УДК 004.4'22

М. Т. Фісун, д. т. н., проф.; І. О. Кандиба; Г. В. Горбань, к. т. н., доц.; М. В. Фаленкова

ВИКОРИСТАННЯ МЕТОДУ АНАЛІЗУ ІЄРАРХІЙ ДЛЯ ВИБОРУ ЗАСОБІВ РОЗРОБКИ СИНТАКСИЧНИХ АНАЛІЗАТОРІВ ПРИ СТВОРЕННІ DSL

Представлено аналіз програмних інструментів генерації синтаксичних аналізаторів, що базуються на мові Python. Виділено дві основні категорії: універсальні інструменти генерації, що не прив'язані до певної мови програмування, та інструменти призначення для застосування виключно мовою загального призначення Python.

Проведено детальний аналіз низки найрозповсюдженіших програмних інструментів. На основі стандарту ISO/IEC 2510 визначено основні характеристики генераторів аналізаторів. До обраних характеристик належать: продуктивність, покриття контексту, задоволеність, функціональна придатність, переносимість, зручність використання.

Побудовано ієрархію критеріїв для реалізації методу аналізу ієрархій. Ієрархія включає в себе обидві моделі якості стандарту ISO/IEC 2510 та набір критеріїв, описаних цими моделями. Використано експертне оцінювання для розрахунку вектору локальних пріоритетів, що лягли в основу розрахунку вектору глобальних пріоритетів. З вектору глобальних пріоритетів визначено альтернативу, що має найбільшу оцінку. Обрана альтернатива вказує на найефективніший програмний інструмент генерації аналізаторів.

Ключові слова: *реляційна алгебра, Python, DSL, Pyparsing, Parglare, PLY, ANTLR, Unicc.*

Вступ

Domain-Specific Languages (DSL) – це клас мов програмування, що використовуються виключно в певних предметних сферах (галузях) [1]. До цього класу: відносять мови маніпуляції даними [2, 3], мови представлення знань [4, 5], тощо.

DSL розділяють на два основні види: зовнішні та внутрішні. Внутрішні DSL будуються на ланцюжку виклику методів та не потребують наявності додаткових компонентів, але мають обмежену область застосування. Зовнішні DSL реалізуються окремими мовами програмування та не мають обмежень попереднього класу, але потребують наявності лексичного та синтаксичного аналізаторів. Засоби автоматичної генерації лексичних та синтаксичних аналізаторів мають різни можливості [6,7].

У роботах, що відображають проведені дослідження якості роботи аналізаторів, основою є показники швидкості обробки вхідної мови [7] або особливості побудови правил та роботи інструментів генерації аналізаторів [6, 8, 9]. Проте створення аналізаторів предметно-орієнтованих мов має декілька особливостей: підтримка кирилиці та додавання коду на мові загального призначення. Для вибору кращої альтернативи серед засобів генерації синтаксичних аналізаторів необхідно враховувати всі перераховані характеристики та особливості. Аналіз джерел свідчить про відсутність комплексного підходу щодо розв'язання такої задачі, тому тема дослідження представляється актуальною.

Актуальність

В останні роки прослідковується зростання інтересу до предметно-орієнтованих мов [1]. Найчастіше цей клас мов програмування в англомовних джерелах називають Domain-Specific Languages (DSL). Популярність DSL обумовлена двома фактами:

– підвищення продуктивності роботи розробників ПЗ за рахунок меншого об'єму коду DSL, ніж при використанні мов загального призначення;

– поліпшення взаємодії з фахівцями в предметній сфері за рахунок особливостей синтаксису, що складається з термінів предметної сфери.

Найбільш відомими прикладами є: мова реляційної алгебри [2], мова маніпуляції даними у графовій БД Cypher [3], мова опису онтологій OWL [4]. Окремо варто виділити мови моделювання, що найчастіше представляються саме як DSL [5].

Однією із складних задач при розробці DSL є реалізація обробників синтаксису вхідної мови (синтаксичних аналізаторів). Сам процес створення DSL включає в себе декілька різних етапів: формування та формальний опис структури вхідної мови, розробка лексичного аналізатору, розробка синтаксичного аналізатору, генерація вихідного коду [6].

Мета

Генератори синтаксичних аналізаторів мають широкий набір характеристик [6], які впливають на результати їх роботи, що, в свою чергу, вказує на необхідність визначити набір критеріїв для вибору найефективнішого засобу генерації коду. Вибір найефективнішого засобу генерації аналізаторів є складною задачею, вирішення якої вимагає розв'язку багатокритеріальної задачі прийняття рішень. Побудова ієрархії критеріїв та вибір найефективнішого засобу генерації аналізаторів є **метою статті**. При цьому дослідження стосуються як універсальних генераторів, що не прив'язані до певної мови програмування, так і засобів, призначених виключно для однієї мови програмування.

