Третья международная научно-практическая конференция: Инструменты и методы анализа программ, TMPA-2015
12–14 ноября, Санкт-Петербург

Лексический анализ динамически формируемых строковых выражений

Автор: Полубелова Марина

Санкт-Петербургский государственный университет

12 ноября 2015г.

Примеры

- Встроенный SQL в C#

```
private void Go (int cond){
  string columnName = cond > 3 ? "X":(cond < 0 ? "Y":"Z");
  string query =
    "SELECT name" + columnName + " FROM table";
  Program.ExecuteImmediate(query);
}
```

- Динамически генерируемый HTML в PHP-программах

```
<?php
    $name = 'your name';
    echo '<table>
         <tr><th>Name</th></tr>
         <tr><td>'.$name.'</td></tr>
         </table>';
?>
```

Мотивация

Использование динамически формируемых строковых выражений

- Уменьшает надежность
  - Нет статического поиска ошибок
- Увеличивает уязвимость
  - SQL инъекции
  - Межсайтовый скриптинг

Статический анализ программ

Статический анализ позволяет получать знания о коде без его запуска

- Лексический анализ
- Синтаксический анализ
- Семантический анализ

Обзор существующих инструментов

- Проверка выражения на соответствие описанию некоторой эталонной грамматики
  - Java String Analyzer
  - PHP String Analyzer
  - Alvor
- Статический анализ программы на уязвимость
  - Pixy
  - Stranger
  - SAFELI

Разработка инструментов

Возможны два подхода

- Создание универсального инструмента
- Создание набора генераторов и библиотек стандартных функций
  - По описанию языка генерируются анализаторы (Lex, Yacc, ANTLR и т.д.)
  - Для создания конечных решений можно использовать стандартные функции

Проект YaccConstructor

- YaccConstructor — модульный инструмент, предназначенный для проведения лексического и синтаксического анализа
- YaccConstructor — платформа для поддержки встроенных языков

- В лексическом анализе не поддерживаются циклы и строковые операции

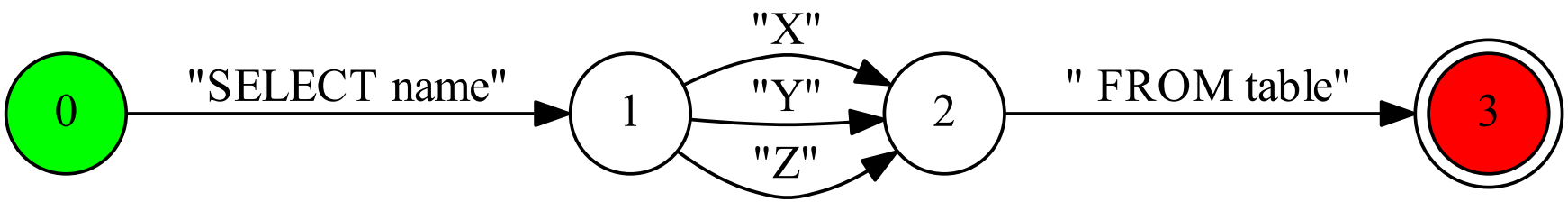
Постановка задачи

Цель: разработать автоматизированный подход создания лексического анализатора для динамически формируемого кода

- разработать алгоритм лексического анализа выражений, формируемых с помощью строковых операций и циклов
- сохранить привязку лексических единиц к исходному коду
- реализовать генератор лексических анализаторов

Аппроксимация

- ```
  private void Go (int cond){
   string columnName = cond > 3 ? "X":(cond < 0 ? "Y":"Z");
   string query =
     "SELECT name" + columnName + " FROM table";
   Program.ExecuteImmediate(query);
  }
  ```
- Множество значений
  { "SELECT nameX FROM table"; "SELECT nameY FROM table"; "SELECT nameZ FROM table"}
- Результат аппроксимации

