УДК 004.4
DOI https://doi.org/10.32782/2663-5941/2023.2.1/25

Кандиба І.О.
Чорноморський національний університет імені Петра Могили

Фісун М.Т.
Чорноморський національний університет імені Петра Могили

Горбань Г.В.
Чорноморський національний університет імені Петра Могили

Степанчук Д.К.
Чорноморський національний університет імені Петра Могили

ГЕНЕРАТОР МУЛЬТИАЛФАВІТНИХ СИНТАКСИЧНИХ АНАЛІЗАТОРІВ З ГРАФІЧНИМ ВІДОБРАЖЕННЯМ СИНТАКСИЧНОГО ДЕРЕВА

У статті досліджено особливості реалізації предметно-орієнтованих мов програмування (ПОМ). Виконано аналіз основних досліджень в галузі генераторів синтаксичних аналізаторів: алгоритми реалізації, підвищення швидкодії, особливості інтеграції генерованих аналізаторів з мовами програмування загального призначення. Проаналізовано існуючі генератори аналізаторів в контексті розробки ПОМ, визначено їх основні недоліки: відсутність підтримки кількох алфавітів у синтаксисі граматик та відсутність засобів візуалізації абстрактного синтаксичного дерева (АСД). Наведено опис особливостей створення генератора аналізаторів на основі опису граматик за нотацією нормальних форм Бекуса-Наура (БНФ). Описано особливості процесу проєктування ПОМ за допомогою використання БНФ. Представлено опис частини ПОМ реляційної алгебри у вигляді БНФ. Проаналізовано особливості реалізації ПОМ з використанням методу LL (Left-to-right & Leftmost derivation). Запропоновано реалізацію генерації множин FIRST та FOLLOW для побудови розбору предикативного синтаксичного аналізатора (ПСА). Наведено алгоритм побудови названої таблиці розбору. Описано алгоритм автоматизованої генерації ПСА. Запропоновано найбільш доцільну мову програмування загального призначення для реалізації генератора синтаксичних аналізаторів для ПОМ. Описано недоліки інструментарію відображення АСД у аналогічних синтаксичних генераторах. Наведено опис інструментарію для візуалізації АСД, отриманого в результаті роботи згенерованого аналізатора. Продемонстровано представлення АСД засобами розробленого генератора аналізаторів на прикладі запиту ПОМ реляційної алгебри. Проаналізовано інструментарій Python для роботи різними типами кодування та реалізацією підтримки різних алфавітів. Запропоновано шляхи подальшого розвитку генератору синтаксичних аналізаторів для створення ПОМ.

Ключові слова: *ПОМ, генератор аналізаторів, синтаксичний аналіз, Python, Graphviz.*

Постановка задачі. ПОМ широко розповсюджені в сучасному світі інформаційних технологій [1 с. 23]: консольні інтерфейси, мови запитів, засоби математичного моделювання тощо. Цей клас мов дозволяє вирішити дві задачі: зменшити синтаксис необхідний для реалізації певних частин програмного коду, та спростити інформаційними технологіями (ІТ) для фахівців предметних галузей, що не мають поглиблених знань з програмування.

Реалізувати синтаксичні аналізатори для ПОМ можливо шляхом написання коду самих аналізаторів або за рахунок генерації цього коду спеціалізованим інструментарієм. Написання коду власноруч є складним та трудомістким процесом, а застосування спеціалізованих інструментів генерації аналізаторів має ряд обмежень [2 с. 3]:

− відсутня або частково підтримується кирилиця, що обмежує синтаксис вхідної мови;

− спеціалізований синтаксис побудови регулярних виразів для обробки лексем, що також має власні обмеження;

− відсутня можливість візуалізації абстрактного синтаксичного дерева (АСД) вхідного коду, що ускладнює процес реалізації семантичного аналізу.

