УДК 681.31

А.Н. Афанасьев, Р.Ф. Гайнуллин

# МЕТАКОМПИЛЯТОР ДИАГРАММНЫХ ЯЗЫКОВ[1]

***Афанасьев Александр Николаевич,*** *доктор технических наук, окончил радиотехнический факультет Ульяновского политехнического института. Директор Института дополнительного дистанционного образования Ульяновского государственного технического университета. Имеет статьи, монографии, изобретения в области разработки и применения автоматизированных систем. [e-mail: a.afanasev@ulstu.ru].*

***Гайнуллин Ринат Фаязович,*** *аспирант кафедры «Вычислительная техника» УлГТУ, окончил факультет информационных систем и технологий УлГТУ. Программист ООО «Эквид». Имеет статьи, изобретения в области разработки и применения автоматизированных систем. [e-mail: r.gainullin@gmail.com].*

Аннотация

Предложен метакомпилятор графических языков на основе автоматных RV-грамматик. Метакомпилятор позволяет строить автоматную грамматику диаграммного языка и генерировать необходимые операции с внутренней памятью. Построенная грамматика обладает полнотой контроля. Рассмотрен алгоритм построения грамматики на примере диаграммы активности языка UML.

Ключевые слова: RV-грамматика, метакомпиляция, диаграммные языки.

***Alexander Nikolaevich Afanasev,*** *Doctor of Engineering; graduated from the Faculty of Radioengineering of Ulyanovsk Polytechnical Institute; Director of Institute of Additional distant education; author of articles, monographs, and inventions in the field of development and application of automated systems. e-mail: a.afanasev@ulstu.ru.*

***Rinat Faiazovich Gainullin,*** *Graduate Student at the Deartment of Computer Engineering of Ulyanovsk State Technical University; graduated from the Faculty of Information Systems and Technologies of Ulyanovsk State Technical University; a programmer of Ekvid LLC; author of articles, inventions in the field of development and application of automated systems. e-mail: r.gainullin@gmail.com.*

Abstract

A metacompiler of graphics languages on the basis of finite-state RV grammars is offered. The metacompiler allows to build the finite-state grammar of chart language and to generate necessary operations with an internal memory. The constructed grammar possesses a monitoring completeness. The grammar building algorithm exemplified by the UML language activity chart is considered.

Key words: RV-grammar, metacompiler, diagram languages.

## Введение

Для анализа визуальных диаграммных языков (UML, IDEFx, DFD, SDL, ARIS eEPC, ER и др.) и их синтаксически управляемого перевода используются графические грамматики. В [1] предложена автоматная RV-грамматика, обеспечивающая полноту контроля топологии диаграмм и обладающая линейной временной характеристикой анализа.

Метакомпилятор используется на этапе создания схемы RV-грамматики. Основная цель его применения – упростить работу по составлению правил грамматики, сохраняя эффективность парсинга диаграммы. На входе метакомпилятор получает контекстно-свободное описание диаграммного языка, на выходе выдает описание автомата RV-грамматики в формате XML.

Метакомпилятор по своим характеристикам похож на компиляторы компиляторов. Современные компиляторы компиляторов позволяют по синтаксическому описанию грамматики $G$ построить синтаксический анализатор предложений языка $L(G)$. Большинство компиляторов компиляторов принимают на вход БНФ-нотацию языка [2] (нотация в Форме Бэкуса-Наура), либо специализированную модификацию БНФ. Примерами служат следующие продукты:

- YACC (Yet Another Compiler Compiler) – стандартный генератор синтаксических анализаторов в Unix-системах [3];
- ANTLR (Another Tool For Language Recognition) – позволяет генерировать синтаксические анализаторы на различных целевых языках [4];

1 Работа поддержана грантом РФФИ № 13-07-00483.

• GOLD – использует конечные автоматы для анализа и LALR(1) алгоритм для разбора [5];

• Spirit – часть библиотеки Boost, предназначенная для написания парсеров напрямую в C++ тексте программы в виде, близком к форме Бэкуса-Наура [6];

• LAPG (lexical analyzer and parser generator) – позиционируется как более «дружелюбная» замена YACC, поддерживает 5 различных конечных языков [7];

• Coco/R – входным является грамматика в формате РБНФ, сканер работает как конечный автомат, парсер использует методологию нисходящего рекурсивного спуска [8].

