УДК 004:37

І. Б. Швороб
Національний університет “Львівська політехніка”,
кафедра інформаційних систем та мереж

ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ СИНТАКСИЧНОГО РОЗБОРУ ТЕКСТІВ

Описано деякі алгоритми синтаксичного аналізу. Також порівняно продуктивність роботи обраних алгоритмів

Ключові слова: парсинг, контекстно-вільна граматика, аналіз тексту.

Some parsing algorithms have been shown and described in the article. The performance comparison of the selected algorithms is made.

Key words: parsing, context-free grammar, text analysis.

Вступ. Загальна постановка проблеми

Нині людство все частіше використовує інформацію в цифровому вигляді. І часто виникає необхідність для конкретного аналізу цієї інформації та її структурування. З цією метою використовується синтаксичний розбір (парсинг) у лінгвістиці та інформатиці – процес порівняння лінійної послідовності лексем (слів) природної або формальної мови з її формальною граматикою. Результатом є, як правило, дерево розбору (синтаксичне дерево). Інакше кажучи, парсинг – це процес аналізу або розбору тексту на компоненти з використанням спеціального програмного забезпечення. Парсер – це програма, яка аналізує текстові документи, зберігає аналіз даних у своїй базі даних, а потім видає їх при пошуку актуальних і поточних даних. Аналізатор може виявити велику кількість корисної інформації та обробляти її, залежно від завдань.

Синтаксичний аналіз є важливою складовою опрацювання тексту і спрямований на розпізнавання, виділення та групування даних.

У галузі пошукової оптимізації парсинг використовується дуже часто. Всі SEO-інструменти щось аналізують (посилання, ключові слова) і на цій основі забезпечують корисні дані для аналізу.

Використовуючи парсинг, можна дуже швидко опрацьовувати великі обсяги інформації, оскільки вручну це робити практично не можливо. Загалом парсинг є ефективним рішенням для автоматизації збору та зміни інформації.

Парсер повинен мати такі характеристики:

- забезпечувати швидкий обхід великої кількості інформації;
- грамотно і акуратно відділяти технічну інформацію від нетехнічної;
- безпомилково вибирати потрібну інформацію і відкидати зайву;
- ефективно подавати і зберігати дані у потрібному форматі.

Будь-який аналізатор складається з трьох частин, які відповідають за три окремі процеси розбору:

- отримання тексту у його первісному вигляді. Отримання тексту часто означає завантаження текстового документа, з якого необхідно отримати певні дані.
- вилучення та перетворення даних. Необхідні дані, які були отримані на першому етапі на цій фазі «витягаються». Для вилучення найчастіше використовуються регулярні вирази. Крім того, на цьому етапі здійснюється перетворення витягнутих даних у певний формат, якщо це необхідно;
- генерування результатів. Це заключний етап аналізу. Він досягається шляхом виводу або запису даних, отриманих на другому етапі, у бажаному форматі. Часто записують безпосередньо в базі даних.

Завдання синтаксичного аналізатора, по суті, полягає у визначенні, чи є і в який спосіб вхідні дані можуть бути отримані з початкового символу граматики.

Метою роботи є вивчення декількох алгоритмів для синтаксичного розбору та аналіз їх роботи.

Аналітичний огляд алгоритмів синтаксичного розбору

Класифікація методів синтаксичного розбору

Існують різні класифікації синтаксичного розбору тексту. За основою класифікації їх поділяють на такі види:

1) за методом синтаксичного розбору:
   - нисхідний;
   - висхідний;
   - комбінований;
2) за послідовністю у розборі:
   - зліва направо;
   - справа наліво;
   - довільний;
3) за переглядом наперед:
   - на один символ;
   - на два символи;
   - на *n* символів;
4) за використанням повторень:
   - є повторення;
   - нема повтореннь.

Нисхідний синтаксичний аналіз можна розглядати як спробу знайти найлівіший диференційований елемент вхідного потоку за рахунок пошуку дерев розбору, використовуючи розширення даних формальних правил граматики проходом згори донизу. Лексеми «поглинаються» зліва направо. Виключний вибір використовується для узгодження неоднозначностей, використовуючи розширення всіх альтернативних правобічних граматичних правил [1]. Такий аналіз можна легко здійснити вручну, наприклад методом рекурсивного спуску.

