Конвертер из языка QPILE в язык Lua

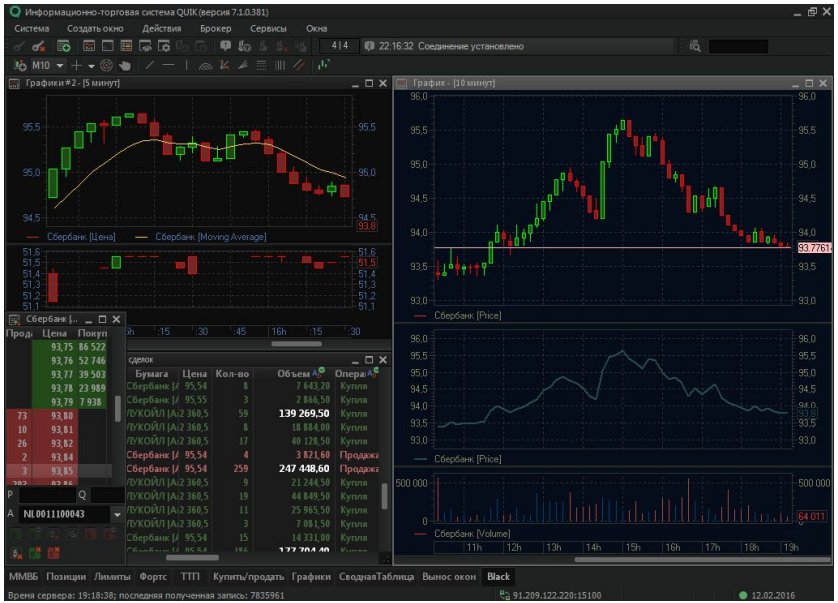
Выполнил: Андреев С.И.
241гр.
Научный руководитель:
Григорьев Д.А.

Санкт-Петербург, 2019

Мотивация

Алгоритмический язык QPILE используется в терминале QUIK (программный комплекс для организации доступа к биржевым торгам) для создания программ, взаимодействующих с системой QUIK:

- автоматизация торговли
- анализ биржевой ситуации
- экспорт данных терминала
- связь с внешними программами
- ...



Структура программы на языке QPILE

Блок описания базовых параметров таблицы

Блок алгоритма
(его трансляция и была реализована в данной работе)

Блок описания столбцов таблицы

```
PORTFOLIO_EX ROBOT_STEP_1;
DESCRIPTION ROBOT_STEP_1;
CLIENTS_LIST ALL_CLIENTS;
FIRMS_LIST ALL_FIRMS;

PROGRAM
SERVER_TIME=GET_INFO_PARAM("SER
VERTIME")
OUTPUT=CREATE_MAP()
OUTPUT=SET_VALUE(OUTPUT,"TIME",S
ERVER_TIME)
DELETE_ALL_ITEMS()
ADD_ITEM(1,OUTPUT)
END_PROGRAM

PARAMETER TIME;
PARAMETER_TITLE TIME;
PARAMETER_DESCRIPTION none;
PARAMETER_TYPE STRING(10);
END
END_PORTFOLIO_EX
```