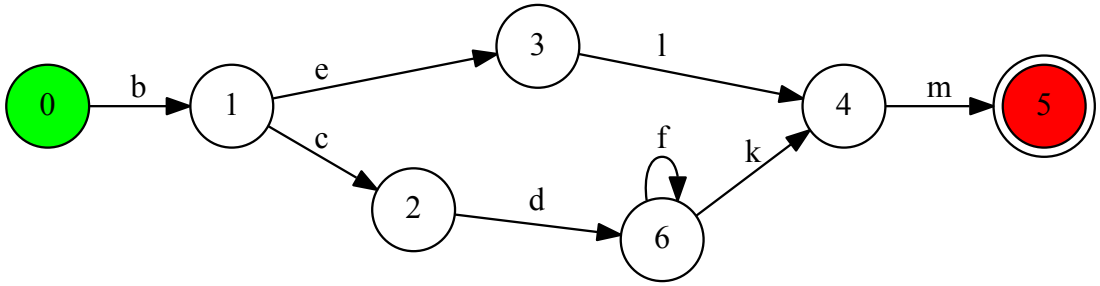

  ```

Строковые операции

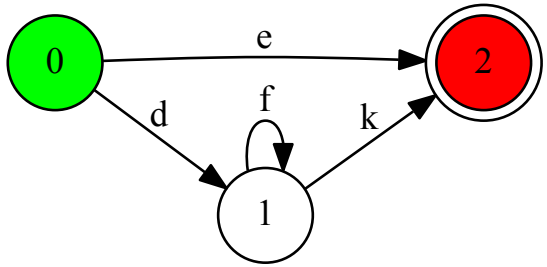
- ```
  string s = "SELECT nameX FROM tableY";
  s = s.Replace("SELECT nameX", "b");
  ```
- Многие строковые операции могут быть выражены с помощью конечных автоматов
- Для построения аппроксимации множества значений выражения использовался алгоритм, описанный в статье Fang Yu “Automata-based symbolic string analysis for vulnerability detection”
  ```

