Анотації

Досліджено метод тришарової декомпозиції предикатів, який поєднує декомпозицію на рівні предикатів та на рівні змінних. Наведено застосування методу на прикладі словозміни повних прикметників російської мови (для трьох змінних). Проведено аналіз методу розшарування предиката та методу побудови реляційних мереж як засобів формального подання довільних відношень.

Исследован метод трехслойной декомпозиции n-арных предикатов, объединяющий декомпозицию отношений на уровне предикатов и на уровне переменных. Представлено использование метода на примере словоизменения прилагательных русского языка (для трех переменных). Проведен анализ метода расслоения предиката и метода построения реляционных сетей как средств формального представления произвольных отношений.

The method of pair predicate three-layer decomposition which unites decomposition of relations on the level of predicates and variables is researched. The usage of this method on example of Russian adjectives' changes (for three variables) is presented. The method of predicate's exfoliation and the method of relational systems' building as means for formal presentation of random relations are analyzed.

УДК 004.4'413

Г. Ф. ДЮБКО,
кандидат технічних наук,
професор кафедри програмного забезпечення електронних обчислювальних машин,
Харківського національного університету радіоелектроніки,
В. В. ГОЛЯН,
кандидат технічних наук,
доцент кафедри програмного забезпечення електронних обчислювальних машин
Харківського національного університету радіоелектроніки,
Р. О. ГЛУЩЕНКО,
магістр
Харківського національного університету радіоелектроніки

СИНТАКСИЧНИЙ АНАЛІЗ ІЗ ВИКОРИСТАННЯМ АНОТОВАНИХ ГРАМАТИК

Синтаксичний аналіз, або розбір, як його ще називають, – це процес, у якому досліджується ланцюжок лексем і встановлюється, чи задовольняє він структурні умови, сформульовані у визначенні синтаксису мови. Розбір – це одна з найбільш важливих фаз компіляції. За сукупністю синтаксичних правил для деякої формальної мови можна автоматично побудувати синтаксичний аналізатор, який буде перевіряти, чи має вихідна програма синтаксичну структуру, обумовлену цими правилами.

В основі будь-якого синтаксичного аналізатора лежить алгоритм аналізу. Найбільш прості алгоритми використовують перебиральні стратегії [1], що призводить до набагато більших витрат апаратних ресурсів, зокрема часу. Проблема побудови ефективних синтаксичних аналізаторів актуальна донині, оскільки досить часто створюються нові мови програмування, мови розмітки (RDF, OWL) тощо, і для них необхідно створювати синтаксичні аналізатори з оптимальними тимчасовими характеристиками.

За допомогою певних теоретичних знань можна вручну створити такий аналізатор, але це завдання досить трудомістке й потребує багатьох обчислень, перевірок і інших операцій. Однак процес створення синтаксичного аналізатора можна автоматизувати, тим самим значно скоротивши витрати тимчасових ресурсів. **Мета** даної статті – проаналізувати існуючі методи й підходи до завдання автоматизації побудови синтаксичних аналізаторів і запропонувати альтернативне й ефективне рішення.

Огляд існуючих методів вирішення проблеми. Сьогодні розроблені генератори синтаксичних аналізаторів, що підтримують найпоширеніші методи граматичного розбору.

Одним із найбільш ранніх інструментів подібного типу стало поєднання YACC та LEX. Ці утиліти з'явилися на початку 1970-х рр. для системи UNIX. YACC розшифровується як Yet another compiler-compiler (ще один компілятор компіляторів). Він був розроблений С. Джонсоном і застосовувався для створення багатьох компіляторів.

Для створення парсерів використовується YACC, для лексичного аналізу – утиліта LEX. Ці програми добре інтегруються одна з одною. Для них створена спеціальна метамова з метою опису конструкцій аналізованї мови. Вона дозволяє описувати формальні граматики у формі, що нагадує форму Бекуса – Наура.

