САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Александрия Георгий Гуливерович

Сравнение инструментов анализа динамически формируемых строковых выражений

Курсовая работа

Научный руководитель:
ст. преп. С. В. Григорьев

Санкт-Петербург
2015

SAINT-PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Chair of Software Engineering

Alexandria George

Comparison analysis tools of dynamically generated string expressions

Course Work

Scientific supervisor:
senior lecture S. V. Grigorev

Saint-Petersburg
2015

Оглавление

Введение

Программное обеспечение может содержать строковые выражения в виде исходного кода на другом, встроенном, языке. Такие выражения формируются динамически из строковых литералов во время выполнения программы и передаются во внешнюю среду на исполнение. Соответственно, выражения встроенного языка — это тоже код на некотором языке программирования, и с ними возникают те же проблемы, что и при разработке на обычных языках.

Одна из главных проблем — отсутствие возможности статической проверки динамически формируемых строковых выражений, и, как следствие, любая незначительная ошибка в них обнаруживается лишь на этапе исполнения. Это значительно увеличивает затраты на разработку, отладку и сопровождение продуктов и приложений, в которых используются встроенные языки.

Другой проблемой является то, что при разработке приложений, содержащих выражения на встроенном языке, нет возможности использовать функции для работы с этими выражениями, которые могут упростить процесс разработки и облегчить поддержку кода. Например простые функции интегрированных сред разработки: подсветка синтаксиса, автодополнение, рефакторинг, некоторые подсказки.

Анализ динамически формируемых выражений также необходим и при реинжиниринге программного обеспечения. Например, при наличии динамически формируемых строковых SQL запросов, не проанализировав строковые выражения, нельзя точно сказать, с какими объектами в базе данных взаимодействует система. А значит, нет возможности гарантировать безопасность изменений в структуре базы данных при реинжиниринге.

Кроме того, встроенные языки, зачастую, являются серьёзным источником уязвимостей информационных систем. Например, атаки основанные на SQL-инъекциях возможны благодаря использованию в приложениях встроенного SQL. Поэтому возникает необходимость в разработке инструментов, которые умеет анализировать конструкции встроенных языков, а также которые могут быть использованы в качестве инструмента для проведения реинжиниринга программного обеспечения.

На сегодняшний день существует ряд инструментов для анализа строковых выражений. Несмотря на то, что они решают схожие задачи, все они имеют свои достоинства и недостатки, отличающие их друг от друга. Таким образом, целью работы является сравнение инструментов анализа динамически формируемых строковых выражений и выявление направления дальнейшего пути развития разработки инструментов анализа.

1. Постановка задачи

Цель работы – сравнить инструменты анализа динамически формируемых строковых выражений и выявить направления дальнейшего пути развития разработки инструментов анализа. В рамках данной работы были поставлены следующие задачи:

- Выбрать сравниваемые инструменты
- Определить критерии сравнения
- Составить краткое описание сравниваемого инструмента
  - Основные функции
  - Поддерживаемые языки
- Провести сравнение инструментов
- Сделать выводы из результатов

2. Обзор

2.1. Инструменты обнаружения SQL-инъекций

Обнаружение SQL-инъекций – актуальная проблема, для решения которых существует ряд инструментов. Краткое описание некоторых из них представлено ниже.

- **SAFELI [13]** — инструмент статического анализа, предназначенный для определения возможности SQL-инъекций в Web-приложениях на этапе компиляции приложений. SAFELI работает с MSIL байткодом ASP.NET приложений.

- **A Static Analysis Framework for Database Applications [3]**

  Данный инструмент производит анализ скомпилированного кода, использующего ADO.NET для доступа к данным. Может быть использован для определения возможных мест SQL-инъекций, для определения "узких мест" по производительности запросов.

2.2. Инструменты анализа динамически формируемых выражений

Ниже представлены основные особенности некоторых инструментов для работы с динамически формируемыми строковыми выражениями.

- **Alvor [9]** – это плагин к среде разработки Eclipse, предназначенный для статической валидации SQL-выражений встроенных в код на языке Java. Он может использоваться как для разового запуска на всем исходном коде, так и для запуска в режиме реального времени, который работает в процессе написания кода. Найденные в коде SQL-запросы проверяются на основе SQL-грамматики. Так же они могут быть проверены исполнением в указанной базе данных.

- **Java String Analyzer (JSA) [2][5]** – это инструмент для анализа динамически формируемых строк и строковых операций в программах на языке Java. Для каждого строкового выражения он строит конечный автомат, представляющий приближенное значение всех значений данного выражения, которые могут быть получены во время выполнения программы.

- **PHP String Analyzer (PHPSA) [10]** – это статический анализатор для строковых выражений, порождаемых программами на PHP. Он аппроксимирует значения выражений некоторой контекстно-свободной грамматикой. Инструмент может использоваться, например, для проверки корректности динамически генерируемых программой на PHP Web страниц.

- **IntelliLang [7]** – плагин к средам разработки PHPStorm и IntelliJ IDEA, позволяющий подсвечивать и указывать об ошибках во встроенных языках таких, как HTML, SQL, XML, JavaScript в указанных средах разработки. Для среды разработки IDEA плагин IntelliLang также предоставляет отдельный текстовый редактор для работы со встроенным языком.

- **YaccConstructor [14]** – модульный инструмент для проведения лексического анализа и динамического разбора, а также платформа для исследования и разработки генераторов лексических и синтаксических анализаторов. Данный инструмент также является платформой для поддержки встроенных языков.

- **PHPStorm [8]** – интегрированная среда разработки для PHP, которая осуществляет подсветку и автодополнение встроенного кода на HTML, CSS, JavaScript, SQL. Однако такая поддержка осуществляется только в случаях, когда строка получена без использования каких-либо строковых операций. Также PHPStorm для каждого встроенного языка предоставляет отдельный текстовый редактор.

- **Varis [12]** – инструмент, который предоставляет сервисы для работы с языком HTML в PHP программах. Производит аппроксимацию продукций PHP программ, используя сивольный механиз выполнения [11], преобразуя продукцию в VarDOM, что есть компактное представление всех возможных DOM.

- **Automata-based Symbolic String Analysis for Vulnerability Detection [1]** В данной статье описан алгоритм, базирующийся на конечных автоматах, поддерживающий обработку строковых операций, в частности операцию replace. Данный алгоритм реализован в инструменте Stranger [15], который предназначен для валидации строковых операций в PHP программах.

2.3. Выбор сравниваемых инструментов

В качестве сравниваемых инструментов были выбраны следующие: Alvor, JSA, PHPSA, IntelliLang. Данные инструменты были отобраны потому, что они предоставляют стабильную работающую реализацию необходимых функций для анализа динамически формируемых строковых выражений. Остальные же либо не имеют опубликованных реализаций, либо используют нестабильные реализации.