Розв'язання задач

Для скорочення часу програмування аналізаторів частіш за все використовуються спеціалізовані програмні засоби – генератори аналізаторів. Ці засоби здатні автоматично генерувати частину програмного коду аналізаторів (у деяких випадках повністю), що дозволяє скоротити не лише час, необхідний на розробку програмного забезпечення, а й запобігти помилкам під час написання коду.

Важливими елементом генераторів мовних аналізаторів є спосіб опису правил синтаксису вхідної мови. Створення правил для лексичного аналізатору, у більшості випадків здійснюється за допомогою регулярних виразів, але розробка синтаксичного аналізатору є більш складною задачею, що потребує програмування алгоритму розбору вхідного набору термінальних символів і токенів.

Перед початком розробки аналізаторів синтаксис вхідної мови треба її формально описати. Опис структури вхідної мови за звичай здійснюється за допомогою форм Бекуса-Наура (БНФ) [6] або їх модифікації. БНФ – це формальний засіб опису вхідної мови, у якій синтаксичні сутності описуються за допомогою інших синтаксичних сутностей. Наприклад, для реалізації мови програмування маніпулювання даними – реляційної алгебри (РА), можна використати наступні правила в БНФ:

<union>::=<attribute_name> **UNION** <attribute_name>**;**

<attribute_name>::=< braket> | <table_name>

<table_name>::=<symbol>{<symbol>}|<symbol>{<symbol> <number>}

<symbol>::=**a**|**b**|…|**z**

<number>::=**0**|**1**|**2**|…|**9**

<braket>::=**(**< braket_expr >**)**

<braket_expr>::=<minus>|<union>|<join>|<projection>|<extend>|<semiminus>|<semijoin >|< divideby>|<braket>

Після формального опису вхідної мови необхідно почати розробку програмного коду аналізаторів. На цьому етапі необхідно обрати інструмент для генерації аналізаторів. Більшість інструментів цього типу прив'язана до певної мови

програмування, наприклад JLanguageTool, JavaCC, PLY, PyParsing та ін., але існують також універсальні: ANTLR, Unicc та ін..

За даними порталу «ain.ua» мова програмування Python утримує одну з лідируючих позицій у рейтингах вакансій на ринку праці. Окрім того Python підтримує декілька різних парадигм програмування та велику кількість додаткових інструментів, наприклад бібліотеки для різних СКБД, веб фреймворки (Flask, Django, тощо), бібліотеки для роботи з апаратним забезпеченням та ін.

Генератори аналізаторів можна поділити на два типи: універсальні – окремі інструменти, що дозволяють генерацію аналізаторів для різних мов програмування; вбудовані в певну мову програмування. Виходячи з цього факту, можливо виділити наступні генератори аналізаторів виключно для Python:

а) **Pyparsing**. Бібліотека для генерації лексичного та синтаксичного аналізатору на основі спеціалізованих правил. Особливість цього інструменту закладається у способі побудови правил, що вимагає використання спеціалізованого синтаксису [7]. Pyparsing використовує вбудовані методи та змінні для опису синтаксису (табл. 1).

Таблиця 1

Основні функції Pyparsing

Метод/Змінна	Значення
DEFAULT_KEYWORD_CHARS	Набір символів, що не є роздільникам: '$_ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'.
Literal()	Пошук точного збігу з рядком.
Keyword()	Схожа за принципом дії на Literal (), але при цьому заданий рядок має бути відділений символом, що не входить у змінну DEFAULT_KEYWORD_CHARS.
CaselessKeyword()	Аналогічно Keyword(), але без врахування регістру.
Word()	Використовується для створення шаблону рядку, що складається з символів, які входять до змінної DEFAULT_KEYWORD_CHARS.
CharsNotIn()	Символи, шо не мають входити у рядок.
Regex()	Порівняння з шаблоном регулярного виразу.
QuotedString()	Визначення доступних роздільників: відключення або підключення можливості використання проміжків, табуляції, використання символу нового рядку та ін.
SkipTo()	Метод дозволяє пропуск неспівпадаючих фрагментів.
White()	Аналогічна Word (), але включає пробільні символи.
Empty()	Перевірка на наявність пустого рядку.
NoMatch()	Протилежність Empty()
Alphas	Рядок, що містить лише літери.
Nums	Рядок, що містить лише цифри.
Alphanums	Рядок, що містить літери та цифри.
Printables	Рядок, що складається з будь яких символів, що виводяться на друк окрім проміжку.
And() або +	Логічне «і».
Or() або ^	Логічне «або».
ZeroOrMore()	Нуль або більше.
OneOrMore()	Один або більше.
Optional()	Пошук рядку, що може бути наявним або відсутнім

Методи та змінні, що наведені у таблиці 1, дозволяють описувати вхідну мову для проведення подальшого аналізу, але окрім наведених у табл. 1, бібліотека містить додаткові методи, що призначені для специфічної обробки, наприклад htmlComment() – виділення коментарів у розмітці HTML (' <!-- ' і ' --> '), с StyleComment() – виділення коментарів у розмітці CSS (' /* ' і ' */ '). Доступні також інструменти обмеження

довжини вхідного рядку: min – встановлення мінімальної довжини, max – максимальної довжини, exact – точно довжина рядку та ін.

