УДК 004.423.25-047.44

Фісун М. Т., Яблонська К. Ю.

РОЗРОБКА ПРЕДИКТИВНОГО СИНТАКСИЧНОГО АНАЛІЗАТОРА ДЛЯ СПРОЩЕНОЇ МОВИ ОПИСУ СЦЕНАРІЇВ

Статтю присвячено опису етапу побудови синтаксичного аналізатора проблемно-орієнтованої мови опису сценаріїв для роботи вітрової електростанції після її перетворення у неліворекурсивну лівофакторизовану. Запропоновано загальний підхід до генерації сценаріїв шляхом побудови проблемно-орієнтованих мов програмування і наведено приклад такої мови високого рівня. Описано етапи розробки компілятора, що здатен компілювати вказані моделі об'єктів у цільову програму залежно від сценарію.

Ключові слова: *контекстно-вільна граматика, нерекурсивний предиктивний аналіз, проблемно-орієнтована мова сценаріїв, синтаксичний аналізатор.*

Статья посвящена описанию этапа построения синтаксического анализатора проблемно-ориентированного языка описания сценариев для работы ветровой электростанции после ее преобразования в нелеворекурсивную левофакторизированную. Предложен общий подход к генерации сценариев путем построения проблемно-ориентированных языков программирования и приведен пример такого языка высокого уровня. Описаны этапы разработки компилятора, который способен компилировать указанные модели объектов в целевую программу в зависимости от сценария.

Ключевые слова: *контекстно-свободная грамматика, нерекурсивный предиктивный анализ, проблемно-ориентированный язык сценариев, синтаксический анализатор.*

This study describes a stage of the parser construction of the scenario problem-oriented language for a wind-power station after its transformation into a not left recursive and a left factorized language. The study presents a general approach to generate scenarios by constructing problem-oriented programming languages and gives an example of a such high-level language. The stages of the compiler development, which is able to compile the specified model objects into a target program, which is depended on the scenario, are described.

Key words: *context-free grammar, non-recursive predictive analysis, problem-oriented scenario language, parser.*

Постановка проблеми та її зв'язок із важливими науковими завданнями. В останні роки в системотехніці велика увага приділяється сценарному аналізу [1-2]. Сценарний підхід використовується для вирішення складних слабоформалізованих задач у системах динамічного та інтелектуального моделювання, а також для опису можливих варіантів поведінки систем, що містять активних учасників (гравців), які, не маючи інформації про стратегії протидіючих сторін, змушені їх генерувати на основі доступних їм знань [1]. У названих та інших роботах пропонується математичний апарат сценарного аналізу, оснований на модифікаціях методів експертного оцінювання. Таким чином, однією із актуальних проблем є генерація різноманітних варіантів сценаріїв для всебічного вивчення поведінки досліджуваного об'єкта.

Аналіз останніх досліджень та публікацій. У роботі [3] запропоновано загальний підхід до генерації сценаріїв шляхом побудови проблемно-орієнтованих мов програмування і наведено приклад такої мови високого рівня. Але цей напрям поки що не знайшов належного розвитку. Публікацій стосовно відповідних мов генерації сценаріїв та їх реалізації відсутні. У роботі [4] такий підхід запропоновано для опису сценаріїв подій вітрової електричної станції (ВЕС). Для практичної реалізації цієї ідеї необхідно розробити комплекс моделей розвитку подій того чи іншого об'єкта, що описуються проблемно-орієнтованою мовою опису сценаріїв (ПОМС), та розробити компілятор, який би на основі опису сценаріїв компілював вказані моделі в цільову програму моделювання об'єкта в цілому залежно від

заданого сценарію. Для ВЕС низку таких моделей наведено, наприклад, у працях [5; 6]. У [4] проаналізовано спробу розробки синтаксичного аналізатора («парсера») такого компілятора на основі генератора «парсерів» ANTLR [7]. *У результаті дослідження автори дійшли висновку про необхідність розробки «власного» синтаксичного аналізатора на основі відомих алгоритмів та з врахуванням особливостей проблемно-орієнтованої мови. Доцільність такого підходу можна обґрунтувати тим, що будувати другу, набагато більш складну частину компілятора – синтез моделей у цільову програму моделювання сценарію – буде легше при використанні «власного» синтаксичного аналізатора.*

Постановка завдання. Необхідно розробити синтаксичний аналізатор, як складову частину компілятора мови опису сценаріїв роботи ВЕС, представленої формальною граматикою у формі синтаксичних правил Бекуса-Наура [8]. При цьому бажано закласти гнучкі засоби для побудови процесора компіляції цільової програми на основі обраного варіанта предиктивного аналізу.