Пример

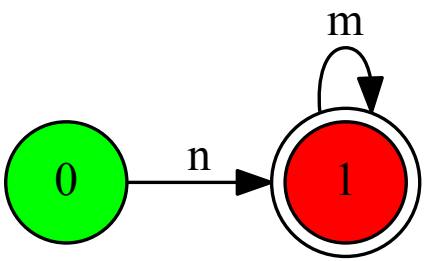
$M = \texttt{replace}(M_1, M_2, M_3)$



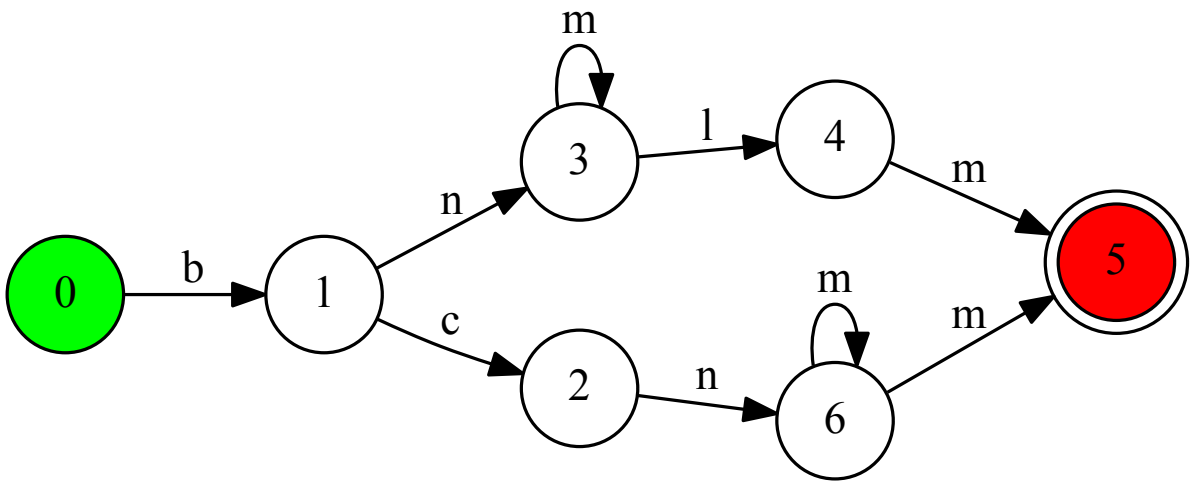
M_1



M_2



M_3



M

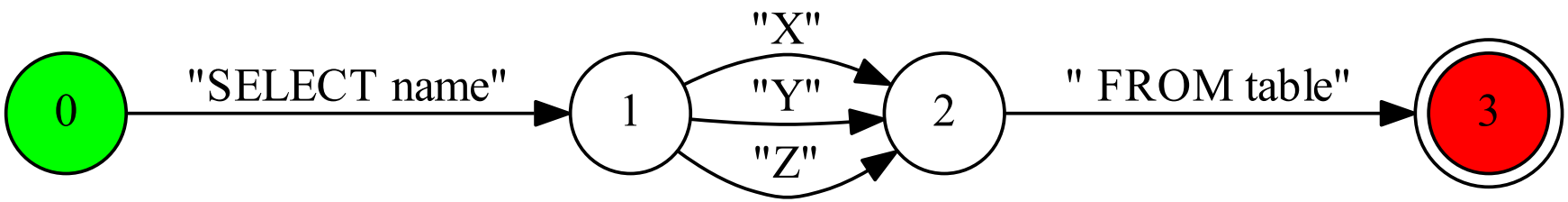
Лексический анализ строковых выражений

- На вход анализатору подается конечный автомат, полученный в результате аппроксимации множества значений строкового выражения
- На выходе получаем либо конечный автомат над токенами, либо список лексических ошибок. Токен содержит в себе:
  - идентификатор токена
  - конечный автомат, описывающий все возможные последовательности символов для данного токена

Задача лексического анализа: получение конечного автомата над алфавитом токенов эталонной грамматики из конечного автомата над алфавитом символов обрабатываемого языка

Пример

- ```
  private void Go (int cond){
    string columnName = cond > 3 ? "X":(cond < 0 ? "Y":"Z");
    string query =
      "SELECT name" + columnName + " FROM table";
    Program.ExecuteImmediate(query);
  }
  ```
- Результат аппроксимации



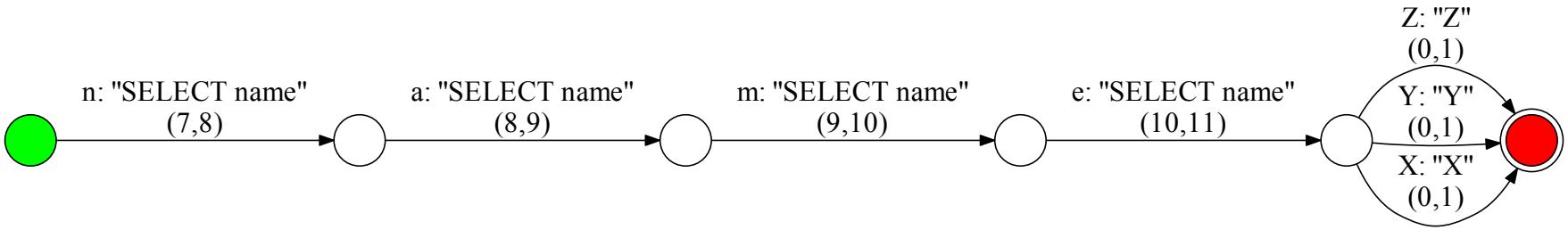
- Результат лексического анализа


  ```

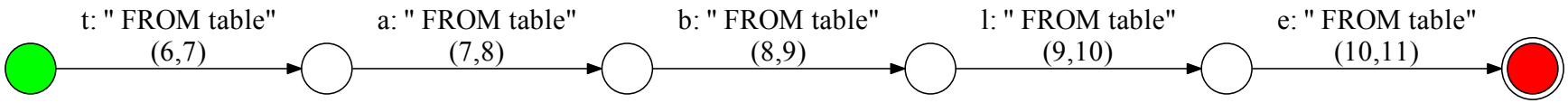
Пример

- Результат лексического анализа

- Конечный автомат первого токена `IDENT`



- Конечный автомат второго токена `IDENT`



Конечный преобразователь

- **Конечный преобразователь** — это конечный автомат, который может выводить конечное число символов для каждого входного символа
- **Композиция** конечных преобразователей — это два последовательно взаимодействующих конечных преобразователя: выход первого конечного преобразователя является входом для второго конечного преобразователя

Генератор лексических анализаторов

Вход:

- Лексическая спецификация языка