Синтаксис аналізатору мови PA [2], створеного за допомогою Pyparsing, виглядатиме наступним чином:

```
TableName=Word(alphas)
UnionOper=TableName+Keyword('UNION')+TableName
MinusOper=TableName+Keyword('Minus')+TableName
JoinOper=TableName+Keyword('Join')+TableName
SemiJoinOper=TableName+Keyword('SemiJoin')+TableName
SemiMinusOper=TableName+Keyword('SemiMinus')+TableName
Prjection=TableName+Word('[')+TableName+Optional(','+TableName)+Word(']')
OperProcesing=TableName^UnionOper^MinusOper^JoinOper^SemiJoinOper^SemiMinus
Oper^Prjection
s = 'Student Union TradeUnion'
try:
   res = OperProcesing.parseString(s)
   print(res.asList())
except ParseException:
   print('Помилка опрацювання оператору ParseException');
```

Правила описані вище дозволяють здійснювати розбір певного оператору та перехоплення виключення у разі помилки Pyparsing дає можливість використання функцій operatorPrecedence та infixNotation для обробки виразів. OperatorPrecedence розділяє вираз на окремі частини використовуючи у якості роздільника круглі дужки, повертає цей метод, об'єкт pyparsing. Forward, що в свою чергу містить реалізацію ітератору для послідовної обробки виразів в дужках.

Основним алгоритмом розбору вхідного рядку інструментом Pyparsing є метод рекурсивного спуску (Recursive descent parser). Суть цього алгоритму полягає у взаємному виклику функцій, які реалізують перевірку правил, реалізація граматичних правил відбувається за допомогою змінних. Запуск першої функції відбувається наступним чином: OperProcesing.parseString(s), де OperProcesing ім'я змінної, що містить опис правила, а s – вхідний рядок, що містить код для аналізу. Варто окремо згадати, що цей інструмент не підтримує можливість роботи з кирилицею.

b) Parglare. Бібліотека для мови python, що включає в себе реалізацію алгоритму GLR (Generalized Left-to-right Right most derivation parser розширена версія методу LR) [8]. Призначений для розбору недетермінованих та неоднозначних граматик. У роботі алгоритму GLR реалізовано роботу LR – парсеру з додатковою перевіркою усіх можливих трактовок граматики шляхом використання пошуку в ширину.

Parglare при створенні аналізаторів вимагає реалізації кількох компонентів: блок Grammar – цей блок описує лексеми (вкладений блок terminals) та синтаксичні правила, блок може бути представлений як окремим файлом так і строкою в середині програмного коду python. Наступний компонент, що необхідно реалізувати – блок actions, що включає в себе опис методів або лямда виразів для обробки операторів заданих у попередньому блоці при цьому кількість описаних дій блоку actions має відповідати кількості варіантів написання оператору у блоці Grammar. Об'єкт класу Parser є також необхідною складовою Parglare, що в конструкторі містить змінні Grammar та actions. Початок аналізу відбувається за допомогою виклику методу Parser.parse( ) у який передається рядок для аналізу. Синтаксис опису правил для цього інструменту генерації аналізаторів схожий на БНФ, але не забороняє наявність лівої рекурсії завдяки використанню алгоритму GLR. Відсутня підтримка кирилиці, що

призводить до виведення помилки при спробі опису оператору з використанням кирилиці.

Підтримується вбудований обробник помилок, що реалізовано у класі parglare.exceptions.ParseError. При виявленні помилки в коді користувачу виводиться повідомлення з варіантами виправлення помилки, наприклад при неправильному написанні оператора union буде виведено наступне повідомлення: parglare.exceptions.ParseError: Error at 1:16:"nt unions **> (TradeUnio" => Expected:) or STOP or join or minus or semijoin or semiminus or union but found.

Граматика аналізатору для операторів [2] мови реляційної алгебри буде виглядати наступним чином:

```
grammar = r"""
E: E 'minus' E  | E 'join' E | E 'union' E  | E 'semijoin' E  | E 'semiminus' E | '(' E ')' | IDs;
terminals
IDs: /[A-Za-z0-9]+/;
"""
actions = {
   "E": [lambda _, n:  n[0] +" minus "+ n[2],
        lambda _, n:  n[0] +" join "+ n[2],
        lambda _, n:  n[0] +" union "+ n[2],
        lambda _, n:  n[0] +" semijoin "+ n[2],
        lambda _, n:  n[0] +" semiminus "+ n[2],
        lambda _, n:  n[1],
        lambda _, n:  n[0] ],}
g = Grammar.from_string(grammar)
parser = Parser(g, actions=actions)
result = parser.parse("(Student union (TradeUnion minus Table)) minus (Table minus Table)")
```

c) PLY (Python Lex-Yacc) [9]. Бібліотека для мови програмування python, що реалізує можливості інструментів LEX та YACC. Lex – являється вбудованим у UNIX системи генератором лексичних аналізаторів, цей інструменту входить до стандарту POSIX. Yacc служить для генерації синтаксичних аналізаторів, що є наступним кроком обробки програмного коду.