Данные компиляторы компиляторов работают для текстовых языков. В статье рассматривается подход по созданию метакомпилятора для диаграммного языка.

**Принципы работы метакомпилятора**

Процесс метакомпиляции состоит из нескольких этапов [9]:

- лексический разбор описания диаграммного языка;
- синтаксический разбор и построение дерева разбора;
- анализ дерева разбора и построение промежуточной структуры грамматики в терминах языка спецификаций;
- трансляция – преобразование внутреннего представления в термины RV-грамматики;
- оптимизация и минимизация;
- сохранение таблиц RV-грамматики в формате XML.

В процессе анализа описания диаграммного языка метакомпилятор формирует набор паросочетаний между терминальными символами языка, строит таблицу переходов автомата RV-грамматики и генерирует набор операций с памятью для контроля соответствия вложенности нетерминальных символов и блоков с количеством входов или выходов больше одного.

Отделение описания RV-грамматики от ее анализатора позволяет использовать полученный XML-файл на разных платформах вне зависимости от того, с каким приложением интегрирован анализатор.

Метакомпилятор представляет собой консольное приложение, написанное на C# с использованием GOLD Parser Builder для разбора языка метаописаний. Структура взаимодействия его компонентов в составе проекта показана на рисунке 1.

Разбор диаграмм происходит с помощью RV-грамматик. Ниже приведено определение грамматики (согласно [1]) и пример построения грамматики для диаграммы активности языка UML.

### RV-грамматика

$RV$-грамматикой языка $L(G)$ называется упорядоченная пятерка непустых множеств $G = \left(V, \Sigma, \tilde{\Sigma}, R, r_0\right)$,

где $V = \left\{v_l, l = \overline{1, L}\right\}$ – вспомогательный алфавит;

$\Sigma = \left\{a_t, t = \overline{1, T}\right\}$ – терминальный алфавит графического языка;

$\tilde{\Sigma} = \left\{\tilde{a}_t, t = \overline{1, \tilde{T}}\right\}$ – квазитерминальный алфавит;

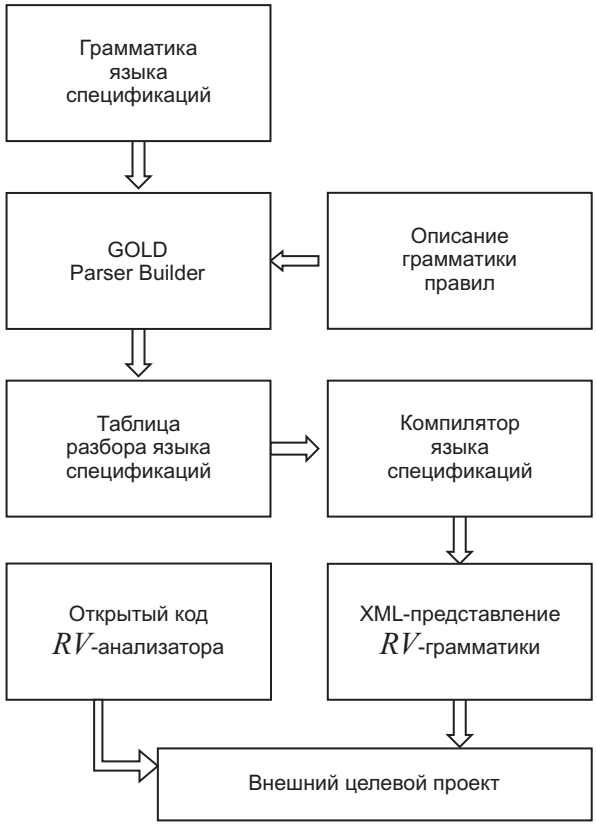



Рис. 1. Структура метакомпилятора диаграммных языков

$R = \left\{r_i, i = \overline{0, I}\right\}$ – схема грамматики $G$;

$r_0 \in R$ – аксиома $RV$-грамматики.

Продукция $P_{ij} \in r_i$ имеет вид $P_{ij}$:

$$\tilde{a}_t \xrightarrow{\Omega_\mu\left[W_\gamma(\gamma_1,\ldots,\gamma_n)\right]} r_m,$$

где

$W_\gamma(\gamma_1, \ldots, \gamma_n)$ – $n$-арное отношение, определяющее вид операции над внутренней памятью в зависимости от $\gamma \in \{0, 1, 2, 3\}$;

$\Omega_\mu$ – оператор модификации, определенным образом изменяющий вид операции над памятью, причем $\mu \in \{0, 1, 2\}$;

$r_m \in R$ – имя комплекса продукции–преемника.