Аналіз останніх досліджень і публікацій. Сучасні дослідження синтаксичного аналізу здебільшого присвячені аналізу особливостей методів цього аналізу. У роботі [3 с. 127] автори дослідили швидкодію аналізаторів, що реалізують спадний та підіймальні методи синтаксичного аналізу. Це дослідження також демонструє особливості створення аналізаторів з використанням моделі магазинного автомату. Однак, у роботі не розглянуто можливість використання згенерованих аналізаторів на основі зазначених методів та особливостям формального опису граматик вхідної мови.

У роботі [4 с. 5] продемонстровано можливість реалізації аналізатору для visibly pushdown languages (VPL), що є підвидом контекстно-вільних граматик [5 с. 4]. У цьому дослідженні наведено аналіз швидкодії аналізаторів побудованих спеціально для VPL, та генератору аналізаторів ANTLR призначеного для більш широкого кола задач. Наведені в роботі описи алгоритму роботи аналізатору та опис особливостей граматик є дуже детальними, але не недостатньо уваги приділено можливості застосування граматик з використанням різних алфавітів та застосуванню засобів відображення дерева розбору.

Стаття [6 с. 121] містить результат дослідження особливостей реалізації алгоритму синтаксичного аналізу GLR (Generalized Left-to-right Rightmost derivation parser). В ній наведено особливості розширеної форми Бекуса-Наура, задекларованої стандартом ISO/IEC 14977:1996(E) [7] та її аналізу алгоритмом GLR. Однак автори приділяють недостатню увагу дослідженню методи опису граматик, а саме можливості використання кирилиці.

Варто відмітити дослідження продуктивності існуючих генераторів аналізаторів при використанні їх у якості інструменту у навчальному процесі [8 с. 2]. У ньому висвітлюється складність опанування інструментарію генерації аналізаторів та особливості роботи згенерованих аналізаторів: використання генерованих обробників помилок, відображення дерева розбору, тощо. Не зважаючи на велику кількість представлених характеристик аналізаторів більш детального дослідження потребує можливість застосування різних алфавітів в процесі опису правил.

Виконаний аналіз продемонстрував, що сучасні дослідження в галузі розробки генераторів аналізаторів хоча й охоплюють різні аспекти цього процесу, але питання використання різних алфавітів у граматиках вхідних мова та генерації дерев розбору залишаються актуальним. Особливо актуальними ці питання є при розробці ПОМ, призначених для спрощення роботи фахівців предметних галузей без поглибленого знання інформаційних технологій, що може потребувати використання різних алфавітів, наприклад кирилиці.

Формулювання цілей та завдань. Метою досліджена є покращення usability [9] за рахунок можливістю використання в ПОМ різних алфавітів та графічного відображення дерев синтаксичного аналізу.

Створення відповідної інформаційної технології дозволить генерувати синтаксичний аналізатор для ПОМ на основі опису граматик з можливістю використання в описі вхідної мови різних алфавітів. Для досягнення зазначеної мети поставлені наступні завдання:

дослідження основних методів аналізу вхідної мови, що застосовуються при обробці ПОМ;

− розробка програмного забезпечення (ПЗ) для обробки вхідних LL граматик;

− реалізація генерації коду синтаксичного аналізатору вхідних ПОМ та відображення дерев граматичного розбору.

Виклад основного матеріалу дослідження.

Реалізація засобів генерації аналізаторів потребує створення ПЗ для обробки граматики. ПОМ відносяться до класу формальних мов, отже, їх можна описати за допомогою формальних граматик. Формальні граматики – це четвірка $G = \{ N, T, P, S \}$, де N – алфавіт нетермінальних символів, нетерміналів; T – алфавіт термінальних символів, терміналів; P – скінченна множина визначення не термінальних символів (продукцій, породжень); S – нетермінальний символ з якого починається опис граматики [10 с. 258].

Визначну роль в розробці ПОМ грає саме множина P, що фактично містить опис синтаксису мови та в певній мірі представляє собою модель досліджуваної предметної сфери. Перед розробкою аналізаторів ця множина має бути представлена у формалізованому вигляді, наприклад, у вигляді форми Бекуса-Наура (БНФ) [9 с. 76]. Наявність опису засобами БНФ при створенні ПОМ дає змогу:

− залучити до створення мови фахівців предметної галузі без поглибленого знання інформаційних технологій;

− вдосконалювати ПОМ, шляхом додавання нових мовних конструкцій;

− виправити неоднозначності граматик;

− застосувати автоматизовані генератори для скорочення часу необхідного на розробку аналізаторів.

