер правила$\rangle$ вигляду $A_i \mapsto \omega_i$, такого, що $b \in \mathrm{First}_1(\omega_i \cdot \mathrm{Follow}_1(A))$
- $M(A, b) = \langle$ім'я допоміжної програми$\rangle$ за умови, що

$$b \in \mathrm{First}_1(\omega_i \cdot \mathrm{Follow}_1(A)) \cap b \in \mathrm{First}_1(\omega_j \cdot \mathrm{Follow}_1(A)), \quad i \neq j$$

Програма, яка виконує додатковий аналіз вхідного ланцюжка, повинна:

- прочитати додатково одну лексему;
- на основі двох вхідних лексем вибрати необхідне правило або сигналізувати про синтаксичну помилку;
- у випадку, коли правило вибрано, необхідно повернути додатково прочитану лексему у вхідний файл.

Звичайно, необхідно модифікувати алгоритм $LL(1)$-синтаксичного аналізатора.

При цьому підпрограма аналізу конфліктної ситуації повинна додатково прочитати нову вхідну лексему, далі скориставшись контекстом з двох лексем, визначити номер правила, яке замість нетермінала на вершині стека та повернути додатково прочитану лексему у вхідний файл.

2.10.2 Контрольні запитання

1. Які дві умови повинна задовольняти граматика щоб бути $LL(1)$-граматикою?

2. Що таке таблиця керування синтаксичного аналізатора на основі $LL(1)$-граматики?

3. Який автомат і яка таблиця використовуються в алгоритмі роботи $LL(1)$-синтаксичного аналізатора?

4. Опишіть алгоритм роботи $LL(1)$-синтаксичного аналізатора.

5. Як необхідно модифікувати таблицю керування для сильної $LL(2)$-граматики яка є майже $LL(1)$-граматикою?

6. Як необхідно модифікувати алгоритм роботи синтаксичного аналізатора для сильної $LL(2)$-граматики яка є майже $LL(1)$-граматикою?

2.11 Програмування синтаксичних аналізаторів

2.11.1 Синтаксична діаграма

Синтаксична діаграма — це орієнтований граф, дуги котрого позначені елементами $(N \cup \Sigma)^\star$. Синтаксична діаграма будується для кожного A-правила КС-граматики мови програмування.

Оскільки вершини такого графа не іменуються, то вони припускаються неявно. Синтаксична діаграма позначається іменем нетермінала, для якого вона будується.

Мета побудови синтаксичних діаграм для мови програмування на основі КС-граматики:

- для кожного A-правила КС-граматики будується синтаксична діаграма;
- на основі побудови синтаксичної діаграми для деякого нетермінала $A \in N$ будуємо підпрограму, яка аналізує ту частину головної програми, яку вона визначає.

Оскільки у більшості випадків при визначенні синтаксису мови програмування ми користуємося множиною рекурсивних правил, то серед підпрограм, які будуються на основі правил граматики, будуть і рекурсивні процедури (рекурсія буде як явна, так і неявна).

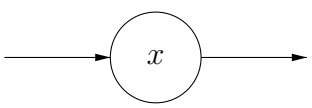
Сформулюємо правила побудови синтаксичного графа:

1. Кожен нетермінал з відповідною множиною породжуючих правил $A \mapsto \omega_1 \mid \omega_2 \mid \ldots \mid \omega_p$, $\omega_i \in (N \cup \Sigma)^\star$, $i = \overline{1..p}$ відображається в один синтаксичний граф.

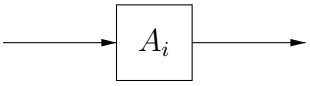
   Отже, кількість синтаксичних графів рівна кількості нетерміналів граматики G.

