УДК 004.415.2.043

САЛАПАТОВ В.І.

СИНТАКСИЧНИЙ АНАЛІЗ ІЗ РОЗПОДІЛОМ ЛЕКСЕМ НА ГРУПИ

Традиційний розбір операторів мов програмування із розбиттям їх на окремі лексеми як рівноправні одиниці мови достатньо ускладнює синтаксичний аналіз. Природні мови спілкування мають при їхньому сприйнятті відокремлення підмету, присудку та інших частин мови для усвідомлення сенсу речення. Дещо подібне пропонується на етапі лексичного розбору операторів мов програмування. Всі лексеми при цьому розбиваються на три групи: лексеми-об'єкти, лексеми дії та інші. Особливість мов програмування полягає у обов'язковій наявності пари лексем: лексеми-об'єкту та лексеми дії. Причому з цієї пари починаються всі оператори і обов'язковим елементом у цій парі завжди є лексема дії. Тому, якщо при лексичному аналізі з'являється лексема-об'єкт, то доречним є одночасний пошук відповідної лексеми дії. Такий підхід дозволяє значно спростити синтаксичний аналіз операторів мови та прискорити його виконання.

The traditional analysis of program language operators if with laying out of them on separate lexemes as equal in rights units of language complicates enough the syntactic analysis. The natural languages of intercourse are had at their perception of separation to the subject, to the predicate and other parts of language for the awareness of sense of suggestion. Something similar is offered on the stage of lexical analysis of program language operators. All lexemes are here divided into three groups: lexemes-objects, lexemes of action at al. A feature as programming if consists in the obligatory presence of pair of lexemes: lexeme-object and lexeme of action. Thus from this pair all operators begin and in this pair it is always the lexeme of action is a mandatory member. Therefore, if a lexeme-object appear at the lexical analysis, the simultaneous search of the proper lexeme of action is appropriate. Such approach allows considerably simplify the syntactic analysis of operators of language and accelelerate his implementation.

Ефективність засобів компіляції

Потреба в нових мовах програмування та створенні для них систем компіляції ставить задачу щодо їхньої ефективної та швидкої розробки. Сучасні системи компіляції ще й досі породжують надто надмірні коди програм. Тому методи створення ефективних систем компіляції містять великий резерв щодо підвищення продуктивності роботи комп'ютерів і на даний час залишаються вельми актуальними. Важливим компонентом систем компіляції є синтаксичний аналізатор, який перетворює текст програми у внутрішнє уявлення. Математичною моделлю аналізатора є граматика мови програмування. Контекстно вільна граматика описує всі оператори мови і складається з низки правил, у правій частині яких можуть бути присутніми нетермінали. У мовах програмування такими нетерміналами є такі поняття, як вираз, оператор або інструкція та окремі фрази від часток операторів.

Створення сучасних синтаксичних аналізаторів пов'язано з двома різновидами аналізу – згори донизу та знизу догори, а точніше їхніми різновидами LL-аналізу та LR-аналізу [1, 2]. Останні два алгоритми передбачають посимвольний перегляд тексту програми з кроком уперед. При цьому всі елементи оператора мають оброблятися за єдиним алгоритмом. Це вносить певні складнощі, оскільки нетермінали, які мають бути присутніми у певних місцях правил граматики, описуються за своїми власними правилами. Ці правила мають вигляд $G = f(T, N, P, N_s)$, де N_s – стартовий символ (спеціальний нетермінал), який визначає тип оператора. T являє собою множину терміналів (в мові програмування це константи, ідентифікатори, ключові слова, символи пунктуації і т.і.). N – це нетермінали (в нашому випадку такі поняття, як вирази, оператори та окремі частки операторів). В зв'язку з цим доречно розглядати граматику кожного нетерміналу та його обробку окремо. Так, якщо згідно з граматикою у певному місці оператора має бути вираз, то обробку оператора мови в цій частині треба виконувати згідно граматики виразу, тобто окремою підпрограмою. При цьому ознакою кінця виразу є або ключове слово з конструкції оператора, або спеціальні знаки-розподільники.

