Санкт-Петербургский государственный университет

Кафедра системного программирования

Синтаксический анализ динамически формируемых программ

Григорьев Семён Вячеславович

Научный руководитель: к.ф.-м.н., доцент Д.В. Кознов

Официальные оппоненты: д.т.н., профессор А.Р. Лисс

к.т.н., доцент В.М. Ицыксон

Санкт-Петербург

2015

Предметная область

Статический анализ — получение знаний о коде без его запуска

Необходим для увеличения надёжности, улучшение качества, упрощение разработки и сопровождения кода

- Статический поиск ошибок
- Поддержка в средах разработки
- Реинжиниринг ПО

Динамически формируемые программы

- 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!!!" );
```

Мотивация

- Динамически формируемые программы необходимо учитывать при обработке информационных систем
  - Рефакторинг
  - Реинжиниринг
  - Статический анализ
- Стандартные инструменты данные задачи не решают
- Для эффективного решения этих задач необходимо структурное представление динамически формируемых программ

Существующие работы

- **Doh Kyung-Goo, Kim Hyunha, et al** (Hanyang University, South Korea)
  - Abstract Parsing: Static Analysis of Dynamically Generated String Output Using LR-Parsing Technology. 2009
  - Abstract LR-parsing, 2011
  - Static Validation of Dynamically Generated HTML Documents Based on Abstract Parsing and Semantic Processing. 2013
- **Aivar Annamaa, Andrey Breslav, et al** (University of Tartu, Estonia; State University ITMO, Russia)
  - An Interactive Tool for Analyzing Embedded SQL Queries. 2010
  - Using Abstract Lexical Analysis and Parsing to Detect Errors in String-Embedded DSL Statements. 2010
- **Christensen Aske Simon, Moller Anders, Schwartzbach Michael I.** Precise Analysis of String Expressions. 2003, University of Aarhus, Denmark
- **Minamide Yasuhiko.** Static Approximation of Dynamically Generated Web Pages. 2005, University of Tsukuba, Tsukuba, Japan

Существующие работы

- **Nguyen Hung Viet, Kastner Christian, Nguyen Tien N.** Varis: IDE Support for Embedded Client Code in PHP Web Applications. 2015, Iowa State University
- **Smith Zachary.** Development of Tools to Manage Embedded SQL. 2011, University of Alabama
- **Huib van den Brink, Rob van der Leek, Visser Joost.** Quality Assessment for Embedded SQL. 2007, Utrecht University, The Niderlands
- **Huib van den Brink.** A Framework to Distil SQL Queries Out of Host Languages in Order to Apply Quality Metrics. 2007, Utrecht University, The Niderlands

Существующие инструменты

- Проверка включения языков
  - Java String Analyzer — регулярная аппроксимация строкового выражения
  - PHP String Analyzer — контекстно-свободная аппроксимация строкового выражения
- Поддержка встроенных языков в IDE