YACC дозволяє працювати із граматиками класу LALR; цей клас граматик дуже широкий, і конструкції багатьох мов програмування можуть бути описані за допомогою цих граматик. За вхідною граматикою YACC генерує програму мовою C, яка є синтаксичним аналізатором для вихідної граматики. Цей аналізатор реалізує алгоритм «перенесення-згортання», і є досить ефективним і швидкодіючим. YACC має кілька механізмів рішення конфліктів неоднозначності (ЗГОРТАННЯ-ЗГОРТАННЯ, ПЕРЕНЕСЕННЯ-ЗГОРТАННЯ).

Відсутність графічного інтерфейсу й зручних засобів розробки ускладнюють роботу із цією системою. Користувач змушений використовувати найпримітивніші засоби для створення компілятора, такі, як текстовий редактор і командний процесор. Відсутність засобів «налагодження» або перевірки граматики ускладнює її написання, тому що розробка коректної формальної граматики є складним процесом. Людина може не врахувати деяких конструкцій мови або «випадково» додати нові. Для багатьох виявиться незручним той факт, що можна згенерувати синтаксичний аналізатор лише для мови C. На сьогодні програмісти використовують багато мов програмування й напевно захочуть використовувати C++, Java, Delphi (Pascal), Visual Basic, Smalltalk тощо. До переваг цього пакета слід віднести високу надійність (з його допомогою були створені сотні компіляторів), наявність великої бібліотеки граматик, написаних для YACC–LEX (користувач може вибрати необхідну мову з уже наявних і модифікувати її під свої потреби) і, звичайно ж, його ціну – дана утиліта поставляється безкоштовно.

Генератор синтаксичних аналізаторів на базі анотованих граматик. Граматика – це математична система, що визначає мову. Одночасно вона є обладнанням, яке надає ланцюжкам мови корисну структуру. У граматиці, що визначає мову L, використовуються дві кінцеві непересічні безлічі символів – безліч нетермінальних символів, яка часто буде позначатися буквою N, і безліч термінальних символів, позначувана $\sum$. З термінальних символів утворюються слова (ланцюжка) обраної мови. Нетермінальні символи служать для породження слів мови L [2].

Творцем граматик уважають Н. Хомський, який запропонував класифікувати граматики за видом їх продукцій: загальні, контекстно-залежні, контекстно-вільні й регулярні. Із чотирьох класів граматик ієрархії Хомського клас контекстно-вільних граматик найбільш важливий з погляду додатків до мов програмування й компіляції. За допомогою Кс-Граматики можна визначити більшу частину синтаксичної структури мови програмування. Крім того, вона є основою різних схем задання перекладів.

Маючи граматику, можна побудувати висновок якого-небудь термінального ланцюжка. Завдання синтаксичного аналізу протилежне: за даним термінальному ланцюжкові потрібно відновити її висновок з аксіоми. Синтаксичний аналіз можна інтерпретувати і як відновлення синтаксичного дерева даного вхідного ланцюжка. Для цього дерева відомі листки (термінальні символи) і корінь (аксіома). Міркування про відновлення синтаксичного дерева еквівалентні міркуванням про відновлення висновку ланцюжка з аксіоми: відновлення кожного кроку висновку відповідає додаванню групи вузлів у синтаксичне дерево. У зв'язку з цим розрізняють дві стратегії – згортання й розгортання. При згортанні висновок (або синтаксичне дерево) будується від термінальних символів до аксіоми, при розгортанні – від аксіоми до термінальних символів.

Найпростіші алгоритми синтаксичного аналізу використовують перебиральні стратегії, що призводить до набагато більших витрат часу. Але кількість перебирань можна скоротити за допомогою керування висновком, яке полягає у використанні додаткової інформації. Така інформація може мати вигляд керуючих таблиць, які допомагають обирати потрібну продукцію на кожному кроці висновку. Для складання й використання керуючих таблиць мову описують спеціальними типами граматик. Часто граматики, що не є спеціальними, можна звести до таких формальними перетвореннями.