3. Реализация

3.1. Критерии сравнения

Ниже представлены основные критерии, по которым будут сравниваться все выбранные инструменты.

- Обнаружение ошибок – возможность оповестить пользователя при наличии ошибок в анализируемом выражении
- Позиционирование ошибок – способность указать на местоположение обнаруженных ошибок
- Расширяемость языков – наличие способа дополнить множество анализируемых языков инструмента каким-нибудь новым языком
- Подсветка синтаксиса – выделение ключевых и не ключевых слов определенными цветами
- Проверка "на лету" – возможность проверки кода в режиме реального времени, на момент написания кода
- Поддержка ветвлений – поддержка анализа выражений, полученных при помощи условных конструкций
- Поддержка циклов – поддержка анализа выражений, полученных при помощи циклических итераций

Выбор таких критериев был обоснован тем, что более всего требовалось при разработке приложений, содержащих выражения на встроенных языках. Так критерии обнаружение и позиционирование ошибок определяют главную задачу данных инструментов – статическая валидация динамически формируемых выражений, расширяемость – позволяет использовать один и тот же инструмент для различных языков, подсветка синтаксиса и проверка на лету – сокращение времени на разработку и отладку программного обеспечения, а поддержка ветвлений и циклов – новые возможности генераций выражений.

Сравнение инструментов происходит следующим образом:

- Выбор встроенного языка сравнения для каждого инструмента
- Обзор и классификация тестов
- Тестирование интересных особенностей инструмента при их наличии
- Сбор и анализ результатов для конкретного инструмента
- Сравнение всех инструментов по основным критериям

3.2. Сравниваемые инструменты

3.2.1. Alvor

Описание. Alvor [9] – плагин к среде разработки Eclipse, предназначенный для статической валидации SQL-выражений, встроенных в Java программ. Он не требует указания явного атрибута языка, с помощью которого составляются динамически формируемые выражения. Структура инструмента представлена на рисунке 1.

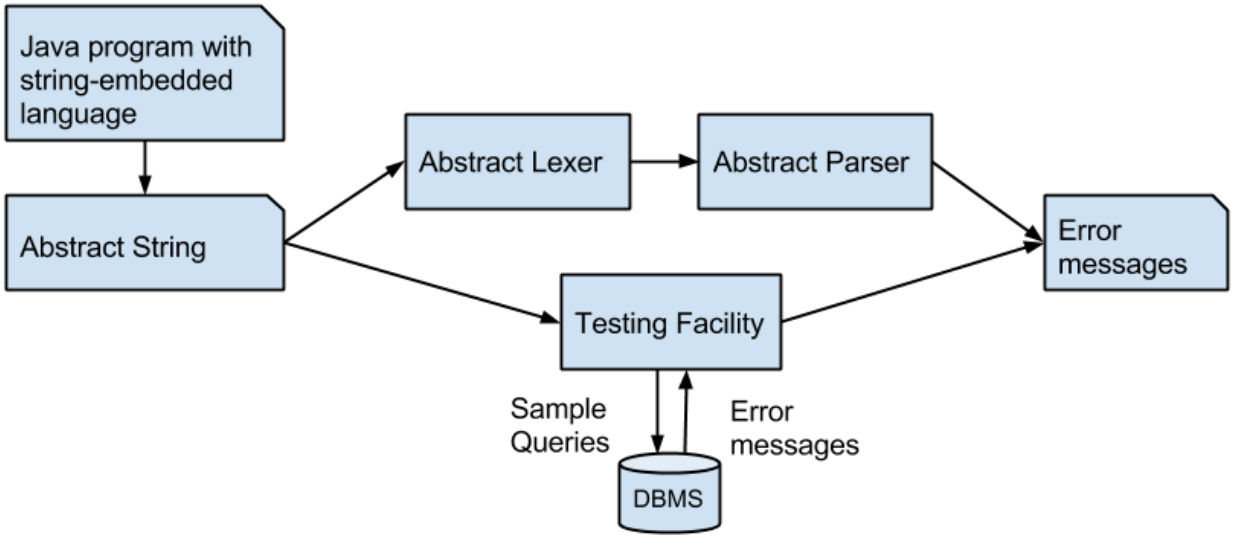


Рис. 1: Структура инструмента Alvor

Для компактного представления множества динамически формируемого строкового выражения используется понятие абстрактной строки, которая фактически представляет собой регулярное выражение над используемыми в строковом выражении символами. Поскольку абстрактную строку можно преобразовать в конечный автомат, то лексический анализ заключается в преобразовании этого конечного автомата в конечный автомат над токенами при использовании конечного преобразователя [6]. После осуществляется абстрактный анализ, базирующийся на GLR-анализе [16], который сообщает об обнаруженных лексических и синтаксических ошибках.

Alvor осуществляет статическую проверку строковых выражений на языке SQL и поддерживает несколько его диалектов: PLSQL, MySQL.

Тесты. Так как Alvor реализует мощный алгоритм анализа, то интерес вызвали производительность анализа данного инструмента и возможность поддержки тех или иных видов SQL-запросов. Ниже представлена выбранная классификация тестов.

- Простые – это строковые выражения, представляющие из себя короткие, не вложенные, SQL-запросы, которые часто используются для получения относительно мощного множества.

```
String sql = "select id, first_name from persons";
```

Listing 1: Пример простого теста

- Вложенные – это строковые выражения, которые содержат несколько уровней вложенности SQL-запроса. Такие запросы часто используются для более точной выборки из баз данных.

```
String example = "SELECT address FROM person WHERE FullName IN (SELECT
    fullName FROM employee WHERE exp>6";
```

Listing 2: Пример вложенного теста

- С ветвлениями – это строковые выражения, содержащие в себе ветвление некоторых частей. Такие запросы возникают при необходимости по какому-то условию создать несколько различных запросов имеющих некую общую часть.

```
int count = 0;
...
String sqlWithCase = "Select ref_num FROM TBC.PARTIES"
switch(count)
{
    case 0:
        sqlWithCase += "WHERE reg_num = 5";
        break;
    case 1:
        sqlWithCase += "WHERE post = \"employee\"";
        break;
    default:
        sqlWithCase += "WHERE social_id = 2";
        break;
}
```

Listing 3: Пример теста с ветвлением

- С циклами – это строковые выражения, имеющие какие-то схожие части, которые можно перечислить при использовании циклов. Такие запросы используются часто, когда используемые выражения имеют общие части, которые могут быть получены при итерации циклов.

```
int count = 1;
String sql = "SELECT";
while(count < 5)
{
    sql = sql + "arg" + count;
    count++;
}
```

```
sql = sql + "FROM person";
```

Listing 4: Пример теста с циклом

Данная классификация была выбрана, учитывая наиболее часто используемые SQL-запросы и наиболее часто возникающие ситуации генерации SQL-запросов при разработке программного обеспечения.

