УДК 519.685.3

Я.А. Кириленко, С.В. Григорьев, Д.А. Авдюхин

РАЗРАБОТКА СИНТАКСИЧЕСКИХ АНАЛИЗАТОРОВ В ПРОЕКТАХ ПО АВТОМАТИЗИРОВАННОМУ РЕИНЖИНИРИНГУ ИНФОРМАЦИОННЫХ СИСТЕМ

I.A. Kirilenko, S.V. Grigoriev, D.A. Avdiukhin

SYNTAX ANALYZERS DEVELOPMENT IN AUTOMATED REENGINEERING OF INFORMATIONAL SYSTEM

Рассмотрены особенности разработки синтаксических анализаторов при автоматизации реинжиниринга. Сформулированы требования, предъявляемые к генераторам парсеров. Описан инструмент, удовлетворяющий этим требованиям.

РЕИНЖИНИРИНГ. СИНТАКСИЧЕСКИЙ АНАЛИЗ. ТРАНСЛЯЦИЯ. ГЕНЕРАТОРЫ ПАРСЕРОВ.

In this article features of syntax analyzer development in automatic reengineering problems are considered. Requirements, laid on parser generators, are stated. Tool, satisfying these requirements, is described.

REENGINEERING. SYNTAX ANALYSIS. TRANSLATION. PARSER GENERATORS.

Проекты по автоматизации реинжиниринга программ [1] выдвигают особые требования к генераторам синтаксических анализаторов по сравнению с требованиями при разработке оптимизирующих компиляторов. Причины принципиального отличия кроются в процессе разработки средства автоматизации реинжиниринга по сравнению с процессом разработки транслятора, но немаловажную роль играет история развития мигрируемых языков: большинство из них уже существовали, когда теория синтаксически управляемой трансляции только развивалась. Такие языки – сейчас их относят к устаревшим (legacy) языкам – проектировались, когда еще не были получены основные теоретические результаты, положенные в основу наиболее распространенных современных генераторов. Поэтому устаревшие языки не учитывают эти результаты и трудны для синтаксического анализа с использованием современных инструментов, которые в свою очередь значительно упрощают создание анализаторов, но лишь для относительно современных или заново проектируемых языков.

В отличие от проектов по разработке оптимизирующих компиляторов, при реинжиниринге систем актуальна проблема отсутствия подходящей спецификации языка. Нередко документация вовсе неполна или содержит ошибки – работающие программы даже синтаксически не соответствуют документации. Кроме того, для описания грамматик устаревших языков использовались синтаксические диаграммы, обладающие значительно большими выразительными возможностями, чем популярная ныне и хорошо формализованная форма Бэкуса–Наура (БНФ) [2].

Отдельного внимания заслуживает проблема множества схожих вариантов синтаксиса – диалектов (ярким примером является COBOL[1]).

Если посмотреть на жизненный цикл синтаксического анализатора в проекте по реинжинирингу информационой системы,

[1] Common Business Oriented Language — один из старейших языков программирования (первая версия увидела свет в 1959 г.), широко используемый в разработке бизнес-приложений.

то разница видна уже на этапе предпродажной подготовки: необходимо быстро реализовать прототип инструмента, позволяющий продемонстрировать заказчику возможность создания конечного инструмента. В связи с этим важным требованием является возможность быстрого создания простой версии парсера, обрабатывающего основные конструкции языка. Однако при продолжении работ иногда выясняется, что были выбраны неподходящие инструменты, а их смена приводит к необходимости переписывания грамматики заново, в формате, принимаемом новыми инструментами. Чтобы этого избежать, необходима возможность автоматического преобразования грамматики в новый формат.