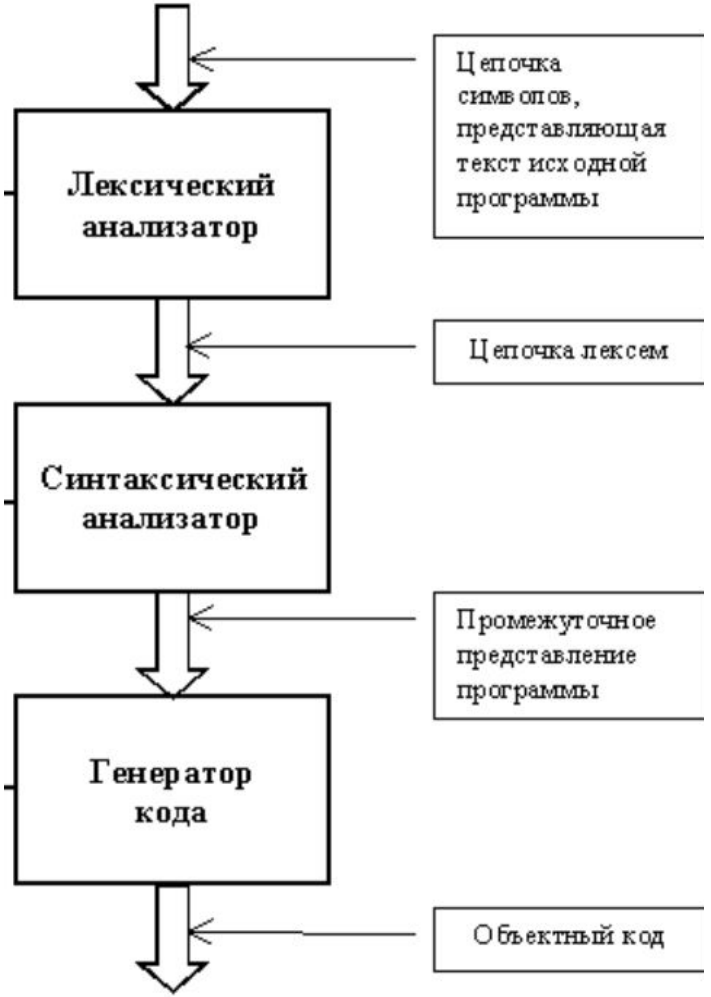
Цель и задачи курсовой работы

Цель курсовой работы: частично реализовать конвертер программ на языке QPILE в программы на языке Lua

Задачи:

- Изучить программные средства, направленные на создание трансляторов
- Адаптировать формальную грамматику языка QPILE под выбранный инструмент
- Реализовать лексический анализатор (лексер) и синтаксический анализатор (парсер)
- Реализовать генератор кода на язык Lua (проделать обход созданного парсером синтаксического дерева)

Структура транслятора



Обзор средств для создания парсера

1) Писать парсер на том же языке, на котором будет производиться разбор синтаксического дерева вручную

   + бОльшая производительность

   + “читабельность”

   - занимает очень много времени, часто платформа не предназначена для такого

   - достаточно тяжело отлаживать

Обзор средств для создания парсера

2) Использовать генераторы парсера:

GNU Bison и Yacc VS **ANTLR** VS **CoCo/R**

Алгоритм их работы был востребован, когда аппаратное обеспечение было ограничено (1960г.-1970г.)	Быстрее работает	Работает медленнее, чем ANTLR
	Наличие собственной среды разработки	
Поддержка леворекурсивных правил	Поддержка появилась в 4 версии	Выбор поддерживаемых грамматик меньше, чем у ANTLR
Тяжелее отладить	Более простой для начинающего разработчика; Более “читабельный” код после генерации	Не поддерживает левую рекурсию, поэтому будет неудобен для решения поставленной задачи

Описание грамматики в ANTLR

```
IDENT
    : ('a' .. 'z' | 'A' .. 'Z') ('a' .. 'z' | 'A' .. 'Z' | '0' .. '9' | '_')*
    ;

NUM_INT
    : ('0' .. '9') +
    ;
```

Пример правил лексера

```
6    PROGRAM
7    CURMONTH=SUBSTR(FSERVERDATE,3,2)
8    newname = Myfun(4+8-6,"classic")
9    FirmCode = "ms0012300000" & "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

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>]
```

Код программы на QPILE,
разбитый лексером на токены
(среда разработки ANTLRworks)

Описание грамматики в ANTLR

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

```
ifOperator
    : IF condition NEWLINE
     statementList
     ELSE NEWLINE
     statementList
     END_IF
    ;
```