Результаты. В ходе сравнения было выяснено, что несмотря на возможность абстрактной строки конструировать строковые выражения с циклами, плагин сообщает о не поддержке таких языковых конструкций (см. рис. 2). Также Alvor не поддерживает обработку строковых операций, за исключением конкатенации (см. рис. 3). Если же регулярное выражение порождает несколько ошибок, то обнаруживается и позиционируется лишь первая из них (см. рис. 4). Если не учесть позиционирование первой ошибки, то данный инструмент не поддерживает подсветку синтаксиса.

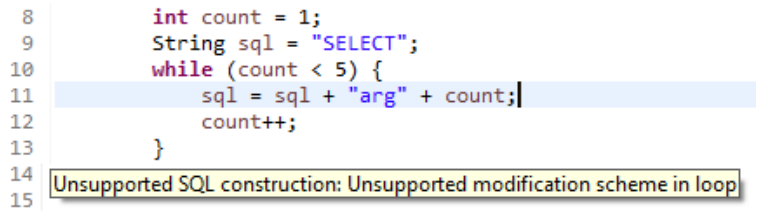


Рис. 2: Формирование строкового выражения с помощью цикла while в среде Eclipse с установленным плагином Alvor

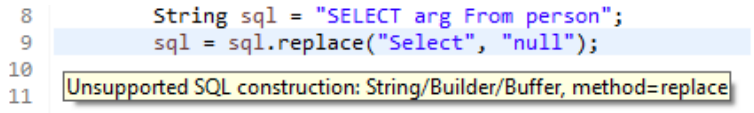


Рис. 3: Формирование строкового выражения с помощью строковой операции replace в среде Eclipse с установленным плагином Alvor

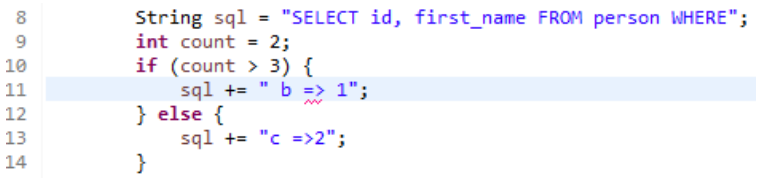


Рис. 4: Сообщение об ошибке в среде Eclipse с установленным плагином Alvor

Добавление новых анализируемых языков – трудозатратный процесс, требующий ручного изменения кода, помимо описания грамматики языка, в частности потребуется реализовать лексические данные, по которым надо сгенерировать абстрактный лексер, на основе которого будет построен конечный преобразователь.

При анализе тестов с глубокой вложенностью запроса или с большой длинной инструмент сообщит о том, что такие тесты он не способен анализировать. Однако при увеличении глубины вложенности и длинны тестов Alvor прекращает работу с генерацией ошибки о переполнении стека.

На графике 5 отображено время, потраченное инструментом на анализ таких типов тестов, как простые и вложенные. На графиках 6, 7, 8, 9, представлены время анализа инструментом тестов с ветвлением. В данных тестах варьирующимися параметрами были количество ветвлений и количество веток в одном ветвлении. Данные результаты были получены на машине с процессором Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, 2501 МГц, ядер: 2, логических процессоров: 4, с установленной оперативной памятью DDR3 8.00 ГБ, под управлением ОС Microsoft Windows 7, SP1 x64.

На основе данных графиков, можно сделать вывод о том, что Alvor анализирует простые и вложенные тесты за короткий промежуток времени, в то время как тесты с ветвлениями проанализированы за время, близкое к экспоненциальному.

