УДК 681.3.068

В.В. Гулаков, Т.В. Ковалюк

МОДЕЛЮВАННЯ ОПИСУ ПРЕДМЕТНОГО СЕРЕДОВИЩА ЗАСОБАМИ СИНТАКСИЧНО-ОРІЄНТОВАНОЇ ТРАНСЛЯЦІЇ

Вступ

Об'єктно-орієнтована розробка програмних систем та їх компонентів базується на застосуванні об'єктно-орієнтованих моделей, технологій та інструментальних засобах, що їх підтримують. Об'єктно-орієнтовані методології дозволяють зрозуміти різні аспекти та властивості програмної системи, що розробляється, і згодом суттєво покращити її реалізацію, тестування, супровід, розробку нових версій та модифікацію. До об'єктно-орієнтованої технології відносяться як до засобу подолання складності, що властива реальним систем. Ця складність обумовлена проблемами опису властивостей та поведінки об'єктів предметного середовища. Опис предметного середовища задається природною мовою, яку можна вважати вхідною мовою на етапі об'єктно-орієнтованого аналізу. Синтаксично-орієнтована обробка речень вхідної мови формує на виході модель програмної системи. Перетворення вхідної мови у певну вихідну зводиться до побудови деякого транслятора, який для будь-яких перетворень речень вхідної мови використовує структуру цього речення. В коло ідей, на яких базується ця концепція, входять ідеї формальних граматик, та семантичних обчислень для символьних ланцюжків. Саме ці питання і покладені в основу розробки, що розглядається.

Аналіз проблеми моделювання предметного середовища

Проектування програмної системи починається з аналізу вимог, яким вона має задовольняти. Етап формування вимог передбачає дослідження бізнес-процесів, аналіз предметного середовища та розробку моделей цієї системи, які формально описують систему у вигляді об'єктів що її складають, та відношень між ними. Об'єкти є екземплярами класів, які характеризуються спільністю атрибутів, властивостей та поведінки. Визначення класів предметного середовища найчастіше здійснюється шляхом вербального аналізу опису постановки прикладної задачі (технічного завдання, анкет, інтерв'ю тощо). Зокрема, кожному іменнику, що зустрічається в постановці задачі, може відповідати клас певних об'єктів. Цей етап є найбільш відповідальним в технології програмування та найменш формалізованим. Усі інструментальні засоби, що підтримують процес автоматизації програмування, визначення класів залишають розробнику. Сам процес класифікації включає велику кількість рутинної роботи, яку бажано звести до мінімуму. Одним з можливих шляхів розв'язання цієї задачі є спроба застосувати лексичний та синтаксичний аналіз опису предметного середовища для машинного розуміння природної мови.

© В.В. Гулаков, Т.В. Ковалюк, 2008

 ISSN 1562-9945

Метою даної роботи є вирішення задач з прикладного моделюванню предметного середовища, а саме розробка програмних засобів, що дозволяють автоматизувати процес лексичного та синтаксичного аналізу опису предметного середовища шляхом створення мови, близької до природної англійської, яка дозволяє задати об'єктно-орієнтовану модель предметного середовища. Результатом є створення транслятору, який аналізує вхідні дані, що є описом предметного середовища природною мовою, і формує оголошення класів мовою програмування.

Розробка базується на методах і алгоритмах лексичного аналізу із застосуванням регулярних виразів [1] та синтаксичного аналізу з використанням алгоритму низхідного синтаксичного аналізу [2].

Основні положення теорії синтаксично-орієнтованої трансляції

Трансляція включає кілька фаз: лексичний, синтаксичний і семантичний аналіз. У результаті трансляції отримується певна структура даних, яка потім використовується для виконання конкретних завдань.

