COMPUTER SYSTEMS AND NETWORKS

Vol. 7, No. 1, 2025

DOI: https://doi.org/10.23939/csn2025.01.015

УДК 621.3

МЕХАНІЗМ РОЗШИРЕННЯ СИНТАКСИСУ МОВИ ПРОГРАМУВАННЯ НА ЕТАПІ КОМПІЛЯЦІЇ

Ю. В. Берегуляк, Р. В. Бачинський

Національний університет "Львівська політехніка",
кафедра електронних обчислювальних машин
E-mail: yurii.v.berehuliak@lpnu.ua, ruslan.v.bachynskyi@lpnu.ua

У статті досліджується проблема гнучкості граматик з фіксованим набором правил, що обмежує можливості інтеграції спеціалізованих синтаксичних конструкцій. Запропоновано модель розширення таких граматик, яка дає змогу модульно доповнювати базову систему новими правилами без порушення її цілісності. Завдяки такому підходу синтаксична система набуває здатності адаптуватися до нових вимог, забезпечуючи автоматичне розширення формальних описів.

Розроблено механізм інтеграції нових синтаксичних конструкцій, які ґрунтуються на перевірці їх узгодженості за допомогою аналізу FIRST та FOLLOW множин. Крім того, застосовано методи усунення неоднозначностей із використанням поточного контексту розбору, що дає змогу автоматично визначати оптимальні правила для побудови абстрактного синтаксичного дерева. Такий підхід гарантує однозначність та коректність синтаксичного аналізу, адже навіть у випадках потенційних конфліктів система, враховуючи локальні семантичні та синтаксичні особливості, усуває неоднозначність.

Цей підхід демонструє високу адаптивність і модульність механізму розширення граматики, що має значне практичне значення для подальшої еволюції компіляційних систем та створення спеціалізованих діалектів. Запропонована методологія відкриває нові перспективи для розробки гнучких і масштабованих систем синтаксичного аналізу, здатних оперативно реагувати на зміни у вимогах сучасного програмування.

Ключові слова: архітектура компілятора, граматика, лексичний аналіз, синтаксичний аналіз.

Вступ

Сучасні мови програмування використовують граматику, яка є попередньо визначеною і незмінною. Це не дає змоги легко впроваджувати нові синтаксичні конструкції. Основною проблемою є те, що граматика з фіксованим набором правил не дозволяє швидко адаптувати її до змінних вимог сучасного програмування, що знижує ефективність розробки програмного забезпечення. Зростає потреба у підвищенні зручності розробки, зменшенні когнітивного навантаження на розробників у спеціалізованих сферах, кількості помилок та покращенні якості кінцевих продуктів. Запропонований підхід до використання граматики зі змінним набором правил, який дає змогу розробнику "на льоту" додавати нові правила без зміни механізму аналізу коду, є перспективним рішенням для подолання цих обмежень.

Актуальність дослідження визначається необхідністю адаптації мов програмування до сучасних вимог індустрії та наукової спільноти. Граматика зі змінним набором правил відкриває можливості для створення спеціалізованих діалектів, що враховують специфіку конкретних завдань

і доменів застосування, сприяючи підвищенню продуктивності та якості розробки. У статті розглядаються формальні аспекти інтеграції нових синтаксичних конструкцій, аналізуються сучасні методи лексичного та синтаксичного аналізу, компіляції, що дає змогу встановити принципи розширення правил граматики динамічно. Отже, дослідження сприяє розвитку теоретичних основ мов програмування та практичній реалізації інноваційних технологій у сфері розробки програмного забезпечення.

