Оптимизация алгоритма лексического анализа динамически формируемого кода

Автор: Александр Байгельдин, студент СПбГУ
Научный руководитель: ст.пр., мастер ИТ С.В. Григорьев

Санкт-Петербургский государственный университет
Кафедра системного программирования

18 мая 2016г.

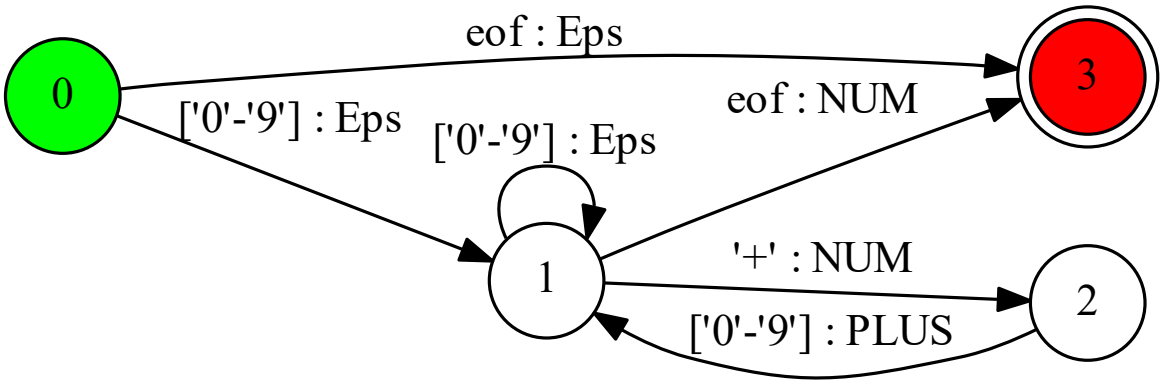
Динамически формируемый код

```
1  private void Go(bool cond)
2  {
3    string tableName = cond ? "Sold" : "OnSale ";
4    string queryString =
5        "SELECT ProductID, UnitPrice, ProductName "
6            + "FROM dbo.products_" + tableName
7            + "WHERE UnitPrice > 1000 "
8            + "ORDER BY UnitPrice DESC;";
9    Program.ExecuteImmediate(queryString);
10 }
```

Лексический анализ

В классическом лексическом анализе применяются конечные преобразователи (Finite State Transducers)

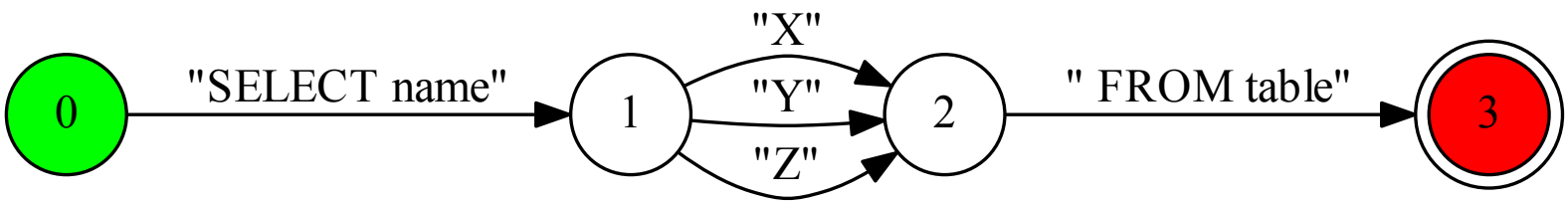
```
rule token = parse
| ['0'-'9']  { Some(NUM(gr)) }
| '+'  { Some(PLUS(gr)) }
```



