Синтаксический анализ контекстно-свободной аппроксимации

Автор: Дмитрий Ковалев

15 октября 2016г.

Динамически формируемый код

- SQL-запросы в C#

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

Динамически формируемый код

- SQL-запросы в C#

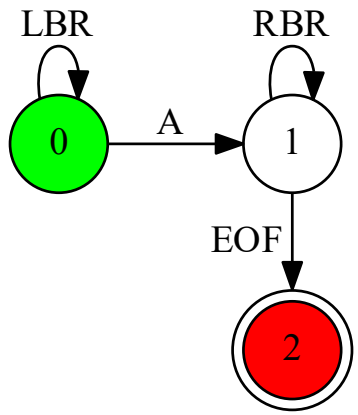
```
private void Example (int cond) {
    string columnName = cond > 42 ? "X" : "Y";
    string queryString =
        "SELECT name" + columnName + " FROM table";
    Program.ExecuteImmediate(queryString);
}
```

- Генерация HTML-страниц в PHP

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

Существующие решения

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



- Java String Analyzer, Alvor, PHP String Analyzer
  - Регулярная/КС-аппроксимация
  - Не строят структурного представления кода

Абстрактный синтаксический анализ

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

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

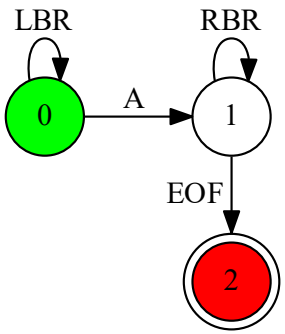
```
X0 ::= a
X1 ::= X0 | X2
X2 ::= [ X1 ]
X3 ::= X1
```

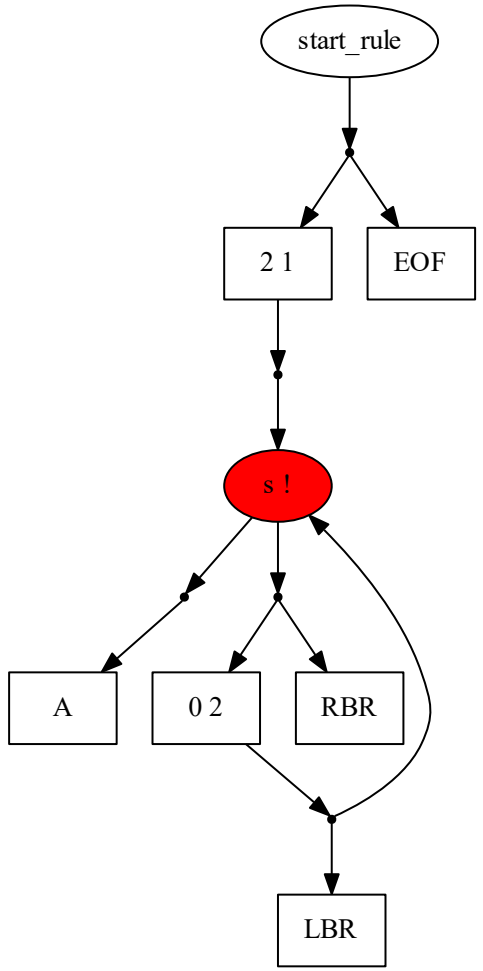
- Kyung-Goo Doh, Hyunha Kim, David A. Schmidt “Abstract LR-parsing”, 2011
- Решение dataflow-уравнений (LFP) в домене LR-стеков
- Отсутствует структурное представление результатов
- KC-аппроксимация

YaccConstructor

- Анализ регулярной аппроксимации на основе алгоритма Generalized LL (Анастасия Рагозина, 2016)

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





$$start ::= s$$
$$s ::= \texttt{LBR}\ s\ \texttt{RBR} \mid \texttt{A}$$

Generalized LL (GLL)

- Работает с произвольными KC-грамматиками
- Стандартные LL(k)-таблицы, допускаются конфликты в ячейках

Generalized LL (GLL)

- Работает с произвольными KC-грамматиками
- Стандартные LL(k)-таблицы, допускаются конфликты в ячейках
- Дескриптор — (L, u, i, w), где
  - L — позиция в грамматике вида $A \to \alpha \cdot X \beta$
  - u — вершина стека
  - i — позиция во входном потоке
  - w — узел SPPF

Generalized LL (GLL)