  - Varis — плагин к Eclipse IDE для поддержки JS и HTML в PHP: подсветка синтаксиса, навигация
  - IntelliLang — поддержка встроенных языков в IntelliJ IDEA
  - PHPStorm — IDE для PHP с поддержкой встроенных языков
  - Alvor — плагин к Eclipse IDE для проверки встроенного в Java SQL

Регулярная аппроксимация

- Существует множество способов построения
  - JSA, Alvor, Stranger и т.д.
- **Fang Yu, Muath Alkhalaf, Tevfik Bultan, Oscar H. Ibarra.** Automata-based Symbolic String Analysis for Vulnerability Detection, 2014
  - Обработка строковых функций (replace и т.д.)
  - Приближение сверху

Выводы

- Теоретические возможности синтаксического анализа встроенных языков хорошо изучены, однако
  - В существующих работах не предъявлен целостный алгоритм синтаксического разбора динамически формируемых программ
  - Существующие инструменты не предоставляют платформу для создания инструментов статического анализа динамически формируемых программ
  - В существующих работах не уделяется должного внимания задаче реинжиниринга встроенного программного кода

Разработка инструментов

Возможны два подхода

- Создание универсального инструмента
- Создание набора генераторов и библиотек стандартных функций
  - По описанию языка генерируются анализаторы
  - Для создания конечных решений можно использовать стандартные функции

Цели работы

- Целью данной работы является создание метода статического анализа динамически формируемых программ, который уменьшил бы затраты по созданию целевых инструментов, обеспечивающих
  - Поддержку в средах разработки: подсветку синтаксиса, статический поиск ошибок
  - Реинжиниринг информационных систем, содержащего большое количество динамически формируемых строковых выражений

Положения, выносимые на защиту

1. Разработан алгоритм синтаксического разбора динамически формируемых программ, гарантирующий конечность представления леса вывода. Доказана завершаемость и корректность предложенного алгоритма
2. Предложена архитектура инструментария для разработки программных средств статического анализа динамически формируемых строковых выражений
3. Разработан метод анализа встроенного программного кода в проектах по реинжинирингу информационных систем

Методы исследования

- Алгоритм обобщённого восходящего синтаксического анализа RNGLR (Elizabeth Scott, Adrian Johnstone, Royal Holloway University of London)
- Компактное хранение леса вывода SPPF (Jan Rekers, University of Amsterdam)
- Приближение множества значений динамически формируемого выражения регулярным множеством, описываемым с помощью конечного автомата
- Теория формальных языков, теория графов и теория сложности алгоритмов для доказательства завершаемости и корректности предложенного алгоритма

