



# Синтаксический анализ регулярных множеств

**Автор:** Екатерина Вербицкая

Семинар научно-исследовательских лабораторий JetBrains

21 ноября 2015г.

# Динамически формируемые выражения

- *Динамически формируемые выражения* генерируются из строковых литералов во время исполнения программы
  - ▶ Также могут называться *динамическим* или *встроенным кодом*
- Программа, формирующая такие выражения — *генератор* или *основная программа*
- *Точка интереса* — строка основной программы, в которой осуществляется чтение значения формируемого выражения
- *Множество значений (динамического) выражения* — множество всех выражений, которые могут породиться в процессе выполнения основной программы в данной точке интереса

# Динамически формируемые выражения: примеры

- Встроенный SQL

```
SqlCommand myCommand = new SqlCommand(  
    "SELECT * FROM table WHERE Column = @Param2",  
    myConnection);  
myCommand.Parameters.Add(myParam2);
```

- Динамический SQL

```
IF @X = @Y  
    SET @TBL = ' #table1 '  
ELSE  
    SET @TBL = ' table2 '  
SET @S = 'SELECT x FROM' + @TBL + 'WHERE ISNULL(n,0) > 1'  
EXECUTE (@S)
```

- Встроенный код — код на некотором языке программирования, поэтому, как следствие, необходимы:
  - ▶ Поддержка в IDE (подсветка кода, сообщение об ошибках, рефакторинги)
  - ▶ Трансляция (миграция с устаревших технологий на новые платформы)
  - ▶ Обнаружение уязвимостей

# Статический анализ встроенного кода

- Производится без выполнения основной программы
- Проверяет выполнение некоторых свойств для *каждого* возможного выражения
- В общем случае задача статического анализа встроенных языков неразрешима
- При *регулярной аппроксимации* множества значений выражения некоторые задачи становятся разрешимыми
  - ▶ *Регулярная аппроксимация* — аппроксимация (сверху) регулярным языком множества значений выражения

# Существующие решения

- PHP String Analyzer, Java String Analyzer, Alvor
  - ▶ Статические анализаторы PHP, Java и SQL, встроенных в Java
- Kyung-Goo Doh et al.
  - ▶ Проверяет синтаксическую корректность встроенного кода
- PHPStorm
  - ▶ IDE для PHP с поддержкой HTML, CSS, JavaScript
- IntelliLang
  - ▶ PHPStorm и IDEA plugin, поддержка многих языков
- STRANGER
  - ▶ Обнаружение уязвимости в коде на PHP
- Недостатки
  - ▶ Ограниченная функциональность
  - ▶ Сложно расширить новой функциональностью или поддержать новые языки
  - ▶ Не создают структурного представления динамического кода

# Статический анализ встроенных языков: схема

- Определение точек интереса
- Построение аппроксимации
- Лексический анализ
- **Синтаксический анализ**
  - ▶ Относительно эталонной грамматики
- Семантический анализ

# Статический анализ встроенных языков: схема

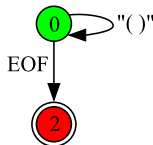
Код: выделена точка интереса

```
string res = "";  
for(i = 0; i < l; i++)  
    res = "()" + res;  
use(res);
```

Порождаемые значения { "", "()", "()", ..., "()"^l }

Регулярная аппроксимация ("()")\*

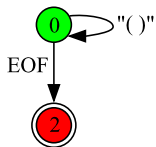
Аппроксимация



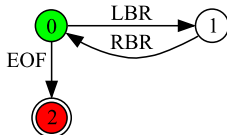


# Статический анализ встроенных языков: схема

Аппроксимация



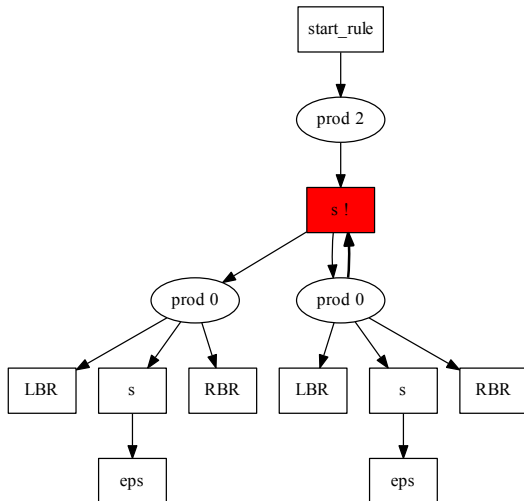
После лексического анализа



Грамматика

$start ::= s$   
 $s ::= LBR \ s \ RBR \ s$   
 $s ::= \epsilon$

Лес разбора



**Цель:** разработать алгоритм, подходящий для синтаксического анализа встроеного кода

- *Синтаксический анализ* — сопоставление каждого выражения из аппроксимирующего языка с некоторой эталонной грамматикой и построение деревьев разбора