- Работает с произвольными KC-грамматиками
- Стандартные LL(k)-таблицы, допускаются конфликты в ячейках
- Дескриптор — (L, u, i, w), где
  - L — позиция в грамматике вида $A \to \alpha \cdot X \beta$
  - u — вершина стека
  - i — позиция во входном потоке
  - w — узел SPPF
- В случае конфликта создаются дескрипторы для каждого правила в ячейке таблицы

Generalized LL (GLL)

- Работает с произвольными KC-грамматиками
- Стандартные LL(k)-таблицы, допускаются конфликты в ячейках
- Дескриптор — (L, u, i, w), где
  - L — позиция в грамматике вида $A \to \alpha \cdot X \beta$
  - u — вершина стека
  - i — позиция во входном потоке
  - w — узел SPPF
- В случае конфликта создаются дескрипторы для каждого правила в ячейке таблицы
- В процессе анализа поддерживаются очередь обработки и мн-во созданных ранее дескрипторов

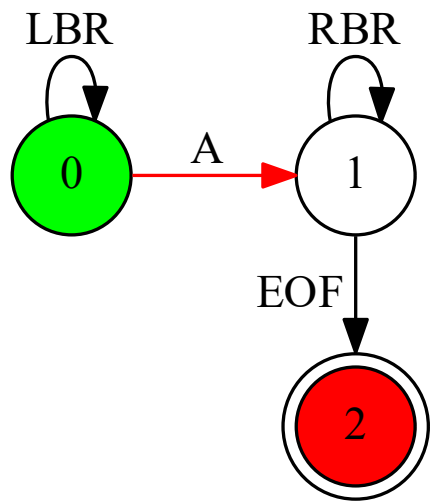
Модификация GLL

- Вход — детерминированный конечный автомат
- Вместо позиции в строке — номер вершины автомата

Модификация GLL

- Вход — детерминированный конечный автомат
- Вместо позиции в строке — номер вершины автомата
- Для текущей вершины просматриваются все исходящие ребра

$$
\begin{array}{rcl}
\mathit{start} & ::= & s \\
s & ::= & \cdot\ \texttt{LBR}\ s\ \texttt{RBR} \\
s & ::= & \texttt{A}
\end{array}
$$



Некорректные строки, генерируемые автоматом : “[A“, “[[A“, “[A]]“, ...

GLL & KC-аппроксимация

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

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

- Необходимо представление аппроксимирующей грамматики, подходящее для GLL

Постановка задачи

Целью работы является повышение точности диагностики ошибок в динамически формируемом коде (в рамках YC)
Задачи:

- Найти подходящее представление KC-аппроксимации
- Адаптировать GLL для работы с таким представлением

Замечание

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

Замечание

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

$\sim$

```
X1 ::= a | X2
X2 ::= [ X1 ]
X3 ::= X1
```

Замечание

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

$\sim$

```
X1 ::= a | X2
X2 ::= [ X1 ]
X3 ::= X1
```

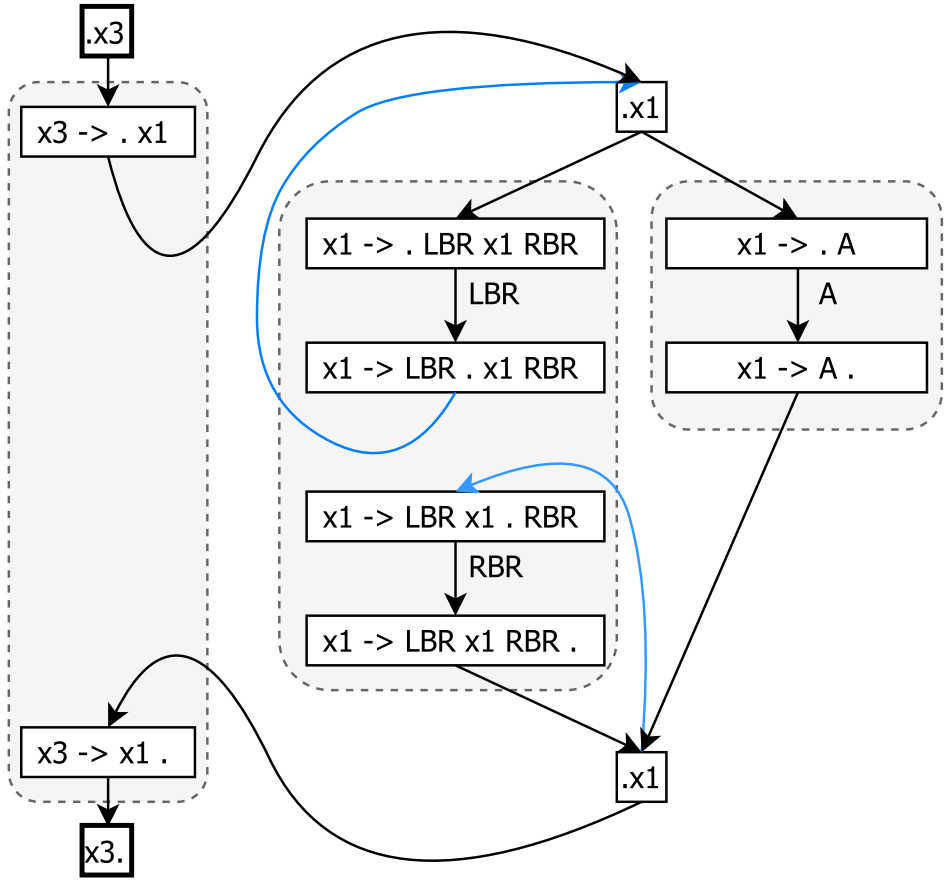
$\sim$

```
X1 ::= a
X1 ::= [ X1 ]
X3 ::= X1
```