Огляд літературних джерел

Одним із ранніх та фундаментальних підходів до побудови компіляторів є використання атрибутивних граматик. Автори [1] запропонували модель побудови компіляторів на основі атрибутивних граматик, що дає змогу визначати синтаксичний та семантичний аналіз в єдиній формальній системі. Вони демонструють, як атрибутивні граматики можуть бути ефективно використані для генерації компіляторів із синтаксичним та семантичним аналізом в одному механізмі. Ще одним фундаментальним питанням є визначення пріоритетів у специфікаціях мов програмування. Автор [2] розглянув, як пріоритети операцій можуть бути визначені та реалізовані в мовах програмування, що впливає на спрощення побудови парсерів. У книзі [3], яка є одним із найбільш впливових підручників у цій галузі, охоплено всі аспекти розробки компіляторів, зокрема лексичний аналіз, синтаксичний аналіз, оптимізацію та генерацію коду. Видання 2006 року містить оновлені розділи про сучасні техніки компіляції та оптимізації коду.

Значний прогрес у створенні компіляторів став можливим завдяки розробці автоматизованих інструментів. Автор [4] запропонував методологію побудови компіляторів за допомогою сучасних (на той час) інструментів, що дають змогу значно скоротити час розробки. Дослідники [5] представили набір інструментів для навчання створенню компіляторів, що охоплює автоматизовані засоби аналізу та трансляції мов програмування. Це дослідження має значення як для освітньої сфери, так і для розробників, які прагнуть освоїти основи компіляції. Ще одним підходом є методологія “навчання через приклади”. У публікації [6] показано практичні аспекти побудови компіляторів.

З розвитком багатоплатформних середовищ з’явилася потреба в ефективних методах трансляції коду для різних архітектур. Автори [7] запропонували техніки розробки компіляторів для підтримки багатоцільової генерації коду, що забезпечує адаптацію до різних архітектур.

У книзі [8] розглянуто сучасні методи розробки компіляторів, зокрема підтримку проміжного представлення, що дає змогу створювати більш ефективні компілятори. Автори [9] запропонували модульний підхід до створення компіляторів, який дозволяє розділити процес трансляції на окремі частини з використанням проміжного представлення.

Значна увага приділяється оптимізації продуктивності компіляторів. У [10] досліджено розробку та продуктивність ELI-to-C компілятора, проаналізовано його ефективність у генерації коду та зменшенні витрат на трансляцію.

Дослідження [1–10] демонструють, що інтеграція адаптивних механізмів у компілятори стають ключовими факторами еволюції мов програмування, а базові принципи, викладені у класичних працях, забезпечують міцну основу для розробки та впровадження нових технологій. Це все у сукупності дозволяють архітектурі компіляторів ефективно реагувати на виклики сучасної індустрії інформаційних технологій.

Постановка задачі

У сучасних мовах програмування їх стала граматика обмежує можливості адаптації до нових вимог та впровадження сучасних парадигм без суттєвих змін у базовій архітектурі компілятора. Основною проблемою є відсутність гнучкого механізму для її розширення, що унеможливлює інтеграцію нових синтаксичних конструкцій “на льоту” без перебудови або модифікації лексичного та синтаксичного аналізатора компілятора.

Отже, завдання дослідження полягає у розробці формальної моделі, яка дасть змогу інтегрувати нові граматичні правила під час процесу компіляції, зберігаючи коректність синтаксичного та семантичного аналізу, коректної побудови гілки абстрактного синтаксичного дерева (АСД), яка відповідає новому правилу, а також забезпечує сумісність із базовою граматикою. Серед основних характеристик, що мають бути покращені, є продуктивність компіляції, зниження складності розробки, когнітивного навантаження на розробників, а також можливість створення спеціалізованих діалектів мови для спеціалізованих напрямів.

Метою цієї роботи є формальне обґрунтування механізму розширення граматичних правил мов програмування "на льоту", що дає змогу інтегрувати нові синтаксичні конструкції без зміни базового компілятора. Досягнення цієї мети сприятиме створенню гнучких мовних середовищ розробки, здатних адаптуватися до швидкоплинних вимог сучасної індустрії програмування.

