

Санкт-Петербургский Государственный университет
Математическое обеспечение и администрирование информационных
систем

Андреев Сергей

Конвертер из языка QPILe в язык Lua

Курсовая работа

Научный руководитель
к.ф.-м.н. доц. Д. А. Григорьев

Санкт-Петербург, 2019 г.

СОДЕРЖАНИЕ

1.	ВВЕДЕНИЕ.....	2
1.1.	Введение в предметную область	2
1.2.	Постановка задачи	3
2.	ОБЗОР ПРОГРАММНЫХ СРЕДСТВ, НАПРАВЛЕННЫХ НА СОЗДАНИЕ ТРАНСЛЯТОРОВ.....	4
2.1.	Структура транслятора.....	4
2.2.	Средства создания лексических анализаторов и синтаксических анализаторов.....	5
3.	АДАПТАЦИЯ ГРАММАТИКИ QRILE ПОД ANTLR.....	7
4.	СОЗДАНИЕ ГЕНЕРАТОРА КОДА	9
5.	РЕЗУЛЬТАТЫ И АПРОБАЦИЯ.....	10
5.1.	Графическое оформление	10
5.2.	Возникшие трудности	10
5.3.	Тестирование	11
6.	ЗАКЛЮЧЕНИЕ	12
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	13

1. ВВЕДЕНИЕ

1.1. Введение в предметную область

Многие из людей, увлекающихся вопросами биржевой торговли, знакомы с системой QUIK (от англ. Quickly Updatable Information Kit — Быстро-Обновляемая Информационная Панель) — комплекс программ для доступа к биржевым торгам [6]. Для работы с системой пользователь разрабатывает необходимую ему программу — “биржевого робота”. Изначально система QUIK поддерживала только роботов, написанных на языке QPILE [3]. В 2012г. Разработчики системы QUIK приняли решение об отсутствии перспектив его дальнейшего развития, в результате чего в Рабочее место QUIK был встроен интерпретатор скриптового языка Lua. QPILE прекратил свое развитие, но поддержка была сохранена. Последующие 4 года разработчики пропагандировали среди клиентов применение Lua, который в текущий момент является основным инструментом, используемым для разработки клиентских скриптов в QUIK. Весной 2016г. разработчики QUIK заявили, что в течение года они планируют полностью прекратить поддержку QPILE. В дальнейшем прекращение поддержки QPILE было отложено на неопределенный срок. Но тем не менее вопрос конвертации отлаженных и проверенных роботов на QPILE в Lua стоит остро. На данный момент в открытом доступе не было найдено конвертеров, решивших бы эту проблему, поэтому было решено создать такой конвертер. Для формулировки цели работы следует описать структуру программ на QPILE.

Любая программа на языке QPILE состоит из трех частей [3]:

- описание основных параметров таблицы в рабочем месте QUIK;
- алгоритм для анализа биржевой ситуации, автоматизации торговли, взаимодействия с данными терминала;
- описание параметров каждого из столбцов таблицы.

Конвертер второго блока программ на QPILE в программы на Lua было решено реализовать в данной работе.

1.2. Постановка задачи

Цель работы: частично реализовать конвертер программ на языке QPILE в эквивалентные им на языке Lua.

Под словом “частично” понимается трансляция второго блока программ на QPILE, содержащего любые конструкции, описанные в грамматике этого языка, а также некоторые встроенные функции.

Задачи:

- изучить программные средства, направленные на создание трансляторов;
- адаптировать формальную грамматику языка QPILE под выбранный инструмент, реализовать лексический и синтаксический анализаторы;
- реализовать генератор кода на язык Lua;

2. ОБЗОР ПРОГРАММНЫХ СРЕДСТВ, НАПРАВЛЕННЫХ НА СОЗДАНИЕ ТРАНСЛЯТОРОВ

2.1. Структура транслятора

Транслятор (конвертер) – программа или техническое средство, выполняющее преобразование программы, представленной на одном из языков программирования, в программу на другом языке [1].

Этапы трансляции:

- лексический анализ;
- синтаксический анализ;
- генерация кода.

Лексический анализатор осуществляет чтение входной цепочки символов и их группировку в элементарные конструкции, называемые лексемами. Каждая лексема имеет класс и значение. Обычно претендентами на роль лексем выступают элементарные конструкции языка, например, идентификатор, действительное число, комментарий. Полученные лексеммы передаются синтаксическому анализатору.

Синтаксический анализатор осуществляет разбор исходной программы, используя поступающие лексеммы, построение синтаксической структуры программы и семантический анализ с формированием объектной модели языка.

Генератор кода строит код на другом языке программирования на основе анализа объектной модели или промежуточного представления.