```
let digit = ['0'-'9']
let whitespace = [' ' '\t' '\r' '\n']
let num = ['-']? digit+ ('.'digit+)? (['e' 'E'] digit+)?
```

FsLex

```
rule token = parse
| whitespace  token lb
| num   { NUMBER(lexeme lb) }
| '-'   { MINUS(lexeme lb) }
| '/'   { DIV(lexeme lb) }
| '+'   { PLUS(lexeme lb) }
| "**"  { POW(lexeme lb) }
| '*'   { MULT(lexeme lb) }
```

YaccConstructor

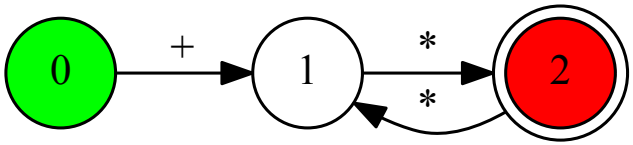
```
rule token = parse
| whitespace { None }
| num   { Some(NUMBER(gr)) }
| '-'   { Some(MINUS(gr)) }
| '/'   { Some(DIV(gr)) }
| '+'   { Some(PLUS(gr)) }
| "**"  { Some(POW(gr)) }
| '*'   { Some(MULT(gr)) }
```

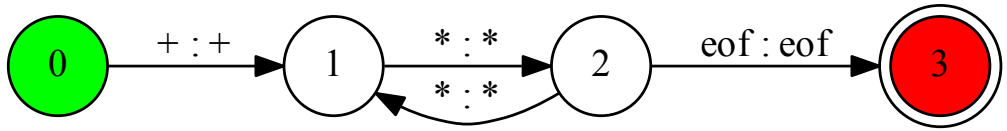
- Описание токенов

Выход: Описание конечного преобразователя и вспомогательные функции

Алгоритм лексического анализа

- **Этап 0**.
