Синтаксический анализ динамически формируемых строковых выражений

Автор: Григорьев Семён Вячеславович
Научный руководитель: кандидат физико-математических наук, доцент Д.В. Кознов

Санкт-Петербургский государственный университет
Математико-Механический факультет
Кафедра системного программирования

2015г.

Динамически формируемые строковые выражения: пример

- Встроенный SQL

```
let p cond fldLst =
    let mutable flds = "id"
    for fld in fldLst do
        flds <- flds + ", " + fld
    let tbl = if cond then "table1" else "table2"
    execute ("SELECT" + flds + "FROM" + tbl)
```

- JavaScript в Java

```
String script =
    "function hello(name)  print('Hello, ' + name); ";
engine.eval(script);
Invocable inv = (Invocable) engine;
inv.invokeFunction("hello", "Scripting!!!" );
```

Динамически формируемые строковые выражения: проблемы

- Значение выражения — код на некотором языке и его нужно соответствующим образом поддерживать и обрабатывать
- Однако для стандартных инструментов это просто строки
  - Ошибки в динамически формируемых выражениях обнаруживаются лишь во время выполнения
  - Нет поддержки в средах разработки
  - Затруднён реинжиниринг ПО, разработанного с использованием встроенных языков
- Существует множество различных языков и задач

Обзор

- Построение аппроксимации $L_R \rightarrow$ проверка включения $L_R \subseteq L(G)$, G – КС грамматика референсного языка
  - JSA, PHPSA
- Alvor – построение регулярной аппроксимации $\rightarrow$ лексический анализ $\rightarrow$ синтаксический анализ
- Kyung-Goo Doh – data-flow уравнения + LALR(1) анализ + абстрактная интерпретация стеков + атрибутные грамматики
- Для работы с сематникой нужно иметь лес вывода – множество деревьев для всех $\omega \in L_R$ выводимых в G

Цели и задачи работы

- Создать подход к статическому синтаксическому анализу динамически формируемых выражений, позволяющего обрабатывать произвольную регулярную аппроксимацию без потери точности.
  - Обработка регулярной аппроксимации
  - Построение леса вывода
- Разработать технологию для создания инструментов статического анализа встроенных текстовых языков, реализующей данный алгоритм.

Положения, выносимые на защиту

- Разработан алгоритм синтаксического анализа динамически формируемых выражений, позволяющий обрабатывать произвольную регулярную аппроксимацию множества значений выражения в точке выполнения, реализующий эффективное управление стеком и гарантирующий конечность представления леса вывода. Доказана завершаемость и корректность предложенного алгоритма.
- Создана архитектура инструментария для разработки программных средств синтаксического анализа динамически формируемых строковых выражений.
- Разработана методика анализа динамически формируемых строковых выражений в проектах по реинжинирингу информационных систем.