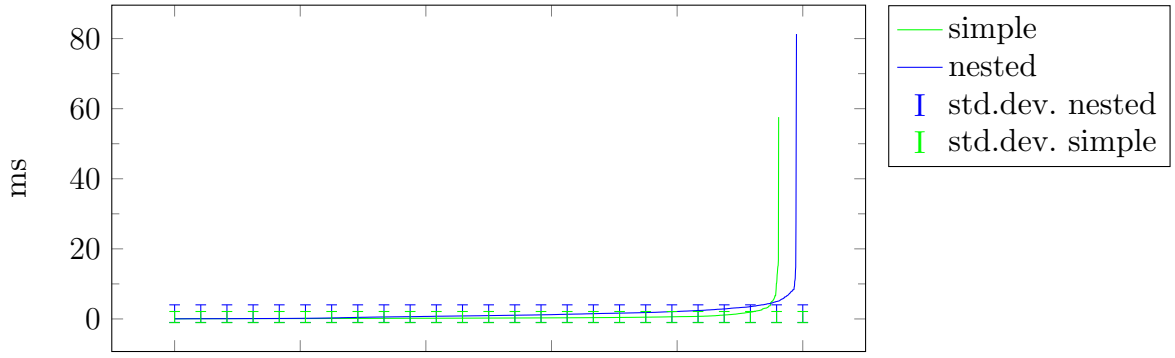


Рис. 5: Время работы Alvor в мс на простых и вложенных тестах

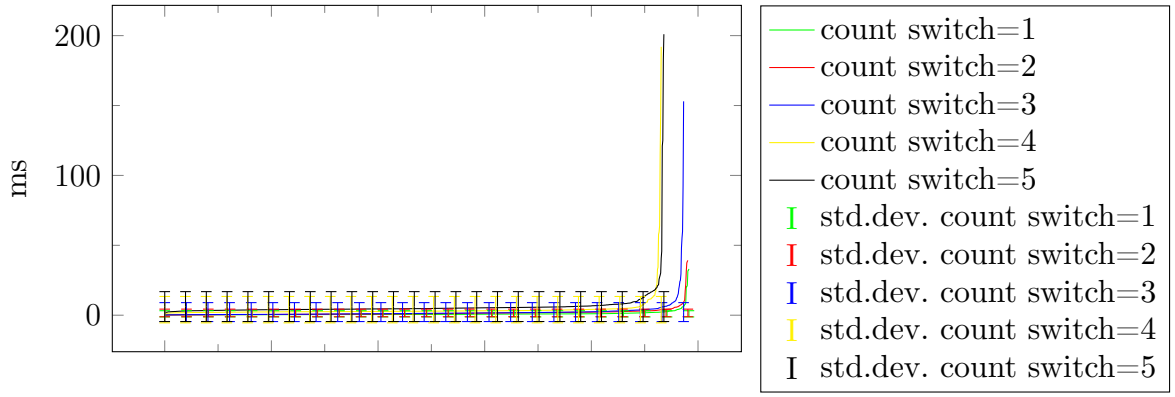


Рис. 6: Время работы Alvor в мс на тестах с ветвлением при двух ветках

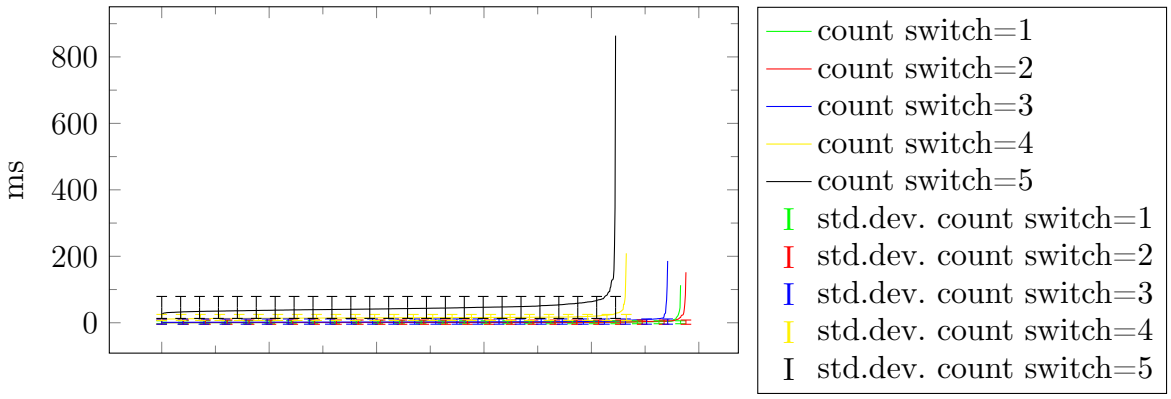


Рис. 7: Время работы Alvor в мс на тестах с ветвлением при трех ветках

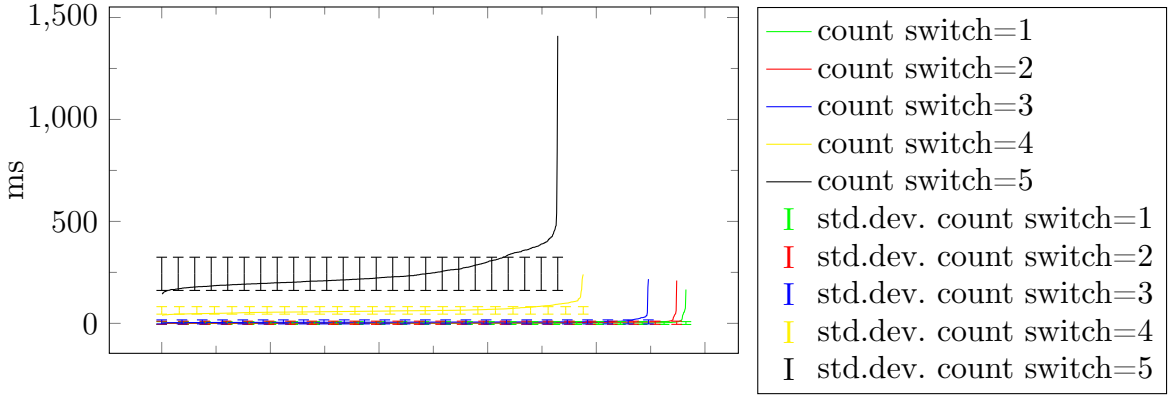


Рис. 8: Время работы Alvor в мс на тестах с ветвлением при четырех ветках

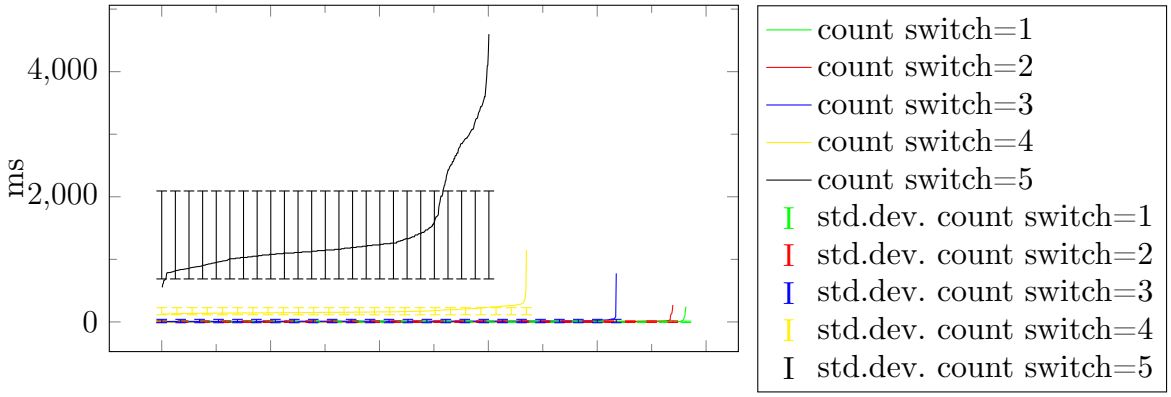


Рис. 9: Время работы Alvor в мс на тестах с ветвлением при пяти ветках

3.2.2. JSA

Описание. JSA [2][5] – инструмент для анализа динамически формируемых строк и строковых операций в программах на языке Java. Структура инструмента представлена на рисунке 10.