$RV$-грамматика является автоматом, изменяющим состояния под действием входных символов. При переходе из состояния в состояние выполняется операция над внутренней памятью.

### Правила описания синтаксиса языка диаграмм активности UML

Описание спецификаций диаграммных языков состоит из правил, содержащих имя нетерминального символа, следующего после зарезервированного слова Rule, перечисления набора терминальных и нетерминальных символов, входящих в правило, а также описания отношений между терминальными и нетерминальными символами внутри правила.

На рисунке 2 приведены конструкции диаграмм активности и метаописания их правил. Рамка, в которую заключены составляющие правил, обозначает границу сущности правила и иллюстрирует соответствующие входы и выходы.

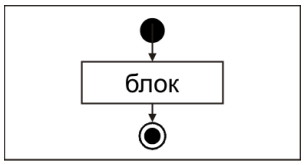
Первое правило описывает диаграмму в целом. Далее правила описывают множество допустимых блоков в диаграмме активности. Последнее правило «Продолжение» рекурсивно ссылается на себя, благодаря чему может содержать несколько параллельных ветвей «Блоки».

На основе приведенных правил генерируются терминальные и квазитерминальные символы.

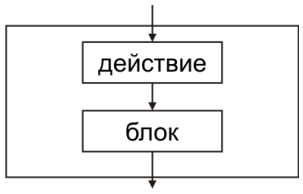
### Синтез операций с внутренней памятью

Из приведенных выше примеров видно, что правила имеют общую структуру. У каждого правила «Блоки» есть вход и выход, следовательно, для контроля контекста можно использовать стековый механизм. В начале обработки правила в стек заносится первый элемент. В конце обработки из стека извлекается этот элемент. Если путей в правиле более одного, то возникает вопрос, когда заканчивается обработка одного правила и начинается обработка следующего.

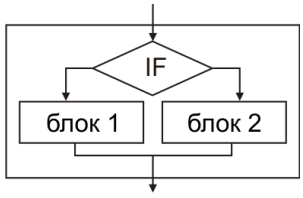
Для контроля количества путей необходимо определить, сколько выходов имеет стартовый блок и сколько входных связей проанализировано в конечном блоке. Рассмотрим в качестве примера правило условного ветвления. Как видно из рисунка 2, из стартового элемента выходят две метки-связи. В начале обработки правила в стек записывается вершина условия, в ленту в ячейку $i$, соответствующую номеру условной вершины, записывается число выходов из вершины (для нашего случая 2). При достижении точки слияния в ячейку $j$, соответствующую номеру слияния условных ветвей, записывается значение на единицу большее предыдущего (начальное значение ноль). Когда значения в ячейках $i$ и $j$ совпадут, считается, что все возможные пути проанализированы, и дальнейший анализ продолжается по связи метки, соответствующей выходу из конечного элемента. Для правила распараллеливания данный механизм аналогичен.



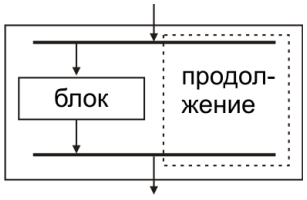

Rule Диаграмма
Consist of Начало начало, Конец конец, Блоки блок
Internal Relationships: блок.Вход = начало.Выход, блок.Выход = конец.Вход
External Relationships:



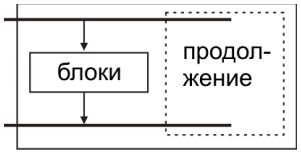

Rule Блоки
Consist of Действие действие, Блоки блок
Internal Relationships: блок.Вход = действие.Выход
External Relationships: Блоки.Вход = действие.Вход, Блоки.Выход = блок.Выход




Rule Блоки
Consist of Блоки блок1, Блоки блок2, Условие условие , Слияние слияние
Internal Relationships: условие.Выход1 = блок1.Вход, условие.Выход2 = блок2.Вход,
слияние.Вход1=Блок1.Вход,
слияние.Вход2=Блок2.Вход
External Relationships: блоки.Вход = условие.Вход, блоки.Выход = слияние.Выход