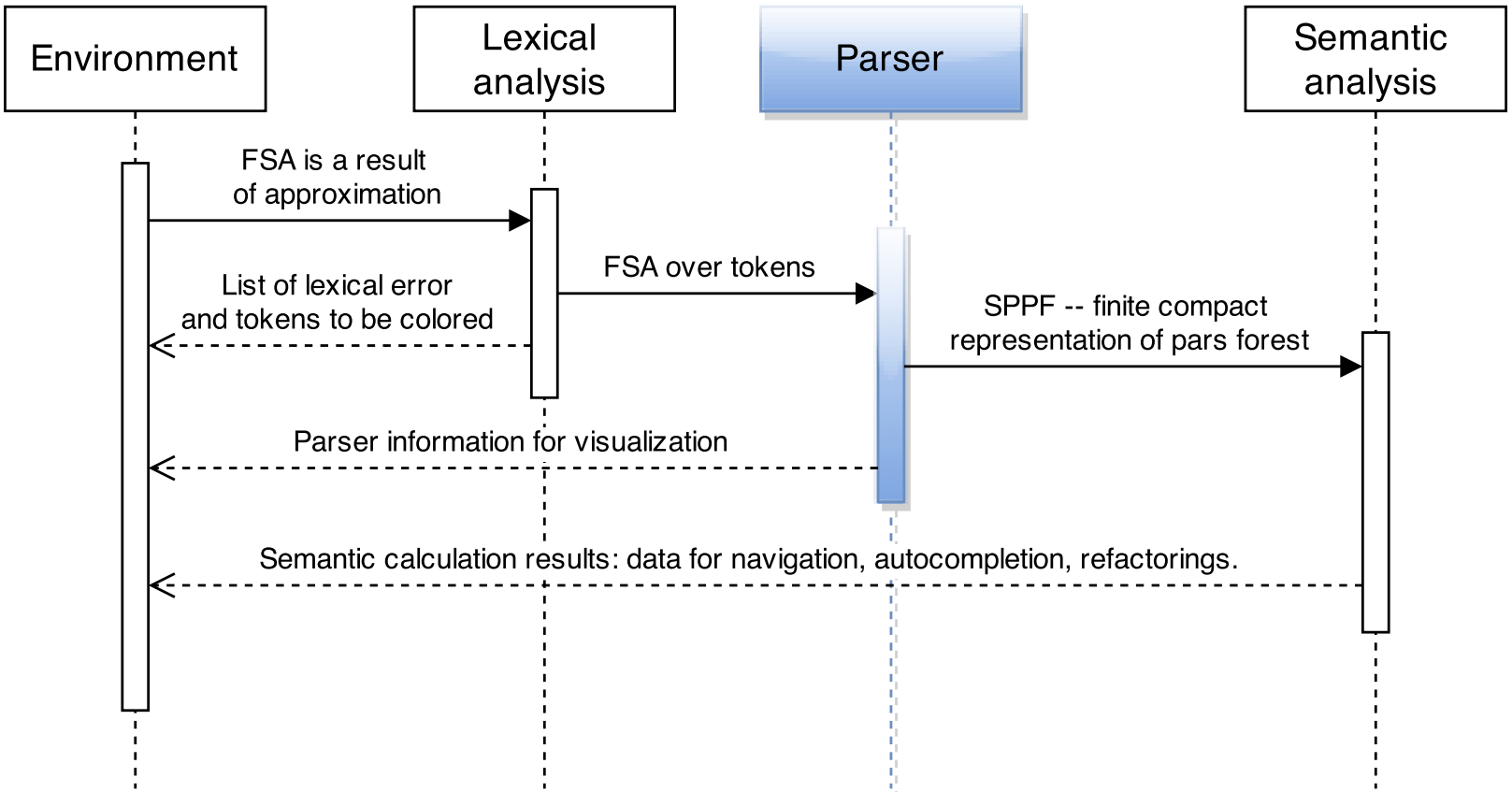
Научная новизна

- Предложенный алгоритм предназначен для синтаксического анализа динамически формируемых выражений и построения компактной структуры данных, содержащей для всех корректных значений выражения их деревья вывода.
- Предложена архитектура инструментального средства, позволяющего упростить создание новых инструментов для анализа динамически формируемых выражений на любых языках программирования.