Основною ідеєю PLY є спрощення створення аналізаторів, на основі одного набору правил. Lex генерує набір правил, які за своїм синтаксисом схожі на форму представлення БНФ, але задаються за допомогою спеціалізованих сутностей. Підключення реалізації інструменту Lex відбувається за допомогою команди import ply.lex.

Задача Lex полягає у розбитті вхідного рядку на токени. Робота з ply.lex потребує створення спеціалізованого масиву токенів, що має ім'я tokens та подальшого опису токену у вигляді змінної, що починається з «t_»:

```
tokens = ('MINUS', ' UNION ', 'JOIN', ' SEMIMINUS ', ' SEMIJOIN ','ident','open','close')
t_UNION = r'union'
t_ident=r'\w+'
t_open=r'\('
t_close=r'\)'
```

Другою частиною бібліотеки є ply.yacc, що служить для реалізації синтаксичного аналізу. Використання функцій цього інструменту вимагає підключення цієї бібліотеки: import ply.yacc.

Створення правил ply.yacc відбувається за допомогою комбінування токенів описаних у ply.lex. Кожен вираз мови, що реалізується має бути описаний за

допомогою методів, які починаються з «p_». Наприклад синтаксис опису оператору PA UNION:

```
def p_expr(p):
    '''expr :  open close |  open expr close |  expr UNION ident  |  expr UNION expr | ident UNION expr'''
    print(p[2])
def p_UNION (p):
    '''union : ident UNION ident'''
    p[0]+=" "+p[1:]+" "+p[2]+" "+p[3]
```

Результатом роботи одного з методів синтаксичного аналізу є рядок, що зберігається у змінній «p[0]», де «p» є об'єктом класу ply.yacc - YaccProduction (ім'я змінної «p» може бути змінене розробником). Змінна «p» містить вхідний рядок розбитий роздільниками, а також результат роботи методу синтаксичного аналізу.

Як і попередній інструмент генерації аналізаторів PLY не підтримує можливість роботи з кирилицею. Обробка помилок здійснюється за допомогою методу «p_error».

PLY реалізовує алгоритм LALR(1) (Lookahead Left to Right) [6], що базується на побудові таблиці розбору, а сформована таблиця розбору зберігається у окремому файлі «parser.out».

Розповсюдженими універсальними генераторами, що підтримують можливість генерації для Python є:

d) ANTLR (ANother Tool for Language Recognition) [10]. Універсальний генератор аналізаторів, що розроблений на мові програмування java. Цей програмний продукт реалізований окремим застосунком, що на вхід приймає набір спеціалізованих правил у файлі *.4g, а на виході генерує набір класів для лексичного та синтаксичного аналізу. Команда генерації аналізаторів для ANTRL виглядає наступним чином: **antlr4 -Dlanguage=Python3 "F:\real\REAL_Lang.g4"**, де ключ **–Dlanguage** задає мову програмування для якої генеруються аналізатори, а **"F:\real\REAL_Lang.g4"** файл, що містить опис граматик вхідної мови.

Після генерації у директорії з файлом опису граматик з'являються файли: REAL_LangLexer.py ,REAL_LangListener.py, REAL_LangParser.py, REAL_Lang.tokens, REAL_LangLexer.tokens, де REAL_Lang – назва вхідної мови.

Кожен згенерований файл виконує свою роль:

a) REAL_LangLexer.py містить правила лексичного розбору;

b) REAL_LangParser.py містить правила синтаксичного розбору;

c) REAL_LangListener.py додатковий клас, що містить опис методів, які виконуються під час обходу синтаксичного дерева (на мові загального призначення);

d) REAL_Lang.tokens, REAL_LangLexer.tokens – допоміжні файли з описом токенів.

В середині описаного файлу REAL_LangParser.py знаходяться згенеровані класи для кожного граматичного правила, ці класи містять механізм обробки правила та успадковують від ParserRuleContext.

Файл типу *.g4 містить опис правил у спеціалізованому синтаксисі подібному до форми Бекуса-Наура:

```
parseoperator: minus_stmt |where_stm |projection_stmt |union_stmt |join_stm |matching_stm |notmatching_stm;
args_stm: table_name|projection_stmt| bracket;
minus_stmt: args_stm MINUS args_stm;
table_name: IDENTIFIER;
IDENTIFIER: [a-zA-Z_] [a-zA-Z_0-9]*;
```

ANTLR генерує лексичний та синтаксичний аналізатор на основі одного і того ж набору правил прописаних у файлі *.g4. Проте для коректної роботи згенерованого коду потрібно завантажити пакунок antlr4-python3-runtime (pip install antlr4-python3-runtime).