Розглянемо основні терміни, що застосовуються в теорії трансляції [1,2]. Під алфавітом розуміємо скінченну множина символів. Послідовність символів з алфавіту складає ланцюжок w мови. Множина ланцюжків над певним алфавітом означує формальну мову L. Формальна граматика є сукупністю правил, що застосовуються для завдання формальної мови. Контекстно вільною формальною граматикою G називається множина $\{N, T, S, P\}$, де N – скінченна непорожня множина нетермінальних символів (нетерміналів), кожний з яких задає множину слів формальної мови; T – скінченна непорожня множина термінальних символів (терміналів), кожний елемент котрої задає слово формальної мови на множині T; $S \in N$ – початковий символ, P – скінченна непорожня множина продукцій (або правил перетворення) вигляду $A \to \alpha$, $A \in N$, $\alpha \in N \cup T$.

Якщо дві та більше продукцій мають однакову ліву частину, то вони можуть бути об'єднані за допомогою символу | (операції "або"). Послідовність кроків для отримання ланцюжка α_n з α_0 називається виведенням. У мовах, породжених контекстно вільними граматиками, виведення можна зображати графічно за допомогою орієнтованих кореневих дерев, які називаються деревами виведення або деревами синтаксичного розбору.

Трансляція починається з лексичного аналізу програми. Лексика мови – це правила правопису слів, таких, як ідентифікатори, константи, службові слова, коментарі тощо. Лексичний аналіз (lexical analysis) – це лінійне сканування вхідної програми, при якому символи групуються в лексеми (lexemes) – послідовності, що мають певне сукупне значення. На основі лексем генеруються токени (tokens), найменші лексичні одиниці, які характеризуються класом (наприклад, ідентифікатори, константи, ключові слова тощо) та атрибутами (які можуть бути відсутніми), що формують значення токена.

Синтаксис мови – це правила складання речень мови з окремих слів. Синтаксичним аналізом або розбором (parsing) називається групування токенів у граматичні фрази. Синтаксису притаманний принцип рекурсивності правил побудови. Тому граматичну структуру виразу можна представити у вигляді дерева синтаксичного розбору.

Семантика мови – це зміст, який закладається в кожну конструкцію мови. Зокрема, у цьому випадку семантичний аналіз полягає у побудові об'єктної моделі. Процес трансляції зображено на рисунку 1:

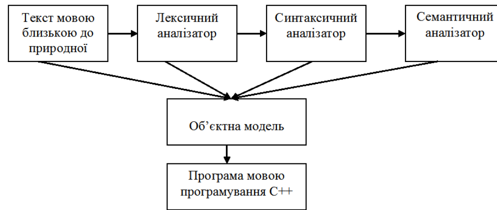

Рис. 1 – Процес трансляції природної мови

Усунення лівої рекурсії

Граматика називається ліворекурсивною, якщо у неї є продукції виду $A \to A\alpha$, $A \in N$, $\alpha \in A \cup N \cup \{\varepsilon\}$, де ε – пустий ланцюжок символів. Якщо є правила виду $A \to A\alpha|\beta$, вони створюють явну ліву рекурсію. Позбавитися від неї можна замінивши кожне із таких правил на $A \to \beta|\beta A'$, де $A' \to \alpha|\alpha A'|\varepsilon$.

В загальному випадку правила виду $A \to A\alpha_1|A\alpha_2|\ldots|A\alpha_n|\beta_1|\beta_2|\ldots|\beta_m$ слід замінити на $A \to \beta_1 A'|\beta_2 A'|\ldots|\beta_m A'$, де $A' \to \alpha_1 A'|\alpha_2 A'|\ldots|\alpha_m A'|\varepsilon$. Однак така заміна не усуває ліву рекурсію, викликану кількома породженнями. Її можна усунути за допомогою наступного алгоритму.

Алгоритм усунення лівої рекурсії

Вхідні дані представлені граматикою G без циклів, тобто породжень виду $A \overset{+}{\Rightarrow} A$ та без ε продукцій, тобто продукцій виду $A \to \varepsilon$. Вихідними даними є еквівалентна граматика G' без лівої рекурсії. Сам алгоритм складається з таких кроків. Нехай заданий $A_1, A_2, \ldots, A_n$ – набір усіх нетерміналів граматики G. Для усіх нетерміналів $A_i, i = 1, 2, \ldots, n$ та $A_j, j = 1, 2, \ldots, i-1, j > i$ замінимо продукції виду $A_i \to A_j\gamma$ продукціями $A_i \to \delta_1\gamma|\delta_2\gamma|\ldots.|\delta_k\gamma$, де $Aj \leftarrow \delta_1|\delta_2|\ldots.|\delta_k$. Дії повторюватимуться доти, поки існують продукції виду $A_i \to A_j\gamma$.