2. Для кожного елемента слова $\omega = \alpha_1\alpha_2 \ldots \alpha_p$, $\alpha_i \in (N \cup \Sigma)$, $i = \overline{1..p}$ будуємо ребро синтаксичного графа та покажемо його таким чином що:

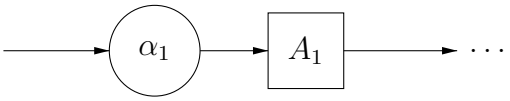
   - якщо $\alpha_i = x$, $x \in \Sigma$, де x — вихідна лексема, то будуємо таке ребро:



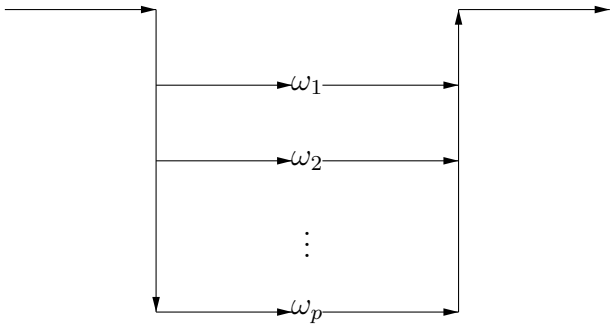
- якщо $\alpha_k = A_i \in N$ — нетермінал граматики, то будуємо таке ребро:



Тоді, коли правило граматики G має вигляд $A_i \mapsto \alpha_1 A_1 \ldots$ для побудови діаграми скористаємося обома способами:



Коли правило граматики G має вигляд $A_i \mapsto \omega_1 \mid \omega_2 \mid \ldots \mid \omega_p$, то відповідний синтаксичний граф буде мати вигляд:



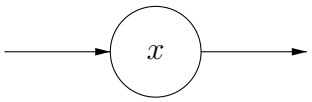
де замість $\omega_1, \omega_2, \ldots, \omega_p$ будуються відповідні синтаксичні діаграми.

Якщо на основі граматики мови програмування побудована множина синтаксичних графів, то можна спробувати зменшити їх кількість, скориставшись підстановкою одних графів у інші. При цьому замість елемента A_i підставляється його синтаксичний граф. Таким чином можна зменшити кількість синтаксичних графів. Для того, щоб забезпечити детермінований синтаксичний аналіз з переглядом вперед на одну лексему, потрібно накласти певні обмеження, а саме: для кожного правила A_i виду $A_i \mapsto \omega_1 \mid \omega_2 \mid \ldots \mid \omega_p$ з синтаксичною діаграмою наведеного вище вигляду необхідне виконання наступної умови: множини $\mathrm{First}_1(\omega_j) \oplus_1 \mathrm{Follow}_1(A_i)$, $j = \overline{1..p}$ повинні попарно не перетинатися. Зрозуміло, що ця умова гарантує детермінований вибір шляху при русі по синтаксичній діаграмі.

Код

Подальше програмування синтаксичного аналізатора можна звести до наступних примітивів:

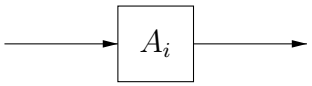
- для фрагмента синтаксичної діаграми вигляду



відповідний фрагмент програми (наприклад, мовою C) матиме вигляд:

```
extern int lexem_code;  // код лексеми, яку виділив сканер
extern char lexem_text[];  // текст лексеми
...
if (lexem_code == code_x) get_lexem();
else error();
```

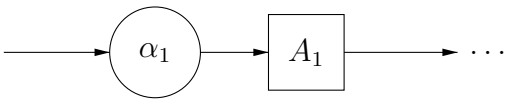
- для фрагмента синтаксичної діаграми вигляду



відповідний фрагмент програми матиме вигляд:

```
// виклик функції, яка побудована для синтаксичної
// діаграми побудованої для нетермінала A_i.
f_A_i();
```