2.2. Средства создания лексических анализаторов и синтаксических анализаторов

Один из способов создания данных анализаторов- написать их вручную. Однако, этот метод обладает рядом серьезных недостатков:

- создание анализаторов вручную требует очень много времени;
- полученный код тяжело отлаживать;
- чаще всего платформы не предназначены для создания анализаторов вручную.

К плюсам можно отнести неплохую производительность и понятность кода.

Так как работу было решено делать на языке C#, то и от средств создания анализаторов требовалась генерация кода на языке C#. Среди популярных анализаторов, удовлетворяющих этому условию, можно отметить следующие:

- Jay;
- CoCo/R;
- ANTLR.

Главными минусами Jay – портированного под C# Yacc, являются отсутствие в нем лексического анализатора (часто для решения этой проблемы выбирают лексический анализатор Lex), устаревший алгоритм работы, сложность отладки, к плюсам можно отнести поддержку леворекурсивных правил грамматики [5].

Что касается CoCo/R, алгоритм его работы современнее, чем у Jay, но все равно работает медленнее, чем у ANTLR. Также CoCo/R поддерживает гораздо меньше грамматик, чем ANTLR, в том числе не поддерживает леворекурсивные правила, поэтому для решения поставленной задачи подходит не слишком хорошо [2].

ANTLR, начиная с последней версии, поддерживает обработку нужных

нам грамматических правил. Также ANTLR обладает собственной средой разработки- ANTLRworks (рис. 1), более прост для начинающих и выдает

The screenshot shows the ANTLRworks interface. The top pane displays a QPILE program with line numbers 6 through 18. The code includes variable assignments, a function definition, and control structures. The bottom pane, titled 'Lexer Debugger Controller Window', shows the 'Tokens' tab with a list of lexical tokens and their positions in the input stream.

```

6  PROGRAM
7  CURMONTH=SUBSTR(FSERVERDATE,3,2)
8  newname = Myfun(4+8-6,"classic")
9  FirmCode = "ms00123000000" & "abc"
10 FUNC Myfun(x, y)
11   If x > 0
12   RESULT = x
13   Else
14   RESULT = 0
15   End If
16   RETURN
17 END FUNC
18 OUTPUT=CREATE MAP(())

```

Lexer Debugger Controller Window x

Types Tokens Channels Modes Lookahead

[@19,[121..129]='CURMONTH',<37=IDENT>]
 [@20,[129..130]='',<47=EQUAL>]
 [@21,[130..136]='SUBSTR',<37=IDENT>]
 [@22,[136..137]='(',<53=LPAREN>]
 [@23,[137..148]='FSERVERDATE',<37=IDENT>]

Рисунок 1. Код программы на языке QPILE, разбитый на лексемы. Скриншот среды разработки ANTLRworks

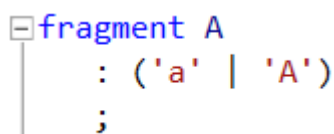
достаточно понятный код анализаторов после их создания [5].

По изложенным выше причинам выбор средства создания лексического и синтаксического анализатора пал на ANTLR.

3. АДАПТАЦИЯ ГРАММАТИКИ QPILE ПОД ANTLR

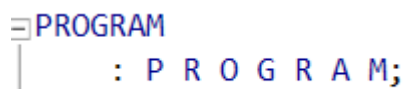
Помимо возможности разработки в собственной среде, ANTLR поддерживает возможность разработки прямо в Visual Studio. Для этого было установлено расширение ANTLR Language Support. Удобство заключается в возможности быстрого изменения лексического и синтаксического анализаторов прямо в проекте с генератором кода с помощью изменений в файле с грамматикой.

Правила лексического анализатора в ANTLR начинаются с большой буквы и выполняются в том порядке, в котором они написаны в файле с грамматикой. Поэтому сначала нужно определять зарезервированные ключевые слова языка, а потом уже переменные и прочие конструкции. В ANTLR нельзя указать независимость правил от регистра, поэтому были созданы правила для каждой буквы алфавита, и ключевые зарезервированные слова определялись комбинацией этих правил (рис. 2 и рис. 3)



```
fragment A
: ('a' | 'A')
;
```

Рисунок 2



```
PROGRAM
: P R O G R A M ;
```

Рисунок 3

Также ANTLR позволяет игнорировать пробелы и пустые линии в коде входной программы.