На відміну від попередніх інструментів ANTLR підтримує можливість роботи з кирилицею на основі використання спеціалізованих кодів unicode. За офіційною документацією ANTLR символи кирилиці входять до діапазону: 0400-04FF. Для обробки синтаксичних та семантичних помилок використовується клас, що успадковується від ANTLRErrorListener, що в свою чергу є складовою частиною ANTLR. Метод синтаксичного аналізу – LL(*) (left to right leftmost derivation).

е) **Unicc** є також універсальним генератором аналізаторів. Цей інструмент написаний на мові C та підтримує можливість роботи з різними операційними системами. Згенерований синтаксичний аналіз працює за методом LALR(1).

Опис граматики вхідної мови для цього інструменту знаходиться у файлі «.pam». Граматики вхідної мови описуються за допомогою синтаксису схожого на БНФ. Файл опису граматик містить також параметр «!language», що вказує мову програмування для якої буде згенеровано аналізатори. Цей та інші параметри починаються з символу #, обов'язковим є позначення токенів для лексичного аналізу :

```
#!language      python;
#lexeme tabl tableList proj;
RelExpr$ ->  expression;
expression : expression oper expression  | '(' expression ')' | proj | tabl ;
oper -> "minus" | "join" | "union" | "semiminus" | "semijoin";
tabl -> 'A-Za-z_' | tabl 'A-Za-z_';
tableList -> tabl | tableList ',' tabl;
proj -> tabl '[' tableList ']';
```

Обробка помилок мови здійснюється за допомогою об'єкту ParseException. Цей об'єкт успадковується від Exception та окрім тексту помилки друкує можливі варіанти написання оператору. Завдяки використанню методу LALR(1) правила опису граматики дозволяють наявність рекурсії.

Unicc також генерує вихідний файл, що має назву вхідного файлу, але з типом відповідно мови програмування (*.py). Всередині згенерованого файлу знаходиться механізм обробки описаних граматик. На рис. 1 представлено цей механізм у вигляді кількох окремих класів, що відображені у вигляді діаграми.

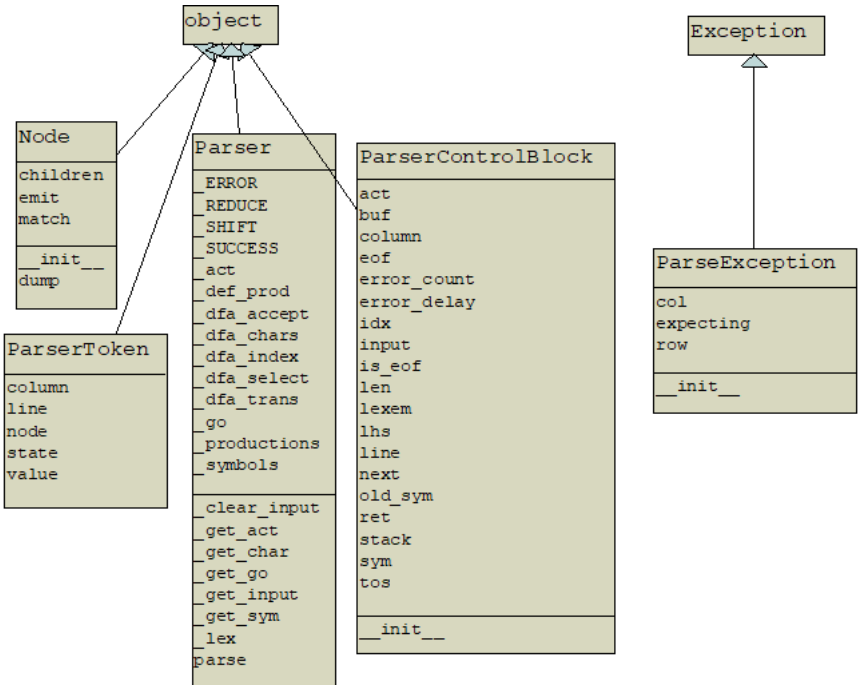

Рис. 1. Діаграма класів аналізаторів, генерованих Unicc

Обробник помилок Unicc генерується окремими класом, що успадковується від Exception, об'єкт якого можливо створити за необхідності.

Обрати оптимальний засіб генерації аналізаторів можливо на основі моделей якості описаних у стандарті ISO/IEC 2510. Цей стандарт пропонує дві моделі визначення якості: якість при використанні та якість продукту. З запропонованих моделей якості можливо вибрати наступні класи критеріїв, що є найбільш вагомими при роботі генератору аналізаторів: продуктивність, покриття контексту, задоволеність, переносимість, функціональна придатність, зручність використання.

Кожен представлений клас характеристик якості включає в себе підкласи, що можуть бути виражені у характеристиках програмного забезпечення. Генератори аналізаторів мають наступні характеристики, що належать до перелічених вище класів.