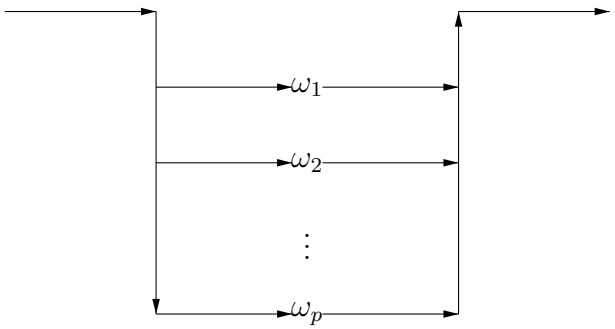
- для фрагмента синтаксичної діаграми вигляду



відповідний фрагмент програми матиме вигляд:

```
extern int lexem_code;
extern char lexem_text[];
...
{
    if (lexem_code == code_alpha_1) get_lexem();
    else error();
    f_A_1();
    ...
}
```

- для фрагмента синтаксичної діаграми вигляду



для кожного ω_i, $i = \overline{1..p}$ знайдемо множину $\mathrm{First}_1(\omega_i) \oplus_1 \mathrm{Follow}_1(A) = L_i = \{a_i^1, a_i^2, \ldots, a_i^{n_i}\}$.

Оскільки за умовою $L_i \cap L_j = \varnothing$, $i \neq j$, то відповідний фрагмент програми на мові C матиме вигляд:

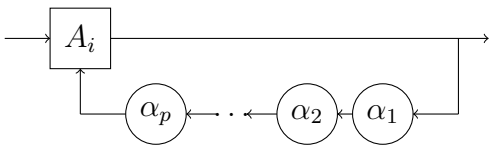
```
extern int lexem_code;
extern char lexem_text[];
...
void f_A_i(void) {
    switch(lexema_code) {
        case code_a_1_1:
        case code_a_1_2:
        ...
        case code_a_1_n_1:
            ...  // фрагмент програми для w_1
           break;
        case code_a_2_1:
        case code_a_2_2:
```

```
        ...
        case code_a_2_n_2:
            ...  // фрагмент програми для w_2
            break;
        ...
        ...
        ...
        case code_a_p_1:
        case code_a_p_2:
        ...
        case code_a_p_n_p:
            ...  // фрагмент програми для w_p
            break;
        default:
            error();
    }
}  // кінець функції для нетермінала A_i
```

Відмітимо, що до того, як зменшувати кількість синтаксичних діаграм шляхом суперпозиції одних діаграм в інші, необхідно знайти контексти виду $\mathrm{First}_1(\omega_i) \oplus_1 \mathrm{Follow}_1(A)$ для тієї синтаксичної діаграми нетермінала A, для якої ми виконуємо операцію суперпозиції. Ці контексти ми використаємо при програмуванні синтаксичного аналізатора на основі синтаксичної діаграми, у яку підставлено синтаксичну діаграму для нетермінала A_i.

Одна особливість

Досить часто при визначенні синтаксису мови програмування користуються синтаксичними правилами виду $A_i \mapsto \alpha_1\alpha_2 \ldots \alpha_p A_i \mid \varepsilon$. Тоді синтаксична діаграма буде мати вигляд:



Для вище наведеної синтаксичної діаграми відповідні множини будуть: $\mathrm{First}_1(\alpha_1\alpha_2 \ldots \alpha_p A_i) \oplus_1 \mathrm{Follow}_1(A_i) = L_1 = \{a_1^1, a_1^2, \ldots, a_1^{n_1}\}$. Відповідний фрагмент програми мовою C матиме вигляд:

```
extern int lexem_code;
```

```
extern char lexem_text[];
void A_i(void) {
    while (
        lexem_code == code_a_1_1 ||
        lexem_code == code_a_1_2 ||
        ... ||
        lexem_code == code_a_1_n_1
    ) {
        ...  // фрагмент програми для слова w
    }
}  // кінець підпрограми для нетермінала A_i
```

Виконавши аналіз варіантів побудови синтаксичного аналізатора на основі синтаксичних діаграм, покажемо вигляд основної — main-програми:

```
int lexem_code;
char lexem_text[1<<8];
int main () {
    get_lexem();
    axiom();  // процедура, пов'язана з аксіомою S граматики
}
```

