Лекція №4 Основи лексичного аналізу.

Основне завдання лексичного аналізу - розбити вхідний текст, що складається з послідовності одиночних символів, на послідовність слів, або лексем, тобто виділити ці слова з безперервної послідовності символів. Всі символи вхідної послідовності з цієї точки зору поділяються на символи, що належать небудь лексемам, і символи, що розділяють лексеми (роздільники). У деяких випадках між лексемами може і не бути роздільників. З іншого боку, в деяких мовах лексеми можуть містити незначущі символи (наприклад, символ пробілу в Фортрані). У Сі розділову значення символів-роздільників може блокуватися («\» в кінці рядка всередині "...").

Зазвичай всі лексеми поділяються на класи. Прикладами таких класів є числа (цілі, вісімкові, шістнадцяткові, дійсні і т.д.), ідентифікатори, рядки. Окремо виділяються ключові слова та символи пунктуації (іноді їх називають символи-обмежувачі). Як правило, ключові слова - це деякий кінцеве підмножина ідентифікаторів. У деяких мовах (наприклад, ПЛ / 1) сенс лексеми може залежати від її контексту і неможливо провести лексичний аналіз у відриві від синтаксичного.

З точки зору подальших фаз аналізу лексичний аналізатор видає інформацію двох сортів: для синтаксичного аналізатора, працюючого слідом за лексичним, істотна інформація про послідовності класів лексем, обмежувачів і ключових слів, а для контекстного аналізу, працюючого слідом за синтаксичним, важлива інформація про конкретних значеннях окремих лексем (ідентифікаторів, чисел і т.д.).

Таким чином, загальна схема роботи лексичного аналізатора така. Спочатку виділяється окрема лексема (можливо, використовуючи символи-роздільники). Ключові слова розпізнаються або явним виділенням безпосередньо з тексту, або спочатку виділяється ідентифікатор, а потім робиться перевірка на приналежність його безлічі ключових слів.

Якщо виділена лексема є обмежувачем, то він (точніше, певний його ознака) видається як результат лексичного аналізу. Якщо виділена лексема є ключовим словом, то видається ознака відповідного ключового слова. Якщо виділена лексема є ідентифікатором - видається ознака ідентифікатора, а сам ідентифікатор зберігається окремо. Нарешті, якщо виділена лексема належить якому-небудь з інших класів лексем (наприклад, лексема являє собою число, рядок і т.д.), то видається ознака відповідного класу, а значення лексеми зберігається окремо.

Лексичний аналізатор може бути як самостійною фазою трансляції, так і підпрограмою, що працює за принципом «дай лексему». У першому випадку виходом аналізатора є файл лексем, у другому (рис. 3.1, б) лексема видається при кожному зверненні до аналізатора (при цьому, як правило, ознака класу лексеми повертається як результат функції «лексичний аналізатор» , а значення лексеми передається через глобальну змінну). З точки зору обробки значень лексем, аналізатор може або просто видавати значення кожної лексеми, і в цьому випадку побудова таблиць об'єктів (ідентифікаторів, рядків, чисел і т.д.) переноситься на більш пізні фази, або він може самостійно будувати таблиці об'єктів. У цьому випадку в якості значення лексеми видається покажчик на вхід у відповідну таблицю.

Робота лексичного аналізатора задається деяким кінцевим автоматом. Однак, безпосереднє опис кінцевого автомата незручно з практичної точки зору. Тому для завдання лексичного аналізатора, як правило, використовується або регулярний вираз, або праволінейная граматика. Всі три формалізму (кінцевих автоматів, регулярних виразів і праволінейних граматик) мають однакову виразну потужність. Зокрема, за регулярним виразом або праволінейной граматиці можна сконструювати кінцевий автомат, що розпізнає ту ж мову.

Регулярні множини і вирази

Введемо поняття регулярного безлічі, що грає важливу роль в теорії формальних мов.

Регулярне безліч в алфавіті Т визначається рекурсивно таким чином:

$\varnothing$ (Порожня множина) - регулярне безліч в алфавіті Т;

{e} - регулярне безліч в алфавіті Т (e - порожній ланцюжок);

{a} - регулярне безліч в алфавіті Т для кожного a $\in$ T;

якщо P і Q - регулярні множини в алфавіті Т, то регулярними є і безлічі

$P \cup Q$ (об'єднання),

PQ (конкатенація, тобто множину $\{pq \mid p \in P,\ q \in Q\}$),

P^* (ітерація: $P^* = \bigcup_{n=0} \infty P^n$);

ніщо інше не є регулярним безліччю в алфавіті Т.