БНФ дозволяє чітко виділити множини *N , T , P* та стартовий символ *S.* Наприклад, частину синтаксису мови реляційної алгебри (РА), що підпадає під визначення ПОМ, можна представити в такому вигляді:

<оператор вибору таблиці> ::= <атрибут> | <вираз> | <вираз> <умовний оператор>

<вираз>::=<атрибут><оператор> <атрибут>

<атрибут> ::= **id | (** <оператор вибору таблиці> **)**

<умовний оператор>::= **WHERE** <перелік умов>

<перелік умов> ::= <умова> <логічний вираз>

<логічний вираз> ::= <логічний оператор> <умова> <логічний вираз> | ε

<умова>: :=**id** <оператор порівняння> <дані>

<дані> ::=**число** |“ **рядок ”** ;

<оператор>::=**UNION** | **JOIN** | **SEMIJOIN** | **SEMIMINUS** | **INTERSECT** | **MINUS**

<логічний оператор> = **OR** | **XOR** | **AND**

<оператор порівняння> = < | >| **=** |!=

де, всі нетермінали записуються в кутових дужках, символ S записується першим, термінали записані у вигляді токенів: **id** – ідентифікатор, **число, атрибут, рядок** – токени виділені на етапі лексичного аналізу, а весь запис є граматикою *P.*

Запис граматики у вигляді БНФ є корисним та важливим етапом процесу розробки трансляторів, але не може бути використаний для генерації аналізаторів сучасними інструментами. В тім перетворивши БНФ в РБНФ з додаванням у якості токенів регулярних виразів генерація стає можливою, але це може ускладнити участь в процесі формування ПОМ фахівців предметних галузей. Враховуючи специфіку використання ПОМ [1, с. 23] участь в її розробці фахівців предметних галузей, що будуть її використовувати є важливою складовою. Отже, можливість використання БНФ у якості мови опису граматик для генерації ПОМ має бути врахована при розробці генератору синтаксичних аналізаторів.

Описаний синтаксис мови дає змогу почати розробляти синтаксичний аналізатор. Перевіряти правильність синтаксису та формувати дерево розбору, що обробляється семантичним аналізатором кількома розповсюдженими методами: LL (Left-to-right Leftmost derivation), LR(Left-to-right Rightmost derivation).

При реалізації предметно-орієнтованих мови найчастіше застосовують LL(1), де в душках показано «lookahead» – кількість символів, що використовуються для визначення наступного породження. Цей метод легко реалізується за допомогою ПСА, що, в свою чергу, не потребує використання рекурсії та побудований на явному використанні стеку породжень.

Використання стеку породжень в предиктивному синтаксичному аналізі базується на таблиці розбору, будується після визначення множин FIRST та FOLLOW за [9 с. 285], де принято такі умовні позначення:

a,в – узагальнення термінального символу;

A, B – не термінальні символи;

T, U, V,X, Y – граматичні символи (термінали або не термінали);

α, β – рядки граматичних символів;

ε – пустий символ.

Першою визначається FIRST:

1) Якщо X – породжує термінальний символ, то {X} ***** FIRST(X);

2) Якщо існує продукція X ::= ε, додамо ε до FIRST(X);

3) Якщо X – нетермінал й існує продукція $\boldsymbol{X ::= Y_1 Y_2 ... Y_k}$, то помістимо $\boldsymbol{a}$ в FIRST(X), якщо для якогось i $\boldsymbol{a}$*****FIRST(Y_i) і $\boldsymbol{\varepsilon}$ входить в усі множини FIRST(Y_1),…, FIRST(Y_{i-1}), тобто $Y_1 \cdots Y_{i-1} \rightarrow^* \varepsilon$. Якщо $\boldsymbol{\varepsilon}$ є у всіх FIRST(Y_i), $\boldsymbol{i = 1 \ldots k}$, то додаємо $\boldsymbol{\varepsilon}$ до FIRST(X). Наприклад, усе, що перебуває в множині FIRST(Y_1), є й у множині FIRST(X). Якщо $\boldsymbol{Y_1}$ не породжує $\boldsymbol{\varepsilon}$, то більше ми нічого не додаємо до FIRST(X), але якщо $\boldsymbol{Y_1 \rightarrow^* \varepsilon}$, то до FIRST($X$) додаємо FIRST($Y_2$) і т.д.