Для реалізації поставленої мети необхідно вирішити такі завдання:

· Опис моделі базової граматики. Необхідно описати формальну модель базової граматики сучасних мов програмування, яка слугуватиме фундаментом для подальших розширень.

· Розробка моделі динамічного розширення. Потрібно створити методологію додавання нових граматичних правил до базової моделі, що забезпечуватиме їх узгодженість із чинними правилами, а також гарантуватиме відсутність неоднозначностей та синтаксичних конфліктів.

· Перевірка узгодженості та коректності. Важливо розробити автоматизовані процедури перевірки нових правил, що додаються до базової граматики, для забезпечення їхньої семантичної цілісності та коректності розбору.

· Розробка алгоритму багатопрохідного синтаксичного аналізу. Передбачається розробка алгоритму багатопрохідного аналізатора, з урахуванням потенційних залежностей між правилами.

· Емпірична оцінка ефективності. Необхідно теоретично визначити оцінки впливу запропонованого підходу на продуктивність компіляції.

Успішне вирішення цих завдань дасть змогу створити більш адаптивні та ефективні системи, здатні оперативно реагувати на потреби сучасних розробників, знижуючи кількість необхідних операцій для внесення змін у синтаксис мови та оптимізуючи процес розробки програмного забезпечення.

Базова модель граматики

Базова модель граматики є фундаментальною складовою для подальшої розробки та інтеграції нових синтаксичних конструкцій. Розглянемо базову граматику мови програмування як

$$G_0 = (N_0, T_0, R_0, S_0), \quad (1)$$

N_0 - це множина базових нетермінальних символів, які представляють логічні категорії або синтаксичні конструкції мови (наприклад, вираз, оператор, оголошення). Ці символи формують "скелет" граматики, завдяки якому визначається структура програмного коду; T_0 - множина термінальних символів, які є найменшими одиницями лексичного аналізу. До них належать ключові слова, літерали, знаки пунктуації, операції тощо. Термінали задають "будівельні блоки" мови, на основі яких створюються синтаксичні конструкції; R_0 - множина правил продукції, що описують, як нетермінальні символи можуть замінюватися послідовностями термінальних і/або нетермінальних символів. Ці правила формалізують синтаксичну структуру мови, використовуючи стандартні нотації, такі як BNF (Backus–Naur Form) або її розширена форма EBNF (Extended Backus–Naur Form). Наприклад, правило може виглядати так:

$$\langle Expr \rangle ::= \langle Term \rangle \mid \langle Expr \rangle" + "\langle Term \rangle \quad (2)$$

S_0 - стартовий символ, що задає початкову точку виведення для побудови синтаксичного дерева. Він визначає, яке правило або набір правил є базовим для аналізу всієї програми.

Базова модель G_0 є абстракцією, що дає змогу формально охарактеризувати множину синтаксично коректних конструкцій, прийнятних у межах заданої системи. Абстрактне визначення

множини синтаксично коректних рядків (позначається як $L(r)$ для правила r) дає змогу здійснити аналіз однозначності граматики. Якщо для кожного правила r множина $L(r)$ не має спільних елементів з множинами, що генеруються іншими правилами, це забезпечує відсутність синтаксичних неоднозначностей. Пряме перерахування всіх елементів $L(r)$ може бути неможливим або неефективним, тому замість цього використовуються методи аналізу, зокрема обчислення $FIRST$ та $FOLLOW$ множин.

Формальна модель G_0 не лише встановлює базову синтаксичну структуру, а й слугує відправною точкою для подальших розширень. Будь-яке оновлення або доповнення граматики має зберігати ключові властивості G_0: однозначність, сумісність з уже визначеними синтаксичними конструкціями. Саме тому детальне формулювання G_0 має критичне значення для побудови гнучкої системи.