Продуктивність згенерованих аналізаторів – швидкість обробки вхідного рядку.

Покриття контексту, оскільки при аналізі засобів генерації важливим є підклас цього класу характеристики використання «повнота контексту». Представником цього підкласу у генераторах аналізаторів є підтримка кирилиці, що є важливим для створення DSL з підтримкою операторів українською мовою.

Задоволеність включає в себе підклас «комфорт», що відображає задоволеність користувача фізичним комфортом. При роботі DSL важливим чинником фізичного комфорту є способи пошуку помилки у введеному коді або інформативність тексту повідомлення про помилку, варіації виправлення, позиція помилки у вхідному рядку. На основі вище написаного представником цього підкласу можна вважати вбудований обробник помилок вхідної мови програмування.

Функціональна придатність. Вона відображає ступінь виконання ПЗ функцій відповідно до потреб користувача. Цей клас містить підклас «функціональна доцільність», що показує ступінь функціонального спрощення виконання певних задач та досягнення цілей. У контексті дослідження генераторів аналізаторів сюди необхідно віднести метод синтаксичного аналізу. Він впливає на багато факторів: можливість використання рекурсії у вхідних правилах, швидкість обробки вхідного рядку та ін.

Переносимість відображає ступінь простоти перенесення розроблених граматик до інших версій або іншого генератору аналізатора. Легкість перенесення граматик напряму залежить від способу опису синтаксису вхідної мови.

Зручність використання містить підклас керованість, що відображає наявність в системі атрибутів для простого керування та контролю. У генераторах аналізаторів для реалізації контролю використовується код мови програмування загального призначення, отже цей підклас характеристик залежить від способу додавання коду мови загального призначення до граматик вхідної мови.

На початкових стадіях розробки вибір найоптимальнішого генератору аналізаторів є складною задачею. Розв'язання цієї задачі можливе за допомогою використання методу аналізу ієрархій (MAI) [11].

MAI вимагає наявності чітко сформульованої цілі, критеріїв та альтернатив. У цьому випадку ціллю є вибір оптимального засобу генерації аналізаторів для предметно-орієнтованих мови програмування та критеріїв. Роль критеріїв тут виконують дві моделі якості ПЗ описані у стандарті ISO/IEC 2510: якість при використанні та якість продукту. Кожна модель має набір класів та підкласів характеристик якості ПЗ. Описані вище характеристики якості аналізаторів можливо відобразити у вигляді ієрархії (рис. 2).

Альтернативами виступають розглянуті вище інструменти генерації аналізаторів: Parglare, ANTLR, PLY, Pyparsing, Unicc.

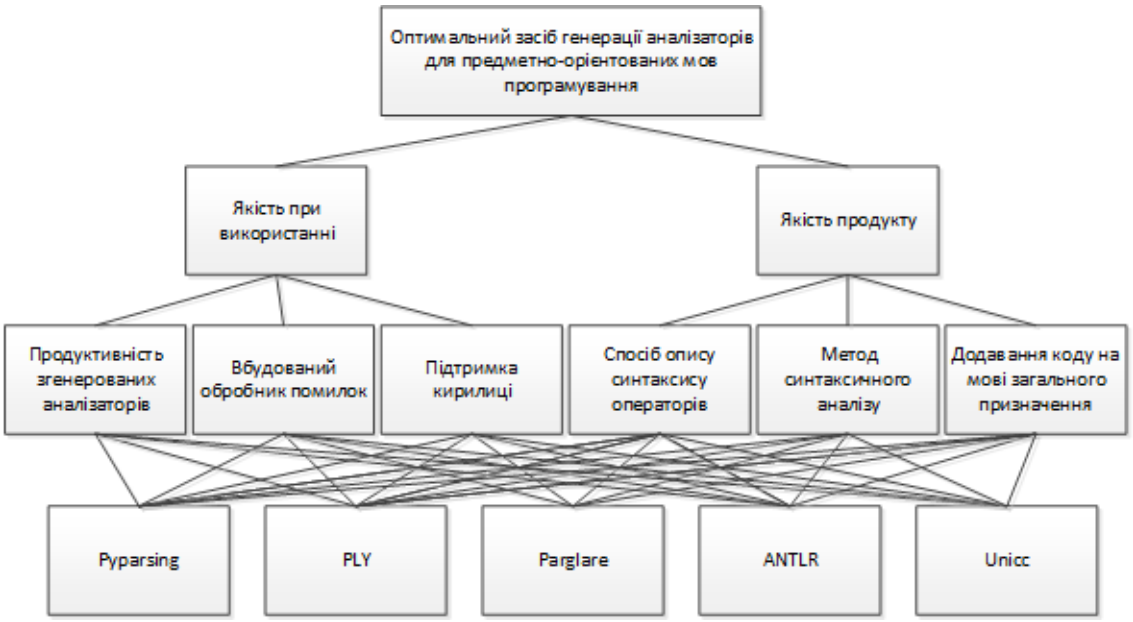

Рис. 2. Відображення ієрархічної моделі прийняття рішень для вибору оптимального генератора аналізаторів для DSL