Ліва факторизація

Лівою факторизацією (left factoring) називається таке перетворення граматики, що вона може бути використана для предикативного аналізу. Суть лівої факторизації полягає в можливості зміни продукцій та їх

 ISSN 1562-9945

застосування до нетермінала А в момент однозначного вирішення питання щодо виду продукції. Опишемо основні кроки алгоритму виконання лівої факторизації вхідної граматики G. Для кожного нетермінала A знаходимо найдовший префікс α, спільний для двох або більше альтернатив. Замінюємо всі продукції виду $A \rightarrow \alpha\beta_1|\alpha\beta_2|\ldots|\alpha\beta_n|\gamma_1|\gamma_2|\ldots|\gamma_m$ на $A \rightarrow \alpha A'|\gamma_1|\gamma_2|\ldots.|\gamma_m$ та $A' \rightarrow \beta_1|\beta_2|\ldots|\beta_n$. Дії повторюватимемо, поки існують альтернативи, що мають спільний префікс. Вихідними даними є еквівалентна лівофакторизована граматика G'.

Нерекурсивний предикативний аналіз

Схематично нерекурсивний предикативний аналізатор зображено на рисунку 2. Вхідний буфер містить вхідний ланцюжок та маркер його кінця $. У стеку знаходяться дані, що визначають обрані правила перетворення. На початку стек містить символ $ на дні та стартовий символ S на верхівці. Таблиця розбору M встановлює відношення для будь-якої пари символів нетермінал A та термінал α. Елемент $M[A,\alpha]$ будується на основі граматики та може містити або правило, або повідомлення про помилку.

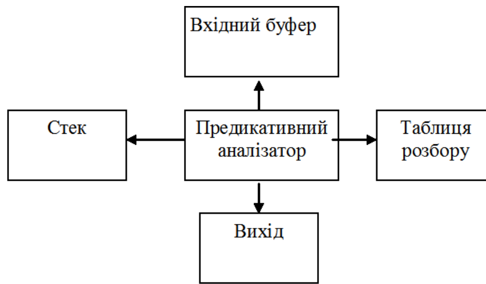

Рис. 2 – Структура нерекурсивного предикативного аналізатору

Вхідні дані представлені вхідним ланцюжком w та таблицею предикативного розбору M для граматики G. Вихідними даними є ліве породження w, якщо $w \in L(G)$, або повідомлення про помилку. Одна ітерація алгоритму має такі кроки. У стеку розміщуємо $\$S$, де S – стартовий символ граматики, $\$$ – маркер завершення, в результаті чого S знаходитиметься на вершині стеку. У вхідному буфері розміщуємо ланцюжок символів $w\$$. Нехай X – символ на вершині стеку, α – поточний символ у вхідному буфері. Поки стек не пустий, перевіряємо, чи його символ X є терміналом, чи маркером завершення $. Якщо X – термінал і дорівнюватиме α, X вибирається із стеку та стає доступним наступний символ α. Якщо X – нетермінал, генеруватиметься повідомлення про помилку та завершуватиметься робота. Якщо у таблиці розбору елемент $M[X,\alpha]$ містить продукцію виду $X \rightarrow Y_1Y_2\ldots Y_k$, зі стеку вибирається X та додається у стек $Y_kY_{k-1}\ldots Y_1$ з Y_1 на верхівці стеку. Отримуємо та виводимо продукцію $X \rightarrow Y_1Y_2\ldots Y_k$.