В реинжиниринге нет необходимости полностью описывать входной язык, поскольку многие конструкции не встретятся в исходном коде. Кроме того, обрабатываемый код часто меняется в процессе работы над ним, и в нём появляются новые синтаксические конструкции. Поэтому разработка грамматики происходит итеративно и представляет собой не разовое действие, а процесс, идущий на протяжении всего проекта по реинжинирингу системы. Из-за этого парсер часто не способен сразу обработать весь исходный код, однако нужно попытаться продолжить разбор и получить информацию о всех возможных ошибках, а в случае небольшого числа ошибок необходимо продолжить работу над разобранным кодом, при этом покрыв большую часть функциональности системы.

Требования к генератору синтаксических анализаторов

Важными характеристиками генераторов синтаксических анализаторов, влияющими на простоту разработки и качество создаваемых анализаторов при разработке средства автоматизации реинжиниринга ИС, являются следующие:

- класс принимаемых языков;
- разрешение неоднозначностей в грамматике;
- скорость работы порождаемого парсера/транслятора;
- возможности языка описания грамматики (как предметно-ориентированного языка программирования);
- возможность быстрого изменения грамматики;
- простота отладки сгенерированного парсера/транслятора;
- наличие восстановления после ошибок;
- сопровождаемость, качество документации;
- возможность модификации инструмента под свои нужны;
- простота интеграции и использования.

Многие из этих свойств связаны с алгоритмом разбора. При использовании инструментов, порождающих LL [3] и LALR [4] парсеры, необходимо избавляться от конфликтов в грамматике. Однако в реинжиниринге грамматика постоянно меняется, и добавление одного правила может порождать десятки конфликтов [5]. Некоторые типы неоднозначностей можно устранить путем преобразований грамматики, но при этом она значительно усложняется, что отрицательно сказывается на ее сопровождаемости. Поэтому входной язык генератора и внутренний алгоритм должны предоставлять возможности по эффективному разрешению конфликтов: использование предикатов, управляющих процессом разбора, либо использование GLR-алгоритма [6] анализа, предназначенного для работы с неоднозначными грамматиками.

При поддержке нескольких диалектов языка нужно в соответствии с принципами качественного кода [7] переиспользовать общие части их грамматик. Помимо этого, многие языки имеют синтаксически схожие конструкции, например, арифметические выражения, поэтому можно повторно использовать некоторые части грамматики одного языка при описании другого. То есть инструмент должен обладать средствами повторного использования элементов грамматик. Одним из них является модульность. Разделение грамматики на модули необходимо не только для повторного использования каких-то ее частей, но и для повышения управляемости и снижения сложности процесса разработки и сопровождения.

Другим способом переиспользования грамматики является параметризация одних правил другими для введения общих концепций. Проводя параллель с языками программирования, правила, параметризованные другими правилами, соответствуют шаблонам (templates) в C++ и generics типам в C# и Java. Одним из примеров является правило для разбора списка элементов с разделителем между ними:

```
list<item delimiter>: item (delimiter item)*
func: typeName funcName list<param ‘,’> funcBody
funcBody: ‘{‘ list<stmt ‘;’> ‘}’
```

В синтаксических диаграммах, которые используются для описания устаревших языков, часто встречаются перестановки, с помощью которых определяются списки опций или атрибутов, где каждый элемент встречается только один раз. Обозначим перестановку как [| *a*, *b*, *c*, ... |], тогда упрощенное описание данных в языке COBOL можно задать так:

```
dataEntry: levelNumber dataName [|pictureClause, occursClause, usageClause|]
```

Однако обычно такой список записывают менее строгим образом, что приводит к большему числу допущений и конфликтов:

```
dataEntry: levelNumber dataName (pictureClause | occursClause | usageClause)+
```

В настоящее время ни один из популярных инструментов не отвечает всем перечисленным выше требованиям. Тем не менее современные инструменты обладают такими удобными средствами для задания грамматик, как

- конструкции EBNF [2] (обычно их обозначают через «*», «+» и «?»);
- правила с параметрами, которые значительно упрощают трансляцию с учетом контекста (в терминах атрибутной грамматики такие параметры называются *наследуемыми атрибутами*);
- специальные конструкции для разрешения неоднозначностей в грамматике — предикаты;
- именование семантических значений (вместо использования $1, $2, $3, . . .), что уменьшает число возможных ошибок при обращении к ним.

