МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное Государственное Бюджетное Образовательное учреждение высшего профессионального образования

«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВПО «КубГУ»)

Кафедра математического моделирования

КУРСОВАЯ РАБОТА

Томита-парсер в системе трансляции из естественного языка в язык SQL

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.А. Бабичев

(подпись, дата)

Факультет ФКТиПМ курс 3

Направление 010400.62 – Прикладная математика и информатика

Научный руководитель,

канд. тех. наук, доцент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.В. Бессарабов

Подпись Дата

Нормоконтролер,

канд. физ.-мат. наук, доцент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. Е. Рубцов

Подпись Дата

Краснодар 2015

РЕФЕРАТ

Курсовая работа 28 с., 4 рис., 5 источников, 4 приложения.

ORM МОДЕЛЬ, TOMITA-ПАРСЕР, PHP, JAVASCRIPT, MYSQL, SQL, FONT, CSP.

Объектом исследования является Tomita-парсер и создание взаимодействия с ним через веб-интерфейс.

Цель работы:

* улучшить взаимодействие с Tomita-парсером;
* реализовать запросы с подзапросами;
* увеличить производительность системы;
* реализация Web-интерфейса доступа к данным;

Исследования проводились с помощью стандартов PSR-0, PSR-1, PSR-2, технологии CSP в СУБД MySQL и технологий: PHP 5.3, JavaScript, PHPixie2, jQuery.

С использованием указанных стандартов и технологии:

* улучшена система взаимодействия с Томита-парсером;
* реализована возможность создавать запросы с подзапросами;
* увеличена скорость получения ответа;
* реализован Web-интерфейс доступа к данным;

Результаты работы могут быть использованы для автоматизации обучения студентов языку SQL.

СОДЕРЖАНИЕ

[Введение 4](#_Toc431419449)

[1 Постановка задачи 5](#_Toc431419450)

[2 Томита-парсер 7](#_Toc431419451)

[3 Взаимодействие с Томита-парсером 11](#_Toc431419452)

[4 Запросы с подзапросами 13](#_Toc431419453)

[Заключение 15](#_Toc431419454)

[Список используемых источников 16](#_Toc431419455)

[Приложение А Вес слова и ранжирование 17](#_Toc431419456)

[Приложение Б Список терминов 19](#_Toc431419457)

[Приложение В Работа с парсером 21](#_Toc431419458)

[Приложение Г Обработка деревьев 24](#_Toc431419459)

# ВВЕДЕНИЕ

Работы последних лет связаны с решением проблемы анализа смысла языка в приложении к созданию систем диалога с программным обеспечением. Подходов к решению задачи понимания естественно-языковых(ЕЯ) запросов несколько. Наиболее распространенными являются подходы, основанные на синтаксическом, семантическом анализах и шаблонах.

Проблема трансляции с ЕЯ в SQL разрабатывалась давно, но даже в таких серьезных работах, как [1] рассматривается упрощённый формат фразы естественного языка.

Основные проблемы возникают из-за не определеннозначности естественного языка. В распространённых формулировках заданий, как минимум используются эллипсис и анафора.

Предлагаемая работа ориентирована на улучшение сервиса программы для трансляции задания с ЕЯ в язык SQL, в которой предоставляет интерфейс к Томита-парсеру. В частности, решена задача повышения быстродействия системы и возможность создания запросов с подзапросами.

# Постановка задачи

Создается сервис для преобразования заданий с естественного языка (ЕЯ) в специализированный декларативный язык SQL.

Существуют словари, SQL схемы c их помощью осуществима работа тривиальных преобразований. Трансляции более сложных заданий, использующих: эллипсис, анафора, должны выполняться выполнятся при помощи парсера, использующего грамматики, или наборов правил схем.

Желательно получить время реагирования меньше секунды для тривиальных преобразований. Для запросов с подзапросами до второго уровня время реагирования должно составлять не более секунды, для второго и третьего не более секунды.