Існує клас граматик, для яких можливий детермінований висхідний аналіз із читанням ланцюжка символів ліворуч-праворуч. Вони називаються LR(k)-граматиками. Буква L у назві граматик указує на те, що вхідний ланцюжок читається ліворуч (left) праворуч,

R – на те, що видається правий (right) розбір, а k позначає кількість символів передогляду, тобто число вхідних символів, на які алгоритм «заглядає наперед». Мовою LR(k) називається мова, яку можна згенерувати за допомогою LR(k)-граматики. Якщо потрібен тільки один символ передогляду, граматику й мову відносять до класу LR(1).

Розбір вхідного ланцюжка для LR(k)-граматик здійснюється за алгоритмом типу «перенесення-згортання». У ході розбору алгоритм повинен ухвалювати рішення трьох типів. По-перше, перед кожним кроком необхідно визначати, чи переносити вхідний символ у магазин, чи виконувати згортання. Це рішення полягає в з'ясуванні того, де у ланцюжку, що виводиться праворуч, знаходиться правий кінець основи.

Друге й третє рішення будуть ухвалюватися після того, як правий кінець основи буде локалізований. Як тільки стає відомо, що у вершині магазину утворилася основа, необхідно знайти в магазині її лівий кінець. Далі, коли основа виділена, потрібно знайти відповідний нетермінал, яким слід її замінити. У загальному випадку необхідний певний механізм для визначення того, яким саме нетерміналом потрібно замінити виділену основу.

Для LR(k)-граматики завжди можна побудувати детермінований правий аналізатор. Тобто граматика належатиме до класу LR(k)-граматик у тому випадку, якщо для довільного правого висновку $S = \alpha_0 \Rightarrow_r \alpha_1 \Rightarrow_r \alpha_2 \Rightarrow_r \ldots \Rightarrow_r \alpha_m = z$ у кожному правому ланцюжку α_i, читаючи його зліва направо, можна виділити основу й однозначно визначити, яким нетерміналом її потрібно замінити, дійшовши при цьому не більш ніж до k-го символу, розташованого праворуч від правого кінця цієї основи.

Припустімо, що $\alpha_{i-1} = \alpha A \omega$ і $\alpha_i = \alpha\beta\omega$, де β – основа ланцюжка α_i. Припустимо далі, що $\alpha\beta = X_1X_2\ldots Xr$. Якщо Кс-Граматика є LR(k)-граматикою, то можна стверджувати наступне:

– знаючи $X_1X_2\ldots Xj$ і перші k символів ланцюжка $Xj_{+1}\ldots Xr\omega$, можна не сумніватися, що правий кінець основи не буде досягнутий доти, поки не стане j = r;

– знаючи не більш k символів ланцюжка, завжди можна визначити, що це – основа і її треба згорнути до A;

– якщо $\alpha_{i-1} = S$, то можна з упевненістю сказати про те, що вхідний ланцюжок потрібно допустити.

При цьому на кожному кроці ланцюжок символів, на які алгоритм «заглядає вперед», складається з k або менше термінальних символів.

Для визначення LR(k)-граматики для початку слід увести визначення поповненої граматики.

Визначення. Нехай G = (N, (, P, S) – Кс-Граматика. Тоді поповненою граматикою, отриманою з G, будемо називати граматику G(= (N ({S(}, (, P ({S((S}, S(). Інакше кажучи, це просто G з новим початковим правилом S((S, де S(– новий початковий символ (аксіома), що не належить N. Нехай S((S – нульове правило граматики G(, а інші правила пронумеровані числами 1, 2, 3, …, p. Це початкове правило додається для того, щоб згортання, у якому використовується нульове правило, можна було інтерпретувати як сигнал про те, що ланцюжок допускається.

Тепер дамо визначення LR(k)-граматики.