Таким образом, генератор синтаксических анализаторов должен обладать как хорошо зарекомендовавшими себя возможностями, так и предоставлять описанные выше средства для задания грамматики и разрешения неоднозначностей в ней.

Инструмент для создания синтаксических анализаторов YaccConstructor

На кафедре системного программирования математико-механического факультета Санкт-Петербургского государственного университета (СПбГУ) разрабатывается инструмент для создания генераторов синтаксических анализаторов YaccConstructor [8], который имеет модульную структуру, позволяющую создавать анализаторы с различными языками описания грамматик и алгоритмами разбора. Такая структура во многом решает проблему изменения алгоритма разбора: если пользователь понимает, что изначально выбран неподходящий алгоритм, ему достаточно взять другой генератор.

Для создания генератора анализаторов необходимы следующие компоненты:

- парсер входной грамматики (фронтенд), преобразующий грамматику языка, записанную в некотором формате, во внутреннее представление;
- генератор (бэкенд), при помощи которого на основе внутреннего представления создается синтаксический анализатор или генерируется исходная грамматика на определенном языке, что позволяет порождать эквивалентное описание языка, принимаемое другим инструментом;
- преобразования грамматики, которые приводят грамматику к виду, удовлетворяющему ограничениям, накладываемым генератором. Например устранение левой рекурсии или раскрытие конструкций EBNF.

В рамках проекта разрабатывается язык описания трансляций Yard, удовлетворяющий описанным ранее требованиям, благодаря мощному синтаксису (EBNF, перестановки, литералы), реализации механизмов переиспользования грамматики (модуль-

ность, параметризованные правила) и поддержке средств спецификации трансляции (наследуемые атрибуты, предикаты, именование семантических значений).

Основываясь на описанных преимуществах GLR алгоритма разбора перед LL и LR, в рамках проекта также реализован GLR-генератор. В результате анализа существующих алгоритмов был выбран алгоритм RNGLR [9], работающий со всеми контекстно-свободными грамматиками. В нем реализована встроенная поддержка наследуемых атрибутов, предикатов и литералов. Также для него реализован алгоритм восстановления после ошибок. На основе его практического использования реализованы различные средства отладки, среди которых вывод информации о неоднозначностях и визуальное представление результатов разбора.

Несмотря на то, что построение синтаксически управляемых трансляций является хорошо изученной областью, особенности проектов по реинжинирингу выдвигают новые требования и формулируют новые задачи в сфере автоматизации построения синтаксических анализаторов. Современные инструменты, известные как «компиляторы компиляторов», предоставляют широкие возможности для автоматического создания синтаксических анализаторов (парсеров), но недостаточно отвечают специальным требованиям проектов по реинжинирингу. Это послужило мотивацией для разработки инструмента, предназначенного для создания генераторов синтаксических анализаторов, в рамках которого предъявленные требования будут выполнены.

Дальнейшая исследовательская работа над инструментом ведется в направлении поддержки различных диалектов языков, а также его совершенствования по результатам практического использования.

СПИСОК ЛИТЕРАТУРЫ

1. **Терехов, А.Н.** Автоматизированный реинжиниринг программ [Текст] / А.Н. Терехов. –Изд-во СПбГУ, 2000. –332 с.

2. **Ахо, А.** Компиляторы: принципы, технологии, инструменты [Текст] / А. Ахо, Р. Сети, Дж. Ульман. –М.: ИД «Вильямс», 2003. – 768 с.

3. **Lewis, P.M. II.** Syntax-Directed transduction [Text] / P.M. Lewis II, R.E. Stearns –ACM 15. –July 1968. –Vol. 3. –P. 465–488.

4. **Deremer, F.L.** Practical translators for LR(k) languages [Text] / F.L. Deremer // Ph.D. thesis. –Massachusetts Institute of Technology, 1969.