Отже, безліч в алфавіті Т регулярно тоді і тільки тоді, коли воно або, $\varnothing$ або {e}, або {a} для деякого a $\in$ T, або його можна отримати з цих множин застосуванням кінцевого числа операцій об'єднання, конкатенації й ітерації.

Наведене вище визначення регулярного безлічі дозволяє ввести наступну зручну форму його записи, звану регулярним виразом.

Регулярний вираз у алфавіті Т і позначається їм регулярне безліч в алфавіті Т визначаються рекурсивно таким чином:

$\varnothing$ - Регулярний вираз, що позначає безліч;

e - регулярний вираз, що позначає множину {e};

a - регулярний вираз, що позначає множину {a};

якщо p і q - регулярні вирази, що позначають регулярні множини P і Q відповідно, то

(P | q) - регулярний вираз, що позначає регулярне безліч PQ, (Pq) - регулярний вираз, що позначає регулярне безліч PQ,

(P *) - регулярний вираз, що позначає регулярне безліч P *;

ніщо інше не є регулярним виразом в алфавіті T.

Ми будемо опускати зайві дужки в регулярних виразах, домовившись про те, що операція ітерації має найвищий пріоритет, потім йде операції конкатенації, нарешті, операція об'єднання має найменший пріоритет.

Крім того, ми будемо користуватися записом p + для позначення pp *. Таким чином, запис (a | ((ba) (a *))) еквівалентна a | ba +.

Нарешті, ми будемо використовувати запис L (r) для регулярного безлічі, позначуваного регулярним виразом r.

Приклад 3.1. Кілька прикладів регулярних виразів і позначаються ними регулярних множин:

a (e | a) | b - позначає множину {a, b, aa};

a (a | b) * - позначає безліч всіляких ланцюжків, що складаються з a і b, що починаються з a;

(a | b) * (a | b) (a | b) * - позначає множину всіх непорожніх ланцюжків, що складаються з a і b, тобто множину {a, b} +;

((0 | 1) (0 | 1) (0 | 1)) * - позначає безліч всіх ланцюжків, що складаються з нулів і одиниць, довжини яких діляться на 3.

Ясно, що для кожного регулярного безлічі можна знайти регулярний вираз, що позначає це безліч, і навпаки. Більш того, для кожного регулярного безлічі існує нескінченно багато позначають його регулярних виразів.

Будемо говорити, що регулярні вирази дорівнюють або еквівалентні (=), якщо вони позначають одне і те ж регулярне безліч.

Існує ряд алгебраїчних законів, що дозволяють здійснювати еквівалентну перетворення регулярних виразів.

Лемма. Нехай p, q і r - регулярні вирази. Тоді справедливі наступні співвідношення:

(1) $p\|q = q\|p$;	(7) $pe = ep = p$;
(2) $\varnothing^* = e$;	(8) $\varnothing p = p\varnothing = \varnothing$;
(3) $p\|(q\|r) = (p\|q)\|r$;	(9) $p^* = p\|p^*$;
(4) $p(qr) = (pq)r$;	(10) $(p^*)^* = p^*$;
(5) $p(q\|r) = pq\|pr$;	(11) $p\|p = p$;
(6) $(p\|q)r = pr\|qr$;	(12) $p\|\varnothing = p$.

Слідство. Для будь-якого регулярного виразу існує еквівалентне регулярний вираз ∅, яке або є, або не містить у своєму записі ∅.

Надалі будемо розглядати тільки регулярні вирази, що не містять у своєму записі ∅.

При практичному описі лексичних структур буває корисно зіставляти регулярними виразами деякі імена, і посилатися на них по цим іменам. Для визначення таких імен ми будемо використовувати запис вигляду

$d_1 = r_1$

$d_2 = r_2$

...

$d_n = r_n$

де d_i - різні імена, а кожне ri - регулярний вираз над символами $T \cup \{d_1, d_2, ..., d_{i-1}\}$, тобто символами основного алфавіту та раніше певними символами (іменами). Таким чином, для будь-якого r_i можна побудувати регулярний вираз над T, повторно замінюючи імена регулярних виразів на позначаються ними регулярні вирази.

Приклад 3.2. Використання імен для регулярних виразів.

Регулярний вираз для безлічі ідентифікаторів.

Letter = a | b | c | ... | x | y | z

Digit = 0 | 1 | ... | 9

Identifier = Letter (Letter | Digit) *

Регулярний вираз для безлічі чисел у десятковому запису.

Digit = 0 | 1 | ... | 9

Integer = Digit +

Fraction =. Integer | e

Exponent = (E (+ | - | e) Integer) | e

Number = Integer Fraction Exponent