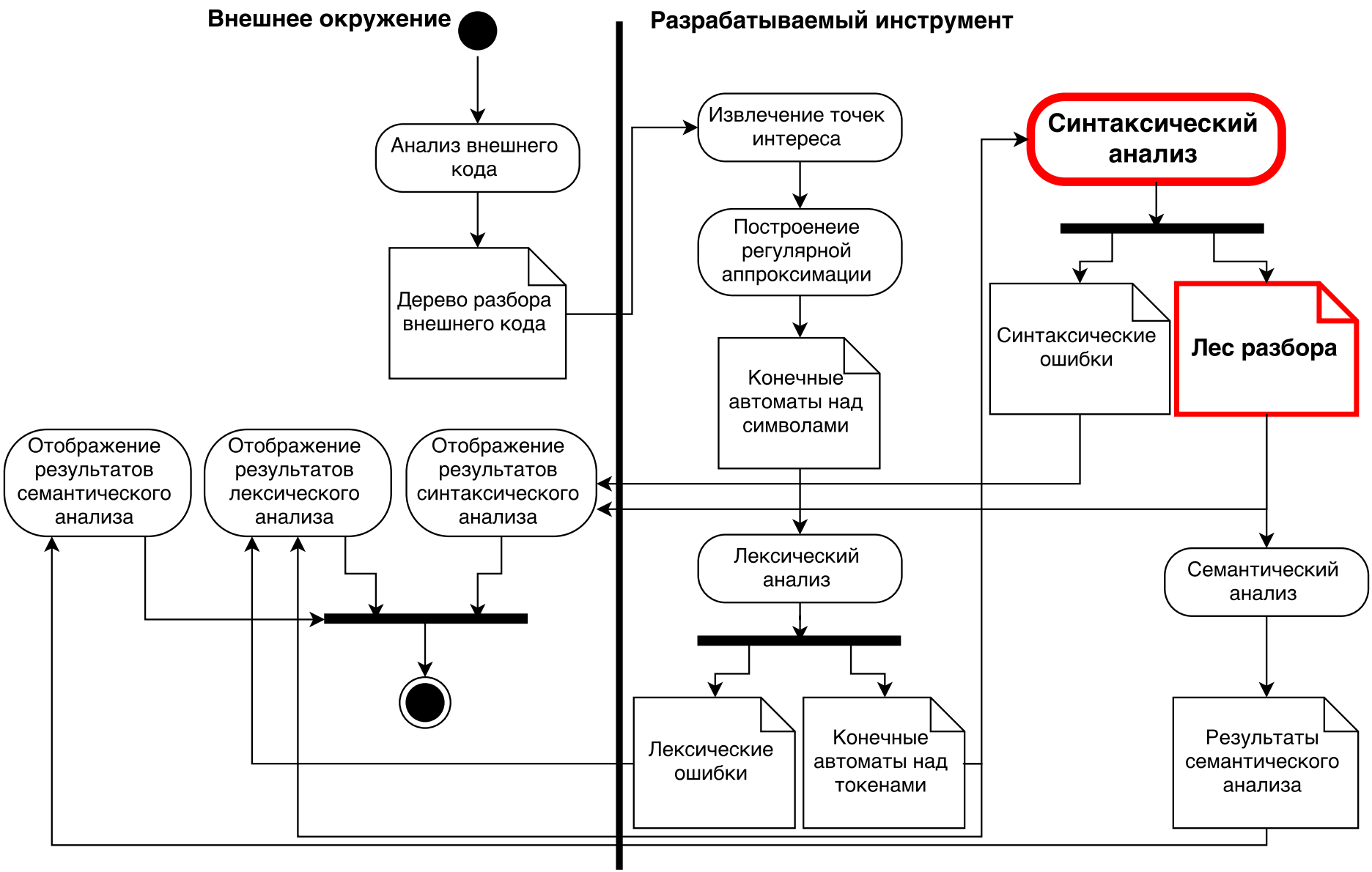
Промышленная реализация созданного подхода

- Промышленный проект компании ЗАО “Ланит-Терком” (Россия) по переносу хранимого SQL-кода с MS SQL Server на Oracle Server
- Расширение для поддержки встроенного T-SQL-кода в Microsoft Visual Studio IDE на основе инфраструктуры проекта ReSharper компании ООО “ИнтеллиДжей Лабс” (Россия)

Положения, выносимые на защиту

1. Разработан алгоритм синтаксического разбора динамически формируемых программ, гарантирующий конечность представления леса вывода. Доказана завершаемость и корректность предложенного алгоритма
2. Предложена архитектура инструментария для разработки программных средств статического анализа динамически формируемых строковых выражений
3. Разработан метод анализа встроенного программного кода в проектах по реинжинирингу информационных систем

Контекст: процесс анализа строковых выражений



Синтаксический анализ: постановка задачи

- $G = \langle N, \Sigma, P, S \rangle$ — однозначная КС грамматика
- R — регулярный язык над алфавитом $\Sigma' \subseteq \Sigma$
- $AST(t, \omega, G)$ — истинен, если t является деревом вывода ω в грамматике G

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

Синтакический анализ: описание алгоритма

- Основан на алгоритме RNGLR
  - Обрабатывает произвольные КС грамматики
  - Основные операции: ***push*** — композиция переноса и goto; ***reduce*** — свёртка
  - Использует компактное представление множества стеков (**GSS**) и компактное представление леса разбора (**SPPF**)
- Замена линейного входного потока на граф конечного автомата
- Обход графа и последовательное построение GSS по аналогии с RNGLR
  - Для каждой вершины входного графа вычисляется множество возможных вершин GSS — состояний LR-анализатора
- Построение SPPF как в RNGLR
- Использование очереди $\mathcal{Q}$ для задания последовательности обхода вершин входного графа
  - Вершина добавляется в $\mathcal{Q}$, если добавляется новое ребро в GSS с концом в этой вершине

Корректность алгоритма

Определение 1

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. *внутренние узлы — нетерминалы G. Все дети нетерминала N соответствуют правой части какой-то продукции для N в G.*

Лемма 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.

Корректность алгоритма

Теорема 1 (Завершаемость)

Алгоритм $\mathbb{P}(R, G)$ завершается для любой однозначной КС грамматики G и любого ДКА R.

Теорема 2 (Корректность)

Любое дерево, извлечённое из SPPF, корректно.

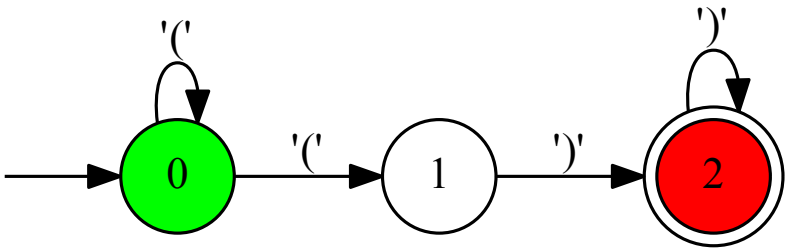
Теорема 3 (Корректность)

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

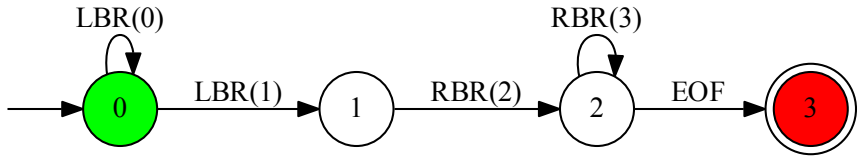
Пример работы: аппроксимация и лексический анализ