Продуктивність згенерованих аналізаторів є важливою складовою оцінки розглянутих інструментів. Для перевірки аналізаторів використано апаратне забезпечення: Mobile DualCore Intel Pentium 2020M, 2400 МГц з об'ємом ОЗУ 4 ГБ. У якості мови тестового рядку було використано декілька виразів реляційної алгебри: Student [IdStudent], Student union cowokers, Student union (TradeUnion minus cowokers), (Student union (TradeUnion minus cowokers)) minus (Student minus TradeUnion). Під час виконання кожного оператору було проведено заміри часу обробки вхідного рядку та розраховано середній час виконання запиту (табл. 2).

Таблиця 2

Час обробки вхідного рядку згенерованими аналізаторами

	Student [IdStudent]	Student union cowokers	Student union (TradeUnion minus cowokers)	(Student union (TradeUnion minus cowokers)) minus (Student minus TradeUnion)	Середній час реакції
ANTLR	0,013	0,017	0,044	0,145	0,055
Unicc	0,009	0,009	0,02	0,025	0,016
PLY	0,05	0,116	0,068	0,088	0,081
Parglare	0,15	0,157	0,161	0,158	0,157
Pyparsing	0,028	0,028	0,033	0,1	0,047

Першим кроком реалізації MAI для вибору генератору аналізаторів є побудова матриці попарних порівнянь альтернатив (А) для кожного критерію (К). Матриця містить порівняльні оцінки експертів (m) відносно альтернатив, при цьому матриця завжди ж квадратною розміром n x n (1). Схожа матриця попарних порівнянь будується і для критеріїв.

$$\begin{array}{c} \\ \mathrm{A}_1 \\ \mathrm{A}_2 \\ \mathrm{A}_3 \\ \cdots \\ \mathrm{A}_i \end{array} \begin{array}{c} \begin{array}{cccccc} \mathrm{A}_1 & \mathrm{A}_2 & \mathrm{A}_3 & \ldots & \mathrm{A}_j \end{array} \\ \begin{vmatrix} m_{11} & m_{12} & m_{13} & \ldots & m_{1j} \\ m_{21} & m_{22} & m_{23} & \ldots & m_{2j} \\ m_{31} & m_{32} & m_{33} & \ldots & m_{3j} \\ \ldots & & \ldots & & \\ m_{i1} & m_{i2} & m_{i3} & \ldots & m_{ij} \end{vmatrix} \end{array} \quad (1)$$

Після виставлення оцінок потрібно розрахувати вектор локальних пріоритетів матриці, що базується на методі геометричного середнього (2).

$$w_i = \sqrt[n]{\prod_{j=1}^{n} m_{ij}} \quad (2)$$

Отриманий вектор необхідно нормувати. При реалізації MAI як правило використовують формулу (3).

$$v_i = \frac{w_i}{\sum_{i=1}^{n} w_i} \quad (3)$$

Таблиця 3

Локальні пріоритети критеріїв

Генератори аналізаторів	Критерії					
	Додавання коду на мові загального призначення	Вбудований обробник помилок	Метод синтаксичного аналізу	Підтримка кирилиці	Спосіб опису синтаксису операторів	Продуктивність згенерованих аналізаторів
	Вектори локальних пріоритетів					
Parglare	0,069	0,091	0,433	0,059	0,298	0,053
ANTLR	0,545	0,232	0,097	0,261	0,298	0,151
PLY	0,253	0,162	0,184	0,114	0,068	0,102
Pyparsing	0,073	0,075	0,074	0,054	0,036	0,176
Unicc	0,0599	0,439	0,211	0,513	0,298	0,517

MAI вимагає виконання дій (2) та (3) для кожного рівня ієрархії, що дає змогу відобразити локальні пріоритети (μ). Починаючи з другого рівня ієрархії для оцінки (λ) альтернативи необхідно застосувати формулу (4).

$$\lambda = v_{i1}\mu_1 + v_{i2}\mu_2 + v_{i3}\mu_3 + \cdots + v_{ij}\mu_j \quad (4)$$

Вибір оптимального засобу генерації аналізаторів залежить від двох критеріїв: Функціональність згенерованих аналізаторів, що відображає деякі характеристики вхідної мови програмування та функціональність генератору аналізаторів, що відображає особливості його роботи. У табл. 3 та табл. 4 відображено локальні та глобальні пріоритети критеріїв (λ).