Правила синтаксического анализатора начинаются с маленькой буквы. Суммарно для описания грамматики QPILE потребовалось 25 правил синтаксического анализатора, пример одного из них показан на рисунке 4. Некоторые из них используют левую рекурсию, особенно это удобно при определении арифметических выражений (рис. 5).


```

statement
: name EQUAL expression
| ifOperator
| forOperator
| funcDescr
| procedureCall
| CONTINUE
| BREAK
| RETURN
;

```

Рисунок 5

```

expression
: expression PLUS term
| expression MINUS term
| expression COMPOUND term
| term
;

```

Рисунок 4

Описание грамматики языка QPILE в руководстве пользователя не поддерживало некоторые возможные конструкции, например, вызов процедур, вероятно это связано с тем, что оно устарело. Поэтому грамматику пришлось немного модифицировать.

После создания и сохранения файла с грамматикой, ANTLR автоматически генерирует лексический и синтаксический анализаторы на выбранном языке.

4. СОЗДАНИЕ ГЕНЕРАТОРА КОДА

На основе полученного от лексического анализатора потока токенов синтаксический анализатор создает синтаксическое дерево кода программы. На этапе генерации кода требуется совершить обход этого дерева и вернуть полученный код на языке Lua. Обход дерева можно организовать вручную, однако, это достаточно сложно и долго. Для упрощения этой задачи ANTLR предлагает использовать один из приемов: Listener или Visitor [4].

С помощью Listener-а можно осуществить неконтролируемый обход. Это значит, что для каждого узла будут посещены все его потомки, а далее сделаны необходимые для генерации действия. Плюсом такого подхода является невозможность “забыть” обойти какой-то узел, но в то же время это уменьшает гибкость программы.

Visitor, как и паттерн с соответствующим названием, позволяет осуществлять контролируемый обход синтаксического дерева. Для решения поставленной задачи больше подходит этот способ, так как он обладает большей гибкостью.

Вместе с анализаторами ANTLR автоматически создает частичный класс BaseVisitor. Он содержит виртуальные методы с названиями, соответствующим названиям правил синтаксического анализатора в грамматике. Для реализации обхода требуется создать класс-наследник и определить действия, которые должны быть выполнены при вызове этих методов — это соответствует посещению Visitor-ом узла дерева, созданного правилом с названием, соответствующему названию вызываемого метода. Например, в методе, соответствующему оператору if, требуется посетить трех потомков: потомка, соответствующего условию оператора, потомка, соответствующего действию при выполнении условия, и потомка, соответствующего действию при его невыполнении, а затем вернуть всю конструкцию, обернутую в синтаксис языка Lua.

5. РЕЗУЛЬТАТЫ И АПРОБАЦИЯ

5.1. Графическое оформление

Для удобства взаимодействия пользователя с конвертером, с помощью .NET Windows Forms была реализована графическая оболочка транслятора (рис. 6). Код программы на языке QPILE можно вставлять в специальное окно и, после нажатия на соответствующую кнопку, код эквивалентной программы на языке Lua появится в окне справа. Также вместо этого можно указать путь к файлу с кодом на языке QPILE и путь к файлу, в который требуется записать результат.