Поєднання в одному аналізі розбору оператора, розбору виразу та інших частин оператора ускладнює та уповільнює процес компіляції в цілому. Тому розробка нових ефективних засо-

бів синтаксичного аналізу у мовах програмування на даний час є все ще актуальною.

Огляд методів синтаксичного аналізу

З моменту появи компіляторів і до даного часу алгоритми синтаксичного аналізу майже не змінились. При цьому всі лексеми мови розбираються однаково згідно одного з обраних алгоритмів [1, 2, 3]. Але одні лексеми вимагають виконання певних дій (оператори, знаки арифметичних операцій, фразові ключові слова), а інші являють собою дані, над якими виконуються дії. Синтаксичний розбір операторів за конкретним алгоритмом згідно граматики кожного разу вносить певні корективи у всі попередні алгоритми синтаксичного розбору. Тому кожного разу, при створенні нової мови програмування і, відповідно, нового компілятора приходиться відтворювати такий алгоритм заново. Хоча на даний час створено багато засобів по створенню систем компіляції (компілятори компіляторів) [2], але за своєю ефективністю вони значно поступаються тим, що заново створюються. На жаль, в цьому компоненті за останні роки майже нічого не змінилось. Тому виникла нагальна потреба створити такий алгоритм аналізатора, який складався б із загальної частини, яка присутня в усіх компіляторах та спеціальної частини, де б можна було врахувати особливості конкретного компілятора. До загальної частини можна віднести обробку ключових слів, констант, змінних, знаків пунктуації та арифметичних дій. Задача полягає у розробці такого універсального алгоритму синтаксичного аналізатора, який би мав можливість налаштовуватись на конкретний алгоритм згідно граматики мови та не вносив би надмірність у машинний код.

Розподіл лексем

Огляд існуючих мов програмування показав, що всі лексеми, які являють собою термінальні символи, можна розділити на дві великі групи: лексеми-об’єкти та лексеми дії. Перші описують данні, а другі – дії над цими даними так само, як це має місце у різних системах: алгебрі, геометрії, базах даних і т.і. Таким чином, $L \rightarrow L_d + L_a$, де L – множина всіх лексем, L_d – лексеми даних, L_a – лексеми дії. Припустимість тих чи інших дій задається правилами граматики. Лексеми, які визначаються за допомогою символів, можуть бути визначені через таблиці символів, ключові слова – за допомогою таблиці ключових слів. У правій частині правил граматики для операторів мов програмування присутні нетермінальні символи, які, у свою чергу, можуть бути описані окремими правилами. До таких нетерміналів відносяться вирази та оператори мови. Вирази описуються граматикою, яка співпадає з правилами арифметичний дій, тобто існує пріоритетність виконання операторів, урахування дужок, які змінюють пріоритетність дій, та деякі інші. Вирази є одним з основних компонентів мов програмування. В залежності від типу результату у виразах припускаються дані певного типу та відповідно дії, що припустимі для обробки цих даних. Те саме виникає і при обробці операторів. За ключовими словами операторів згідно граматик розміщуються вирази. При цьому ознаками кінця виразу можуть бути як ключові слова операторів, так і спеціальні символи.

Лексеми даних – L_d обробляються лексичним аналізатором під час їх появи. Тип результату визначається припустимими терміналами дії – L_a. Ці лексеми визначають порядок їхньої обробки згідно пріоритетів. Це в значній мірі спрощує опис правил граматики. Ключові слова операторів обробляються в останню чергу, тому вони мають найнижчий пріоритет. Результатом їх обробки є породження гілок дерева поточного оператора. Треба зазначити, що даний алгоритм аналізу є модифікацією алгоритму “знизу догори”, тобто від кінцівок дерева до його кореня. Стартовий символ N_s, тобто тип оператора мови, розпізнається через першу або другу лексему, за один крок роботи лексичного аналізатора, про що йдеться далі.