Визначення. Нехай G = (N, (, P, S) – Кс-Граматика й G(= (N(, (, P(, S() – отримана з неї поповнена граматика. Тоді будемо називати G LR(k)-граматикою для k (0, якщо з умов

$$S' \Rightarrow^*_{G'r} \alpha A \omega \Rightarrow_{G'r} \alpha\beta\omega; \quad (1)$$

$$S' \Rightarrow^*_{G'r} \gamma B x \Rightarrow_{G'r} \alpha\beta y; \quad (2)$$

$$\text{Firstk}(\omega) = \text{Firstk}(y) \quad (3)$$

випливає, що $\alpha A y = \gamma B x$ (тобто $\alpha = \gamma$, A = B і x = *y*). Граматика G називається LR(k)-граматикою, якщо вона LR(k)-граматика для деякого k (0.

Вищеописане визначення говорить про те, що якщо $\alpha\beta\omega$ і $\alpha\beta y$ – правий ланцюжок поповненої граматики, у якому Firstk(ω) = Firstk(*y*), і $A \to B$ – останнє правило, використане в правому висновку ланцюжка $\alpha\beta\omega$, те правило $A \to B$ повинне використовуватися також у правому розборі при згортанні $\alpha\beta y$ до $\alpha A y$. Оскільки A дає β незалежно від ω, то LR(k)-умова говорить про те, що в Firstk(ω) утримується інформація, достатня для визначення того, що $\alpha\beta$ за один крок виводиться з αA. Тому ніколи не може виникнути сумнівів щодо того, як згорнути черговий правий ланцюжок поповненої граматики. Працюючи з LR(k)-граматикою, завжди відомо, чи допустити даний вхідний ланцюжок, чи продовжувати розбір. Якщо початковий символ S не зустрічається в правих частинах продукцій граматики, то у визначенні LR(k)-граматики замість S(можна брати S, а саме G = (N, (, P, S) буде LR(k)-граматикою, якщо із трьох умов

$$S' \Rightarrow *_r \alpha A \omega \Rightarrow_r \alpha\beta\omega; \quad (4)$$

$$S' \Rightarrow *_r \gamma B x \Rightarrow_r \alpha\beta y; \quad (5)$$

$$\text{Firstk}(\omega) = \text{Firstk}(y) \quad (6)$$

випливає, що $\alpha A y = \gamma B x$. Але даним визначенням можна користуватися не завжди, оскільки якщо початковий символ зустрічається в правій частині деякої продукції граматики, то не можна сказати точно, чи буде досягнуто кінець вхідного ланцюжка і її потрібно допустити, потрібно продовжувати розбір [1].

Граматика G є LR(k)-граматикою тоді й тільки тоді, коли для неї можна побудувати LR(k)-аналізатор. Для здійснення висхідного синтаксичного аналізу згортанням із використанням LR(1)-граматики насамперед необхідно зробити дві операції – анотування граматики й побудову керуючої таблиці синтаксичного аналізу, виходячи з даних, отриманих після проведення анотування.

Опишемо даний метод більш детально. Анотування граматики відбувається в такий спосіб. Конфігурацією називається позиція в правій частині продукції перед першим символом, після останнього символу або між будь-якими двома символами. У конфігурації може бути кілька позицій.

Анотування починається уведенням початкового стану 1 (позиція перед першим символом правої частини продукції, де ліва частина є аксіомою), стан 2 ставиться перед другим символом цієї правої частини тощо. Припустімо, що отримано сполучення, де *i* – стан, X – символ граматики. Якщо X – нетермінальний символ, то стан *i* з'являється на початку всіх правих частин продукцій, де X є лівою частиною. Якщо X – термінал, то за ним ставиться послідовно стан $_{ixi+1}$. Конфігурації в різних продукціях, що відповідають одному стану, нерозрізнені з погляду синтаксичного аналізатора. Припустімо, що ми маємо конфігурацію $_{i1,i2,i3}Xj_{1,j2}$, де X – нетермінальний символ в одній із продукцій, і i1 відповідає j1 (тобто можливий перехід з i1 в j1), а i2, i3 – j2.