  Вход: конечный автомат, полученный в результате построения аппроксимации
  Выход: конечный преобразователь, построенный из входного конечного автомата

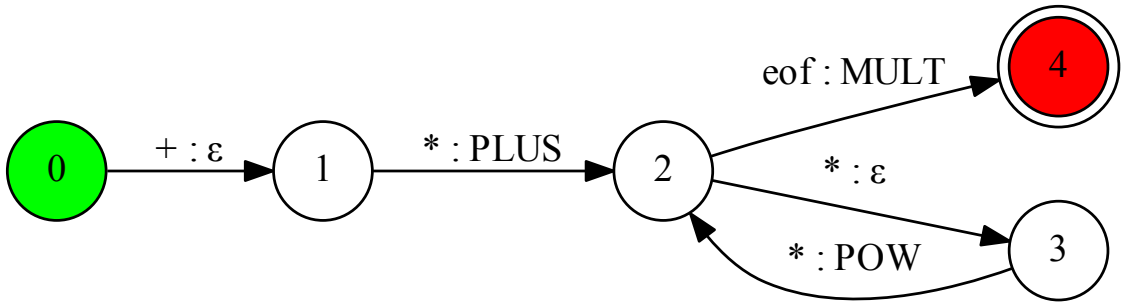




Алгоритм лексического анализа

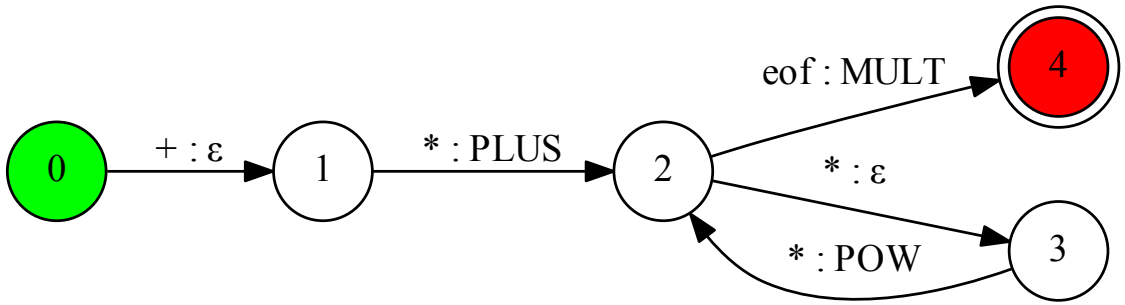
- **Этап 1.**
  Вход:
  - конечный преобразователь, полученный на Этапе 0
  - конечный преобразователь, полученный из описания, построенного генератором лексических анализаторов

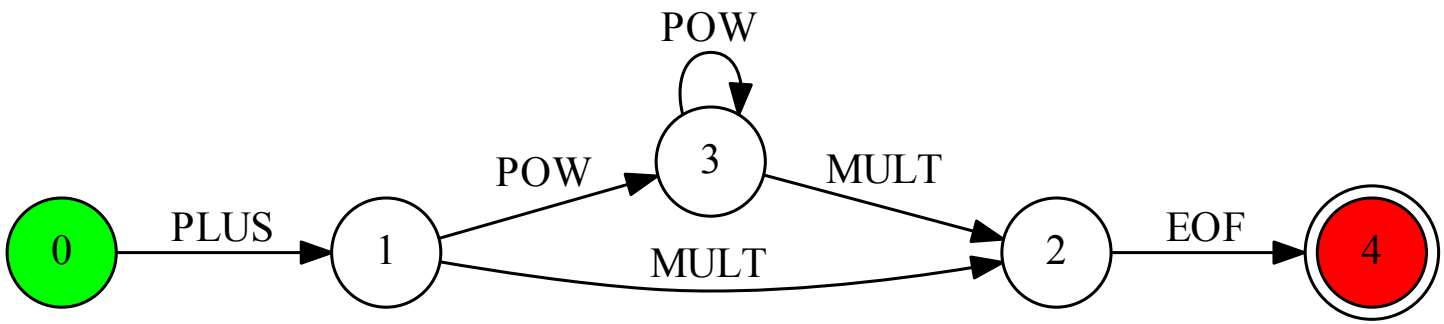
  Выход: конечный преобразователь и набор лексических ошибок



Алгоритм лексического анализа

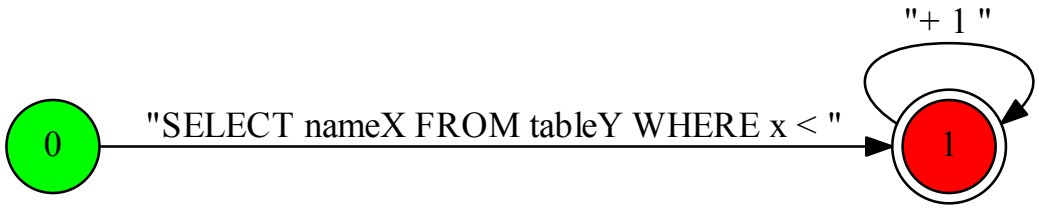
- **Этап 2**. Интерпретация конечного преобразователя
  Вход: конечный преобразователь, полученный на Этапе 1
  Выход: конечный автомат над алфавитом токенов эталонной грамматики



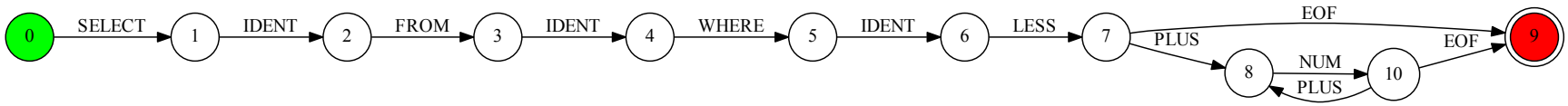


Пример 1