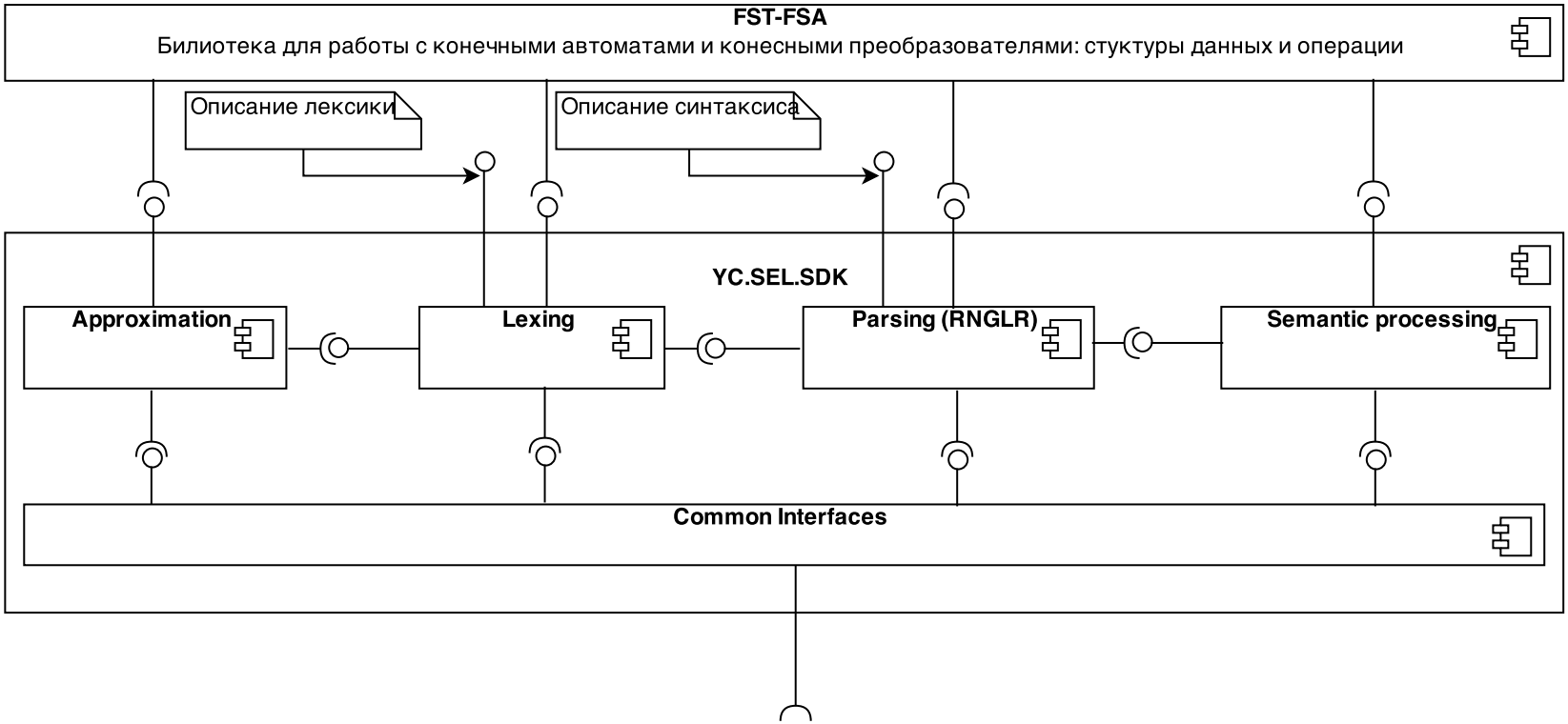
Архитектура: цели и задачи

- Упростить создание инструментов анализа динамически формируемых выражений
- Необходимв: набор готовых компонентов, генераторы лексических и синтаксических анализаторов

Архитектура: процесс анализа



Архитектура: инструментарий



Аппроксимация и лексический анализ

Синтаксический анализ: постановка задачи

- $G = \langle N, \Sigma, P, S \rangle$ – однозначная КС грамматика
- R – регулярное множество над алфавитом $\Sigma' \subseteq \Sigma$
- $AST(t, \omega, G)$ – является ли t деревом вывода ω в грамматике G

Необходимо построить алгоритм $\mathbb{P}$ такой что
$(\forall \omega \in R)(\omega \in L(G) \Rightarrow (\exists t \in \mathbb{P}(R, G))AST(t, \omega, G))$
$\wedge (\forall t \in \mathbb{P}(R, G))(\exists s \in R)AST(t, \omega, G)$

Синтакический анализ: идеи алгоритма

- Основан на RNGLR алгоритме
- Замена линейного входного потока на граф конечного автомата
- Обход графи и последовательное построение GSS по аналогии с RNGLR
- Построение SPPF как в RNGLR
- Использование очереди $\mathcal{Q}$ для задания последовательности обхода вершин

Синтакический анализ: идеи алгоритма

The general idea of the algorithm is to traverse the automaton graph and sequentially construct GSS, similarly as in RNGLR. However, as we deal with a graph instead of a linear stream, the next symbol turns into the *set of terminals* on all outgoing edges of current vertex. This results in a different semantics of pushing and reducing (see line 5, Algorithm **??**, and lines 9 and 21, Algorithm **??**). We use queue $\mathcal{Q}$ to control the order of automaton graph vertices processing. Every time a new GSS vertex is added, all zero-reductions have to be performed and then new tokens have to be shifted, so a corresponding graph vertex has to be enqueueed for further processing. Addition of new GSS edge can produce reductions to handle, so the graph vertex at the tail of the added edge has also to be enqueueed (see Algorithm **??**). Reductions are applied along the paths in GSS, and if we add a new edge to some tail vertex, which was already presented in GSS, we also have to recalculate all *passing* reductions (see *applyPassingReductions* function in Algorithm **??**).

Синтакический анализ: идеи алгоритма

Besides parser *state* and *level* (which is equal to the input automaton state), a collection of *passing reductions* is stored in a GSS vertex. Passing reduction is a triplet $(startV, N, l)$, representing reductions, whose path contains given GSS vertex. This triplet is similar to one describing reduction, where l is a remaining length of the path. Passing reductions are stored for every vertex of the path (except for the first and the last) during path search in *makeReductions* function (see Algorithm **??**).