Grammar Flow Graph (GFG)

```
X3 ::= X1
X1 ::= [ X1 ]
X1 ::= a
```

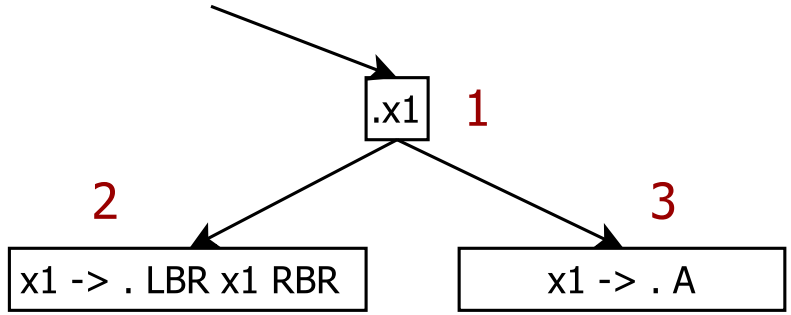
$A \to \alpha \cdot X \beta$	call node
$A \to \alpha X \cdot \beta$	return node



- Keshav Pingali, Gianfranco Bilardi "A Graphical Model for Context-Free Grammar Parsing", 2015

GLL & GFG

- В случае недетерминированного выбора создается дескриптор для каждого пути

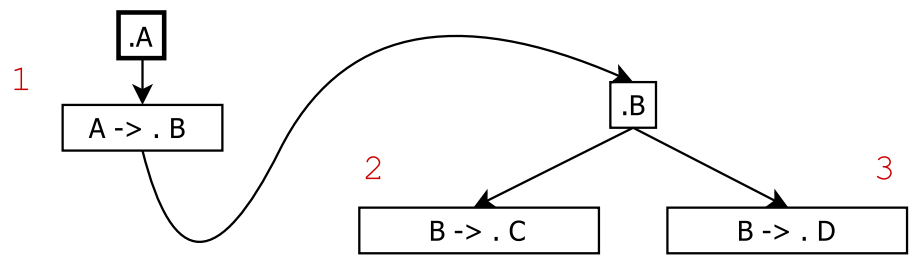


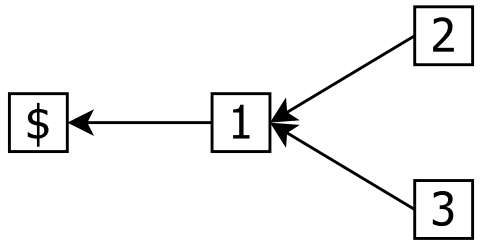
Current : $(L, u, 1, w)$

AddToQueue $(L, u, 2, w)$ $(L, u, 3, w)$

GLL & GFG

- Необходимо поддерживать CR-стек





- Graph-structured stack (GSS)
- Дескрипторы имеют вид (L, u, i, w, v), где v — вершина CR-стека

Эксперименты

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

Аппроксимация	Кол-во ошибок
Регулярная	5
KC (GFG)	0

Результаты

- Реализовано преобразование грамматики в GFG
- GLL адаптирован для работы с GFG