## Задачи:

- Разработать алгоритм для синтаксического анализа регулярного языка, который строит конечный лес разбора
- Лес разбора должен содержать дерево разбора для каждой корректной строки из регулярного языка
- Диагностики ошибок не предусматривается: некорректные выражения игнорируются
- Алгоритм не должен зависеть от языков основной программы и встроенного кода

- **Входные данные**

- ▶ Эталонная КС грамматика  $G$
- ▶ Регулярный язык в виде графа ДКА без  $\epsilon$ -переходов над алфавитом терминалов грамматики  $G$

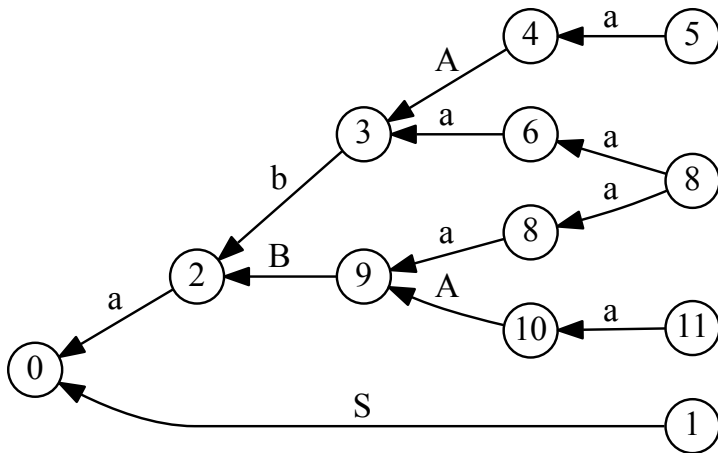
- **Результат**

- ▶ Лес разбора всех корректных выражений, принимаемых входным автоматом

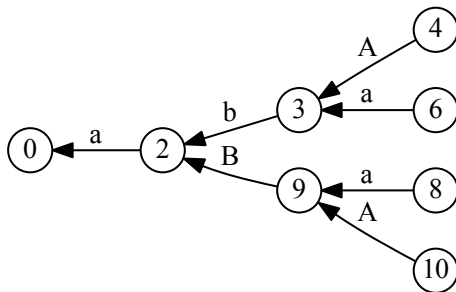
# Right-Nullled Generalized LR алгоритм

- RNGLR обрабатывает произвольные КС грамматики
- Табличный синтаксический анализ
- В случае LR-конфликтов, продолжает обработку всех возможных вариантов
  - ▶ Shift/Reduce конфликт
  - ▶ Reduce/Reduce конфликт
- Использует специальные структуры данных, ограничивающие потребление памяти и гарантирующие полиномиальное время работы

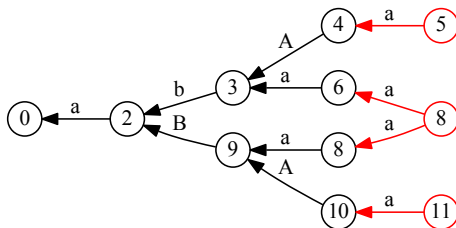
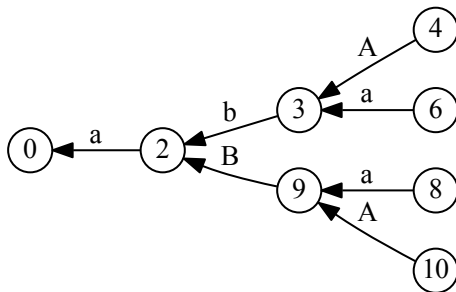
# Структуры данных: Graph-Structured Stack