```
string expr = "()"
while (cond) do
    expr := "(" + expr + ")"
evaluate(expr)
```

Аппроксимация:



Результат лексического анализа:

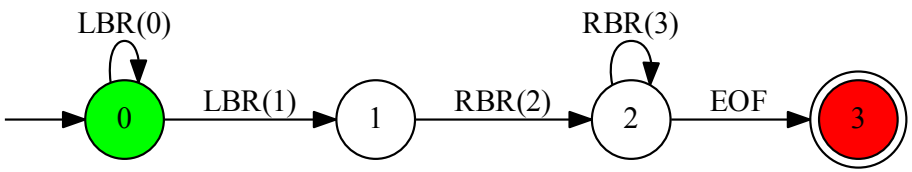


Пример работы: синтаксический анализ

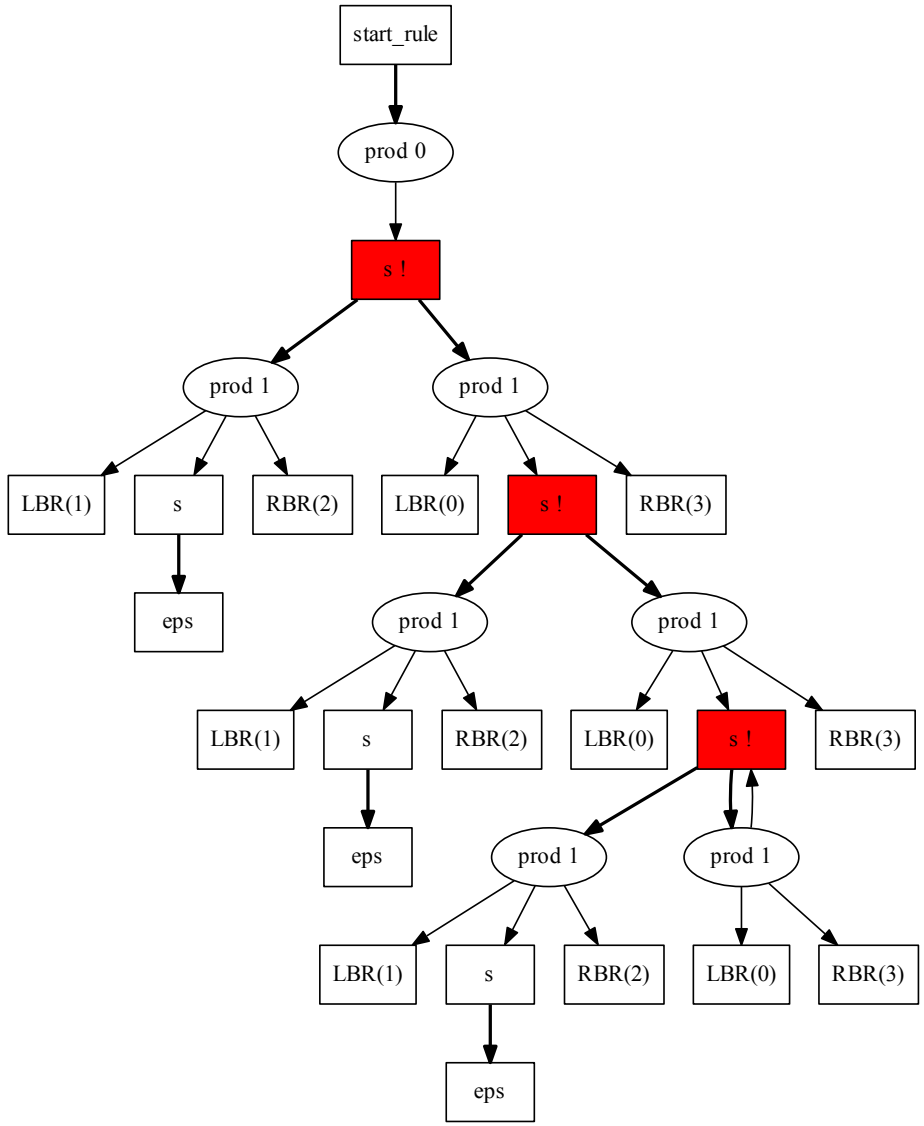
Грамматика:

(0) $start\_rule ::= s$
(1) $s ::= \texttt{LBR}\ s\ \texttt{RBR}\ s$
(2) $s ::= \varepsilon$

Вход:



Результат (SPPF):

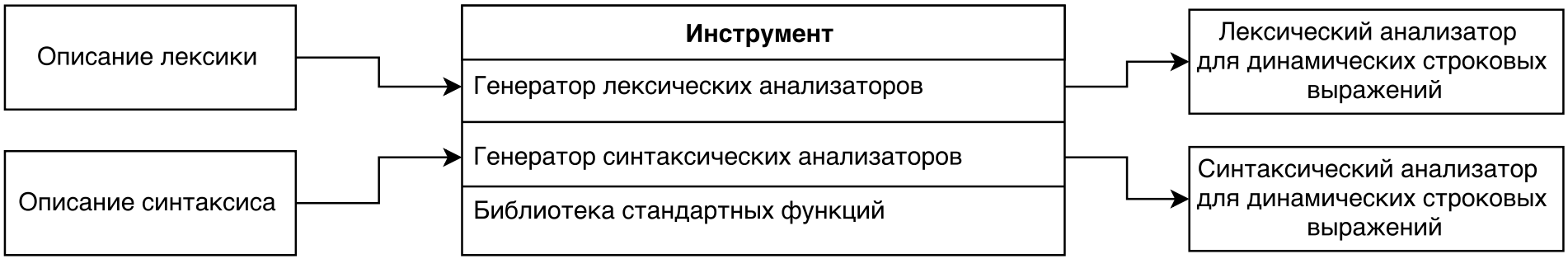


Положения, выносимые на защиту

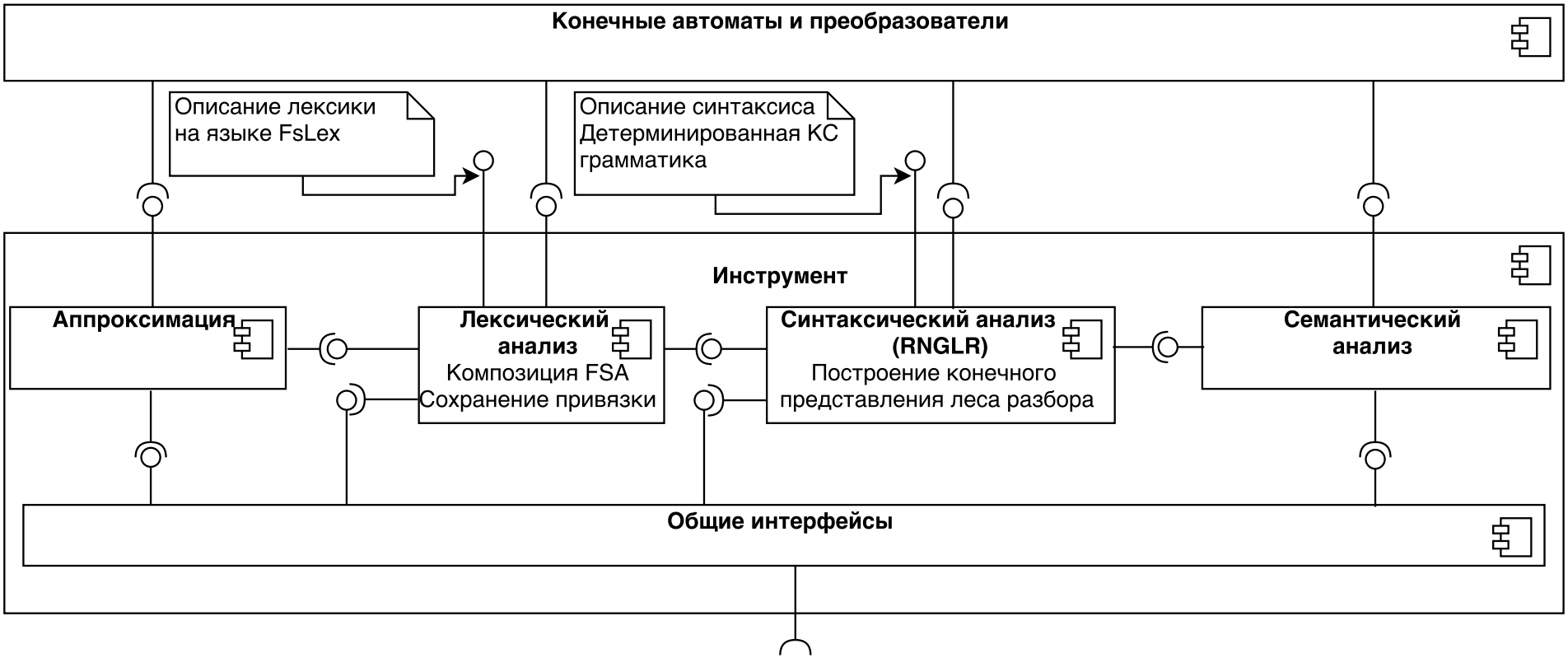
1. Разработан алгоритм синтаксического разбора динамически формируемых программ, гарантирующий конечность представления леса вывода. Доказана завершаемость и корректность предложенного алгоритма
2. Предложена архитектура инструментария для разработки программных средств статического анализа динамически формируемых строковых выражений
3. Разработан метод анализа встроенного программного кода в проектах по реинжинирингу информационных систем

Архитектура: требования

- Упростить создание инструментов для анализа динамически формируемых выражений
- Предоставить набор готовых компонентов, генератор лексических и синтаксических анализаторов



Архитектура



Положения, выносимые на защиту

1. Разработан алгоритм синтаксического разбора динамически формируемых программ, гарантирующий конечность представления леса вывода. Доказана завершаемость и корректность предложенного алгоритма