Цей алгоритм може бути застосований для будь-якої граматики, однак таблицю предикативного аналізу за допомогою алгоритму, наведеного нижче, можна отримати лише на основі граматики класу $LL(1)$. Граматики $LL(1)$ повинні бути однозначними та не можуть бути ліворекурсивними. В позначенні $LL(k)$. значення k визначає кількість необхідних для прийняття рішень символів, що переглядаються. За замовчуванням приймається значення 1. Саме $LL(1)$ граматики найчастіше застосовують на практиці.

Розглянемо дві множини символів [5]. Позначимо через $FIRST(\alpha)$ множину всіх термінальних символів, з яких починається ланцюжки, виведені з α. Тут α –множина термінальних символів, що формують ланцюжки в деякій мові $L = \{\alpha | S \rightarrow \alpha, a \in T\}$. Побудувати $First(\alpha)$ можна, використовуючи правила:

1. $FIRST(\alpha) = \{a\}$, якщо перший символ α термінальний із значенням a.

2. До $First(\alpha)$ додаватиметься пустий ланцюжок ε, якщо існує ε продукція $\alpha \rightarrow \varepsilon$.

3. До $First(\alpha)$ додаватиметься $FIRST(Y_j), j = 1, \ldots, k$, якщо існує продукція $X \rightarrow Y_1 Y_2 \ldots Y_k$ та $Y_1 Y_2 \ldots Y_{j-1} \overset{*}{\Rightarrow} \varepsilon$.

Через $FOLLOW(A)$ позначимо об'єднання всіх множин початкових термінальних символів, які можуть з'являтися в ланцюжках, що стоять справа від нетермінала A у виведенні $S \overset{*}{\Rightarrow} \alpha A a \beta$, де S – стартовий символ. Побудувати множину $FOLLOW(A)$ можна, використовуючи такі правила:

1. Кінцевий символ \$ додаватиметься до $FOLLOW(S)$, якщо справа від нетерміналу A не з'являються ланцюжки символів .

2. Для всіх продукцій виду $B \rightarrow \alpha A \beta$ всі елементи $FIRST(\beta)$ додаватимуться до $FOLLOW(A)$ за винятком пустого ланцюжка ε.

3. Для всіх продукцій виду $B \rightarrow \alpha A$ та для продукцій $B \rightarrow \alpha A \beta$, де $\beta \overset{*}{\Rightarrow} \varepsilon$ всі елементи із $FOLLOW(B)$ додаватимуться до $FOLLOW(A)$.

На основі введених множин $FIRST(\alpha)$, $FOLLOW(A)$ побудуємо алгоритм створення таблиці предикативного аналізу. Вхідними даними вважатимемо граматику G. Вихідними даними буде таблиця предикативного аналізу M. Основні кроки алгоритму такі. Визначити $M[A, \alpha] \Rightarrow \{A \rightarrow \alpha\}$ для кожної продукції $A \rightarrow \alpha$ граматики G та кожного термінала α із $FIRST(\alpha)$. Якщо $\varepsilon \subset FIRST(\alpha)$, до всіх терміналів $b \in FOLLOW(A)$, визначити $M[A, b] = \{A \rightarrow \alpha\}$. Якщо $\varepsilon \subset FIRST(\alpha)$, $\$ \subset FOLLOW(A)$, визначити $M[A, \$] = \{A \rightarrow \alpha\}$.

Складність алгоритму нерекурсивного предикативного низхідного синтаксичного аналізу контекстно вільної граматики класу $LL(1)$ дорівнює $O(n)$, де n – довжина ланцюжка символів граматики. Отже, алгоритм працює з лінійною складністю, оскільки побудова допоміжних таблиць відбувається лише один раз і не береться до уваги.