Таблиця 4

Глобальні пріоритети для критеріїв

	Функціональність згенерованих аналізаторів	**Функціональність генератору**	λ
μ	0,667	0,333	
Parglare	0,062	0,251	0,125
ANTLR	0,234	0,281	0,250
PLY	0,117	0,152	0,129
Pyparsing	0,082	0,055	0,073
Unicc	0,504	0,178	0,396

Результатом використання MAI є матриця, що містить набір глобальних пріоритетів. На основі отриманої матриці можливо визначити альтернативу, що є найбільш оптимальною. Для вибору найоптимальнішого інструменту генерації аналізаторів предметно-орієнтованих мов програмування була сформована ієрархія з п'ятьма альтернативами та шістьма критеріями, за результатами використання MAI є найбільш оптимальною альтернативою Unicc.

Висновки та перспективи подальшого дослідження. Проведено аналіз сучасних засобів генерації синтаксичних аналізаторів при створенні предметно-орієнтованих мов програмування (DSL). Досліджено особливості інструментів цього типу для мови програмування python. Застосовано метод MAI для вибору найоптимальнішого засобу генерації аналізаторів. Побудовано ієрархію з використанням критеріїв якості ПЗ з

основних моделей якості ПЗ стандарту ISO/IEC 2510. Продемонстровано переваги інструменту Unicc для створення предметно-орієнтованих мов програмування: підтримка кирилиці, можливість додавання коду мови загального призначення, швидкість роботи згенерованих аналізаторів та спосіб опису синтаксису вхідних операторів.

Планується використати найбільш оптимальний генератор Unicc для генерації синтаксису мов моделювання графових структур, зокрема топологічної структури тепломережі. Мова будується як надбудова над універсальною графовою СКБД Neo4j з вживанням у синтаксисі специфічних термінів у цій предметній галузі. Так, вона має включати в себе оператори для маніпуляції топологічною структурою тепломережі описані кирилицею, що дасть змогу працювати з нею фахівцям у предметній галузі. Аналогічним чином можуть бути побудовані DSL для інших предметних галузей, що мають мережеву структуру, сферу застосування DSL більш широку, ніж предметна орієнтованість. Вони можуть бути і проблемно-орієнтованими, наприклад, як навігація кодом з метою визначення фрагментів для побудови юніттестів, визначення контуру дерева тощо.

СПИСОК ЛІТЕРАТУРИ

1. Фаулер М. Предметно-ориентированные языки программирования / Мартин Фаулер. – Киев: Диалектика-Вильямс, 2011. – 576 с.

2. Кандиба І. Використання програмної системи ANTLR для створення мови реляційної алгебри / І. Кандиба, М. Фісун // Одинадцята Міжнародна Науково-Практична Конференція ІОН-2018 : збірник праць конф., 22 – 25 травня 2018 р. – Вінниця : ВНТУ, 2018. – 343 с.

3. Эйфрем Э. Базы данных / Э. Эйфрем, Дж. Вебер, Я. Робинсон. – Москва ДМК Пресс, 2016. – 256 с.

4. Глибовець А. М. Інтелектуальні мережі / А. М. Глибовець, М. М. Глибовець, М. В. Поляков. – Дніпропетровськ : НаУКМА, 2014. – 462 с.

5. ANTLR as a Development Platform for the Series DSL for the Learning Process : (2019, 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems) [Електронний ресурс] / / I. Kandyba, Y. Davydenko, V. Panasyuk, A. Shved, M. Fisun // 2019, 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). – 2019. – Режим доступу : https://ieeexplore.ieee.org/document/8924354.

6. Компиляторы: принципы, технологии и инструментарий / [Ахо А. В., Лам М. С., Сети Р., Ульман Д. Д.]. – Київ : Диалектика, 2019. – 1184 с.

7. Dejanović I. Arpeggio: A flexible PEG parser for Python / I. Dejanović, G. Milosavljević, R. Vaderna // Knowledge-Based Syst. – 2016. – Vol. 95. – P. 71 – 74.

8. Parglare [Електронний ресурс] / Офіційний веб-портал інструменту Parglare. – Режим доступу: www.igordejanovic.net/parglare/stable.

9. Behrens S. Prototyping Interpreters Using Python Lex-Yacc. / S. Behrens // Dr Dobb's Journal-Software Tools for the Professional Programme: CMP Technology – 2004. – Vol. 95, № 3. – P. 30 – 35.

10. Parr T. The Definitive ANTLR 4 Reference / T. Parr. – United States, Texas : Pragmatic Bookshelf, 2014. – 322 p.

11. Методы системного анализа в задачах морских кластеров : монографія / [И. И. Коваленко, С. К. Чернов, А. В. Швед та ін.]. – Харьков : Новое слово, 2017. – 268 с.

Стаття надійшла до редакції 15.03.2021.
Стаття пройшла рецензування 23.03.2021.

Фісун Микола Тихонович **–** д. т. н., професор кафедри інженерії програмного забезпечення.

Кандиба Ігор Олександрович **–** викладач кафедри інженерії програмного забезпечення.

Горбань Гліб Валентинович **–** к. т. н., доцент кафедри інженерії програмного забезпечення.

Фаленкова Марина Володимирівна **–** викладач кафедри інженерії програмного забезпечення.

Чорноморський національний університет ім. П. Могили.