Висхідний синтаксичний аналіз може починати розбір з входу і пробувати переписати його на початковий символ. Інтуїтивно зрозуміло, що аналізатор намагається знайти основні елементи, потім елементи, які містять основні, тощо. LR-аналізатоо є прикладом висхідного аналізатора.

Послідовність розбору визначає, в який спосіб буде побудовано дерево розбору на кожному кроці підстановки. Ці підстановки можна здійснювати зліва направо, справа наліво або довільно. Слід зазначити, що використання упорядкованого розбору прискорює його виконання за рахунок зменшення кількості правил, що перебираються.

Послідовність розбору безпосередньо взаємодіє з методом розбору. Тобто, за спадного розбору зліва направо при підстановці правил замість найлівіших нетерміналів вхідний ланцюжок розпізнаватиметься з його початку. При спадному розборі справа наліво — початкове підтвердження символів здійснюється з кінця ланцюжка. І навпаки, для висхідного розбору зліва направо здійснюється заміна на нетермінал символів, розташованих наприкінці ланцюжка. Заміна початкових символів проводиться при висхідному розборі справа наліво. Довільний розбір не обумовлює послідовність підстановки правил. Це веде до більшої кількості переборів.

Підвищення ефективності розбору здійснюється розробкою граматик, що спеціально підтримують погоджені між собою метод і послідовність. Тобто, граматики, призначені для спадного розбору, використовують для виводу прохід зліва направо. Отже, і вхідний ланцюжок буде розбиратися зліва направо. Це дозволяє швидше одержувати потрібні символи, а не чекати кінця ланцюжка і лише потім розбирати. Граматики, орієнтовані на висхідний розбір, зазвичай оптимізовані під правий вивід вхідного ланцюжка, що дозволяє, при синтаксичному розборі, здійснювати підстановки нетерміналів зліва праворуч.

Перегляд наперед — це один з можливих варіантів упорядкування підстановок, що забезпечує вирішення проблеми недетермінованості. Поряд з ним використовуються: перетворення граматик до детермінованого вигляду та аналіз з поверненнями.

Синтаксичний розбір з поверненнями виконується так само, як і непрямий лексичний аналіз. Повернення застосовуються для тих правил, які починаються з однакових підланцюжків. У цьому випадку поява відмови при розборі правила веде до відновлення входу в те положення, у якому воно було до входу в дане правило. Використання повернень може виступати альтернативою перегляду наперед. Пріоритет правил, що визначає порядок їх обходу, призначається також як і при лексичному аналізі і залежить від того, чи є деяке правило підмножиною іншого. Метод універсальний і легкий для розуміння і реалізації. Однак такий підхід сповільнює розбір і може вести до додаткових витрат при подальшому опрацюванні.

Аналіз обраних алгоритмів

Для аналізу було обрано такі алгоритми: алгоритм Earley, LL парсер, метод рекурсивного спуску, CKY парсер, LALR аналізатор і Pratt парсер.

Алгоритм Earley **—** алгоритм розбору запропонованих даних для контекстно-вільної граматики; він заснований на методі динамічного програмування [2]. Цей алгоритм не накладає ніяких обмежень на аналіз контекстно-вільної граматики, яка використовується. Алгоритм Earley реалізує стратегію проходу «зліва направо». Алгоритм синтаксичного аналізу може бути представлений у вигляді обчислюваної функції розбору з двома аргументами *Parse (G, ω)*:

• $G=\{N,T,P,S\}$ — контекстно-вільна граматика з великою кількістю не-термінальних символів N, множиною терміналів T, набором правил P і початковою граматикою нетерміналів S;

• $\omega = a_1 \ldots a_n$ — рядок з n термінальних символів граматики G.

Функція розбору *Parse* повертає множину дерев виведення вхідного рядка ω, якщо вона виведена в граматиці G, і значення *False* в іншому випадку.