Враховуючи, що генератор синтаксичних аналізаторів має опрацьовувати файл з описом синтаксису (набір породжень), то визначення FIRST має відбуватись автоматизовано (рис. 1).

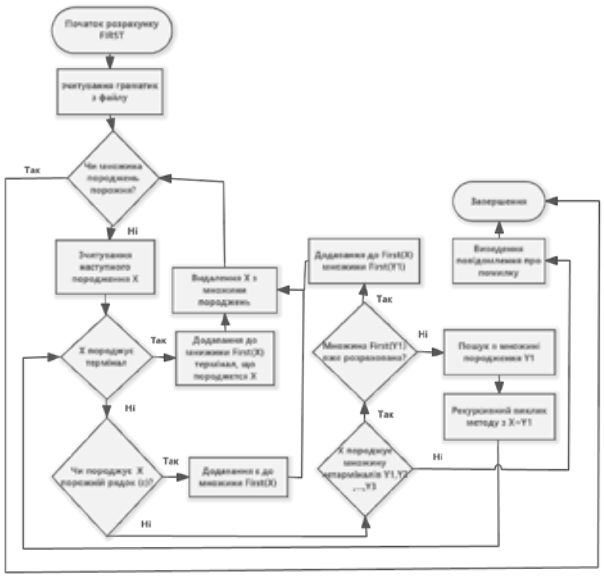

Рис. 1. Блок-схема процесу визначення множини FIRST

Множина FOLLOW(A) є множиною терміналів $\boldsymbol{a}$, що розташовані праворуч від (після) нетермінала

A в деякому породженні (сентенціальній формі), тобто множина терміналів **a**, таких, що існує породження виду, S →* αAaβ для деяких α й β.

Щоб забезпечити роботи генератора аналізаторів необхідно обрахувати FOLLOW(A) для всіх нетерміналів A програмним шляхом на основі вхідного набору породжень. Процес розрахунку FOLLOW полягає в застосування наступних правил доти, доки до жодної із множин FOLLOW не можна буде додати жодного символу [9 с. 285]:

1) поміщаємо $ в FOLLOW(S), де S – стартовий символ, а $ – правий обмежувач вхідного потоку;

2) якщо є продукція A ::= αBβ, то всі елементи множини FIRST(β), крім ε, поміщаються в множину FOLLOW(B);

3) якщо є продукція A ::= αB або A::=αBβ, де FIRST(β) містить ε (β →* ε), то всі елементи із множини FOLLOW(A) містяться в множину FOLLOW(B).

Генератор синтаксичних аналізаторів використовує лише набір вхідних породжень, що вимагає автоматизувати процес обчислення FOLLOW. Блок-схема алгоритму реалізації автоматичного процесу визначення цієї множини представлена на рис. 2.