В іншій продукції цей же символ анотований як $_{i1}Xj_3$ ($_{j1,j2<j3}$). Тоді в першій продукції анотація символу X повинна бути доповнена як $_{i1,i2,i3}Xj_{1,j2,j3}$, і i1 відповідає j1,j3, а i2,i3 – j2. Стан j, що міститься наприкінці продукції, відповідає згортанню. Якщо між перенесенням і згортанням є конфлікт у стані j, він вирішується в такий спосіб. Коли зі стану j існує перехід у стан j1 з термінальним символом x, то здійснюється перенесення. Інакше виконується згортання. Конфлікт згортань між собою вирішується через стан, тому що кожен зі «згорнутих» станів пов'язаний тільки з однією продукцією.

Тепер розглянемо побудову таблиці синтаксичного аналізу. Таблиця синтаксичного аналізу є прямокутною, кожному стану аналізатора (це завжди кінцеве число) відповідає один рядок, а кожному терміналу й нетермінальному граматики відповідає один стовпець.

На початку процесу синтаксичного аналізу аналізатор знаходиться в стані 1, а вхідним символом є перший уведений символ. Кожний крок аналізу визначається позицією таблиці, відповідної до поточного стану, і вхідним символом. Позиція таблиці може належати до одного із двох типів:

– позиція перенесення виду Sm, що змушує аналізатор виконати дію перенесення і змінити поточний стан на стан m;

– позиція згортання виду Rn, що змушує аналізатор виконати дію згортання, використовуючи продукцію n.

Порожні позиції таблиці відповідають синтаксичним помилкам в уведенні. Процес заповнення таблиці відбувається в такий спосіб. По черзі зчитуються анотовані символи правої частини кожної продукції. При цьому згідно з анотаціями в таблицю заносяться значення виду Sm (перенесення). Припустімо, що існував зчитаний символ з анотаціями виду $_{i1,i2,i3}Xj_{1,j2,j3}$, де i1 відповідає j1, i2 – j2, а i3 – j3. Тоді в таблиці в стовпці, що відповідає до символу X, для стану i1 з'явиться значення «Sj1», для стану i2 – значення «Sj2» тощо.

Коли зчитано останній символ правої частини кожної із продукцій, у таблицю необхідно ввести стани згортання. Для цього потрібно обчислити так звані символи попереднього бачення для того нетермінального символу, у який буде проводитися згортання, тобто для лівої частини продукції, яка аналізується в даний момент. Символами попереднього бачення для даного нетермінала будуть символи, які можуть випливати відразу за ним, згідно з продукціями граматики. Далі, коли всі символи попереднього бачення відомі, у стовпці таблиці, що відповідають кожному із символів попереднього бачення, для стану, що перебуває наприкінці поточної продукції граматики, заносяться значення виду Rn (згортання). Наприклад, для символу $_{i1}Xj_1$ останнім символом продукції з номером k у таблиці для стану j1 і символу X буде внесене значення «Rk».