Rule Блоки
Consist of Блоки блок, Распараллеливание распараллеливание, Слияние параллельных ветвей слияние, Продолжение продолжение
Internal Relationships: распараллеливание.Выход1 = блок.Вход,
распараллеливание=продолжение.
Вход, слияние = продолжение. Выход
External Relationships: блок.Вход = распараллеливание.Вход, блок.Выход = слияние.Выход




Rule Продолжение
Consist of Блоки блок, Распараллеливание распараллеливание, Слияние параллельных ветвей слияние, Продолжение продолжение
Internal Relationships: распараллеливание.Выход1 = блок.Вход,
распараллеливание = продолжение.Вход,
слияние = продолжение.Выход
External Relationships: продолжение.Вход = распараллеливание, продолжение.Выход = Слияние

Рис. 2. Конструкции диаграммы активности и метаописания их правил

### Построение квазитермального алфавита

Квазитермальный алфавит строится на основе части правила, начинающегося с Consist of. Построение происходит в два прохода – построение множества имен и фильтрация множества имен.

Во время первого прохода собираются все имена из правил. Для этого в каждом правиле из части Consist of строится множество используемых имен. Во время этого же прохода строится дополнительное множество имен правил. Для первого правила получим следующие множества:

| Правило | Множество имен | Множество имен правил |
|---|---|---|
| Rule Диаграмма Consist of Начало, Конец, Блоки Internal Relationships: External Relationships: Блоки.Вход = Начало. Выход, Блоки.Выход = Конец.Вход | Начало, Конец, Блоки | Диаграмма |

Таблица 1

Условные обозначения

| Терминальный символ | Квазитерминальный символ | Примечание |
|---|---|---|
| ● | A0 | Начало диаграммы |
| ◉ | Ak | Конец диаграммы |
| ▭ | A | Действие |
| ◇ | P | Ветвление |
| ▽ | S | Объединение взаимоисключающих ветвей |
| | R | Параллельное выполнение действий |
| | L | Слияние выполнения параллельных ветвей |
| | label | Связь |
| | ∅ | Отсутствие связей меток |

После окончания первого прохода необходимо отфильтровать множество имен. Целевое отфильтрованное множество равно разности Множества имен и Множества правил. Таким образом, формируется квазитермальный алфавит блоков.

На следующем проходе формируется терминальный алфавит связей. Для диаграммы активности существует только один тип связи. Для других диаграммных языков, где связи могут быть разного типа, алгоритм построения терминального алфавита следующий:

- обрабатываются правила Relation;
- выделяются все типы связей для их правила;
- добавляется квазитерминальный символ в алфавит.

В таблице 1 приведены терминальные и соответствующие квазитерминальные символы. Последняя строка таблицы 1 соответствует случаю, когда анализатор не достигает конечного состояния и во входном потоке нет символов для анализа. В процессе анализа правил присваиваются произвольные квазитерминальные символы (обычно a, b, c…). Для наглядности в таблице 1 приведены семантически понятные имена квазитерминальных символов.

В результате получаем автоматную $RV$-грамматику, содержащую 15 комплексов продукций. Фрагмент грамматики приведен на рисунке 3. $W^1$ обозначает операцию записи в стек, $W^2$ – операцию чтения из стека. После минимизации, заключающейся в исключении одинаковых комплексов продукций, получаем $RV$-грамматику, изображенную на рисунке 4.