Механізм розширення граматики

Нехай після i-го етапу розширення базова граматика набуває вигляду:

$$G_i = (N_i, T_i, R_i, S_0). \quad (3)$$

Нові синтаксичні конструкції вводяться шляхом додавання нових елементів у відповідні множини, що визначають граматику, зберігаючи початковий стартовий символ S_0. Формально, розширення до G_{i+1} записується як:

$$G_{i+1} = \left(N_i \cup N_{i,etx}, T_i \cup T_{i,etx}, R_i \cup R_{i,etx}, S_0\right), \quad (4)$$

$N_{i,etx}$ - множина нових нетермінальних символів, що вводяться розширенням. Вона містить символи, які не належали початковій граматиці N0 і потрібні для опису нових конструкцій. Наприклад, для лямбда-виразів може бути введено новий нетермінал <LambdaExpr>; $T_{i,etx}$ - множина нових термінальних символів, які використовуються виключно для розширень. Це можуть бути нові ключові слова або спеціальні символи, що відокремлюють розширення від звичних конструкцій; $R_{i,etx}$ - множина правил продукції, що описують нові синтаксичні конструкції. Кожне правило $r \in R_{i,etx}$ має своє формальне визначення, яке задає, як новий нетермінальний символ може бути замінений послідовністю термінальних і/або нетермінальних символів.

Цей механізм розширення дає змогу поступово розширювати наявну граматику, додаючи нові можливості без зміни фундаментальної структури мови. Це досягається завдяки збереженню стартового символу S_0 і об'єднанню нових правил із базовими. Важливо, щоб інтеграція відбувалася у такий спосіб, що нові правила не створюють синтаксичних колізій з уже наявними. Для цього використовується умова яка гарантує, що мови (множини рядків), породжені різними правилами, не перекриваються, що дає змогу уникнути неоднозначності під час синтаксичного аналізу:

$$\forall r, r` \in R_0 \cup R_{i,etx}\text{, якщо } r \neq r`\text{, то } L(r) \cap L(r`) = \varnothing\,. \quad (5)$$

Механізм побудови АСД

Кожне нове правило $r \in R_{ext}$ супроводжується відповідною трансформаційною функцією, що генерує фрагмент абстрактного синтаксичного дерева (АСД). Позначимо через ψ функцію трансформації, яка відображає кожне нове правило rRext у відповідний фрагмент АСД:

$$\psi_r : L(r) \to T_r, \quad (6)$$

де $L(r)$ позначає множину лексичних конструкцій, що задовольняють правило r, а T_r - підмножину абстрактних синтаксичних дерев, яка представляє розширену конструкцію. Ця множина функцій (для кожного правила своя) забезпечує відображення формальних правил розширення у семантично значущі елементи АСД.

Нехай

$$P: I \times G_1 \rightarrow AST \tag{7}$$

це відображення, яке, враховуючи вхідний рядок I і оновлену граматику G_1, генерує повноцінне АСД. Інакше кажучи, P є алгоритмічним процесом (наприклад, парсером), який перетворює послідовність символів із вхідного потоку I на дерево, що відображає синтаксичну структуру згідно з правилами оновленої граматики G_1. Коли під час розбору застосовується правило $r \in R_{ext}$, його відповідний фрагмент АСД визначається як:

$$AST_{fragment} = \psi(r). \tag{8}$$

Це означає, що у разі розбору в певному місці вхідного потоку, якщо обрана альтернативна конструкція відповідає правилу $r \in R_{ext}$, то відповідний фрагмент АСД формується шляхом застосування функції ψ до цього правила. Отже, нова синтаксична конструкція, визначена розширенням, має своє чітке відображення у структуру в АСД, що дає змогу потім її обробляти, оптимізувати або транслювати у відповідний код. Це забезпечує модульність і розширюваність системи, адже кожне правило розширення має чітко визначене представлення у фінальному синтаксичному дереві.