- ```
  private void Go (int number){
      string query =
                  "SELECT nameX FROM tableY WHERE x < ";
      while(query.Length < number){ query += "+ 1 ";}
      Program.ExecuteImmediate(query);
  }
  ```
- Результат аппроксимации



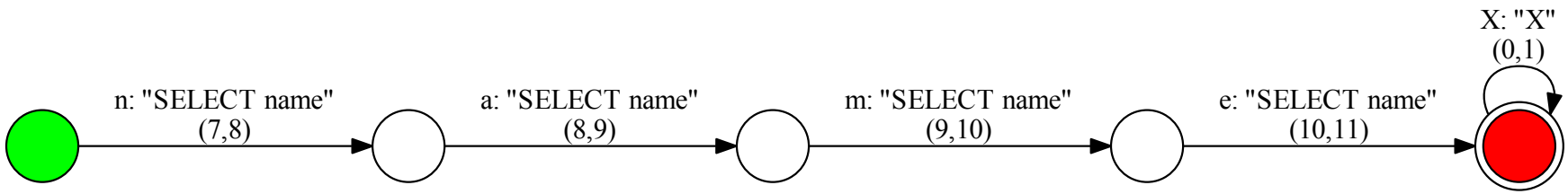
- Результат лексического анализа


  ```

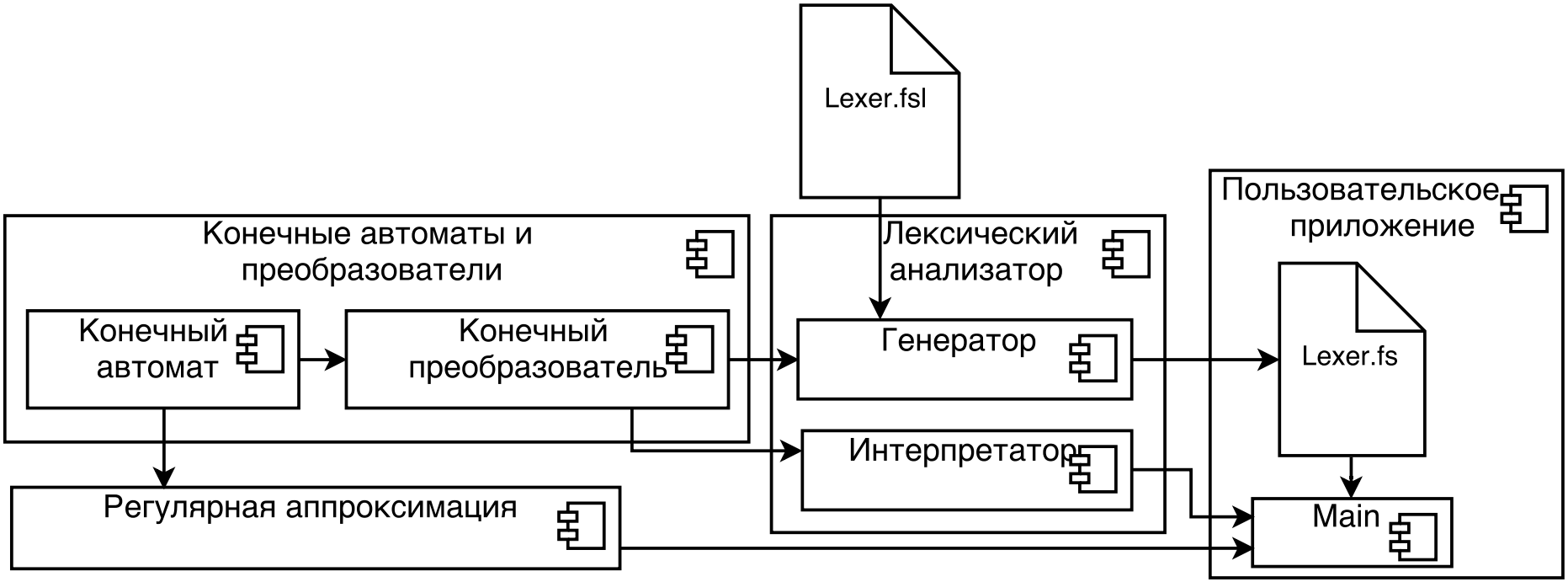
Пример 2

- ```
  string query = "SELECT name";
  for(int i = 0; i < 10; i++){ query += "X";}
  query += " FROM tableY";
  Program.ExecuteImmediate(query);
  ```
- Результат лексического анализа

- Конечный автомат первого токена IDENT


  ```

Архитектура инструмента



Результаты

В рамках данной работы были получены следующие результаты:

- Разработан алгоритм лексического анализа выражений, формируемых с помощью строковых операций и циклов
- Реализован генератор лексических анализаторов на основе предложенного алгоритма

Контактная информация

- Полубелова Марина: polubelovam@gmail.com
- Григорьев Семён: Semen.Grigorev@jetbrains.com
- Исходный код YaccConstructor:
  `https://github.com/YaccConstructor`