Санкт-Петербургский государственный университет

Кафедра системного программирования

Ковалев Дмитрий Александрович

Динамически формируемый код: синтаксический анализ контекстно-свободной аппроксимации

Курсовая работа

Научный руководитель:
к. ф.-м. н., ст. преп. Григорьев С. В.

Санкт-Петербург
2016

Оглавление

Введение

Современные языки программирования общего назначения поддерживают возможность работы со строковыми литералами, позволяя формировать из них выражения при помощи строковых операций. Строковые выражения могут создаваться динамически, с использованием таких конструкций языка, как циклы и условные операторы. Данный подход распространен, например, при формировании SQL-запросов к базам данных из программ, написанных на Java, C# и других высокоуровневых языках (см. листинг **??**).

```
private void Example (int cond) {
    string columnName = cond > 42 ? "X" : "Y";
    string queryString =
        "SELECT name" + columnName + " FROM table";
    Program.ExecuteImmediate(queryString);
}
```

Листинг 1: Динамически формируемый SQL-запрос

Недостаток рассматриваемого метода заключается в том, что формируемое выражение, с точки зрения компилятора, является обычной строкой, и, следовательно, не проходит статических проверок на корректность и безопасность, что приводит к ошибкам времени выполнения и усложняет разработку и сопровождение системы. Более того, использование классических методов статического анализа, таких как синтаксический разбор, для динамически формируемых выражений невозможно, так как, в общем случае, они не представимы в виде линейного потока, который принимают на вход традиционные алгоритмы.

Существует несколько подходов, позволяющих частично решать проблему проверки корректности такого вида выражений. Один из них основан на проверке включения языка, генерируемого программой, в язык, заданный пользователем в виде простого перечисления строк, либо при помощи грамматики [**?**, **?**]. Здесь под языком, который генерирует программа, подразумевается множество значений динамически формируемого выражения, а точнее, некоторая его аппроксимация,

ведь в общем случае такое множество может быть рекурсивно перечислимым. Другой подход заключается в проведении синтаксического анализа аппроксимации множества значений [**?**, **?**]. Стоит отметить, что такой подход позволяет не просто отвечать на вопрос о включении языков, но и реализовать дополнительную функциональность, такую как вычисление семантики, а также, в некоторых случаях, автодополнение и рефакторинг динамически формируемого кода [**?**].

В магистерской диссертации [**?**] был описан алгоритм синтаксического анализа регулярной аппроксимации динамически формируемых выражений на основе алгоритма обобщенного синтаксического анализа GLL [**?**] и подробности его реализации в рамках проекта YaccConstructor [**?**]. Данное решение обладает несколькими важными достоинствами. Во-первых, результатом работы алгоритма является компактное представление множества деревьев разбора корректных значений выражения (при этом некорректные значения отбрасываются), которое может быть использовано для проведения других видов статического анализа. Во-вторых, реализация представляет собой классический генератор синтаксических анализаторов, что позволяет легко добавлять поддержку новых языков.

Так как большинство языков программирования принадлежит к классу контекстно-свободных, использование регулярной аппроксимации влечет за собой появление синтаксических ошибок, отсутствующих в исходном множестве значений выражения. В данной работе, также выполненной в рамках проекта YaccConstructor, будет представлен алгоритм, который основан на описанной модификации GLL, но позволяет производить синтаксический анализ более точной контекстно-свободной аппроксимации, что ведет к сокращению количества ложных синтаксических ошибок. Также, будет описан процесс получения данной аппроксимации из исходного кода и преобразование ее в форму, пригодную для работы с GLL-анализатором.

1. Постановка задачи

Целью дипломной работы является разработка алгоритма синтаксического анализа контекстно-свободной аппроксимации динамически формируемых выражений на базе существующей модификации алгоритма GLL. Для ее достижения были поставлены перечисленные ниже задачи.

- Реализовать подходящее для последующего анализа представление контекстно-свободной аппроксимации.
- Модифицировать существующий алгоритм синтаксического анализа регулярной аппроксимации, основанный на GLL.
- Провести экспериментальное исследование.

2. Обзор

В данной секции описываются существующие методы анализа динамически формируемых выражений, в том числе алгоритм синтаксического анализа регулярной аппроксимации, реализованный в проекте YaccConstructor.

2.1. Существующие инструменты

Под регулярной аппроксимацией далее будем подразумевать регулярное выражение (или соответствующий конечный автомат), аппроксимирующий сверху множество значений динамически формируемого выражения.

В настоящее время существуют два основных подхода к анализу динамически формируемого кода. Первый из них заключается в проверке включения языков. Он позволяет лишь отвечать на вопрос, содержатся ли выражения, генерируемые программой, в эталонном языке, который был задан пользователем при помощи контекстно-свободной грамматики, либо явным перечислением. Следующие инструменты реализуют данный подход.