 ISSN 1562-9945

Граматика, що застосовується

```
data_manager = method >> separator >> data_manager
| attribute >> separator >> data_manager
| inheritance >> separator >> data_manager | epsilon_p;
inheritance=word>>separator>>str_p(L"is")>>separator>>separator>>ch_p(L'.');
by_using = str_p(L"by") >> separator >> str_p(L"using");
contains = (str_p(L"consists") >> separator >> str_p(L"of"))
| str_p(L"contains") | str_p(L"has");
attribute =(
word >> separator >> contains >> separator >> word >> separator >>(
(ch_p(L'(') >> separator >> word >> separator >> ch_p(L')'))
| epsilon_p) ) >>separator >> ch_p(L'.');
method =(
word >> separator >> str_p(L"can") >> separator >> word >> separator >>(
 (by_using | (word >> separator >> by_using)) >> separator >> (
parameter >> separator >> *(',' >> separator >> parameter >> separator) >>
 ((str_p(L"and") >> separator >> parameter >> separator) | epsilon_)))
|epsilon_p) >> separator >> ch_p(L'.');
parameter =(word >> separator >> (
 (ch_p(L'(') >> separator >> word >> separator >> ch_p(L')')) | epsilon_p));
word = (+range_p(L'A', L'z'));
separator = +space_p | epsilon_p;
```

Засоби реалізації

Для реалізації задач синтаксичного аналізу опису предметного середовища з метою автоматизація створення програмного коду, було розроблено програмний комплекс OOP_GEN мовою C++. Для зберігання даних застосовувалися контейнери STL. При програмуванні логіки задіяні алгоритми із бібліотеки boost. Програмний код не залежить від платформи: він може бути скомпільований для роботи під будь-якою операційною системою. Програма OOP_GEN спроектована відповідно до концепції Unix way. Для реалізації лексичного та синтаксичного аналізу використовується бібліотека boost::spirit. Синтаксичний аналізатор побудований на базі алгоритму низхідного синтаксичного аналізу. Тобто для визначення мови використовується ліворекурсивна граматика. Граматика задається прямо у вихідному коді програми в форматі, подібному до EBNF (Extended Backus-Naur Form). Причому визначення граматики є правомірним кодом на C++: для цього використовуються шаблонні вирази (Expression Templates). Отже, код програми цілком відповідає стандарту ISO/IEC 14882. Варто відзначити, що бібліотека сумісна із STL.

Опис контрольного прикладу

Програма отримує вхідні дані із стандартного потоку введення рядків тексту, що описують бізнес процеси предметного середовища. Зчитані дані записуються у файл формату Xml. На виході у вказаному каталозі генерується каркас програми мовою програмування C++. Каркас програми містить оголошення класів, що є описом об’єктів предметного середовища.

Підтримуються конструкція додавання методу класу:

<class_name> **can** <method_name> **by using**
<argument1_type> **[(**<argument1_name>**)], ... [and]**
<argumentn_type> **[(**<argumentn_name>**)].**

Наприклад, речення "Machine can ride by using wheel and motor" матиме вигляд "Machine has color" після додавання атрибута

<class_name> has <attribute_name>

Формат операції успадкування:

<class_name1> is <class_name2>

У результаті операції успадкування отримуємо речення "Car is machine".

Висновки

В результаті об'єктно-орієнтованого аналізу та об'єктно-орієнтованого моделювання була побудована об'єктно-орієнтована модель, що відображає сутність задачі. Розроблений синтаксично-орієнтований транслятор, який виконує аналізує опис предметного середовища, поданого мовою, близької до природної. Особливу увагу приділено технічному документуванню програмного забезпечення: всі класи, методи та атрибути детально задокументовані та, із використанням doxygen, згенерована технічна документація у форматі HTML.

Планується і надалі розвивати проект. Зокрема необхідно розширювати граматику. Планується реалізувати можливість генерувати вихідні дані в інших форматах.

Література

1. Ахо А., Сети Р., Ульман Дж. Компиляторы: принципы, технологии и инструменты. – М.: Издательский дом "Вильямс", 2003. – 768 с.
2. Рейуорд-Смит В. Дж. Теория формальных языков. Вводный курс. – М.: Радио и связь, 1988. – 128 с.
3. Ларман К. Применение UML и шаблонов проектирования. – М.: Издательский дом "Вильямс", 2001. – 496 с. INTERNATIONAL STANDARD. ISO/IEC FDIS 14882. Programming languages – C++.
4. Вирт Н. Алгоритмы+структуры данных =программы. – М.: Мир, 1985. – 406 с.

Получено 22.04.2008