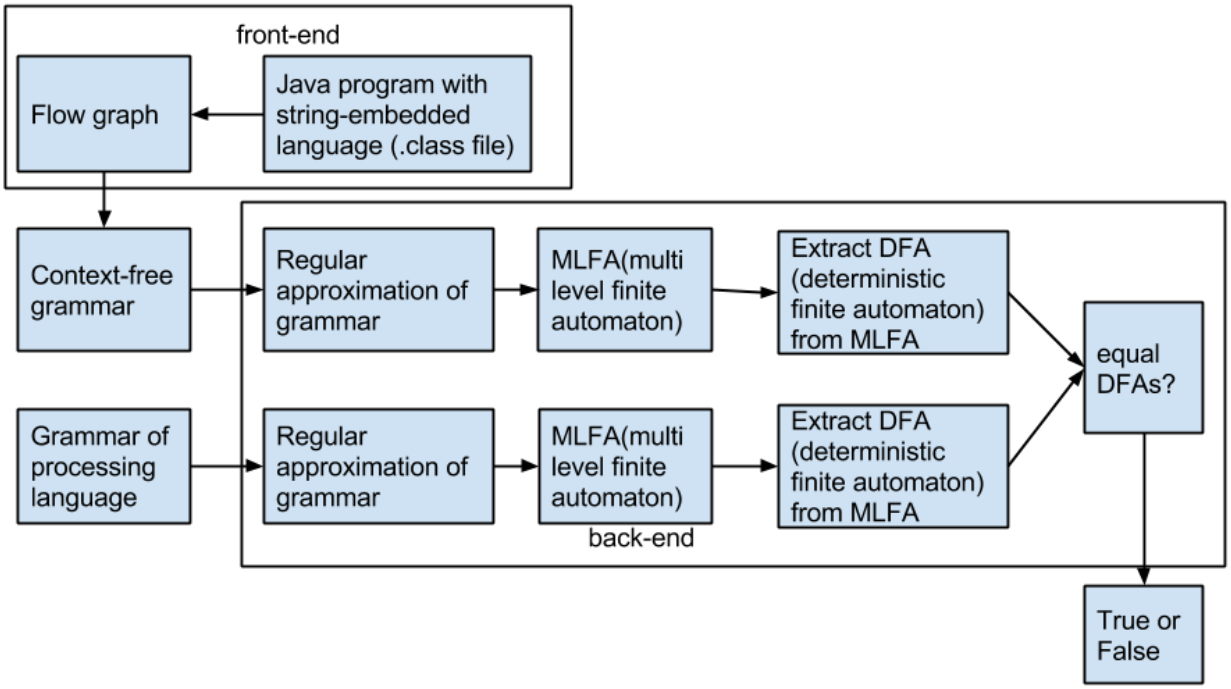


Рис. 10: Структура инструмента JSA

Для каждого строкового выражения строится конечный автомат, представляющий собой аппроксимацию всех значений этого выражения, которые могут быть получены во время выполнения программы. Для того чтобы получить данный конечный автомат, front-end инструмента представляет входные данные в виде flow-graph. Затем back-end инструмента преобразовывает полученный flow-graph в контекстно-свободную грамматику, которая получается в результате замены каждой строковой переменной нетерминалом, а каждой строковой операции правилом продукции. После чего полученная грамматика аппроксимируется регулярной грамматикой и уже по ней строится конечный автомат. По входной грамматике по аналогии с предыдущим строится конечный автомат. Два полученных автомата сравниваются, и если они совпадают, то считается, что динамически формируемое строковое выражение синтаксически корректно (см. листинг 5).

```
static {
    Strings.bind("int", "0|-?[1-9][0-9]*");
}
public String test(int x) {
    String str = "I ate " + x + " apples today";
    Strings.analyze(str, "I ate <int> apples today");
```

```
    return s;
}
```

Listing 5: Пример программы, с синтаксически корректным выражением

Тесты. Так как JSA позволяет работать с разными встроенными языкам, для чего достаточно поменять front-end инструмента, то классификация тестов была взята та же, что и для инструмента Alvor, которая представлена ниже.

- Простые – это строковые выражения, представляющие из себя короткие, не вложенные, SQL-запросы, которые часто используются для получения относительно мощного множества. (см. листинг 1).
- Вложенные – это строковые выражения, которые содержат несколько уровней вложенности SQL-запроса. Такие запросы часто используются для более точной выборки из баз данных (см. листинг 2).
- С ветвлениями – это строковые выражения, содержащие в себе ветвление некоторых частей. Такие запросы возникают при необходимости по какому-то условию создать несколько различных запросов имеющих некую общую часть (см. листинг 3).
- С циклами – это строковые выражения, имеющие какие-то схожие части, которые можно перечислить при использовании циклов. Такие запросы используются часто, когда используемые выражения имеют общие части, которые могут быть получены при итерации циклов (см. листинг 4).

Результаты. К сожалению, JSA не удалось запустить для проведения сравнения. Поэтому результаты, что представлены ниже, основаны на публикациях этого инструмента.

Инструмент умеет анализировать вложенные тесты и тесты с циклом (см листинги 6, 7 соответственно). При помощи фреймворка Soot [4] происходит преобразование исходной программы на языке Java в промежуточное представление Soot – Jimple. После чего JSA преобразовывает Jimple в свое абстрактное промежуточное представление (см рис. 11a, 11b), которое уже будет преобразовано в flow-graph.

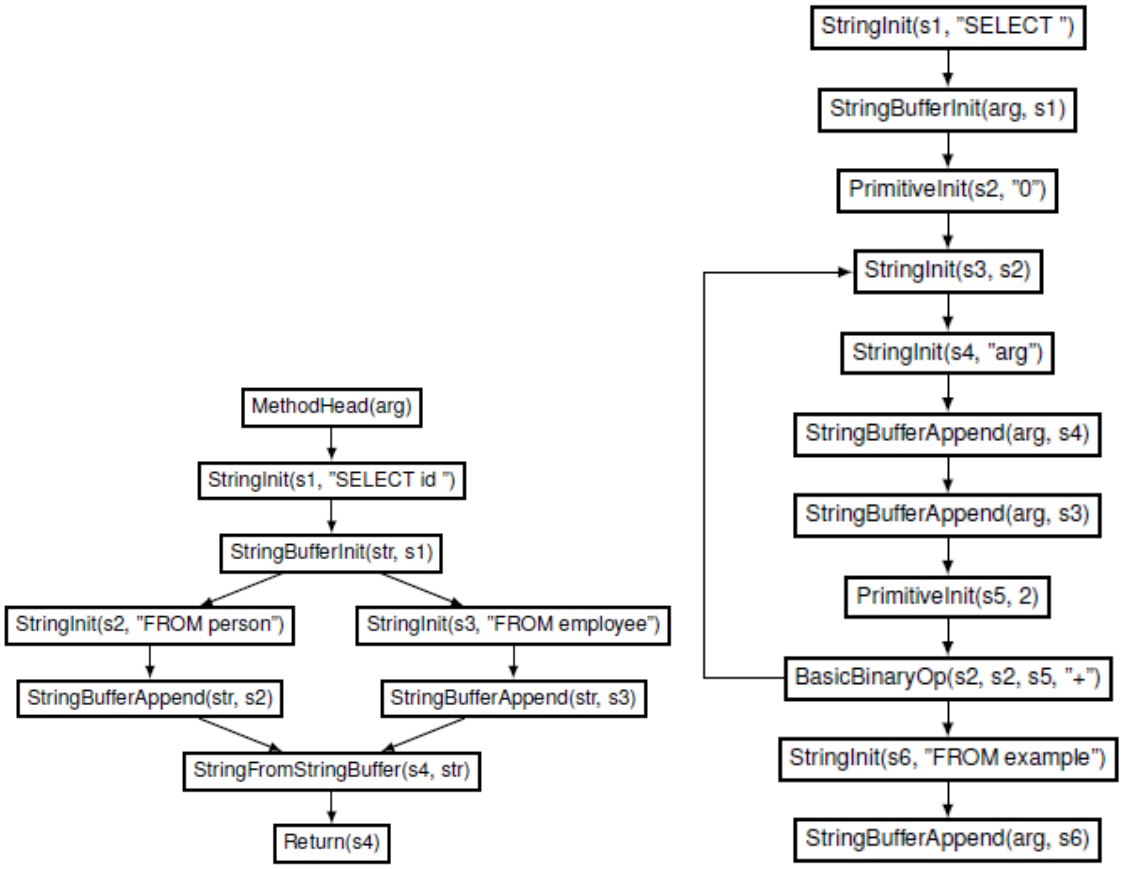
```
public String Ate(int x){
    StringBuffer str = new StringBuffer("SELECT id ");
    if (x > 0) {
        str.append("FROM person");
    } else {
        str.append("FROM employee");
    }
    return str.toString();
```

```
}
```