## Операции в RNGLR-алгоритме: shift (сдвиг)

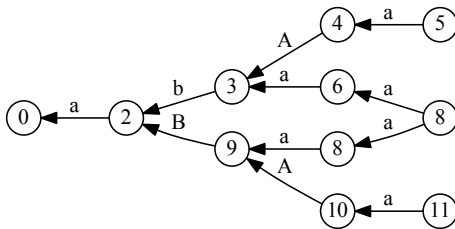


## Операции в RNGLR-алгоритме: shift (сдвиг)

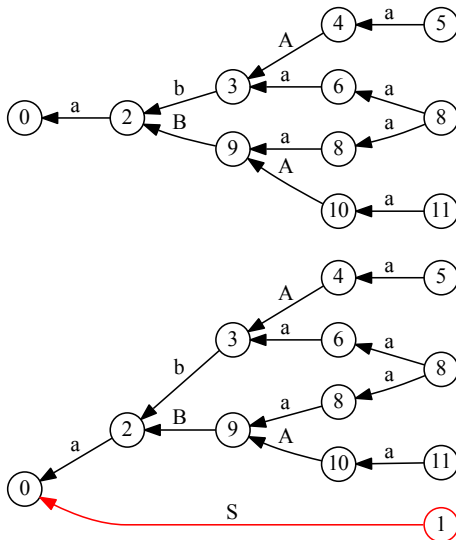




## Операции в RNGLR-алгоритме: reduce (редукция или свертка)



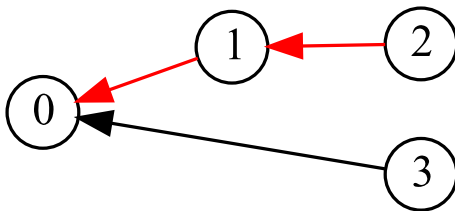
# Операции в RNGLR-алгоритме: reduce (редукция или свертка)



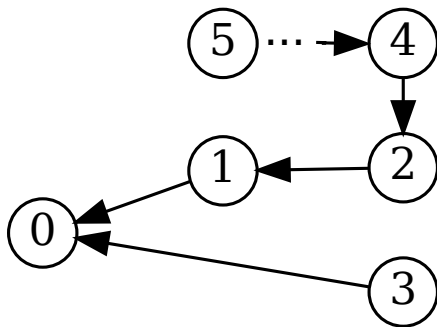
- Входной поток читается последовательно
  - ▶ Обработать все редукции
  - ▶ Сдвинуть следующий символ со входа
- При добавлении новой вершины в GSS вычисляется сдвиг
- При добавлении нового ребра — редукции

- GSS строится последовательно во время обхода графа входного автомата, похожим образом, что в RNLRL
- Новый тип “конфликта”: Shift/Shift
- Множество LR-состояний ассоциируется с каждой вершиной входного графа
- Порядок, в котором обходятся вершины входного графа, определяется очередью. Каждый раз, когда ребро добавляется в GSS, его начальная вершина добавляется в очередь
- Обнаружения ошибок не производится, некорректные строки игнорируются

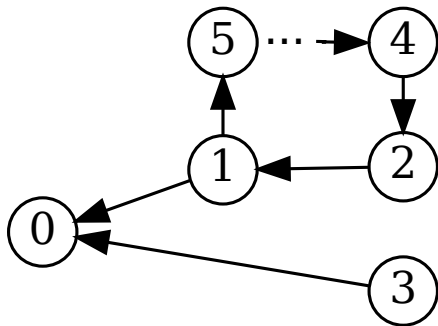
## Обработка циклов: начальный стек



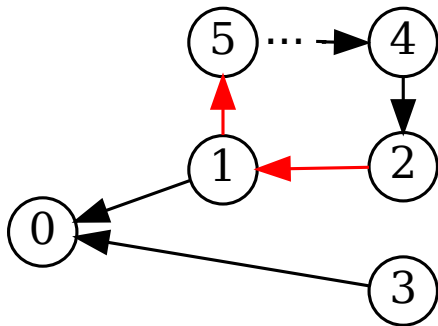
## Обработка циклов: добавили несколько ребер