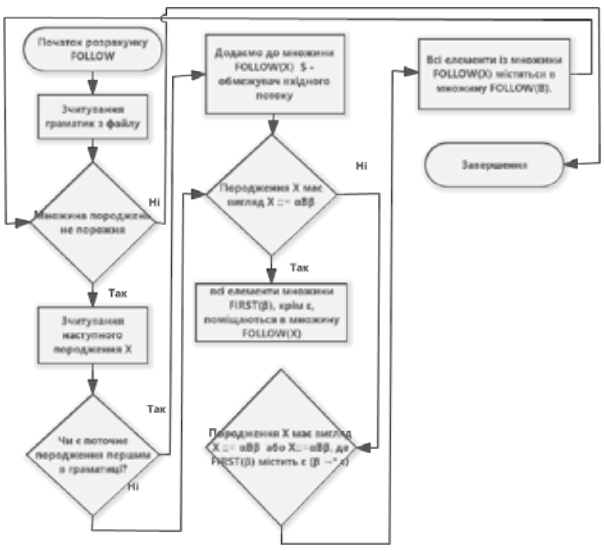

Рис. 2. Блок-схема процесу визначення множини FOLLOW

Подальша обробка LL(1) граматик вимагає наявності таблиці розбору **T**, що фактично є основою для роботи предикативного синтаксичного аналізатора. Генератор аналізаторів для виконання предиктивного синтаксичного аналізу має автоматично будувати цю таблицю розбору. Побудова таблиці розбору здійснюється на основі множин FIRST та FOLLOW за наступним алгоритмом:

1) Для кожного породження A::=α виконуємо кроки 2-4.

2) Для кожного термінала ***a*** з FIRST(α) додаємо A::=α в комірку ***T[A,a]***.

3) Якщо в множину FIRST(α) входить ε, для кожного термінала ***b*** з FOLLOW(A) додаємо A::=α в комірку ***T[A,b]***.

4) Якщо в породженні A::=α, ε входить в FIRST(α), **а** $ – в FOLLOW(A), додаємо A::=α в комірку ***T[A,$]***.

5) Комірки, що залишились порожніми мають містити повідомлення про помилку.

Синтаксичний аналізатор керується результатом порівняння символів на вершині стеку породжень ***X*** та поточним вхідним символом ***a***. Два символи визначають дії синтаксичного аналізатора:

1) Якщо ***X*** = ***a*** = ***$***, робота синтаксичного аналізатору завершена без помилок.

2) Якщо ***X*** = ***a*** ≠ $, синтаксичний аналізатор прибирає зі стека ***X*** і зміщує покажчик вхідного потоку до наступного символу.

3) Якщо ***X*** ::= ε, синтаксичний аналізатор прибирає зі стека ***X***, але не зміщує покажчик вхідного потоку до наступного символу.

4) Якщо ***X*** являє собою нетермінал, відбувається аналіз породження ***T[X,a]*** з таблиці розбору ***T***. Цей запис являє собою або певне породження або запис про помилку. Якщо, наприклад, ***T[X,a]*** = {***X*** ::= ***UTY***}, синтаксичний аналізатор замінює ***X*** на вершині стеку на ***UTY*** (з ***U*** на вершині стека). При цьому, якщо в ***T[X,a]*** немає запису про помилку, додаються елементи ***UTY*** до АСД.

5) Якщо в ***T[X,a]*** повідомлення про помилку, синтаксичний аналізатор виводить користувачу повідомлення про помилку та зупиняє свою роботу.

Реалізація генератора синтаксичних аналізаторів вимагає використання мови програмування загального призначення. Мова Python підтримує велику кількість різного інструментарію, що забезпечує її універсальність. Однією з розповсюджених галузей застосування цієї мови є побудова трансляторів [2], що робить його найбільш доцільним засобом реалізації генератора синтаксичних аналізаторів.

Відображення АСД спрощує процес розробки та відлагодження семантичних аналізаторів. Серед засобів генерації аналізаторів відображення АСД реалізується тільки ANTLR і виключно за умови генерації коду для мови загального призначення Java [2]. Мова загального призначення Python підтримує можливість використання бібліотеки Graphviz.

Graphviz бібліотека з відкритим програмним кодом, що призначена для візуалізації графів