5. **Van den Brand, M.G.J.**. Current Parsing Techniques in Software Renovation Considered Harmful [Text] /M.G.J. Van den Brand, M.P.A. Sellink, C. Verhoef // IEEE Proc. of the 6th International Workshop on Program Comprehension. –1998. –P. 108–117.

6. **Tomita, M.** Efficient Parsing for Natural Language [Text] / M. Tomita. –1986.

7. **Макконнелл, С.** Совершенный код [Текст] / С. Макконнелл. –СПб.: Питер, 2005. –896 с.

8. Домашняя страница YaccConstructor [Электронный ресурс] / Режим доступа: https://code.google.com/p/recursive-ascent/ (Дата обращения 2008)

9. **Scott, Elizabeth.** Right nulled GLR parsers [Text] / Elizabeth Scott, Adrian Johnstone // ACM Transactions on Programming Languages and Systems. –July 2006. –№ 28 (4). –P. 577–618.

REFERENCES

1. **Terekhov A.N.** Avtomatizirovannyi reinzhiniring program. – St Petersburg: Izdatel'stvo Sankt-Peterburgskogo gosudarstvennogo un-ta, 2000. – 332 s. (rus)

2. **Akho A., Seti R., Ul'man Dzh.** Kompiliatory: printsipy, tekhnologii, instrumenty. – Moscow: Izdatel'skii dom «Vil'iams», 2003. – 768 s. (rus)

3. **Lewis P.M. II, Stearns R.E.** Syntax-Directed transduction/ ACM. – July 1968. – Vol. 15. – № 3. – P. 465–488.

4. **Deremer F.L.** Practical translators for LR(k) languages / Ph.D. thesis. — Massachusetts Institute of Technology, 1969.

5. **van den Brand M.G.J, Sellink M.P.A., Verhoef C.** Current Parsing Techniques in Software Renovation Considered Harmful / Proc. of the 6th Internat. Workshop on Program Comprehension. – 1998. – P. 108–117.

6. **Tomita M.** Efficient Parsing for Natural Language, 1986.

7. **Makkonnell S.** Sovershennyi kod. – St. Petersburg: Piter, 2005. – 896 s. (rus)

8. Home page YaccConstructor. Availiable https://code.google.com/p/recursive-ascent/ (Accessed 2008)

9. **Scott E., Johnstone A.** Right nulled GLR parsers / ACM Transactions on Programming Languages and Systems. – July 2006. – № 28 (4). – P. 577–618.

КИРИЛЕНКО Яков Александрович – *старший преподаватель Санкт-Петербургского государственного университета.*
198504, Россия, Санкт-Петербург, Старый Петергоф, Университетский пр., д. 28.
E-mail: jake@math.spbu.ru

KIRILENKO, Iakov A. – *St. Petersburg State University.*
198504, Universitetsky prospekt 28, Peterhof, St. Petersburg, Russia.
E-mail: jake@math.spbu.ru

ГРИГОРЬЕВ Семен Вячеславович – *аспирант Санкт-Петербургского государственного университета.*
198504, Россия, Санкт-Петербург, Старый Петергоф, Университетский пр., д. 28.
E-mail: rsdpisuy@gmail.com

GRIGORIEV Semen V. – *St. Petersburg State University.*
198504, Universitetsky prospekt 28, Peterhof, St. Petersburg, Russia.
E-mail: rsdpisuy@gmail.com

АВДЮХИН Дмитрий Алексеевич – *студент Санкт-Петербургского государственного университета.*
198504, Россия, Санкт-Петербург, Старый Петергоф, Университетский пр., д. 28.
E-mail: dimonbv@gmail.com

AVDIUKHIN, Dmitry A. – *St. Petersburg State University.*
198504, Universitetsky prospekt 28, Peterhof, St. Petersburg, Russia.
E-mail: dimonbv@gmail.com