Синтаксичний аналізатор Earley здійснює аналіз алгоритму вхідного рядка символів за рахунок проходження знизу догори і отримується єдиноправильний вихід вхідного рядка, якщо вхідна граматика є однозначною, або набір правил виведення, якщо вхідна граматика є неоднозначною. Оригінальний алгоритм Earley тільки виявляє вхідний рядок, але не розбирає його.

Алгоритм Earley використовує три обчислювальні процедури для побудови станів:

• *Сканер* (S_i): сканує кожен елемент у стані S_i і, якщо символ X_p дорівнює терміналу a_{i+1} у деяких ситуаціях$[r,p,j]$, додає до значення стану S_{i+1}.

• *Предиктор* $([r,p,j],S_i)$: перевіряє, чи є символ X_p нетермінальним символом граматики G, і якщо так, він перевіряє, чи $L_{r'}=X_p$ виконується для кожного правила r' граматики G, якщо так, то ситуація $[r', 0, i]$ додає до значення стану S_i.

• *Укладач* $([r,p,j],S_i)$: сканує кожну ситуацію $[r', p', k]$ станів S_j, і якщо $X_{p'}=L_r$, то ситуація $[r', p'+1, k]$ додає до значення стану S_i.

Процедуру *Сканер* (S_i) викликають передусім, щоб заповнити стани S_i, тоді до нової ситуації $[r,p,j]$ додавання до стану S_i, потім викликаються процедури *Предикатор*$([r,p,j],S_i)$ і *Укладач*$([r,p,j],S_i)$: для кожної доданої ситуації.

LL аналізатор **—** це нисхідний аналізатор для підмножини контекстно-вільних граматик [3]. Він аналізує вхідні дані зліва направо, виконуючи ліве породження рядка. LL аналізатор називається LL(k) парсером, якщо він використовує k попередніх символів при розборі речення.

Аналізатор працює з рядками з конкретної контекстно-вільної граматики.

Синтаксичний аналізатор складається з:

• вхідного буфера, що зберігає введений рядок (побудований з граматики);

• стека, в якому можна зберігати ще не проаналізовані термінали і нетермінали з граматики;

• таблиці розбору, в якій зазначено, яке (якщо таке є) граматичне правило застосовувати до цього символу на вершині стека і наступного вхідного символу.

Аналізатор застосовує правило, знайдене в таблиці, зіставляючи верхній символ у стеку (рядок) з поточним символом у вхідному потоці (колонка).

На кожному етапі аналізатор читає наступний доступний символ з вхідного потоку і символ у вершині стека. Якщо вхідний символ і символ на вершині стека сходяться, парсер відкидає їх обох, залишивши тільки символ, який не відповідає.

Метод рекурсивного спуску — нисхідний алгоритм синтаксичного аналізу, що здійснюється за рахунок виклику процедур взаємного розбору. Кожна процедура відповідає одному з правил контекстно-вільної граматики. Правила застосовуються послідовно, поглинають елементи, отримані від лексичного аналізатора, зліва направо. Це один з найпростіших алгоритмів для розбору, він підходить для повної ручної реалізації. Основна операція, необхідна для такого розбору, об'єднує зчитування символів з вхідного потоку і узгодження з терміналами від граматики, яка описує синтаксис вхідного сигналу.

Контекстно-вільна граматика може бути використана для створення або визнання рядка на своїй мові. Алгоритм такого розбору можна записати так:

• один метод розбору нетермінального символу;

• нетермінальний символ на правій стороні правила перезапису призводить до виклику методу розбору для цього нетерміналу;

• термінальний символ на правій стороні правила перезапису призводить до «поглинання» лексем вхідного рядка;

• контекстно-вільна граматика призводить до вибору «якщо-інакше» в аналізаторі;

• {...} контекстно-вільна граматика призводить до умови «поки» в аналізатор.

CKY (Cocke-Kasami-Younger) аналізатор є одним з найбільш ранніх алгоритмів розпізнавання та аналізу [4]. Стандартна версія CKY може розпізнавати лише мови, визначені контекстно-вільними граматиками в нормальній формі Хомського (CNF). Цей аналізатор базується на основі динамічного програмування: він будує рішення, що складаються з субрішень. CKY аналізатор використовує безпосередньо граматику.