Результуючий фрагмент вставляється у загальне АСД як піддерево, відповідно до позиції, визначеної синтаксичними правилами базової граматики. Припустимо, що базове правило для виразів визначене як:

$$\langle Expr \rangle ::= \langle SExpr \rangle \mid \langle Extension \rangle \mid \cdots , \tag{9}$$

де $\langle Extension \rangle$ представляє будь-яке нове правило з $r \in R_{ext}$. Під час застосування цього правила, якщо розбір обирає альтернативу, що відповідає новому правилу r, то АСД для цієї конструкції формується за схемою:

$$AST = Merge\left(AST_{prev}, \psi(r)\right), \tag{10}$$

де AST_{prev} - частина абстрактного синтаксичного дерева, що вже була побудована до цієї точки, а $Merge$ - операція інтеграції нового фрагмента $\psi(r)$ у загальну структуру АСД згідно з правилами конкатенації ієрархічних дерев.

Перевірка узгодженості правил

Ключовим аспектом інтеграції нових правил є перевірка їхньої сумісності з базовою граматикою. Для цього пропонується застосувати формальні методи аналізу, які базуються на таких положеннях:

· Синтаксична узгодженість. Для кожного нового правила $r \in R_{ext}$ має виконуватися умова, що не існує іншого правила $r` \in R_0 \cup R_{etx}$, з яким r має неоднозначне перетинання. Ця умова перевірятиметься перед додаванням нового розширення. Формально ця умова *(5)*.

· Семантична узгодженість. Інтегровані розширення не порушують семантичних інваріантів, визначених базовою граматикою, і правильно відображають змістовну інформацію під час формування AST.

Для забезпечення сумісності нових правил із базовою граматикою проводиться формальна перевірка узгодженості. Основна ідея полягає в аналізі множин $L(r)$ для кожного правила r із загальної множини $R_{total} = R_0 \cup R_{ext}$.

Припустимо, що для кожного правила r ми можемо обчислити множину $FIRST$, що містить усі термінальні символи, які можуть з'явитися на початку рядка, породженого r. Аналогічно множина $FOLLOW$ містить символи, які можуть іти після нетермінального, що з'являється у r. Визначення $FIRST$ та $FOLLOW$ дає змогу описати умови відсутності перетину:

· Якщо для двох правил r і r' їхні множини $FIRST$ перетинаються, існує ризик неоднозначності.

· Якщо правило може породжувати порожню послідовність ($Nullable$), додатково враховується перетин $FOLLOW$ із $FIRST$ іншого правила.

Отже, для кожного $r \in R_{total}$ необхідно, щоб:

$\forall r` \in (R_{total} \backslash \{r\})$, якщо $FIRST(r) \cap FIRST(r`) \neq \emptyset$, то додатково $FOLLOW(r) \cap FIRST(r`) = \emptyset$.(11)

Це гарантує, що в жодному моменті аналізу не виникне ситуація, коли один і той самий фрагмент коду може бути витлумачений різними способами, не беручи поточний контекст до уваги.

Усунення неоднозначностей

Іншим важливим аспектом механізму є можливість використання поточного контексту. Під час інтеграції нових правил враховується не лише формальна структура, а й контекст, у якому застосовуються ці правила. Це дає змогу розв'язувати невизначеності, коли декілька правил потенційно можуть відповідати одному і тому самому фрагменту коду. Контекст містить інформацію про позицію у вхідному потоці, поточний стан абстрактного синтаксичного дерева (АСД) та семантичні ознаки, що допомагає визначити найбільш відповідний варіант.

Припустимо, що у процесі синтаксичного аналізу на певному етапі виникає неоднозначність: декілька правил із кандидатської множини $R_{cand} \subset R_{total}$ потенційно можуть застосовуватися до одного й того самого фрагмента вхідного коду. Щоб формально врахувати не лише формальну структуру, а й контекст застосування цих правил, вводимо поняття поточного контексту C. Нехай

· I - вхідний рядок;

· i - поточна позиція у вхідному рядку I;

· A - поточний стан абстрактного синтаксичного дерева (АСД), що відображає вже розібрані частини коду;

· S - семантична інформація, яка містить атрибути, ознаки та інші релевантні характеристики розібраних конструкцій.