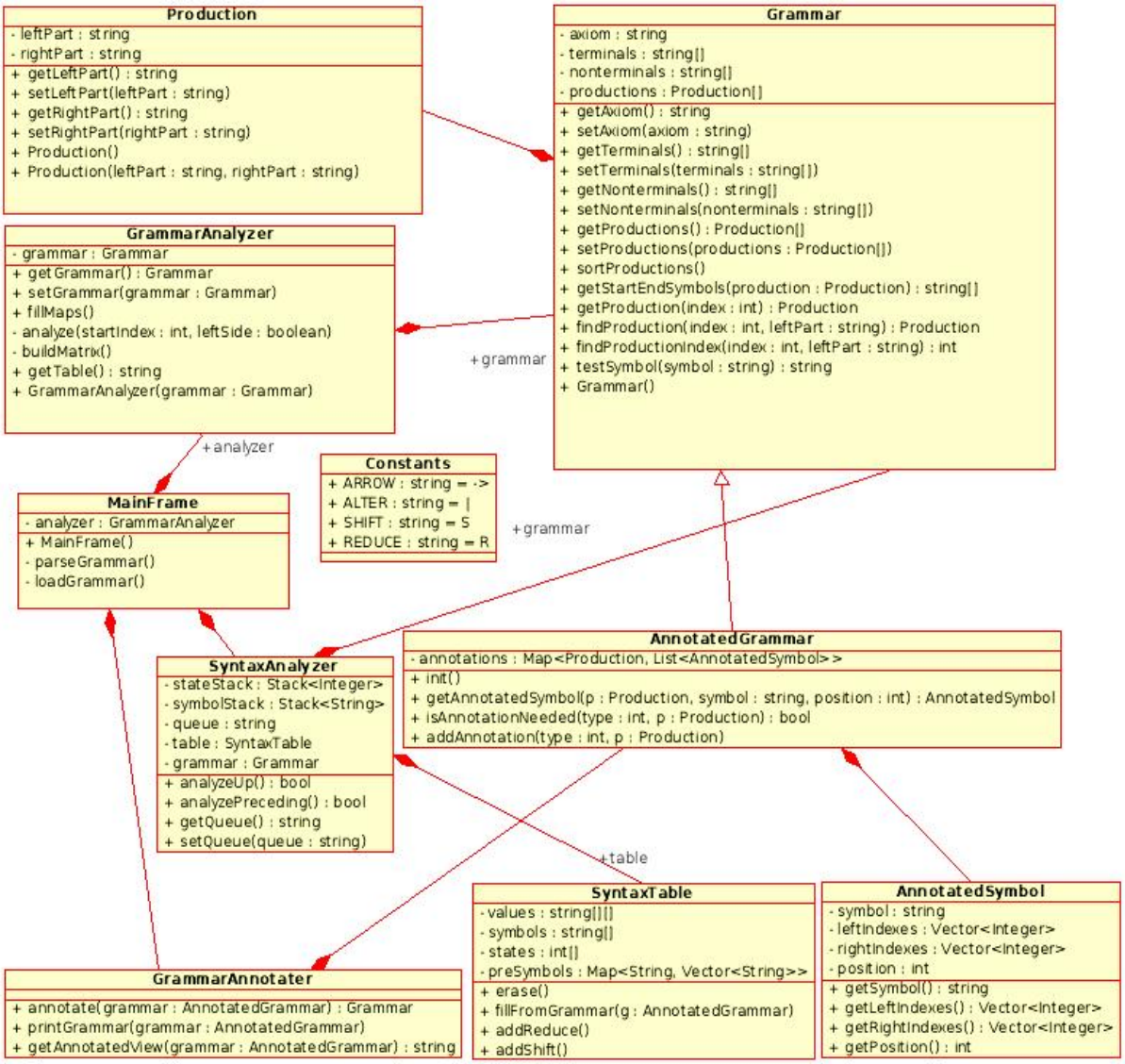

Рис. 1. Umla-діаграма класів додатка

Синтаксичний аналіз із використанням готової таблиці здійснюється за таким алгоритмом. На кожному етапі синтаксичного аналізу аналізатор перебуває в одному з кінцевих станів згідно з рядками таблиці, і цей стан плюс вхідний символ (або символ попереднього бачення, або тільки що згорнутий нетермінал) визначають елемент у таблиці. Припускаючи відсутність синтаксичних помилок, цей елемент – дія перенесення або згортання.

На початку синтаксичного аналізу аналізатор перебуває в стані 1, а вхідним символом є перший символ аналізованого ланцюжка. Якщо позиція таблиці визначає дія перенесення, необхідно виконати такі операції:

– символ, що відповідав стовпцю, у якому перебуває дана позиція таблиці, заноситься в стек символів;

– аналізатор переходить у стек, який визначається позицією перенесення, і цей стан заноситься в стек станів;

– якщо вхідний символ є терміналом, він ухвалюється, і вхідним символом стає наступний термінал ланцюжка (або маркер кінця).