Цей алгоритм враховує всі можливі підпослідовності послідовності слів і встановлює *P[i, j, k]*, щоб мати значення True, якщо підпослідовність слів, починаючи з *i* довжиною *j* можуть бути отримані з R_k. Після того, як він розглянув підпослідовності з довжиною 1, він йде на підпослідовності довжиною 2, і так далі. Для підпослідовностей з довжиною 2 і більше, він розділяє всі можливі підпослідовності на дві частини, і перевіряє, чи $P \rightarrow QR$ спрацьовує так, що *Q* відповідає першій частині і *R* відповідає другій частині. Якщо це так, він записує *P* у відповідність до всієї підпослідовності. Після того, як цей процес буде завершено, речення визнається граматикою, якщо підпослідовність, що містить всі підречення, які поєднуються з початковим символом.

LALR парсер — висхідний алгоритм аналізу. Він є розширенням алгоритму SLR [5]. Клас граматик розбирається за використання LALR ширше, ніж клас SLR граматики. Припустимо, у нас є граматика, яка перетворюється так:

• здійснюється пошук нетермінала, який має скорочення, викликане конфліктом;

• вводяться нові нетермінали A_1, A_2, ..., A_n, по одному для кожної входження *A* в правобічні правила;

• у будь-якому місці правого боку правил *A* замінюється відповідним A_k;

• набір правил з лівого боку повторюється *n* раз для кожного A_k;

• правила з лівого боку видаляються, тим самим повністю знімаючи з A граматики.

LALR парсери доволі ефективні під час пошуку єдино правильного розбору знизу догори в одноразовому скануванні вхідного потоку зліва-направо, тому що не потрібно використовувати відкати. Будучи попередньо оглядовим парсером за визначенням, LALR використовується найпоширеніше.

Prett Parsing (метод Вогана Пратта) – простий і ефективний метод розбору виразів, що не використовує ні автомати, ні граматику як таку. Ідея полягає в тому, що кожен символ наділяється властивостями:

• lbp = пріоритет зв'язування символу ліворуч;

• nud = функція, що визначає результат застосування оператора на початку виразу;

• led = функція, що визначає результат застосування в середині виразу.

Основний розбір здійснюється так. Виконується розбір (за пріоритетом продовження): з вхідного потоку виштовхується символ. Результату присвоюється виклик функції nud для цього символу. Поки пріоритет lbp наступного в потоці символу більший за пріоритет продовження, виштовхується символ з вхідного потоку і результату присвоюється використання функції led для цього символу до поточного результату.

Константи і змінні мають пріоритет зв’язування 0, а функція nud повертає їх значення. Тому застосування розбору до констант відразу поверне їх значення.

Для бінарних операторів функція led рекурсивно викликає продовження розбору (праворуч) аж до нижчого пріоритету, і робить що-небудь з вже накопиченим (ліворуч) результатом, отриманим рекурсивно.

Результат застосування оператора агрегується для зовнішнього виклику. Багатоарні оператори отримують аргументи додатковим викликом функції розбору. Префіксні оператори робляться за допомогою визначення для них функції nud. Для правостороннього зв'язування змінюється пріоритет продовження рекурсивного розбору.

Цей метод не залежить від контекстно-вільних граматик, а використовує для роботи зв’язуваність символів.

У табл. 1 подано узагальнену характеристику проаналізованих алгоритмів.

Таблиця 1

Характеристика обраних алгоритмів

	Earley парсер	LL парсер	Метод рекурсивного спуску	CKY парсер	LALR парсер	Prett парсер
Підтримка граматики	Всі	Всі	Всі	CNF	Всі	—
Метод синтаксичного розбору	Нисхідний	Нисхідний	Нисхідний	Висхідний	Висхідний	Нисхідний
Нарям проходу	Зліва направо	Зліва направо	Зліва направо	Справа наліво	Справа наліво	Зліва направо
Наявність повторень	Є	Є	Є	Є	—	Є

Порівняння роботи розглянутих алгоритмів

Для описаних вище алгоритмів було здійснено програмну реалізацію. В якості контрольного прикладу перевірки роботи алгоритму було обрано речення: «Parsing is the process of analysing a string of symbols». На рис. 1 зображено дерево парсингу для обраного речення.