Тоді поточний контекст C визначається як кортеж:

$$C = (i, A, S), \tag{12}$$

де CQ - простір усіх можливих контекстів.

Для розв'язання неоднозначностей, коли декілька правил з R_{cand} здатні генерувати однаковий фрагмент, визначимо функцію розв'язання неоднозначностей, яка вибирає одне правило з множини кандидатів, враховуючи поточний контекст. Нехай

$$\delta(R_{cand}, C) = arg\ max\ \phi(r, C). \tag{13}$$

Функція $\phi(r, C)$ може враховувати такі параметри:

1. Позиція i: Розташування у вхідному потоці I може впливати на вибір правила.
2. Стан АСД A: Поточна структура абстрактного синтаксичного дерева дає змогу враховувати вже розпізнані конструкції, що може впливати на вибір найбільш узгодженого правила.
3. Семантичні ознаки S: Атрибути і ознаки, отримані під час попереднього аналізу, допомагають розрізнити схожі синтаксичні конструкції на основі їх семантичного значення.

Це дає змогу вибирати правила не лише за формальними ознаками, а й з урахуванням змісту та контексту аналізу.

Формальне включення поточного контексту в процес усунення неоднозначності гарантує однозначність розбору навіть у випадках, коли декілька правил потенційно можуть бути застосовані до одного й того самого фрагмента. Це обґрунтовує інтеграцію контексту як важливий критерій під час вибору оптимального правила для розбору, що забезпечує коректність побудови АСД і узгодженість семантичного аналізу.

Алгоритм багатопрохідного парсингу

Поки i <| I | (тобто, поки не досягнемо кінця вхідного потоку), виконуємо таке:

1. Викликаємо функцію парсингу P з поточним входом та поточною граматикою:

$$\left(AST_{fragment}, j\right) \leftarrow P(I, i, G_{current}), \tag{14}$$

де $P: I \times N \times G \rightarrow (AST_{fragment}, N)$ - відображення, яке з позиції i у I генерує фрагмент абстрактного синтаксичного дерева (АСД) та повертає нову позицію j.

2. Оновлюємо загальне АСД:

$$AST \leftarrow Merge\left(AST, AST_{fragment}\right), \tag{15}$$

де $Merge$ - операція злиття АСД фрагментів.

3. Якщо $AST_{fragment}$ містить опис нових синтаксичних конструкцій (тобто реалізовано правило, яке відповідає опису розширення), то виконуємо:

· функцію витягування розширень

$$E: AST_{fragment} \rightarrow R_{ext}, \tag{16}$$

яка повертає множину нових правил розширення R_{ext}, а також відповідні нові символи N_{ext} та T_{ext}.

· Оновлюємо поточну граматику:

$$G_{current} = \left(N_{current} \cup N_{etx}, T_{current} \cup T_{,etx}, R_{current} \cup R_{etx}, S_0\right). \tag{17}$$

Отже, нові правила у процесі компіляції стають частиною граматики поточної мови програмування.

4. Якщо вхідний потік повністю розібрано, повертаємо фінальне АСД.

Цей алгоритм забезпечує інтегроване розширення граматики “на льоту”: парсер спочатку працює з базовою граматикою, будує АСД, потім при зустрічі опису розширень витягує нові правила, оновлює поточну граматику та продовжує розбір, використовуючи вже оновлену граматику. Отож система здатна адаптуватися до змін у вхідному потоці та інтегрувати нові синтаксичні конструкції без переривання процесу розбору.

Цей формальний опис покроково демонструє, як система розбору динамічно оновлює граматику, витягує додаткові правила розширення, інтегрує їх та будує фінальне АСД, забезпечуючи адаптивність і однозначність синтаксичного аналізу.