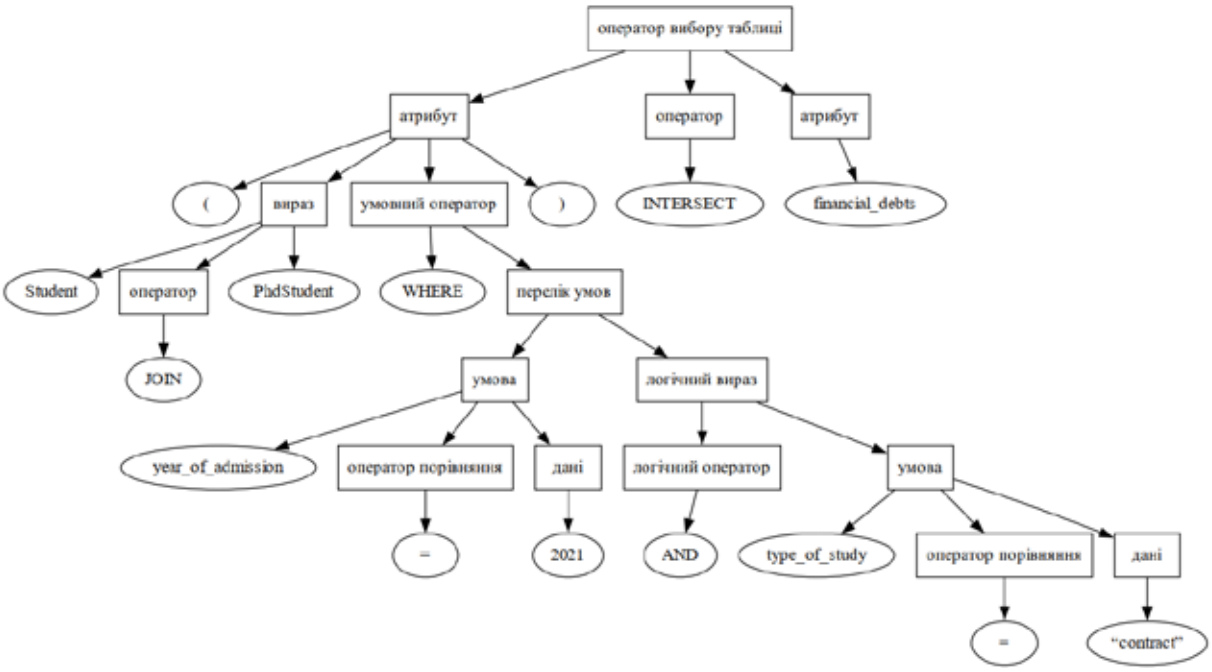

Рис. 3. Представлення АСД засобами розробленого генератора аналізаторів

[11 с. 1985]. Ця бібліотека застосовується в сучасних дослідженнях для візуалізації великих наборів даних. Зазначений факт відображає можливість відображання АСД для об'ємного коду, що дозволить значно спростити відлагодження коду семантичного аналізатора та визначити помилки в синтаксичному аналізі, що можуть виникнути помилках у вхідній множині породжень.

Вхідний рядок реляційної алгебри, що відповідає представленій вище граматиці: (Student JOIN PhdStudent WHERE year_of_admission = 2021 AND type_of_study = "contract") INTERSECT financial_debts. АСД для цього виразу матиме відносно просту структуру (рис. 3).

Особливістю розроблюваного генератору синтаксичного аналізатора ПОМ є підтримка кількох алфавітів у вхідній мові. Використання Python для реалізації зазначених вище кроків дозволяє розв'язувати цю задачу шляхом застосування вбудованих засобів. За замовчуванням Python використовує кодування utf-8. Змінити тип кодування та збільшити кількість доступних алфавітів можливо додавши при зчитуванні з файлу параметр encoding. Для обробки вхідного коду користувача застосовані методи decode( ) та encode( ), що в поєднанні з sys.getdefaultencoding( ) можуть автоматично налаштовуватись на кодування операційної системи користувача та використовувати необхідний алфавіт.

Висновки. Проведено дослідження способів реалізації синтаксичного аналізатора. Запропоновано алгоритм автоматизованої генерації предикативного синтаксичного аналізатора для ПОМ. Розроблено інформаційну технологію з підтримкою кількох алфавітів в описі вхідних граматик. Застосовано інструментарій Python для відображення графових структур, що дозволяє динамічно відображати АСД. Представлено результат роботи системи.

В подальшому планується інтеграція до розробленої системи підсистеми лексичного аналізу, що дозволить повноцінно обробляти вхідний код ПОМ.