2.11.2 Контрольні запитання

1. Який граф називається синтаксичною діаграмою?
2. Як на синтаксичній діаграмі позначаються термінали і нетермінали?
3. Як на синтаксичній діаграмі позначаються прості (без |) і складені (з |) правила?
4. Напишіть фрагмент коду (наприклад на мові C) для обробки терміналів і нетерміналів.
5. Напишіть фрагмент коду (наприклад на мові C) для обробки простих (без |) і складених (з |) правил.
6. Як на синтаксичній діаграмі позначаються правила вигляду $A \mapsto \omega \mid \varepsilon$?
7. Напишіть фрагмент коду (наприклад на мові C) для обробки правил вигляду $A \mapsto \omega \mid \varepsilon$.

2.12 Побудова $LL(k)$-синтаксичного аналізатора

2.12.1 Побудова $LL(k)$-синтаксичного аналізатора

Повернемось до умови, при якій граматика G буде $LL(k)$-граматикою, а саме: для довільного виведення $S \Rightarrow^\star \omega_1 A \omega_2$ та правила $A \mapsto \alpha \mid \beta$ маємо $\mathrm{First}_l(\alpha \cdot L) \cap \mathrm{First}_k(\beta \cdot L) = \varnothing$, де $L = \mathrm{First}_k(\omega_2)$.

Оскільки $L \subseteq \Sigma^{\star k}$ — конструктивна множина, спробуємо побудувати всілякі множини L, які задовольняють попередньо сформульованій умові.

$\mathbf{Local}_k(S, A)$

Визначимо наступну множину:

$$\mathrm{Local}_k(S, A) = \{L \mid \exists x, \omega : S \Rightarrow^\star xA\omega, L = \mathrm{First}_k(\omega)\}$$

Опишемо алгоритм пошуку цієї множини:

1. $\delta_0(S, S) = \{\{\varepsilon\}\}$, в інших випадках — невизначено.
2. $\delta_1(S, A_i) = \delta_0(S, A_i) \cup \{L \mid S \mapsto \omega_1 A_i \omega_2, L = \mathrm{First}_k(\omega_2)\}$, в інших випадках — невизначено.
3. $$\delta_n(S, A_i) = \delta_{n-1}(S, A_i) \cup \\ \cup \{L \mid A_j \mapsto \omega_1 A_i \omega_2, L = \mathrm{First}_k(\omega_2) \oplus_k L_p, L_p \in \mathrm{Local}_k(S, A_j)\},$$ в інших випадках — невизначено.
4. $\delta_m(S, A_i) = \delta_{m+1}(S, A_i) = \ldots, \forall A_i \in N$.

Тоді $\mathrm{Local}_k(S, A_i) = \delta_m(S, A_i)$.

Виходячи з означення $\mathrm{Local}_k(S, A_i)$, умови для $LL(k)$-граматики будуть наступними: для довільного A-правила вигляду $A \mapsto \omega_1 \mid \omega_2 \mid \ldots \mid \omega_p$ маємо:

$$\mathrm{First}_k(\omega_i \cdot L_m) \cap \mathrm{First}_k(\omega_j \cdot L_m) = \varnothing, \quad i \neq k, \quad L_m \in \mathrm{Local}_k(S, A).$$

Як наслідок, з алгоритму пошуку $\mathrm{Local}_k(S, A_i)$ видно, що

$$\mathrm{Follow}_k(A_i) = \bigcup_{j=1}^{m} L_j, \quad L_j \in \mathrm{Local}_k(S, A_i).$$

Таблиці керування

Для побудови синтаксичного аналізатора для $LL(k)$-граматики ($k > 1$) необхідно побудувати множину таблиць, що забезпечать нам безступиковий аналіз вхідного ланцюжка w (програми) за час $O(n)$, де $n = |w|$.