Регулярная аппроксимация

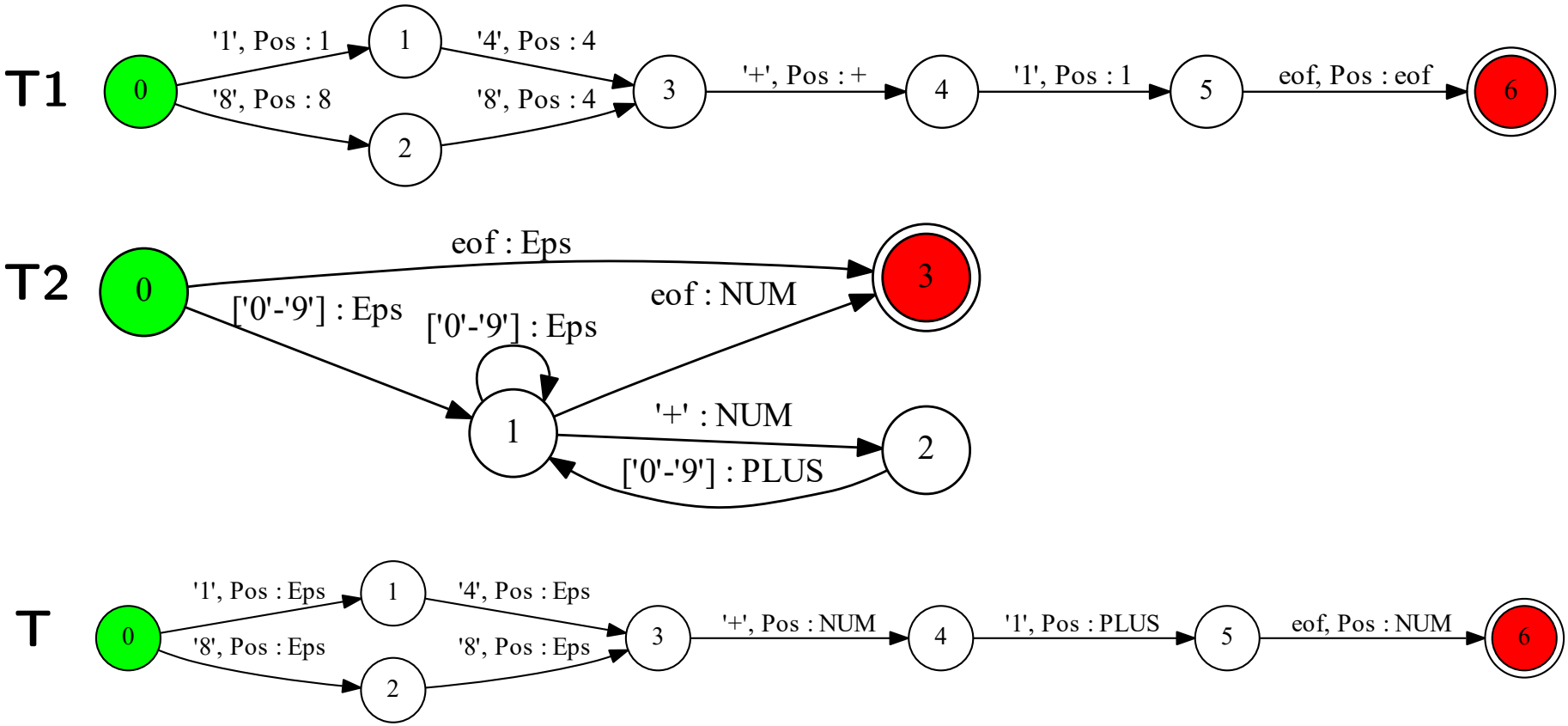
Для динамически формируемого кода можно построить регулярную аппроксимацию

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



Композиция FST

Важной частью лексического анализа динамически формируемого кода является операция композиции FST ($T = T1 \circ T2$)



YaccConstructor

- YaccConstructor — исследовательский проект в области лексического и синтаксического анализа
- В YaccConstuctor для лексического анализа динамически формируемого кода применяется подход, в основе которого лежит построение регулярной аппроксимации и композиция FST
- Реализованный в YaccConstructor алгоритм композиции FST обладает недостаточной производительностью

Постановка задачи

Целью работы является исследование возможности улучшения производительности лексического анализа динамически формируемого кода

Задачи:

- Исследовать алгоритмы композиции FST
- Реализовать и интегрировать в проект YaccConstructor более оптимальный алгоритм композиции
- Сравнить производительность реализаций текущего и выбранного алгоритмов

Решение

- Временная сложность текущего алгоритма:

$$O(V_1 * V_2 * D_1 * D_2)$$

где V — число вершин, D — максимальное количество исходящих ребер

- В текущем алгоритме образуются недостижимые вершины, которые приходится удалять
- Временная сложность выбранного алгоритма:

$$O(V_1 * V_2 * D_1 * (log(D_2) + M_2))$$

где M — степень недетерминированности

- В выбранном алгоритме недостижимых вершин не образуется

Особенности реализации

- F# — язык семейства .NET
- YaccConstructor — исследовательский проект в области лексического и синтаксического анализа
- QuickGraph — библиотека .NET для работы с графами
- Реорганизация проекта YaccConstructor
  - Библиотека для работы с конечными преобразователями YC.FST интегрирована в проект QuickGraph
  - Произведен рефакторинг, заключавшийся в подмене сторонней библиотеки QuickGraph в YaccConstructor на использование собственной сборки

Измерения

Кол-во вершин	Кол-во ребер	Время работы текущего алгоритма (мс)	Время работы выбранного алгоритма (мс)
250	738	16526	1133
711	1766	30285	2068
215	895	4045	248
310	687	7184	394

Математическое ожидание и среднее квадратичное отклонение ускорения (в кол-ве раз) выбранного алгоритма по сравнению с текущим:

$$M = 18.7, \sigma = 3.23$$

Результаты

- Исследованы алгоритмы композиции FST
- Выбранный алгоритм реализован и интегрирован в проект YaccConstructor
- Произведено сравнение производительности реализаций текущего и выбранного алгоритмов
- Выступление на конференции «Современные технологии в теории и практике программирования»
  - Публикация в сборнике конференции