2. Предложена архитектура инструментария для разработки программных средств статического анализа динамически формируемых строковых выражений
3. Разработан метод анализа встроенного программного кода в проектах по реинжинирингу информационных систем

Метод реинжиниринга встроенного программного кода

- Вход — информационная система и список целей и задач реинжиниринга
- Выход — инструменты, необходимые для решения поставленных задач или рекомендации по созданию нового
- Набор шагов и соответствующих инструментов, необходимый для анализ системы и выбора средств решения поставленных задач
- Учитывает
  - Объём динамически формируемого кода
  - Его сложность
  - Особенности поставленных целей и задач

Ограничения

- Регулярность аппроксимации
  - Встроенный язык может быть контекстно-свободным
  - Неполнота информации: пользовательский ввод, взаимодействие с подсистемами и т.д.
  - Потеря точности
- Детерминированность KC-грамматики для “эталонного” языка
- Недостаточная производительность реализации
  - Может быть недостаточна для использования в IDE
  - Конфликт точности и производительности
  - Модульная архитектура — можно найти баланс заменяя модули
  - Возможна оптимизация существующего решения

Апробация

- CEE-SECR (2012, 2013, 2014)
  - Премия Бертрана Мейера за лучшую исследовательскую работу в области программной инженерии (2014)
- PSI-2015
- Parsing@SLE-2013
- Семинар по наукоёмкому программному обеспечению при PSI-2014
- Научный семинар по программной инженерии (2013, 2015, СПбПГУ)
- SYRCoSE-2014

Реализация и внедрение: трансляция динамического SQL

- Промышленный проект ЗАО “Ланит-Терком” по миграции ИС с MS SQL Server на Oracle Server
- 2,7 миллиона строк кода, 2430 точек интереса
  - 75% запросов могли принимать более одного значения
  - До 212 операторов для формирования выражения, среднее: 40
- 2188 разобрано
- 1 не удалось разобрать из-за таймаута
- 241 не разобрано из-за неточностей в используемом окружении

Реализация и внедрение: плагин к ReSharper

- Поддержка встроенных языков в Microsoft Visual Studio IDE
  - Подсветка синтаксиса
  - Подсветка парных скобок
  - Подсветка ошибок

```
19      public static void Go(bool cond)
20      {
21          var query = "varX = 1;";
22          if (cond)
23              query += "varY = 2;";
24          query += "varZ = varX + varY;";
25          Program.ExtEval(query);
26      }
27
```

Публикации (ВАК)

- 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., Рагозина А.К. Обобщённый табличный LL-анализ. Системы и средства информатики. Т. 25, N 1, 2015. С. 89—107.

Другие 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
```