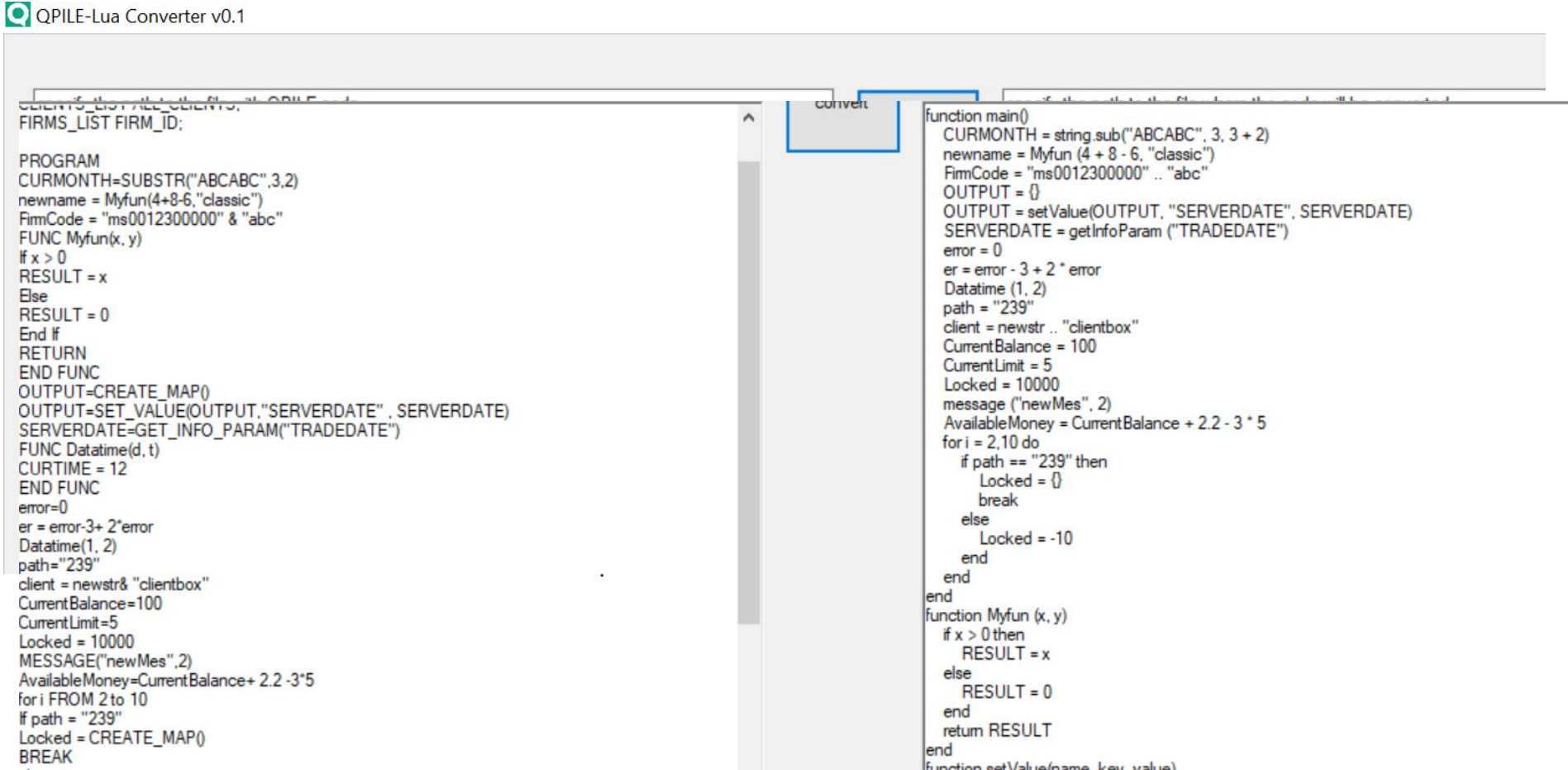
Примеры правил парсера. Суммарно всю грамматику описывают 25 правил парсера

Обход синтаксического дерева

Варианты реализации:

- Написать вручную
- Использовать сгенерированные ANTLR вместе с парсером классы BaseListener или BaseVisitor
    - Listener – “неконтролируемый” обход
      + невозможность “забыть” обойти какой-то узел
      - это убивает гибкость, присущую посетителю.
    - Visitor – обычный паттерн Visitor. На каждом узле мы сами решаем, продолжать обход потомков или нет

Графическая оболочка



Возникшие трудности

- Автоформатирование кода: отступы перед началом в зависимости от структурной вложенности строки
- Описание грамматики языка QPILE в руководстве пользователя пришлось модифицировать, так как она по неизвестным причинам не поддерживала некоторые конструкции данного языка (возможно, являлась устаревшей)
- Перенос объявления пользовательских функций вне функции main()
- Автогенерация методов для функций, не имеющих точных аналогов в lua (например, добавление элемента в массив)

QPILE

```
PROGRAM
SERVER_TIME=GET_INFO_PARAM("SERVERTIME")
OUTPUT=CREATE_MAP()
OUTPUT=SET_VALUE(OUTPUT,"TIME",SERVER_TIME)
END_PROGRAM
```

→

Lua

```
function main()
   SERVER_TIME = getInfoParam ("SERVERTIME")
   OUTPUT = {}
   OUTPUT = setValue(OUTPUT, "TIME", SERVER_TIME)
end
function setValue(name, key, value)
   name [key] = value
   return name
end
```

Итоги

Тестовые скрипты, содержащие только базовые функции взаимодействия с пользовательским интерфейсом транслируются правильно и успешно компилируются средой QUIK.

Результаты работы:

- Изучены основы создания трансляторов
- Создан конвертер, преобразующий блок Program, состоящий из любых конструкций, описанных в грамматике языка QPILE, а также некоторых встроенных в QPILE функций
- В дальнейшем планируется осуществить поддержку практически всех встроенных в QPILE функций