- **Java String Analyzer (JSA, [?, ?])** — инструмент для статического анализа строковых выражений в программах, написанных на Java. Для каждого выражения строит регулярную аппроксимацию и проверяет включение языка, генерируемого данной аппроксимацией, в эталонный контекстно-свободный язык.
- **PHP String Analyzer (PHPSA, [?])** — инструмент для анализа строковых выражений в программах на PHP. Отличается от предыдущего инструмента использованием контекстно-свободной аппроксимации, что позволяет увеличить точность анализа.

Второй подход заключается в проведении синтаксического анализа некоторого представления множества значений динамически формируемого выражения.

- **Alvor [?]** — плагин для IDE Eclipse. Позволяет статически проверять корректность SQL-запросов, генерируемых в программах на Java. Для представления множества значений выражения использует понятие абстрактной строки, которая может быть преобразована в конечный автомат над алфавитом токенов исходного языка. Данный автомат затем передается для синтаксического анализа модифицированному GLR-алгоритму. Alvor не поддерживает обработку строковых операций, за исключением конкатенации, и выражений, которые формируются при помощи циклов.

- **Varis [?]** — плагин для Eclipse, представлен в 2015 году. Поддерживает анализ выражений на HTML, CSS и JavaScript в программах на PHP. Реализует функции подсветки кода, автодополнения и перехода к объявлению.

- **Абстрактный синтаксический анализ [?]** — в данном подходе множество значений выражения описывается при помощи dataflow-уравнений, полученных из исходного кода программы (см. рис. **??**). Эти уравнения затем решаются методом поиска наименьшей неподвижной точки в домене LR-стеков, т.е. одновременно с решением уравнения производится синтаксический анализ с использованием стандартного LR(LALR)-автомата. Результатом анализа является множество LR-стеков, представляющих различные варианты разбора выражения. К сожалению, в статьях, посвященных этому подходу, не было описано эффективное представление результатов, которое могло бы использоваться для более сложных видов анализа.

```
x = ''a''                          X0 = a
while ...                          X1 = X0 ∪ X2
    x = ''['' . x . '']''          X2 = [ . X1 . ]
print x                            X3 = X1
```

Рис. 1: Исходный код и dataflow-уравнения, извлеченные из него

2.2. YaccConstructor

YaccConstructor (далее YC, [**?**]) — исследовательский проект лаборатории языковых инструментов JetBrains на математико-механическом факультете СПбГУ, направленный на исследования в области лексического и синтаксического анализа, а также статического анализа динамически формируемого кода. Проект включает в себя одноименную модульную платформу для разработки лексических и синтаксических анализаторов, содержащую большое количество компонент: язык описания грамматик YARD, преобразования над грамматиками и др. Основным языком разработки является F#.

Ранее, в рамках магистерской диссертации [**?**], на платформе YC был разработан алгоритм синтаксического анализа регулярной аппроксимации динамически формируемого кода, который позволяет получать компактное представление множества деревьев разбора анализируемого выражения.

2.3. Алгоритм синтаксического анализа регулярной аппроксимации (YC)

Предложенный алгоритм является модификацией алгоритма обобщенного синтаксического анализа Generalized LL (GLL, [**?**]). Данная модификация принимает на вход не линейный поток токенов, как оригинальный алгоритм, а детерминированный конечный автомат над алфавитом токенов. Данный автомат является результатом работы специальной процедуры лексического анализа над динамически формируемым выражением. Пример такого автомата приведен на рисунке **??**.

Отметим, что построенный автомат может порождать строки, которые не принадлежат множеству значений исходного выражения. В данном примере программа генерирует строки, которые представляют собой правильные скобочные последовательности: `[a]; [[a]]; ...` . В это же время, множество строк, порождаемых автоматом, содержит, например, такие элементы: `[a; [[a; [a]]` .

```
x = ''a''
while ...
    x = ''['' . x . '']''
print x
```

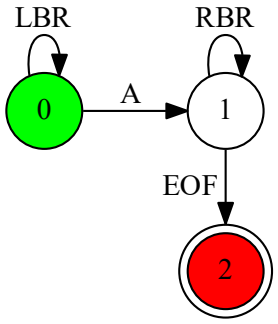

Рис. 2: Пример регулярной аппроксимации

$$\begin{aligned} start &::= s \\ s &::= \texttt{LBR}\ s\ \texttt{RBR} \mid \texttt{A} \end{aligned}$$

Рис. 3: Грамматика G_1

Результатом работы алгоритма будет сжатое представление леса разбора всех корректных значений выражения — Shared Packed Parse Forest (SPPF, [**?**]). Под корректными значениями здесь подразумеваются строки, разобранные анализатором, т.е. выводимые во входной грамматике. Так, если провести синтаксический анализ рассматриваемой аппроксимации по грамматике, представленной на рисунке **??**, то получится SPPF, изображенный на рисунке **??**. При этом после завершения работы алгоритма пользователь получит отчет о том, что во время анализа было обнаружено некоторое количество синтаксических ошибок.

Чтобы сократить количество подобных ложных ошибок, необходимо предоставить более точную аппроксимацию исходного множества значений и изменить алгоритм таким образом, чтобы он мог работать с этой аппроксимацией. Этому и будет посвящена данная работа.