Синтакический анализ: идеи алгоритма

- Вершины GSS связываются с позицией во входном потоке, аналогично RNGLR
  - Позиция — состояние входного конечного автомата
- Строится вспомогательная структура данных (*внутренний граф*) — копия входного автомата с вершинами, хранящими дополнительную информацию
  - *processed*: вершины GSS, для который все свёртки выполнены
  - *unprocessed*: вершины GSS, для которых требуется выполнение переноса токенов
  - *reductions*: очередь свёрток, которые необхоимо выполнить
  - *passingReductionsToHandle*: набор пар из вершины и ребра GSS для выполнения проходящих свёрток

Алгоритм: завершаемость

Theorem

Алгоритм завершается для любой однозначной KC грамматики G и любого KA без $\varepsilon-$переходов.

Доказательство.

Каждая вершина внутреннего графа содержит не более чем N вершин GSS, где N — количество состояний LR-анализатора. Таким образом, всего может быть создано не более $N \times n$ вершин GSS, где n — количество вершин во внутреннем графе. Так как GSS не содержит кратных дуг, то количество дуг в GSS не более, чем $O((N \times n)^2)$. Алгоритм извлекает одну вершину из $\mathcal{Q}$ на каждой итерации. При этом вершины добавляются в $\mathcal{Q}$ только если добавляется новое ребро в GSS. Так как количество рёбер в GSS конечно, то алгоритм завершается. □

Алгоритм: корректность. Определения

Definition

Корректное дерево — это упорядоченное дерево со следующими свойствами:

1. корень дерева — стартовый нетерминал грамматики G
2. листья – терминалы G. При этом последовательность листьев соответствует какому-либо пути в KA
3. The interior nodes are nonterminals of G. All children of nonterminal N correspond to the symbols of the right-hand side of some production for N in G.

Алгоритм: корректность

Lemma (Lemma 1)

Для любого ребра GSS (v_t, v_h), $v_t \in V_t.processed$, $v_h \in V_h.processed$, терминалы соответствующего поддерева соответствуют некоторому пути p из V_h в V_t во внутреннем графе.

Доказательство.

Доказательство проводится по индукции по высоте дерева вывода. □

Алгоритм: корректность

Theorem

Любое дерево, извлечённое из SPPF корректно.

Доказательство.

Consider arbitrary tree, generated from SPPF, and prove that it is correct. The first and the third statements of correctness definition immediately follow from SPPF definition.

LEMMA 1 proves the second item of the definition by consideration of all the edges from the GSS vertex on the last level having accepting state to the vertex on the 0-level with the start parser state.

□

Алгоритм: корректность

Theorem

Для строки, сооответствующей любому пути p во внутреннем графе, имеющей вывод в эталонной грамматике G, корректное дерево, соответствующее p может быть извлечено из SPPF.

Доказательство.

Доказательство аналогично доказательству корректности RNGLR, за исключением следующего момента. RNGLR конструирует GSS по слоям, гарантируя, что $\forall j \in [0..i-1]$ j-й уровень будет зафиксирован к тому моменту, когда будет обрабатываться i-й. В нашем случае это свойство не выполняется, что может привести к появлению новых путей для уже применённых свёрток. Единственный способ добавить новый путь — добавить ребро (v_t, v_h), где v_t уже а GSS и содержит входящие рёбра. Так как алгоритм сохраняет свёртки, проходящие черех каждую вершину, то достаточно продолжить проходящие свёртки для v_t. За это отвечает функция *applyPassingReductions* □

Алгоритм: пример работы

Методология

- Берём и генерируем

Апробация: синтетические замеры

Апробация: трансляция динамического SQL

Апробация: плагин к ReSharper

Публикации

- Кириленко Я.А., Григорьев С. В., Авдюхин Д. А. Разработка синтаксических анализаторов в проектах по автоматизированному реинжинирингу информационных систем. Научно-технические ведомости Санкт-Петербургского государственного политехнического университета информатика, телекоммуникации, управление. Т. 3, N 174, 2013. С. 94 — 98.
- Григорьев С. В., Вербицкая Е. А., Полубелова М. И., Иванов А. В., Мавчун Е. В. Инструментальная поддержка встроенных языков в интегрированных средах разработки. Моделирование и анализ информационных систем. Т. 21, N 6, 2014. С. 131—143.
- Григорьев С.В. Алгоритм синтаксического анализа динамически формируемых выражений.