Listing 6: Пример теста с ветвлением

```
StringBuffer arg = new StringBuffer("SELECT ");
for(int i = 1; i < 9; i = i + 2){
    arg.append("arg");
    arg.append(i);
}
arg.append("FROM example");
```

Listing 7: Пример теста с циклом



(a) Тест с ветвлением (b) Тест с циклом

Рис. 11: Промежуточные представления

JSA не имеет подсветки синтаксиса как и возможности инкрементальной проверки генерируемого кода, однако при этом он умеет обнаруживать и позиционировать ошибки.

3.2.3. PHPSA

Описание. PHPSA [10] – инструмент для статического анализа строковых выражений в программах на языке PHP. Базовая структура инструмента представлена на рисунке 12.

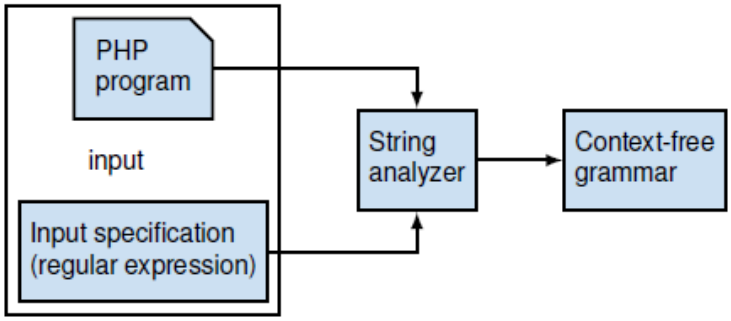


Рис. 12: Базовая структура PHPSA

Данный инструмент базируется на идеях алгоритма JSA. У инструмента отсутствует этап преобразования контекстно-свободной грамматики в регулярный язык, что повышает точность проводимого анализа. Для анализа строковых операций используется конечный преобразователь, что позволяет оставаться в рамках контекстно-свободной грамматики, то есть и входной, и выходной языки конечного преобразователя являются контекстно-свободными. Дальнейший же анализ строковых выражений аналогичен анализу JSA.

Тесты. Встроенными языками, которые поддерживает PHPSA, могут быть HTML и XML, поэтому в качестве основного языка был выбран HTML, так как он более распространен, чем XML. Ниже представлена классификация тестов.

- Простые – это строковые выражения, представляющие из себя HTML страницы, полученные строковыми операциями.

```
<html><head><title>test</title></head>
<body>
<?php
for ($i=1; $i < 10; $i++)
  echo "<p>";
?>
</body>
</html>
```

Listing 8: Пример просто теста

- С ветвлениями – это строковые выражения, содержащие в себе ветвление некоторых частей. Такие случаи возникают при необходимости по какому-то условию создать несколько различных выражений имеющих некую общую часть.

```
<head><title>test</title></head>
<body>
<?php
if ($_POST["abc"] == "abc")
    echo "abc";
else
    echo htmlspecialchars($_POST["xyz"]);
?>
</body>
```

Listing 9: Пример теста с ветвлением

- С циклами – это строковые выражения, имеющие какие-то схожие части, которые можно перечислить при использовании циклов. Случаи использования таких примеров возникают часто, когда используемые выражения имеют общие части, которые могут быть получены при итерации циклов.

```
<html>
<head><title>test</title></head>
<body>
<?php
// (<p></p>)^n<div>(<p></p>)^n</div>
$x = "<div>";
for ($i = 0; $i < 10; $i++)
  $x = "<p></p>".$x."<p></p>";
echo $x;
echo "</div>";
?>
</body>
</html>
```

Listing 10: Пример теста с циклом

Результаты. PHPSA умеет анализировать тесты с ветвлениями и циклами. Так при анализе примера с ветвлением (см. листинг 9) для входящей спецификации **$_POST : [/[a-z<]*/]** анализатор может показать некоторые примеры, удовлетворяющие входным данным (см. листинг 11). При проверке на валидность выражения в примере с циклом (см. листинг 10) PHPSA сообщит о корректности этого выражения.

```
<head><title>test</title></head>
<body>
abc
</body>
```

Listing 11: Пример анализа, удовлетворяющий входной программе

Инструмент также поддерживает работу со строковыми операциями PHP (см. листинг 12). Так операция **str_replace("00","0",$x)** может быть получена с помощью

преобразователя, представленного на рисунке 13, где A – любой символ, отличный от 0. Переходы с $0/\epsilon$ и A/0A означают, что ϵ и 0A продукции для 0 и A соответственно.

```
<?php
for ($i = 0; $i < $n; $i++)
  $x = "0".$x."1";
echo str_replace("00","0",$x);
?>
```

Listing 12: Пример программы со строковыми операциями

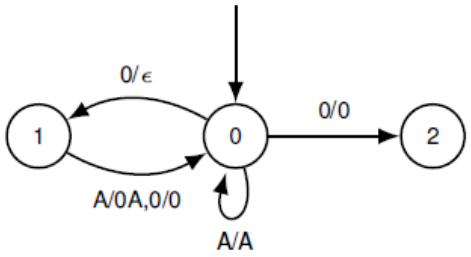


Рис. 13: Преобразователь, реализующий операцию replace

PHPSA дополнительно может проверить, что язык, построенный по PHP программе, и язык, построенная по входной спецификации, имеют пустое пересечение или же первый язык вложен во второй. Для того чтобы преобразовать PHP программу в контекстно-свободную грамматику, PHPSA использует промежуточное представление программы – Constraints (см. рис. 14).

```
$i = 0; $S = "";
echo ""abc;
while ($i < 10) {
    $S = $S"."xx;
    $i = $i + 1;
}
echo $S;
```

(a) PHP программа

```
"" ⊆ S
"abc" ⊆ O1
S."xx" ⊆ S
O1.S ⊆ O
```

(b) Промежуточное представление

```
S → ϵ
O1 → abc
S → Sxx
O → O1S
```

(c) Контекстно-свободная грамматика

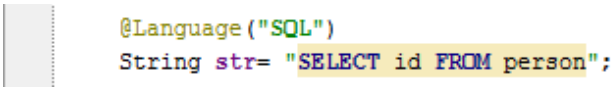
Рис. 14: Процесс преобразования PHP программы в контекстно-свободную грамматику

Что касается обнаружения ошибок, то инструмент обнаруживает лишь последнюю ошибку, при этом не позиционируя ее. Подсветка синтаксиса и анализ в режиме реального времени тоже отсутствуют. Кроме как определить, корректно ли выражение или нет, и построить контекстно-свободную грамматику, инструмент более ничего не умеет.