Якщо згідно з таблицею випливає дія згортання, виконуються такі операції:

– зі стека символів видаляється n символів, а зі стека станів видаляються n станів, де n – число символів у правій частині продукції, що фігурує у згортанні;

– аналізатор переходить у стан на вершині стека станів;

– вхідний символ стає символом у лівій частині продукції, визначеної в позиції згортання [2].

Даний метод реалізований у розробленому додатку, який фактично є генератором синтаксичних аналізаторів для LR(1)-граматик зі

зручним графічним інтерфейсом. Як мова реалізації була обрана мова Java. Завантаження граматики здійснюється з текстового файлу, складеного за певними правилами. Далі користувач додатка може відразу одержати анотований вид граматики й готову таблицю синтаксичного аналізу в наочному виді. Також є текстове поле для введення ланцюжків символів для виконання синтаксичного аналізу.

У розробці додатка використаний підхід об'єктно-орієнтованого програмування, а також розроблені й використані структури даних, що дозволяють реалізувати методи анотування й створення таблиці синтаксичного аналізу оптимальним образом.

Для представлення анотованої граматики на програмному рівні був створений клас Annotatedgrammar, що містить таблицю відповідностей продукцій (клас Production) і списку анотованих символів (клас Annotatedsymbol). Ще один клас Grammarannotater містить реалізацію методу анотації граматики, тобто працює із класом Annotatedgrammar.

Таблиця синтаксичного аналізу представлена у вигляді класу Syntaxtable і містить метод, який ухвалює як параметр екземпляр класу Annotatedgrammar, що й заповнює внутрішню структуру даних, використовуючи описаний вище метод заповнення таблиці синтаксичного аналізу. Ядро додатка – клас Syntaxanalyzer – ухвалює на вхід ланцюжок символів і, відповідно до алгоритму розбору, що використовують таблицю синтаксичного аналізу, ухвалює рішення щодо коректності уведеного ланцюжка. На рис. 1 показана коротка Umla-діаграма класів додатка, на якій більш наочно вимальовується внутрішня структура продукту.

Висновки. Переваги розробленого продукту – це зручний інтерфейс, кросплатформність, використання нововведень мови Java й ефективних структур даних, що дозволяють підвищити швидкість роботи додатка, використання оптимальних безперервних методів аналізу. На відміну від поширеного інструментального засобу YACC, даний додаток не використовує принцип подвійної трансляції.

До недоліків можна віднести відсутність генерації вихідного коду синтаксичного аналізатора, як це робить YACC.

Даний продукт може застосовуватися в будь-яких галузях, пов'язаних із розробкою нових мов програмування, трансляторів, компіляторів, як допоміжний засіб, що дозволяє скоротити тимчасові витрати на створення синтаксичного аналізатора.

У майбутньому до продукту планується додати можливість генерації вихідного коду аналізатора мовою Java, а також розширити можливості опису граматик у файлі (планується використання універсальної мови розмітки XML).

Література

1. Ахо А. Теория синтаксического анализа, перевода и компиляции. Т. 1. Синтаксический аналіз / А. Ахо, Дж. Ульман. – М. : Мир, 1978. – 650 с.

2. Хантер Р. Основные концепции компиляторов / Робин Хантер ; пер. с англ. – М. : Вильямс, 2002. – 256 с.

Надійшла до редколегії 11.03.2010

Анотації

Розглянуто методи автоматичної генерації синтаксичних аналізаторів з алгоритмом синтаксичного аналізу згорткою для LR(1)-граматик, що полягають у використанні методів анотування граматики й автоматичної побудови таблиці синтаксичного аналізу.

В данной статье рассмотрены методы автоматической генерации синтаксических анализаторов с алгоритмом синтаксического анализа сверткой для LR(1)-грамматик, состоящие в использовании методов аннотирования грамматики и автоматического построения таблицы синтаксического анализа.

Methods of automatic generation of parsers with algorithm of parse by convolution for LR (1)-grammatik, consisting in use of methods of annotation of grammar and automatic construction of the table of parse are considered in this article.