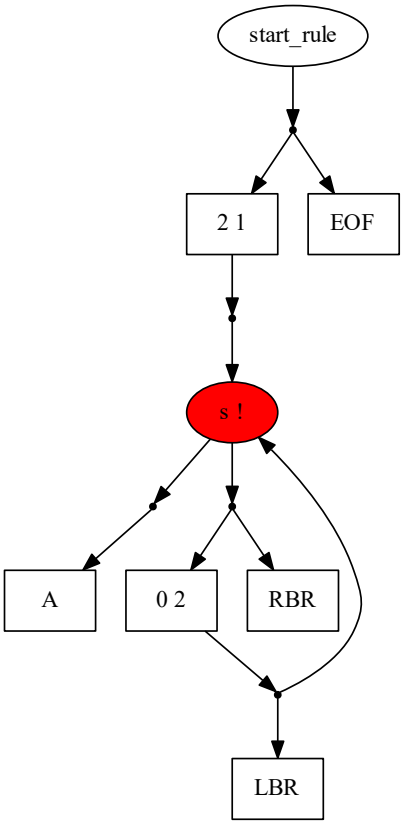

Рис. 4: SPPF для грамматики G_1 и автомата с рис. **??**

3. КС-аппроксимация динамически формируемого выражения

Вернемся к одному из рассмотренных ранее подходов к анализу динамически формируемого кода. На рисунке **??** был приведен пример dataflow-уравнений, получаемых из исходного кода программы для проведения абстрактного синтаксического анализа. Можно заметить, что данные уравнения представляют собой контекстно-свободную грамматику, где нетерминальные символы — имена переменных, а терминалы — строковые литералы (см. рис **??**). При этом оператор объединения будет означать, что для нетерминала существует несколько продукций.

Таким образом, преобразованные dataflow-уравнения можно использовать как контекстно-свободную аппроксимацию динамически формируемого выражения. Для данного примера такая аппроксимация, в от-

```
X0 = a                    X0 ::= a
X1 = X0 ∪ X2              X1 ::= X0 | X2
X2 = [ . X1 . ]           X2 ::= [ X1 ]
X3 = X1                   X3 ::= X1
```

Рис. 5: Представление dataflow-уравнений в виде грамматики

личие от регулярной (рис. **??**), описывает в точности то множество строк, которое генерирует исходная программа.

Предыдущая модификация GLL в проекте YC работает с графовым представлением конечного автомата. Соответственно, для поддержки нового вида аппроксимации необходимо было найти и реализовать удобное для анализа представление KC-грамматики в виде графа.

3.1. Grammar Flow Graph

Такое представление было взято из статьи [?], вышедшей в 2015 году. Grammar Flow Graph (GFG) (так авторы, Keshav Pingali и Gianfranco Bilardi, назвали свою разработку) — это граф с вершинами и ребрами специальных типов, при помощи которого можно смоделировать любую контекстно-свободную грамматику. Пример GFG приведен на рисунке **??**.

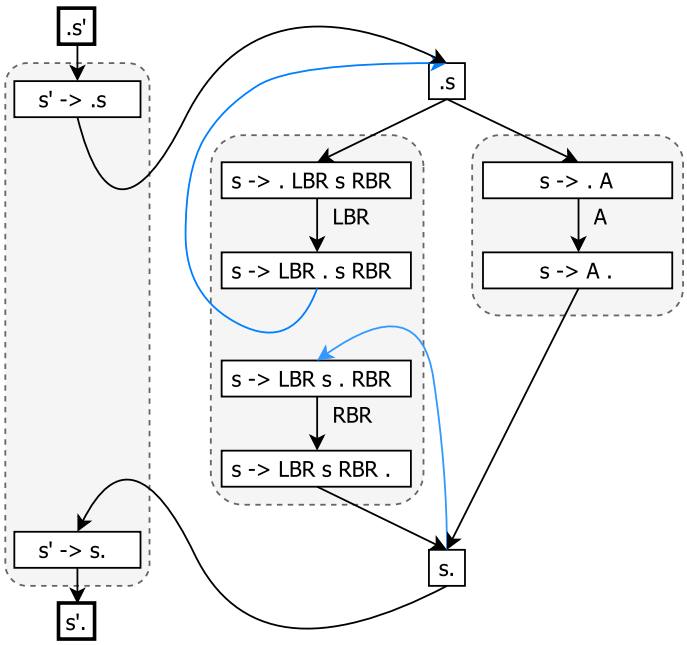

Рис. 6: GFG для грамматики G_1

4. Результаты

- Промежуточные результаты.
  - Написан обзор существующих подходов к анализу динамически формируемых выражений.
  - Реализовано внутреннее представление GFG и преобразование грамматик в это представление.
- В ходе дальнейшей работы планируется следующее.
  - Модифицировать GLL таким образом, чтобы позволить производить синтаксический анализ GFG.
  - Провести экспериментальное исследование полученного алгоритма.