Модифікований стековий алгоритм

У виразах усі дії, за деяким виключенням, є бінарними, тобто вони вимагають наявності двох операндів. Через це доцільно розпізнавати пари лексем у такому порядку: лексема-об’єкт, лексема дії. Лексема-об’єкт у нашому випадку являє собою дані (для виразів це константи, змінні, масиви), а лексема дії вказує на обробку даних. Опис лексем-об’єктів у мовах програмування в основному стандартний, відмінності пов’язані, головним чином, з ідентифікаторами. Лексеми дії також мають стандартний за семантикою набір, що присутній в усіх мовах програмування, та спеціальний набір тільки для конкретної мови.

Унарними є такі дії, як унарний мінус, дужки та деякі інші. Для унарного мінуса можна ство-

рити фіктивний операнд – нуль, а для дужок – пустий операнд.

Пріоритетність дій у виразах є найбільш зрозумілою і визначає порядок виконання цих дій. Якщо пріоритет поточної лексеми дії не перевищує пріоритет попередньої, то виконується обробка попередньої лексеми дії. Інакше обробка відкладається у стек. Таким чином, маємо порядок обробки лексем дії, що представлений нижче у табл. 1.

Табл. 1

Порівняння пріоритету поточної операції з попередньою, перевірка стеку	**Дія**
не перевищує	обробка попередньої операції з поточною лексемою-об’єктом
перевищує	запис у стек попередньої пари: лексема-об’єкт, лексема дії
стек непустий, права дужка або кінець виразу	читання зі стеку пари лексем та їх обробка
стек пустий, кінець виразу	кінець обробки виразу

У цьому алгоритмі необхідно передбачити змінні під *попередню* лексему-об’єкт та лексему дії, а також під *поточну* лексему-об’єкт та лексему дії. В разі запису в стек чергової пари лексем змінні з *попередніх* лексеми-об’єкту та лексеми дії записуються в стек, на їхнє місце переписуються відповідні *поточні* лексеми. Таким чином вивільняються місця під чергові нові *поточні* лексеми з виразу при продовженні обробки виразу.

В разі запису в стек лівої дужки вона записується в стек з “пустим” операндом і продовжується обробка виразу за даним алгоритмом пріоритету. Коли зустрічається права дужка, здійснюється так би мовити зворотний хід у стеку, тобто обробка лексем у стеку з наступним їх виштовхуванням звідти. Ознакою кінця виразу є або спеціальний знак, або ключове слово відповідного оператора згідно його граматики. Кінцем обробки виразу є наявність пустого стеку та досягнення в обробці виразу кінця оператора. Обробка передбачає побудову гілок синтаксичного дерева розбору оператора.

Відсутність паритету дужок одразу ж виявляється через стек як помилка. В разі невідповідності типів операндів стандартний алгоритм має передбачати в разі необхідності перетворення типів (за допомогою спеціальних підпрограм). Тип результату виразу має відповідати опису оператора.

Якщо за основу взяти цей алгоритм, то опис граматики зводиться до опису ключових слів (стандартних та нестандартних) та їхніх семантик як стандартних, так і нестандартних.

Лексеми дії можуть являти собою як спеціальні символи, так і ключові слова, але в обох випадках їхній код складається з коду пріоритету та власно коду лексеми дії, що породжує певний тип обробки.

Ключові слова операторів мови та окремих фраз операторів повинні також визначати пріоритет обробки та відповідну семантику. Так, наприклад семантика ключового слова IF умовного оператора породжує команди порівняння та відповідного умовного переходу.

Всі ключові слова можна звести в одну таблицю та застосувати їх кодування, щоб відрізнити лексеми дії виразу від лексем дії операторів.

Синтаксичний аналіз операторів з таким розподілом лексем доречно розбити на дві частини: обробку виразів та обробку операторів. При цьому обробку виразів необхідно виділити в окрему підпрограму, що породжує дерево розбору саме виразу, а фразові ключові слова операторів мови обробляти окремо. Такий підхід дозволяє достатньо легко вносити зміни синтаксису у мови програмування. Причому окремо в обробку виразів і окремо в обробку операторів.

Синтаксис лексем-об’єктів може відрізнятись в різних мовах і охоплює ідентифікатори, константи (цілого, дійсного, символьного та інших типів) і визначається у кожному випадку окремо. Їхній опис не складає великих труднощів щодо конкретної мови.