Методика побудови синтаксичного аналізатора. Оскільки ПОМС є проблемно-орієнтованою мовою, то вона є мовою більш високого рівня, ніж універсальні мови програмування, а її граматика є менш складною, оскільки в ній відсутні такі синтаксичні конструкції, як цикл, розгалуження тощо. Вона спирається на поняття предметної сфери, і її можна віднести до породжувального програмування [9]. Породжувальне програмування («generate programming») – це парадигма розробки програмних систем (ПС), заснована на моделюванні груп або окремих елементів ПС, таких, що при описі списку конкретних вимог до системи з цих елементів автоматично генерується кінцевий продукт [10].

Синтаксична простота мови опису сценаріїв дозволяє використовувати більш прості методи побудови синтаксичних аналізаторів. Одним із таких методів є нерекурсивний предиктивний синтаксичний аналіз. Цей метод можна застосовувати до LL(n)-граматик [8]. Для приведення граматики ПОМС, наведеної в [4], до LL(1)-граматики, треба усунути ліву рекурсію та провести ліву факторизацію. Після застосування відповідних алгоритмів [8] отримаємо LL(1)-граматику опису сценаріїв роботи ВЕС (спрощений варіант граматики з [4]), яку наведено нижче. При цьому термінальні символи виділені жирним шрифтом. Нумерація правил потрібна для побудови синтаксичного аналізатора.

(1) <сценарій> → **сценарій** <номер> **:** <перелік_ подій> **$**

(2) <перелік_подій> → <подія> <перелік_ подій2>

(4) <перелік_подій2> → ; <подія> <перелік_ подій2>

(5) <перелік_подій2> → **ε**

(6) <подія> → <дія_фактору>

(7) <дія_фактору> → <фактор>, <дія>

(9) <фактор> → <фактор_природ>

(12) <фактор_природ> → **швидкість_вітру =** <знач_шв>

(15) <номер_ВЕУ> → **ВЕУ** <номер>

(20) <дія> → <процедура> <номер_ВЕУ>

(21) <дія> → **повідомлення П** <номер>

(22) <дія> → **розрахувати_склад_ВЕС_ за_алгоритмом А** <номер>

(23) <процедура> → **увімкнути**

(24) <процедура> → **вимкнути**

(30 <номер> → **1**

(31) <номер> → **2**

(91) <знач_шв> → **4** <знач_шв2>

(92) <знач_шв2> → **1**

(93) <знач_шв2> → **2**

У назві LL(1) перше «L» означає перегляд вхідного потоку зліва направо, друге «L» – ліве породження, а «1» – перегляд одного символу (токена – якщо синтаксичному аналізу передує лексичний) з вхідного потоку на кожному кроці для прийняття рішення щодо наступних дій. До LL(1)-граматик належать такі граматики, в яких для будь-яких двох різних продукцій $A \to \alpha \mid \beta$ виконуються нижченаведені умови.

1. Не існує такого термінала a, для якого і α, і β породжує рядок, що починається з a.
2. Порожній рядок може породжувати тільки одну з продукцій α або β.
3. Якщо $\beta \to^* \varepsilon$, то α не породжує ні один рядок, що починається з термінала з множини *FOLLOW(A)* [8].

Першим підготовчим етапом побудови нерекурсивного синтаксичного аналізатора є побудова таблиці *FIRST* [8].

Якщо α – довільний рядок символів граматики, то *FIRST(α)* визначається як множина терміналів, з яких починаються рядки, виведені з α. Якщо $\alpha \to^* \varepsilon$, то $\varepsilon \in FIRST(\alpha)$, де позначення «$\to^*$» означає досягнення пустого символу «ε» за нуль або більше кроків.

Для того, щоб обчислити *FIRST(X)* для всіх символів X деякої граматики, необхідно застосовувати наступні правила доти, поки до жодної із множин *FIRST* не зможуть бути додані ні термінали, ні ε.

1. Якщо X – термінал, то $FIRST(X) = \{X\}$.
2. Якщо існує продукція $X \to \varepsilon$, додамо ε до *FIRST(X)*.
3. Якщо X – нетермінал й існує продукція $X \to Y_1Y_2...Y_k$, то помістимо a в *FIRST(X)*, якщо для якогось i $a \in FIRST(Y_i)$ і ε входить в усі множини $FIRST(Y_1),...,FIRST(Y_{i-1})$, тобто $Y_1...Y_{i-1} \to^* \varepsilon$. Якщо ε є у всіх $FIRST(Y_i)$, $i = 1...k$, то додаємо ε до *FIRST(X)*. Наприклад, усе, що перебуває в множині $FIRST(Y_1)$, є й у множині *FIRST(X)*. Якщо Y_1 не породжує ε, то більше нічого не додаємо до *FIRST(X)*, але якщо $Y_1 \to^* \varepsilon$, то до *FIRST(X)* додаємо $FIRST(Y_2)$ і т. д.