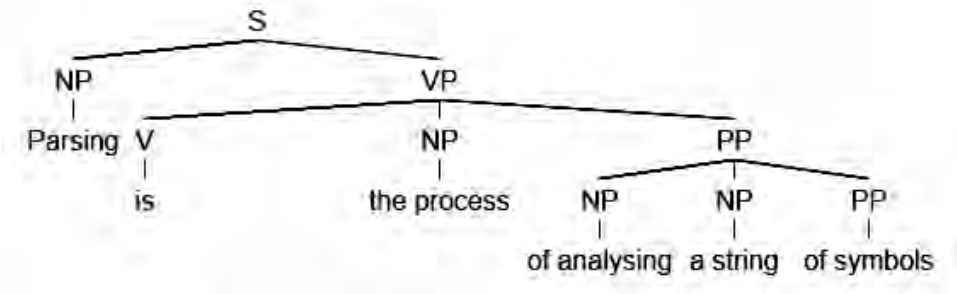

Рис. 1. Дерево синтаксичного аналізу для контрольного прикладу

Алгоритми порівнювались за двома критеріями: швидкодією опрацювання граматики та швидкодією опрацювання поданого на вхід тексту.

Кожен алгоритм був реалізований за принципом, зображеним у вигляді діаграми на рис. 2.

Рис. 2. Контекстна діаграма загального алгоритму парсингу

Порівняння характеристик виконання вищеописаних алгоритмів наведені в табл. 2.

Таблиця 2

Результати порівняння

Назва	Earley парсер	LL парсер	Метод рекурсивного спуску	CKY парсер	LALR парсер	Prett парсер
Опрацювання граматики, мс	59	4	36	20	19	46
Опрацювання тексту, мс	59	4	35	20	19	44

У дослідженні ми не будуємо повний граматичний аналіз, але тексти з відомою структурою аналізуються швидше. Набагато легше реалізувати синтаксичний аналіз в напівструктурованих текстах, які розділені на блоки [6]. Вони можуть бути організовані за допомогою особливих характеристик даних (metafeatures), які створюються з використанням вже отриманої інформації. Такі характеристики витягуються з вхідного документа і використовуються для ідентифікації інформації. Цей підхід може бути використаний для всіх видів інформації

Висновки та перспективи подальших наукових розвідок

Варто відзначити, що деякі алгоритми працюють швидше, ніж інші. Але водночас не всі алгоритми можуть працювати з усіма граматиками. Наприклад, CKY парсер швидший, ніж алгоритм Earley, але для CKY потрібно, щоб граматика була в CNF, а алгоритм Earley працює для будь-якої граматики. Вирішення цієї проблеми може полягати в удосконаленні існуючих алгоритмів або створенні нових алгоритмів шляхом їх комбінацій.

1. Aho A. V., Sethi R., Ullman J. D. (1986). Compilers: principles, techniques, and tools. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA. 2. Earley J. An efficient context-free parsing algorithm // ACM 13, 2 (Feb 1970). – P. 94–102. 3. Rosenkrantz D. J.; Stearns R. E. (1970). Properties of Deterministic Top Down Grammars. Information and Control 17: 226–256. 4. Sariel Har-Peled and Madhusudan Parthasarathy. Lecture 15: CYK Parsing Algorithm 3 March 2009. 5. DeRemer, Frank; Penello, Thomas (October 1982). "Efficient Computation of LALR(1) Look-Ahead Sets". Tansactions on Programming Languages and Systems (ACM) 4 (4): 615–649. Retrieved July 25, 2014. 6. Kluegl P., Atzmueller M., Puppe F. Meta-level Information Extraction. KI 2009: Advances in Artificial Intelligence. Lecture Notes in Computer Science Volume 5803, 2009. – P. 233–240. 7. Pratt Vaughan. Top down operator precedence. Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (1973).