Лексеми дії крім стандартних дій повинні передбачати ще й нестандартні згідно правил граматики. Визначення тієї чи іншої дії легко задається в таблиці ключових слів спеціальним кодом, що являє собою посилання і забезпечує перехід на відповідну частину програми синтаксичного аналізатора. Структура елементу таблиці ключових слів має наступний вигляд.

мнемоніка MNEM	тип нетерміналу TYP	код CODE	пріоритет PRIOR

Рис. 1

Такий термінал розпізнається шляхом пошуку у таблиці ключових слів за мнемонікою MNEM.Тип нетерміналу TYP визначає, до якого типу нетерміналу відноситься дана лексема –

до виразу чи до оператора. Це значною мірою спрощує перевірку правильності синтаксису оператора і дозволяє швидко виявити синтаксичні помилки. Код CODE разом з пріоритетом PRIOR визначають послідовність обробки та власно перехід на відповідний алгоритм обробки згідно коду. Таким чином, визначаючи нову лексему дії як термінал, для її обробки необхідно додати відповідний програмний фрагмент.

До стандартних дій по обробці операторів таких, як умовний оператор (IF), оператори циклу (FOR, WHILE, REPEAT), оператори введення/виведення (READ, WRITE) [3, 4] можна додавати й нестандартні оператори (наприклад побітової обробки). Для цього необхідно визначити правило обробки та додати підпрограми, які забезпечують синтаксичний розбір таких нестандартних операторів. Звернення до цих операторів забезпечується через посилання, яке подано як код CODE у таблиці ключових слів.

Ключові слова в синтаксисі операторів визначають їхню послідовність в тій чи іншій конструкції оператора мови і їхня обробка виконується у самому кінці, тобто з найнижчим пріоритетом. Як бачимо, пріоритет використовується, в основному, при обробці виразів і досить зручно вписується у загальний алгоритм синтаксичного аналізу. Причому ключові слова операторів у нашому випадку використовуються також як ознаки кінця виразів у мовних конструкціях.

Запропонований алгоритм синтаксичного розбору покладає на лексичний аналізатор розпізнавання не однієї лексеми, а пари лексем: лексеми-об’єкту та лексеми дії і повертати їх синтаксичному аналізатору для подальшого розбору. При цьому на першому кроці аналізу оператора лексичний аналізатор розпізнає стартовий символ N_s – тип оператора мови та подальшій алгоритм аналізу згідно правил граматики.

За даним алгоритмом синтаксичного аналізу було створено кросс-компілятор мови С для мікроконтролера AVR, в якому реалізовані методи машинно-залежної оптимізації коду програм.

Висновки

Запропонований алгоритм синтаксичного розбору дозволяє легко вносити корективи у граматику і, відповідно, у алгоритми обробки виразів та як власно операторів, так і сполучених операторів. За допомогою коду лексеми забезпечується посилання на підпрограму обробки, а за допомогою її пріоритету – порядок обробки у синтаксичному у дереві розбору. Оскільки аналіз здійснюється через розбір пари лексем, то швидкість обробки операторів вища, ніж за класичними алгоритмами через направлений перебір варіантів. При цьому дещо ускладнюється алгоритм лексичного аналізатора, що суттєво не впливає на загальну швидкість синтаксичного аналізу.

Другою перевагою даного методу є простота опису правил синтаксичного розбору операторів та виразів.

Перелік посилань

1. Хопкрофт Джон Э., Мотвани Раджив, Ульман Джефри Д. Введение в теорию автоматов, языков и вычислений. – М.: Издательский дом “Вильямс”, 2002. – 528 с.
2. Ахо Альфред В., Сети Равви, Ульман Джефри Д. Компиляторы: принципы, технологии и инструменты. – М.: Издательский дом “Вильямс”, 2003. – 768 с.
3. Себеста Роберт У. Основные концепции языков программирования. – М.: Издательский дом “Вильямс”, 2001.– 672с.
4. Бен Ари М. Языки программирования. Практический сравнительный анализ. – М; Мир, 2000. – 366 с.