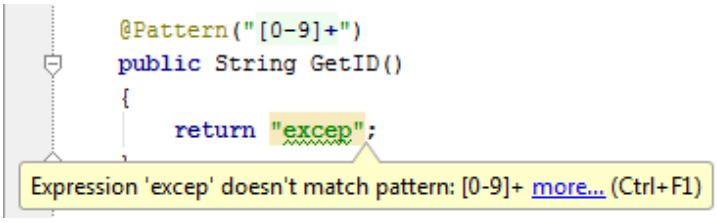
3.2.4. IntelliLang

Описание. IntelliLang [7] — плагин к средам разработки PHPStorm и IntelliJ IDEA. Он является комбинацией трех основных функций:

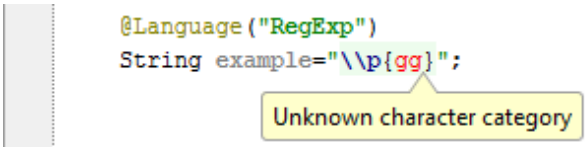
- Language Injection — редактирование строковых выражений на встроенном языке с помощью некоторых стандартных функций интегрированных сред разработак.



- Pattern Validation – проверка того, что строковые выражения задаются указанным регулярным выражением.



- Regular Expression Support – поддержка пользовательских регулярных выражений. К примеру, валидация имен символьных свойств.



Тесты. Так как IntelliLang поддерживает такие языки, как SQL, JavaScript, HTML, XML, CSS, JSON, то основными языками были выбраны SQL и JavaScript. Классификация же тестов выбрана была та же, что и у Alvor.

- Простые – это строковые выражения, представляющие из себя короткие, не вложенные? SQL-запросы, которые часто используются для получения относительно мощного множества. (см. листинг 1).

- Вложенные – это строковые выражения, которые содержат несколько уровней вложенности SQL-запроса. Такие запросы часто используются для более точной выборки из баз данных (см. листинг 2).

- С ветвлениями – это строковые выражения, содержащие в себе ветвление некоторых частей. Такие запросы возникают при необходимости по какому-то условию создать несколько различных запросов имеющих некую общую часть (см. листинг 3).

- С циклами – это строковые выражения, имеющие какие-то схожие части, которые можно перечислить при использовании циклов. Такие запросы используются часто, когда используемые выражения имеют общие части, которые могут быть получены при итерации циклов (см. листинг 4).

Результаты. Инструмент умеет анализировать все типы тестов.

```
@Language("SQL")
String sql = "SELECT id FROM" +
        "(SELECT id FROM person WHERE exp>5)";
```

Рис. 15: Вложенный тест

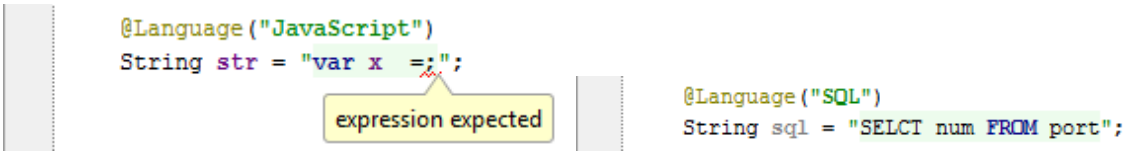
```
public String GetEmployee(int arg) {
    @Language("SQL")
    String str = "SELECT id FROM person ";
    if (arg > 0) {
        str += "WHERE id=arg";
    } else {
        str += "WHERE experience=3";
    }
    return str;
}
```

Рис. 16: Тест с ветвлением

```
public String GetPerson(int count) {
    @Language("SQL")
    String str = "SELECT ";
    int temp = 0;
    while (temp < count - 1) {
        str += "arg" + temp + ", ";
        temp++;
    }
    str += "arg" + temp + " ";
    str += "FROM person";
    return str;
}
```

Рис. 17: Тест с циклом

IntelliLang поддерживает автодополнение, подсветку синтаксиса, обнаружение ошибок и их позиционирование (см. рис 18a), однако обнаружение распространяется не на все встроенные языки, к примеру SQL, HTML (см. рис 18b).



(a) Обнаружение и позиционирование ошибок

(b) Не обнаружена ошибка в выражении на языке SQL

Рис. 18: Обнаружение ошибок

У инструмента есть особенность в виде prefix/sufix – выражения, которые должны предшествовать анализу или следовать за ним. Prefix может быть использован для подачи контекстной информации. Так в JavaScript может передаться информация о существовании какой-то функции (см. рис 19). При отсутствии prefix в этом примере будет выдано соответствующее предупреждение.

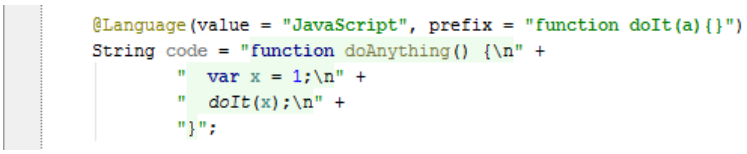


Рис. 19: Передача контекстной информации через prefix

Все функции инструмента работают на основе аннотаций, которые можно создавать, ”отнаследовавшись” от имеющихся (см. рис 20). Однако именно в этих аннотациях кроются все недостатки инструмента, а именно те, что данные аннотации могут быть применены только к полям и параметрам типа String или методам, возвращающим указанный тип, и то, что аннотации нужно указывать для каждой строковый переменной.

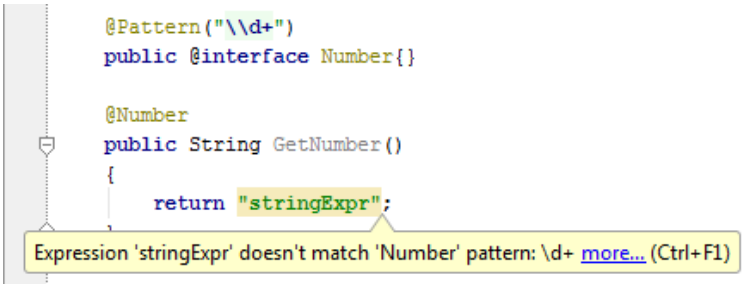


Рис. 20: Создание новых аннотаций

Заключение

В рамках выполнения данной работы были получены следующие результаты.