Публикации

- Semen Grigorev, Iakov Kirilenko. GLR-based abstract parsing. In Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR '13). 2013. ACM, New York, NY, USA. 1-9 p.
- Semen Grigorev, Ekaterina Verbitskaia, Andrei Ivanov, Marina Polubelova, Ekaterina Mavchun. String-embedded language support in integrated development environment. In Proceedings of the 10th Central and Eastern European Software Engineering Conference in Russia (CEE-SECR '14). 2014. ACM, New York, NY, USA. 1-11 p.
- Semen Grigorev, Iakov Kirilenko. From Abstract Parsing to Abstract Translation. Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering. 2014. Saint Petersburg, Russia. 1-5 p.

Алгоритм: псевдокод

```
1:  function PARSE(grammar, automaton)
2:      inputGraph ← construct inner graph representation of automaton
3:      parserSource ← generate RNGLR parser tables for grammar
4:      if inputGraph contains no edges then
5:          if parserSource accepts empty input then report success
6:          else report failure
7:      else
8:          ADDVERTEX(inputGraph.startVertex, startState)
9:          Q.Enqueue(inputGraph.startVertex)
10:         while Q is not empty do
11:             v ← Q.Dequeue()
12:             MAKEREDUCTIONS(v)
13:             PUSH(v)
14:             APPLYPASSINGREDUCTIONS(v)
15:         if v_f.level = q_f and v_f.state is accepting then report success
16:         else report failure
```

Алгоритм: псевдокод

```
1:  function MakeReductions(innerGraphV)
2:      while innerGraphV.reductions is not empty do
3:          (startV, N, l) ← innerGraphV.reductions.Dequeue()
4:          find the set of vertices 𝒳 reachable from startV
5:           along the path of length (l − 1), or 0 if l = 0;
6:          add (startV, N, l − i) in v.passingReductions,
7:           where v is an i-th vertex of the path
8:          for all v_h in 𝒳 do
9:              state_t ← calculate new state by v_h.state and nonterminal N
10:             AddEdge(v_h, startV, state_t, (l = 0))
```

Алгоритм: псевдокод

```
1: function PUSH(innerGraphV)
2:     U ← copy innerGraphV.unprocessed
3:     clear innerGraphV.unprocessed
4:     for all v_h in U do
5:         for all e in outgoing edges of innerGraphV do
6:             push ← calculate next state by v_h.state and the token on e
7:             ADDEDGE(v_h, e.Head, push, false)
8:             add v_h in innerGraphV.processed
9: function APPLYPASSINGREDUCTIONS(innerGraphV)
10:    for all (v, edge) in innerGraphV.passingReductionsToHandle do
11:        for all (startV, N, l) ← v.passingReductions.Dequeue() do
12:            find the set of vertices X,
13:             reachable from edge along the path of length (l − 1)
14:            for all v_h in X do
15:                state_t ← calculate new state by v_h.state and
   nonterminal N
16:                ADDEDGE(v_h, startV, state_t, false)
```

Алгоритм: псевдокод

```
1:  function AddVertex(innerGraphV, state)
2:      v ← find a vertex with state = state in
3:       innerGraphV.processed ∪ innerGraphV.unprocessed
4:      if v is not null then                ▷ The vertex have been found in GSS
5:          return (v, false)
6:      else
7:          v ← create new vertex for innerGraphV with state state
8:          add v in innerGraphV.unprocessed
9:          for all e in outgoing edges of innerGraphV do
10:             calculate the set of zero-reductions by v
11:              and the token on e and add them in
    innerGraphV.reductions
12:         return (v, true)
```

Алгоритм: псевдокод

```
1:  function AddEdge(v_h, innerGraphV, state_t, isZeroReduction)
2:      (v_t, isNew) ← AddVertex(innerGraphV, state_t)
3:      if GSS does not contain edge from v_t to v_h then
4:          edge ← create new edge from v_t to v_h
5:          Q.Enqueue(innerGraphV)
6:          if not isNew and v_t.passingReductions.Count > 0 then
7:              add (v_t, edge) in innerGraphV.passingReductionsToHandle
8:          if not isZeroReduction then
9:              for all e in outgoing edges of innerGraphV do
10:                 calculate the set of reductions by v
11:                  and the token on e and add them in
    innerGraphV.reductions
```