Побудову множини таблиць для управління $LL(k)$-аналізатором почнемо з таблиці, яка визначає перший крок безпосереднього виводу w в граматиці G: $T_0 = T_{S,\{\varepsilon\}}(u) = (T_1\alpha_1 T_2\alpha_2 \dots T_p\alpha_p, n)$, де n — номер правила вигляду $S \mapsto A_1\alpha_1 A_2\alpha_2 \dots Ap\alpha_p$, а $A_i \in N$, $\alpha_i \in \Sigma^\star$, і $u = \mathrm{First}_k(A_1\alpha_1 A_2\alpha_2 \dots A_p\alpha_p)$, і нарешті $i = \overline{1..p}$. Зрозуміло, що в інших випадках (якщо такого правила немає абощо) T_0 не визначена.

Неформально, коли в магазині автомата знаходиться аксіома S, то нас цікавить перших k термінальних символів, які можна вивести з S (аксіома — поняття "програма") при умові, що після неї (програми) буде досягнуто EOF.

Імена інших таблиць $T_1, T_2, \dots, T_p$ визначаються так: $T_i = T_{A_i,L_i}$, де $L_i = \mathrm{First}_k(\alpha_i A_{i+1}\alpha_{i+1} \dots A_p\alpha_p)$, $i = \overline{1..p}$.

Наступні таблиці визначаються так: $T_i = T_{A_i,L_i}(u) = (T_1\alpha_1 T_2\alpha_2 \dots T_p\alpha_p, n)$, де n — номер правила вигляду $A_i \mapsto A_1\alpha_1 A_2\alpha_2 \dots Ap\alpha_p$, а $A_j \in N$, $\alpha_j \in \Sigma^\star$, і $u = \mathrm{First}_k(A_1\alpha_1 A_2\alpha_2 \dots A_p\alpha_p) \oplus_k L_i$, і нарешті $j = \overline{1..p}$. Зрозуміло, що в інших випадках (якщо такого правила немає абощо) T_i не визначена.

Імена інших таблиць $T_1, T_2, \dots, T_p$ визначаються так: $T_j = T_{A_j,L_j}$, де $L_j = \mathrm{First}_k(\alpha_j A_{j+1}\alpha_{j+1} \dots A_p\alpha_p) \oplus_k L_i$, $j = \overline{1..p}$.

Приклад

Побудувати множину таблиць управління для $LL(2)$-граматики з наступною схемою правил:

$$S \mapsto abA, \tag{2.1}$$
$$S \mapsto \varepsilon, \tag{2.2}$$
$$A \mapsto Saa, \tag{2.3}$$
$$A \mapsto b. \tag{2.4}$$

Для вищенаведеної граматики множини $\mathrm{First}_2(A_i)$, $A_i \in N$ будуть такі: $\mathrm{First}_2(S) = \{ab, \varepsilon\}$, $\mathrm{First}_2(A) = \{aa, ab, b\}$, а множини $\mathrm{Local}_2(S, A_i)$, $A_i \in N$ будуть такі: $\mathrm{Local}_2(S, S) = \mathrm{Local}_2(S, A) = \{\{\varepsilon\}, \{aa\}\}$.

Побудуємо першу таблицю $T_0 = T_{S,\{\varepsilon\}}$. Для S-правила відповідні множини u будуть такі:

- $S \mapsto abA$, $u \in \text{First}_2(abA) = \{ab\}$.
- $S \mapsto \varepsilon$, $u \in \text{First}_2(\varepsilon) = \{\varepsilon\}$.

Таблиця T_0 визначається так:

	aa	ab	ba	bb	a	b	ε
$T_0 = T_{S,\{\varepsilon\}}$		abT_1, 1					ε, 2

Нова таблиця управління $T_1 = T_{A,\{\varepsilon\}}$. Для A-правила відповідні множини u будуть такі:

- $A \mapsto Saa$, $u \in \text{First}_2(Saa) \oplus_2 \{\varepsilon\} = \{ab, aa\}$.
- $A \mapsto b$, $u \in \text{First}_2(b) \oplus_2 \{\varepsilon\} = \{b\}$.