Результати дослідження

У ході дослідження було розроблено формальну модель динамічного розширення граматики, яка базується на базовій моделі $G_0 = (N_0, T_0, R_0, S_0)$ та її подальшому оновленні за допомогою додаткових множин N_{ext}, T_{ext}, R_{ext}. Розроблена модель дає змогу математично обґрунтувати інтеграцію нових синтаксичних конструкцій без порушення однозначності синтаксичного аналізу, що підтверджується аналізом множин рядків $L(r)$, які генеруються окремими правилами. Отримані результати демонструють, що якщо для кожного правила r із загальної множини $R_0 \cup R_{ext}$ виконується умова $L(r) \cap L(r`) = \varnothing$ для всіх $r \neq r`$, то система не містить синтаксичних неоднозначностей.

Однозначність синтаксичного аналізу залежить від властивостей кожного окремого правила продукції. Навіть якщо певні правила можуть породжувати перекриття, неоднозначність розбору може бути усунута, якщо хоча б для одного з таких правил визначена контекстно-залежна функція усунення неоднозначності.

Продуктивність розбору визначається базовим часом аналізу T_0 для граматики G_0 та додатковими витратами, які виникають через неоднозначні ситуації. Позначимо:

· f - частоту появи неоднозначних правил, що потребують застосування функції усунення неоднозначностей (тобто відношення кількості випадків, коли викликається функція δ, до загальної кількості застосувань правил);

· k - обчислювальну складність функції усунення неоднозначності, що представляється як додатковий часовий коефіцієнт.

Отже, загальний час розбору T_1 можна апроксимувати за формулою:

$$T_1 \approx T_0 \times (1 + f \cdot k). \quad (18)$$

Це означає, що приріст часу розбору безпосередньо залежить від частоти появи неоднозначних ситуацій та складності функції δ.

Оскільки кожне правило вже має чітко визначене представлення у вигляді АСД фрагмента, інтеграція нових правил не створює додаткових витрат на продуктивність, що забезпечує ефективну побудову абстрактного синтаксичного дерева.

Це показує ефективність запропонованого підходу, забезпечуючи як однозначність синтаксичного аналізу, так і прийнятну продуктивність та ефективну інтеграцію нових синтаксичних конструкцій у АСД. Отримані дані свідчать про високу адаптивність системи, що є критично важливим для сучасних систем розробки програмного забезпечення.

Подальші дослідження

У подальших дослідженнях основна увага буде зосереджена на розвитку та удосконаленні динамічно розширюваного синтаксичного аналізатора, що поєднує чотири підходи: Безлексичний парсинг, Композиційні парсери, Граматика Розбору Виразів та Пакрат-парсинг або пакетний парсинг із мемоізацією. Комбінування цих методологій дає змогу створити гнучкий та потужний інструмент для побудови синтаксичного аналізатора, здатного адаптуватися до змін у мові під час її використання.

Це відкриває перспективи для створення мов програмування з відкритою структурою, що підтримує метапрограмування, створення доменно-специфічних розширень та інтерактивну еволюцію синтаксису.

У межах досліджень буде проведено:

· аналіз ефективності парсера з частковою мемоізацією;

· тестування можливостей розширення граматики у реальному часі;

· порівняння з іншими парсерними підходами;

· дослідження практичної застосовності підходу на прикладі граматик, подібних до S-виразів, які легко адаптуються до змін.

Ці кроки дадуть змогу оцінити переваги та обмеження розробленого рішення, а також сформувати основи для подальшої розробки гнучких мовних інструментів.

Висновки

Визначено формальну модель динамічного розширення граматики, яка ґрунтується на базовій моделі $G0 = (N0, T0, R0, S0)$ та її подальшому доповненні додатковими множинами $N_{ext}, T_{ext}, R_{ext}$. Запропонований підхід формалізує інтеграцію нових синтаксичних конструкцій, гарантує однозначність синтаксичного аналізу навіть у випадку потенційних колізій. Навіть якщо певні правила породжують конфлікти, контекстно-залежна функція усуває неоднозначності, забезпечуючи 100 % коректний розбір.