Побудова таблиці *FOLLOW(A)* для нетермінала A починається з визначення множини терміналів a, які можуть розташовуватися безпосередньо справа від A в деякої сентенціальній формі, тобто

множини терміналів a, таких, що існує породження виду $S \rightarrow^{*} \alpha A a \beta$ для деяких α й β.

Слід зауважити, що в процесі приведення між A і a можуть з'явитися символи, але вони породжують ε і в остаточному підсумку зникають. Якщо A може виявитися крайнім справа символом деякої сентенціальної форми, то $\$ \in FOLLOW(A)$.

Щоб обчислити *FOLLOW(A)* для всіх нетерміналів A, будемо застосовувати наступні правила доти, поки до жодної із множин *FOLLOW* не можна буде додати жодного символу.

1. Помістимо *$* в *FOLLOW(S)*, де S – стартовий символ, а *$* – правий обмежувач вхідного потоку.

2. Якщо існує продукція $A \rightarrow \alpha B \beta$, то всі елементи множини *FIRST(β)*, крім ε, поміщаються в множину *FOLLOW(B)*.

3. Якщо існує продукція $A \rightarrow \alpha B$ або $A \rightarrow \alpha B \beta$, де *FIRST(β)* містить ε $(\beta \rightarrow^{*} \varepsilon)$, то всі елементи із множини *FOLLOW(A)* містяться в множину *FOLLOW(B)*.

У таблиці 1 наведено результат побудови функції *FIRST* та *FOLLOW* для цієї граматики.

Наступним етапом є побудова таблиці переходів предикативного синтаксичного аналізатора.

1. Для кожної продукції граматики $A \rightarrow \alpha$ виконуємо кроки 2 і 3.

2. Для кожного термінала a з *FIRST(α)* додаємо $A \rightarrow \alpha$ в комірку *M[A,a]*.

3. Якщо в *FIRST(α)* входить ε, для кожного термінала b з *FOLLOW(A)* додамо $A \rightarrow \alpha$ в комірку *M[A,b]*. Якщо ε входить в *FIRST(α)*, а *$* – в *FOLLOW(A)*, додамо $A \rightarrow \alpha$ в комірку *M[A,$]*.

4. Зробимо кожну невизначену комірку таблиці M такою, що вказує на помилку.

Предиктивний синтаксичний аналізатор [8] має вхідний буфер, стек, таблицю розбору й вихідний потік. Вхідний буфер містить аналізований рядок із маркером її правого кінця – спеціальним символом. Стек містить послідовність символів граматики з *$* на дні. Завжди стек містить стартовий символ граматики безпосередньо над символом *$*. Таблиця розбору являє собою двомірний масив *M[A,a]*, де A – нетермінал, a – термінал або символ *$* (табл. 2).

Таблиця 1

Результат побудови функції FIRST та FOLLOW

№	Нетермінал A	Множина FIRST(A)	Множина FOLLOW(A)
1	**<сценарій>**	{сценарій}	{ $ }
2	**<номер>**	{1, 2}	{ : }
3	**<перелік_подій>**	{швидкіть_вітру}	{ $ }
4	**<фактор>**	{швидкіть_вітру}	{ , }
5	**<фактор_природ>**	{швидкіть_вітру}	{ , }
6	**<дія_фактору>**	{швидкіть_вітру}	{ ; }
7	**<подія>**	{швидкіть_вітру}	{ ; }
8	**<перелік_подій2>**	{ε, «;»}	{ $, ε }
9	**<дія>**	{повідомлення, розрахувати_склад_ВЕС_за_алгоритмом, увімкнути, вимкнути}	{ ; }
10	**<процедура>**	{увімкнути, вимкнути}	{ ВЕУ }
11	**<номер_ВЕУ>**	{ВЕУ}	{ ; }
12	**<номер>**	{1, 2}	{ : }
13	**<знач_шв>**	{4}	{ , }
14	**<знач_шв2>**	{1, 2}	{ , }

Синтаксичний аналізатор керується програмою, що працює таким чином. Програма розглядає X – символ на вершині стека, і a – поточній вхідний символ. Ці два символи визначають дії синтаксичного аналізатора. Є такі варіанти:

1. Якщо $X ::= a ::= \$$, синтаксичний аналізатор припиняє роботу й повідомляє про успішне завершення розбору.

2. Якщо $X ::= a \neq \$$, синтаксичний аналізатор знімає зі стеку X і переміщує покажчик вхідного потоку до наступного символу.

3. Якщо $X ::= \varepsilon$, синтаксичний аналізатор знімає зі стеку X, але не переміщує покажчик вхідного потоку до наступного символу.