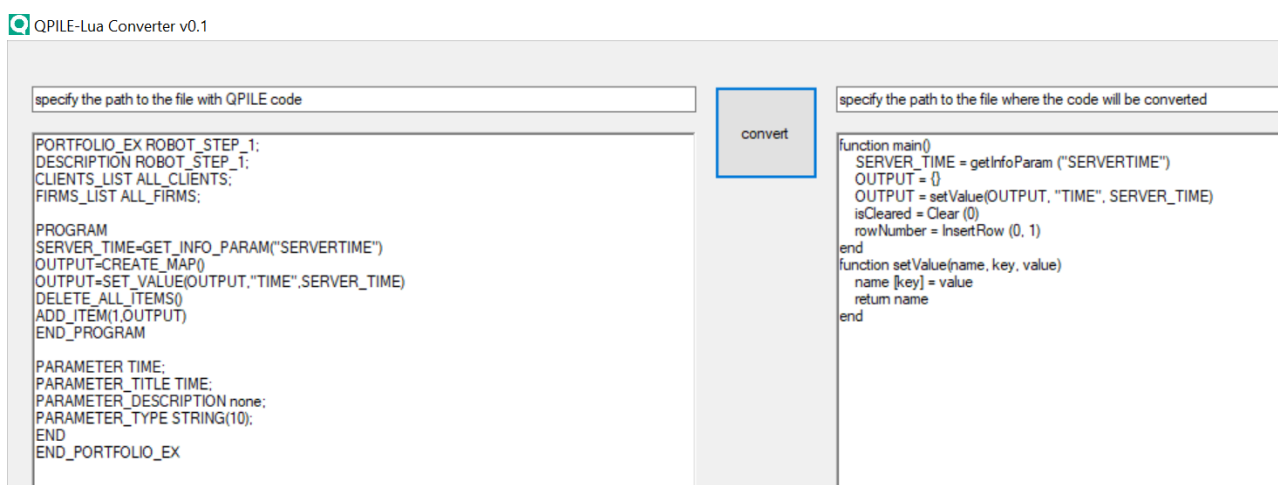


Рисунок 6

5.2. Возникшие трудности

Языки QPILE и Lua достаточно схожи, но имеют ряд серьезных отличий. Например, объявление пользовательских функций и процедур происходит в QPILE прямо в блоке PROGRAM ... END_PROGRAM, который соответствует функции main () в Lua. Но в Lua функции не могут определяться прямо внутри других функций. Поэтому их надо вынести за пределы метода main (). Проблема была решена следующим образом: при обходе синтаксического дерева функции генерируются прямо внутри метода main (). После завершения обхода осуществляется поиск функций и вынесение их вне метода main ().

Также сложность возникла с конвертацией некоторых встроенных в QPILE функций (например, для работы с массивами). Некоторые из них не

имеют точных аналогов в Lua, а некоторые возвращают не то же самое значение, или вообще не возвращают его. Эта проблема была решена следующим образом: при вызове такой функции генератор сам создает метод с необходимым функционалом и вставляет его в код, а в нужном месте просто вызывает этот метод.

Наконец, хотелось создать автогенерацию отступов перед строками в зависимости от их вложенности. Для этого был реализован дополнительный метод, сдвигающий все передаваемые ему строки некой программной конструкции на четыре пробела. Вызов этого метода при генерации кода позволил сделать сгенерированную на Lua программу более читабельной и приятной для человека.

5.3. Тестирование

Для тестирования были созданные программы с практически всевозможными сложными вложенными структурами, а также скачаны рабочие скрипты на языке QPILE. Проверка семантической корректности трансляции проводилась вручную а синтаксической – с помощью загрузки скриптов в рабочее место QUIK. Было создано 5 больших и сложных по структуре программ и скачано 10 рабочих скриптов. Созданные вручную скрипты транслируются и компилируются средой QUIK успешно, а вот с некоторыми из скачанных возникают проблемы: виной тому обилие встроенных в QPILE функций для работы с пользовательским интерфейсом, осуществление поддержки всех из которых не входило в рамки данной работы.

В дальнейшем планируются следующие усовершенствования данного транслятора: поддержка практически всех встроенных в QPILE функций, а также возможность конвертации также первой и третьей частей, то есть объявления параметров таблиц и их столбцов в рабочем месте QUIK, программ на QPILE.

6. ЗАКЛЮЧЕНИЕ

Подводя итоги курсовой работы, можно заключить, что были решены следующие задачи:

- изучены программные средства, направленные на создание трансляторов;
- адаптирована формальная грамматика языка QPILE под выбранный инструмент, реализованы лексический и синтаксический анализаторы;
- реализован генератор кода на язык Lua.

Итак, был реализован конвертер, преобразующий второй блок (блок алгоритма) программ на языке QPILE, состоящий из любых конструкций, описанных в грамматике языка QPILE, а также некоторых встроенных в QPILE функций, в программы на языке Lua. Это позволяет заключить, что цель курсовой работы была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. А.И. Легалов. Общие сведения о трансляторах // SoftCraft разноликое программирование
URL: <http://www.softcraft.ru/translat/lect/t01/> —
(дата обращения:07.06.2019)
2. Иван Кошуркин. Теория и практика парсинга исходников с помощью ANTLR и Roslyn // habr.com коллективный блог статей, связанных с информационными технологиями – 3 марта 2016 г., 16:39.
URL: <https://habr.com/ru/company/pt/blog/210772/#comments-processing> – (дата обращения:07.06.2019)
3. Раздел 8. Алгоритмический язык QPILE // Руководство пользователя QUIK v7.27
URL: <https://arqatech.com/ru/support/files/> —
(дата обращения:07.06.2019)
4. Evgeni Petrov. Antlr4 Listeners and Visitors - which to implement? // stackoverflow.com – форум посвященный информационным технологиям и разработке ПО
URL: <https://stackoverflow.com/questions/20714492/antlr4-listeners-and-visitors-which-to-implement> – (дата обращения:07.06.2019)
5. Daniel Spiewak. Advantages of Antlr // stackoverflow.com – форум посвященный информационным технологиям и разработке ПО
URL: <https://stackoverflow.com/questions/212900/advantages-of-antlr-versus-say-lex-yacc-bison> – (дата обращения:07.06.2019)
6. Никита Михайлов. Торговая система Quik – отличный терминал для работы // Школа инвестирования и трейдинга
URL: <https://investment-school.ru/torgovaya-sistema-quik/> –
(дата обращения:08.06.2019)