- Были выбраны инструменты для сравнения
- Были сформированы критерии сравнения инструментов
- Было составлено описание инструментов: основные возможности, поддерживаемые языки
- Было проведено сравнение инструментов по составленным критериям (см. таблицу 1). В данной таблице под "Да*" подразумевается наличие данной особенности, но лишь для первого значения, "Да˜" – для последнего, "Да̀" – не во всех встроенных языках, "Да́" – наличие возможности конструировать такие выражения, хотя плагин сообщает о не поддержке таких конструкций.
- Данная работа была представлена на конференции "Современные технологии в теории и практике программирования". Тезисы опубликованы в сборнике материалов конференции.

Инструмент	Alvor	JSA	PHPSA	IntelliLang
Обнаружение ошибок	Да*	Да	Да˜	Да̀
Позиционирование ошибок	Да*	Да	Нет	Да̀
Расширяемость языков	Нет	Да	Нет	Да
Подсветка синтаксиса	Нет	Нет	Нет	Да
Инкрементальный анализ	Да	Нет	Нет	Да
Поддержка ветвлений	Да	Да	Да	Да
Поддержка циклов	Да́	Да	Да	Да

Таблица 1: Результаты сравнения инструментов

В результате работы выявлено, что существующие инструменты либо решают узкую задачу, как правило, проверка корректности выражений, и расширение их возможностей затруднено (к таким анализатором относятся JSA и PHPSA), либо предоставляют широкий набор функций, но не поддерживают сложные и частые случаи формирования выражений, например IntelliLang. При этом существуют многофункциональные инструменты, такие как Alvor, реализующие мощный алгоритм анализа, но не обладающие архитектурой, позволяющей создавать новые инструменты.

Таким образом, необходима разработка набора готовых унифицированных компонент, упрощающих создание инструментов для решения различных задач анализа динамически формируемых строковых выражений. Разработка такого инструментария ведётся на основе проекта YaccConstructor, так как он изначально разрабатывался как модульный инструмент и предоставляет необходимую инфраструктуру для расширяемости.

Список литературы

[1] Automata-based Symbolic String Analysis for Vulnerability Detection / Fang Yu, Muath Alkhalaf, Tevfik Bultan, Oscar H. Ibarra. — Hingham, MA, USA : Kluwer Academic Publishers, 2014. — . — Vol. 44. — P. 44–70. — URL: http://dx.doi.org/10.1007/s10703-013-0189-1.

[2] Christensen Aske Simon, Møller Anders, Schwartzbach Michael I. Precise Analysis of String Expressions. SAS'03. — Berlin, Heidelberg : Springer-Verlag, 2003. — P. 1–18. — ISBN: 3-540-40325-6. — URL: http://dl.acm.org/citation.cfm?id=1760267.1760269.

[3] Dasgupta Arjun, Narasayya Vivek, Syamala Manoj. A Static Analysis Framework for Database Applications. ICDE '09. — Washington, DC, USA : IEEE Computer Society, 2009. — P. 1403–1414. — ISBN: 978-0-7695-3545-6. — URL: http://dx.doi.org/10.1109/ICDE.2009.98.

[4] Einarsson Arni, Nielsen Janus Dam. A survivor's guide to Java program analysis with soot. — 2008.

[5] Feldthaus Asger, Møller Anders. The Big Manual for the Java String Analyzer / Department of Computer Science, Aarhus University. — 2009. — November. — Available from http://www.brics.dk/JSA/.

[6] Hanneforth Thomas. Finite-state Machines: Theory and Applications. — 2008.

[7] JetBrains. IntelliLang. — 2015. — URL: https://www.jetbrains.com/idea/help/intellilang.html (online; accessed: 04.05.2015).

[8] JetBrains. PHPStorm. — 2015. — URL: https://www.jetbrains.com/phpstorm/ (online; accessed: 04.05.2015).

[9] Klein Gerwin. From a Verified Kernel Towards Verified Systems. APLAS'10. — Berlin, Heidelberg : Springer-Verlag, 2010. — P. 21–33. — ISBN: 3-642-17163-X, 978-3-642-17163-5. — URL: http://dl.acm.org/citation.cfm?id=1947873.1947877.

[10] Minamide Yasuhiko. Static Approximation of Dynamically Generated Web Pages. WWW '05. — New York, NY, USA : ACM, 2005. — P. 432–441. — ISBN: 1-59593-046-9. — URL: http://doi.acm.org/10.1145/1060745.1060809.

[11] Nguyen Hung Viet, Kästner Christian, Nguyen Tien N. Building Call Graphs for Embedded Client-side Code in Dynamic Web Applications. FSE 2014. — New York, NY, USA : ACM, 2014. — P. 518–529. — ISBN: 978-1-4503-3056-5. — URL: http://doi.acm.org/10.1145/2635868.2635928.

[12] Nguyen Hung Viet, Kästner Christian, Nguyen Tien N. Varis: IDE Support for Embedded Client Code in PHP Web Applications. — New York, NY : ACM Press, 2015. — . — Formal Demonstration paper.

[13] A Static Analysis Framework For Detecting SQL Injection Vulnerabilities / Xiang Fu, Xin Lu, Boris Peltsverger et al. COMPSAC '07. — Washington, DC, USA : IEEE Computer Society, 2007. — P. 87–96. — ISBN: 0-7695-2870-8. — URL: http://dx.doi.org/10.1109/COMPSAC.2007.43.

[14] String-embedded Language Support in Integrated Development Environment / Semen Grigorev, Ekaterina Verbitskaia, Andrei Ivanov et al. CEE-SECR '14. — New York, NY, USA : ACM, 2014. — P. 21:1–21:11. — ISBN: 978-1-4503-2889-0. — URL: http://doi.acm.org/10.1145/2687233.2687247.

[15] Yu Fang, Alkhalaf Muath, Bultan Tevfik. STRANGER: An Automata-based String Analysis Tool for PHP. TACAS'10. — Berlin, Heidelberg : Springer-Verlag, 2010. — P. 154–157. — ISBN: 3-642-12001-6, 978-3-642-12001-5. — URL: http://dx.doi.org/10.1007/978-3-642-12002-2_13.

[16] van den Brand Mark G. J., Sellink A., Verhoef C. Current Parsing Techniques in Software Renovation Considered Harmful. IWPC '98. — Washington, DC, USA : IEEE Computer Society, 1998. — P. 108–. — ISBN: 0-8186-8560-3. — URL: http://dl.acm.org/citation.cfm?id=580914.858216.