4. Якщо X являє собою нетермінал, програма розглядає запис *M[X,a]* з таблиці розбору M. Цей запис являє собою або X-продукцію граматики, або запис про помилку. Якщо, наприклад, *M[X,a] = {X ::= UVW}*, синтаксичний аналізатор заміщує X на вершині стеку на *WVU* (з U на вершині стека). Ми вважаємо, що як вихід синтаксичний аналізатор просто виводить використану продукцію, але, звичайно ж, тут може виконуватися будь-який необхідний код.

5. Якщо *M[X,a] = error*, синтаксичний аналізатор викликає програму відновлення після помилки.

Таблиця 2

Таблиця предиктивного аналізу для заданої граматики

Нетермінал	Вхідний символ													
	сценарій	**швидкість_вітру**	**;**	**ВЕУ**	**повідомлення**	**розрахувати_склад_ВЕС_за_алгоритмом**	**увімкнути**	**вимкнути**	**4**	**1**	**2**	**П**	**А**	**$**
<сценарій>	1													
<перелік_подій>		2	2											2
<подія>		6												
<дія_фактору>		7												
<фактор>		9												
<фактор_природ>		12												
<номер_ВЕУ>				15										
<перелік_подій2>			4											5
<дія>					21	22	20	20						
<процедура>							23	24						
<номер>										30	31			
<знач_шв>									91					
<знач_шв2>										92	93			

Примітка. На перетині комірок «Нетермінал» та «Вхідний символ» заходиться номер правила.

Поведінка синтаксичного аналізатора може описуватися його конфігураціями, які дають вміст стека й вхідний потік, що залишився.

Висновки та напрями подальших досліджень. Закладено основу створення предикативного синтаксичного аналізатора компілятора мови опису сценаріїв для роботи вітрової електростанції: початкову мову перетворено до LL(1)-граматики та побудовано таблицю переходів ПСА. Надалі планується програмна реалізація аналізатора, дослідження й розробка синтаксично-орієнтованого компілятора проблемно-орієнтованих мов типу мови опису сценаріїв ВЕС.

ЛІТЕРАТУРА

1. Коваленко И. И. Системные технологии генерации и анализа сценариев [Текст] / И. И. Коваленко, А. П. Гожий. – Николаев : ЧГУ им. Петра Могилы, 2006. – 160 с.
2. Самойленко Л. И. Разработка методологии оценки сценариев в задачах планирования космической деятельностью [Текст] / Л. И. Самойленко // Проблеми управления и информатики. – 2005. – № 5. – С. 143–156.
3. Андрейчиков А. В. Интеллектуальные информационные системи [Текст] : учеб. пособие / А. В. Андрейчиков, О. Н. Андрейчикова. – М. : Финансы и статистика, 2004. – 422 с.
4. Фісун М. Т. Генерація сценаріїв подій з використанням формальних граматик [Текст] / М. Т. Фісун, К. Ю. Яблонська, О. Б. Шуневич. – Миколаїв : Збірник наукових праць. – НУК, 2013. – № 3. – С. 86–91.
5. Медиковський М. О. Багатокритеріальний метод оцінювання ефективності вітроенергетичної установки [Текст] / М. О. Медиковський, О. Б. Шуневич // Вісник інженерної академії України. –2010. – № 3-4. – С. 240–245.
6. Медиковський М. О. Модифікована мережа Петрі для аналізу структури вітрової електростанції [Текст] / М. О. Медиковський, О. Б. Шуневич // Motrol – Commission of motorization and energetic in agriculture – Lublin, 2012. – Vol. 14. No. 4. – P. 178–184.
7. Офіційний сайт проекту ANTLR [Електронний ресурс]. – Режим доступу : http://www.antlr.org.
8. Ахо А. В. Компилятори: принципы, технология и инструменты [Текст] / А. В. Ахо, Р. Сети, Д. Д. Ульман. – М. : Вильямс, 2003. – 768 с.
9. Лаврищева Е. М. Современные методы программирования: возможности и инструменты [Текст] / Е. М. Лаврищева // Проблемы программирования. – 2006. № 2-3. – С. 60–74.
10. Чернецки К. Порождающее программирование. Методы, инструменты, применение [Текст] / К. Чернецки, У. Айзенекер. – М. : «Питер», 2005. – 730 с.

 Дата надходження статті до редколегії 05.05.2014 р.

ФІСУН Микола Тихонович – доктор технічних наук, професор, завідувач кафедри інтелектуальних інформаційних систем Чорноморського державного університету імені Петра Могили.

Коло наукових інтересів: інтелектуальні інформаційні системи та CASE-засоби, OLAP-технології, Textmining, формальні граматики.

ЯБЛОНСЬКА Катерина Юріївна – аспірант, викладач кафедри інтелектуальних інформаційних систем Чорноморського державного університету імені Петра Могили.

Коло наукових інтересів: інтелектуальні інформаційні системи, формальні граматики, нечітка логіка, сценарний аналіз.