Таблиця T_1 визначається так:

	aa	ab	ba	bb	a	b	ε
$T_1 = T_{A,\{\varepsilon\}}$	T_2aa, 3	T_2aa, 3				b, 4	

Нова таблиця управління $T_2 = T_{S,L}$ де $L = \text{First}_2(aa) \oplus_2 \{\varepsilon\} = \{aa\}$. Для таблиці T_2 та S-правила множини u будуть такі

- $S \mapsto abA$, $u \in \text{First}_2(abA) \oplus_2 \{aa\} = \{ab\} \oplus_2 \{aa\} = \{ab\}$.
- $S \mapsto \varepsilon$, $u \in \text{First}_2(\varepsilon) \oplus_2 \{aa\} = \{aa\}$.

	aa	ab	ba	bb	a	b	ε
$T_2 = T_{S,\{aa\}}$	ε, 2	abT_3, 1					

Наступна таблиця $T_3 = T_{A,L}$ де $L = \text{First}_2(\varepsilon) \oplus_2 \{aa\} = \{aa\}$. Для таблиці T_3 та A-правила множини u будуть такі:

- $A \mapsto Saa$, $u \in \text{First}_2(Saa) \oplus_2 \{aa\} = \{ab, aa\}$.
- $A \mapsto b$, $u \in \text{First}_2(b) \oplus_2 \{aa\} = \{ba\}$.

Таблиця T_3 визначається так:

	aa	ab	ba	bb	a	b	ε
$T_3 = T_{A,\{aa\}}$	T_2aa, 3	T_2aa, 3	b, 4				

Нова таблиця $T_4 = T_{S,L} = T_2$, оскільки $L = \mathrm{First}_2(aa) \oplus_2 \{aa\} = \{aa\}$.

Ми визначили чотири таблиці-рядки (а їх кількість для довільної $LL(k)$-граматики визначається як $\sum_{i=1}^{n} n_i$, де n_i — кількість елементів множини $\mathrm{Local}_k(S, A_i)$, $m = |N|$.

Об'єднаємо рядки-таблиці в єдину таблицю та виконаємо перейменування рядків:

	aa	ab	ba	bb	a	b	ε
T_0		abT_1, 1					ε, 2
T_1	T_2aa, 3	T_2aa, 3				b, 4	
T_2	ε, 2	abT_3, 1					
T_3	T_2aa, 3	T_2aa, 3	b, 4				

Алгоритм

Синтаксичний аналізатор для $LL(k)$-граматики ($k > 1$).

1. Прочитати k лексем з вхідного файла програми (звичайно, інколи менше ніж k). В магазин занести таблицю T_0.

2. Загальний крок:

   - Якщо на вершині магазина знаходиться таблиця T_i, то елемент таблиці $M(T_i, \langle\text{k вхідних лексем}\rangle)$ визначає ланцюжок, який T_i заміщає на вершині магазина.
   - Якщо на вершині магазина $a_i \in \Sigma$ перша поточна лексема з k прочитаних лексем рівна a_i, то з вершини магазина зняти a_i та прочитати з вхідного файла додатково одну лексему (звичайно, якщо це можливо).
   - Якщо досягли кінця вхідного файла програми та магазин порожній, то програма не має синтаксичних помилок.
   - В інших випадках — синтаксична помилка.

2.12.2 Контрольні запитання

1. Наведіть визначення множини $\mathrm{Local}_k(S, A)$.
2. Опишіть алгоритм побудови $\mathrm{Local}_k(S, A)$.
3. Опишіть алгоритм побудови таблиць керування (або рядків великої результуючої таблиці керування).
4. Якою формулою визначається кількість рядків таблиці керування?
5. Опишіть алгоритм синтаксичного аналізу для $LL(k)$-граматики.