Алгоритм для багатопрохідного парсера, який використовується в системі, дає змогу динамічно витягувати розширення з вхідного потоку та оновлювати граматику “на льоту”, забезпечуючи ефективну інтеграцію нових правил у побудову абстрактного синтаксичного дерева (АСД).

Майбутні дослідження можуть бути спрямовані на подальшу оптимізацію алгоритмів перевірки узгодженості, розширення моделі для підтримки ще більш складних синтаксичних конструкцій та інтеграцію методів семантичного аналізу. Крім того, перспективним напрямом є розробка інтерфейсів для експериментальної роботи з динамічними розширеннями граматики в режимі реального часу, що дозволить подальше покращення зручності та ефективності програмування.

Список літератури

1. *Koskimies K., Räihä K.-J., Sarjakoski M. Compiler construction using attribute grammars. 1982. Doi: https://doi.org/10.1145/800230.806991.*

2. *Aasa A. Precedences in specifications and implementations of programming languages. 1995. Doi: http://dx.doi.org/10.1016/0304-3975(95)90680-J.*

3. *Noonan R. E. Compiler construction usingmodern tools. 1986. Doi: https://doi.org/10.1145/5600.5697.*

4. *Guilan D., Suqing Z., Jinlan T., Weidu J. A study of compiler techniques for multiple targets in compiler infrastructures. 2002. Doi: https://doi.org/10.1145/571727.571735*

5. *Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, Compilers: Principles, Techniques, and Tools, 2nd edition, Addison Wesley, 2006.*

6. *De Oliveira Guimarães, J. Learning compiler construction by examples. 2007. Doi: https://doi.org/ 10.1145/1345375.1345418.*

7. *Demaille A., Levillain R., Perrot B. A set of tools to teach compiler construction. 2008. Doi: https:// doi.org/ 10.1145/1384271.1384291.*

8. *Grune D., van Reeuwijk K., Bal H. E., Jacobs C. J. H., Langendoen K. Modern Compiler Design. 2012. Doi: https://doi.org/10.1007/978-1-4614-4699-6.*

9. *Li H., Hu C., Zhang P., Xie L. Modular SDN Compiler Design with Intermediate Representation. Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference - SIGCOMM '16. 2016. Doi: https:// doi.org/10.1145/2934872.2959061.*

10. *Chen H., Ching W.-M., Hendren L. An ELI-to-C compiler: design, implementation, and performance. 2017. Doi: https://doi.org/10.1145/3091966.3091969.*

MECHANISM FOR EXPANDING PROGRAMMING LANGUAGE SYNTAX AT COMPILE TIME

Y. V. Berehuliak, R. V. Bachynskyi

Lviv Polytechnic National University,
Electronic Computational Machines Department

The article examines the issue of the flexibility of grammars with a fixed set of rules, which limits the ability to integrate specialized syntactic constructs. A model for extending such grammars is proposed, allowing for modular supplementation of the base system with new rules without compromising its integrity. This approach enables the syntactic system to adapt to new requirements, ensuring the automatic expansion of formal descriptions.

A mechanism for integrating new syntactic constructs has been developed, based on verifying their consistency using FIRST and FOLLOW set analysis. Additionally, methods for resolving ambiguities have been applied by utilizing the current parsing context, allowing the automatic determination of optimal rules for constructing an abstract syntax tree. This approach ensures the unambiguity and correctness of syntactic analysis, as even in cases of potential conflicts, the system eliminates ambiguity by considering local semantic and syntactic features.

This approach demonstrates the high adaptability and modularity of the grammar extension mechanism, which is of significant practical importance for the further evolution of compilation systems and the development of specialized dialects. The proposed methodology opens new perspectives for the creation of flexible and scalable syntactic analysis systems capable of responding promptly to changes in modern programming requirements.

Keywords: compiler architecture, grammar, lexical analysis, syntactic analysis.