Список літератури:

1. Fowler M. Domain-Specific Languages. Boston: Addison-Wesley Professional, 2010. 640 p.
2. Фісун М. Т., Кандиба І. О., Горбань Г. В., Фаленкова М. В. Використання методу аналізу ієрархій для вибору засобів розробки синтаксичних аналізаторів при створенні DSL. Наукові праці Вінницького національного технічного університету. 2021. № 1. URL: https://praci.vntu.edu.ua/index.php/praci/issue/view/49 (дата звернення: 25.12.2022).
3. Гавриленко С. Ю., Прохорова Т. М., Давидов В. В. Дослідження методів побудови синтаксичних аналізаторів. Системи обробки інформації. 2015. Вип. 136, № 11. С. 125–127.

4. Jia X., Kumar A., Tan G. A derivative-based parser generator for visibly Pushdown grammars. Proceedings of the ACM on Programming Languages. 2021. Vol 5, №151. P. 1–24.

5. Alur R., Madhusudan P. Adding nesting structure to words. Journal of the ACM (JACM). 2009. Vol. 56, № 3. P. 1–43.

6. Scott E., Johnstone A. GLL syntax analysers for EBNF grammars. Science of Computer Programming. 2018. Vol. 166. P. 120–145.

7. ISO/IEC 14977:1996(E) Syntactic metalanguage - Extended BNF [The standard is valid from 1996-12-15]. Switzerland, Geneve. 1996. P.

8. Ortin F., Quiroga J., Rodriguez-Prieto O., Garcia M. An empirical evaluation of Lex/Yacc and ANTLR parser generation tools. Plos one. 2022. Vol. 17, № 3. P. 1-16.

9. ДСТУ ISO/IEC 25010 Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Моделі якості системи та програмних засобів; чинний від 2018-10-10. Вид. офіц. Київ: УкрНДНЦ, 2018. 22 с.

10. Aho A., Lam M., Sethi R., Ullman J. Compilers: Principles, Techniques, and Tools. Boston:Addison Wesley, 2006. 1040 p.

11. Krommyda M., Kantere V., Vassiliou Y. Ivlg: Interactive visualization of large graphs. 2019 IEEE 35th International Conference on Data Engineering (ICDE)(2019). P. 1984–1987.

Kandyba I.O., Fisun M.T., Horban H.V., Stepanchuk D.K. GENERATOR OF MULTI-ALPHABETIC SYNTACTIC ANALYZERS WITH GRAPHICAL DISPLAY OF THE SYNTACTIC TREE

The article presents a study of the peculiarities of implementation and use of domain-specific programming languages (DSL). The basic directions of modern research in the field of parser generators are determined: implementation algorithms, increasing speed, features of integration of generated parsers. The parser generators are analyzed in the context of the development of a DSL, their main disadvantages are identified: the lack of support for several alphabets in the syntax of grammars and the lack of visualization tools for the abstract syntactic tree (AST). The features of creating an analyzer generator based on the description of grammars are described. The peculiarities of the process of designing a DSL using the Backus-Naur form (BNF) are described. A description of a part of the domain-oriented programming language of relational algebra in the form of BNF is presented. The peculiarities of the implementation of the DSL using the LL method (Left-to-right Leftmost derivation) are analyzed. The implementation of the generation of FIRST and FOLLOW sets for building a lookup table is proposed. An algorithm for constructing a lookup table based on the generated sets is given. A method of automated generation of a predicative parser is presented. The most appropriate general-purpose programming language for the implementation of a parser generator for a computer-aided design is proposed. The disadvantages of the tools for displaying the ASD in similar parsers are described. A description of the tools for visualization of the ASD obtained as a result of the generated analyzer is given. The representation of the ASD by means of the developed analyzer generator is demonstrated on the example of a relational algebra DSL query. The Python tools for working with different types of encoding and implementing support for different alphabets are analyzed. The ways of further development of the parser generator for DSL are proposed.

Key words: *DSL, analyzer generator, syntax analysis, Python, Graphviz.*