## Обработка циклов: образовался цикл

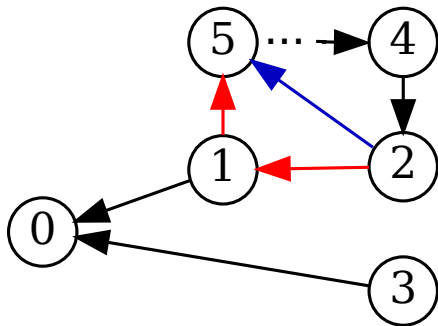


## Обработка циклов: новая редукция





## Обработка циклов: добавили новую редукцию

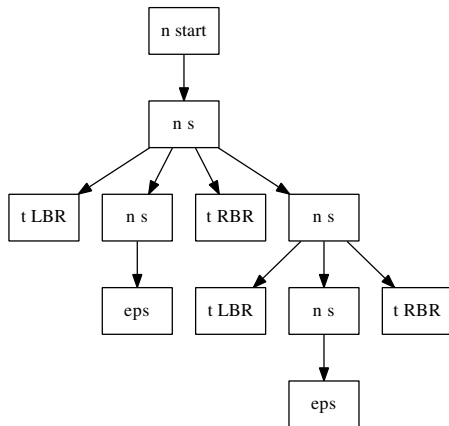
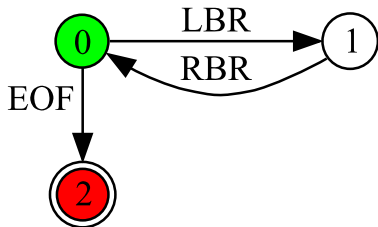


# Построение леса разбора

- Shared Packed Parse Forest — граф, в котором объединено множество деревьев вывода
- Строится так же, как в RNGLR-алгоритме
- С каждым ребром GSS ассоциируется фрагмент дерева вывода
- При обработке shift, создается дерево из одной вершины, соответствующей терминалу
- При обработке reduce, создается дерево, детьми корня которого становятся деревья, ассоциированные с ребрами путей
  - ▶ Фрагменты деревьев переиспользуются, не копируются
- Корень результирующего леса разбора ассоциирован с ребром GSS, соответствующим свертке к стартовому нетерминалу
  - ▶ Все недостижимые вершины удаляются из графа-леса

# Корректность алгоритма

*Корректное дерево* — дерево вывода строки, накопленной вдоль некоторого пути во входном графе



# Алгоритм: корректность

## Свойство (Завершаемость)

Алгоритм завершается при любом входе

## Свойство (Корректность)

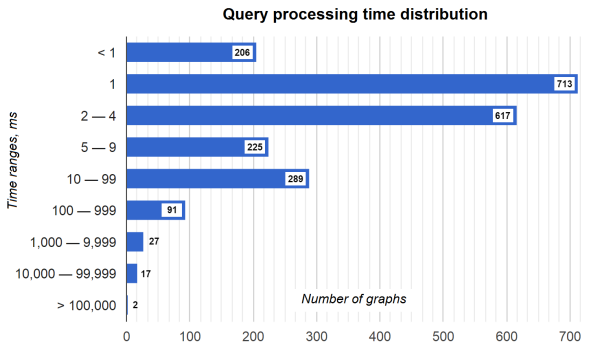
Каждое дерево, генерируемое из SPPF, корректно

## Свойство (Корректность)

Для каждой строки из входного регулярного множества, корректной относительно эталонной грамматики, из SPPF можно извлечь корректное дерево

- Алгоритм реализован как часть проекта YaccConstructor на языке программирования F#
- Генератор таблиц RNLGR и описание структур данных GSS и SPPF переиспользованы

- Данные взяли из промышленного проекта по миграции с MS-SQL на Oracle Server
- Всего 2,7 строк кода, 2430 запросов, 2188 успешно обработаны
- С 45 до 1 сократилось количество необработанных запросов



- Разработан алгоритм синтаксического анализа регулярной аппроксимации динамически формируемых выражений, который строит конечное представление леса разбора
- Доказаны завершаемость и корректность алгоритма
- Алгоритм реализован как часть проекта YaccConstructor
  - ▶ <https://github.com/YaccConstructor/YaccConstructor>
- Продемонстрирована применимость алгоритма для решения сложных задач

- Обнаружение ошибок и сообщение о них
- Теоретическая оценка сложности
- Вычисление семантики над полученным лесом разбора
- Интеграция в плагин к ReSharper