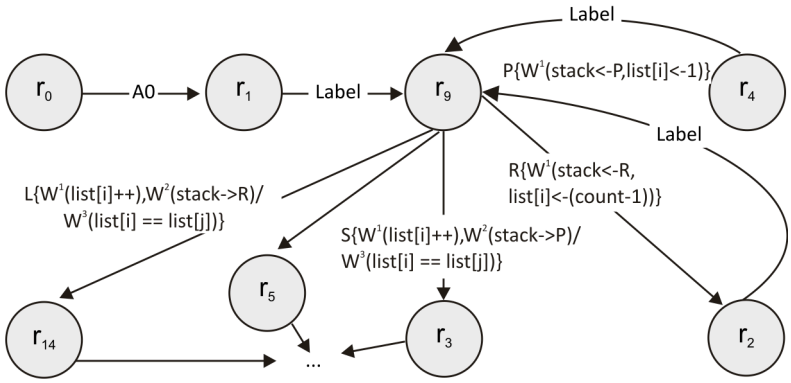



Рис. 3. Фрагмент автоматной $RV$-грамматики Диаграммы активности

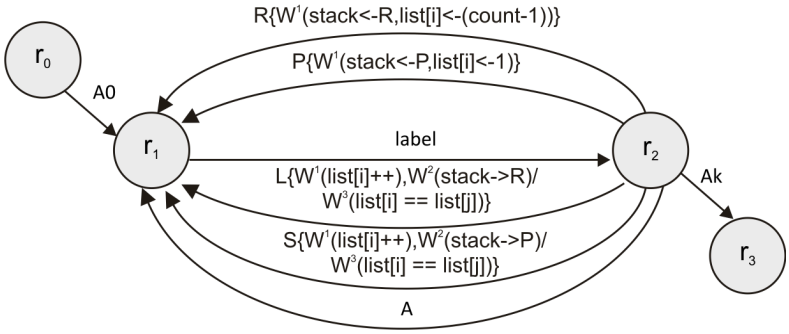



Рис. 4. Автоматная $RV$-грамматика Диаграммы активности после минимизации

Рассмотрим одно из правил подробнее. Например,

$$S\{W^1(list[i]{+}{+}),W^2(stack\text{->}P)/W^3(list[i]==list[j])\}.$$

По символу $S$ производится переход из комплекса продукций $r_2$ в комплекс $r_1$. При этом выполняются операции с памятью: запись в ленту в ячейку $i$ значения на 1

больше предыдущего, извлечение из стека символа $P$ при условии, что значения ячеек $i$ и $j$ совпадают. Фрагмент XML-описания $RV$-грамматики приведен на рисунке 5.

```
<production>
    <currentState>1</currentState>
    <nextState>2</nextState>
    <term>3</term>
    <operations>
        <operation storageId=»1» key=»t» value=»1»/>
        <operation storageId=»3» key=»t» value=»k»/>
    </operations>
    <conditions>
        <condition storageId=»2» key=»i» value=»x» />
        <condition storageId=»1» key=»j» value=»x» />
    </conditions>
</production>
```

Рис. 5. Фрагмент XML-описания грамматики диаграммного языка

На выходе метакомпилятора получается $RV$-грамматика в формате XML, которая используется кодом $RV$-анализатора для разбора конкретных графических предложений.

## Заключение

Таким образом, при создании собственной библиотеки графических примитивов (такая возможность предоставляется, например, в редакторах Visio, Dia) пользователь может описать графический язык, задав метаописание графических конструкций. Проводятся работы по созданию графического редактора, который позволяет заменить текстовый язык спецификаций графических грамматик графическим и в режиме «рисования» проектировать грамматику.

## СПИСОК ЛИТЕРАТУРЫ

1. Шаров О.Г., Афанасьев А.Н. Синтаксически–ориентированная реализация графических языков на основе автоматных графических грамматик // Программирование. – 2005. – № 6 – С. 56–66.

2. Ахо А., Лам М., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструментарий. – 2 изд. – М. : Вильямс, 2008.

3. The Lex & Yacc Page [Электронный ресурс]. – URL: http://dinosaur.compilertools.net/#yacc (дата обращения 27.04.2013).

4. ANTLR [Электронный ресурс]. – URL: http://www.antlr.org/ (дата обращения 27.04.2013).

5. GOLD Parsing System. Multi-Programming Language, Parser [Электронный ресурс]. – URL: http://goldparser.org/ (дата обращения 27.04.2013).

6. Boost c++ libraries [Электронный ресурс]. – URL: http://www.boost.org/doc/libs/1_53_0/libs/spirit/doc/html/index.html (дата обращения 27.04.2013).

7. Lapg Home [Электронный ресурс] – URL: http://lapg.sourceforge.net/ (дата обращения 27.04.2013).

8. The Compiler Generator Coco/R [Электронный ресурс]. – URL: http://www.ssw.uni-linz.ac.at/coco/ (дата обращения 27.04.2013).

9. Wilson T. et al. An ILP-Based Approach to Code Generation. // In Code Genera-tion for Embedded Processors. – Kluwer Academic Publishers, 1995. – pp. 103–118.