Предполагается что разработанная программа будет использована как для обучения языку SQL, так и в качестве рабочего инструмента. Интерфейс должен обеспечить удобную работу пользователя (UX). Так же в интерфейсе должна быть возможность работать с подзапросами. Подзапросы должны поддерживать три уровня вложенности.

Томита-парсер – создан для извлечения структурированных данных из текста на естественном языке. Вычленение фактов происходит при помощи контекстно-свободных грамматик и словарей ключевых слов.

Существует словарь ключевых слов и грамматика определенной базы данных. Именно для неё и будем решать задачу, используя Томита–парсер.

Взаимодействие с Томита-парсером должно осуществляться при помощи языка PHP. Предполагается что результатом взаимодействия с парсером будет файл формата JSON, для удобного взаимодействия с ним в дальнейшем.

# Томита-парсер

Подходов к решению задачи понимания естественно-языковых запросов несколько. Наиболее распространенными являются подходы, основанные на синтаксическом, семантическом анализах и шаблонах. Первый подход основан на использовании синтаксических конструкций. Синтаксическое представление запроса строится на основе подлежащего, сказуемого, прямого дополнения и т.п., которые определяются с помощью морфологических характеристик. Такое представление ничего не говорит о смысле запроса.

Второй подход, основанный на семантике, гораздо ближе к смыслу запроса в нем используется синтаксическая информация из предыдущего подхода, а также информация из семантических словарей. Каждое слово в словаре имеет характеристики, позволяющие определять смысловые отношения между ним.

Основная задача при этом - отсечь ненужные смыслы, постараться выделить с помощью синтаксических связей достоверные семантические конструкции. В больших предложениях, особенно с многозначными словами, это часто приводит к перебору множества смыслов и связей между ними. Это лишь одна проблема, стоящая на пути понимания естественно-языковых запросов.

Вторая сложность - типичные естественно-языковые запросы, которые, как правило, не имеют правильных синтаксических конструкций. На это влияют вольное словоизменение и словообразование в виде неологизмов сетевой общественности, большой процент имен собственных и сокращений, игнорирование правил пунктуации, что приводит к тому, что от естественного языка во всем его многообразии иногда остается лишь лексика. Так же необходимые в этом подходе семантические словари - очень трудоемкая составляющая.

Третий подход к анализу естественно-языковых запросов основан на шаблонах. Он появился самым первым и с точки зрения программной реализации наиболее прост. Суть его в том, что возможные запросы покрываются набором шаблонов-конструкций, позволяющих отождествляться с запросом и выдавать в результате предопределенные конструкции.

Основной недостаток такого подхода заключается в необходимости предусмотреть все возможные способы выражений на естественном языке, т.е. исчислить грамматику. Однако современный пользовательский язык не похож на литературный. Поэтому поисковые запросы синтаксическими шаблонами в чистом виде покрыть довольно трудно.

Томита-парсер создан для извлечения структурированных данных из текста на естественном языке. Вычленение фактов происходит при помощи контекстно-свободных грамматик и словарей ключевых слов. Парсер позволяет писать свои грамматики и добавлять словари для нужного языка.

Томита-парсер позволяет по написанным пользователем шаблонам выделять из текста разбитые на поля цепочки слов или факты. Например, можно написать шаблоны для выделения адресов. Здесь фактом является адрес, а его полями — «название города», «название улицы», «номер дома» и т.д.

Парсер включает в себя три стандартных лингвистических процессора: токенизатор (разбиение на слова), сегментатор (разбиение на предложения) и морфологический анализатор (mystem).

Основные компоненты парсера: газеттир, набор КС-грамматик и множество описаний типов фактов, которые порождаются этими грамматиками в результате процедуры интерпретации.

**Газеттир** — словарь ключевых слов, которые используются в процессе анализа КС-грамматиками. Каждая статья этого словаря задает множество слов и словосочетаний, объединенных общим свойством. Например, «все города России». Затем в грамматике можно использовать свойство «является городом России».

Слова или словосочетания можно задавать явно списком, а можно «функционально», указав грамматику, которая описывает нужные цепочки. Например, цепочка ключевых слов «адрес» описывается соответствующей грамматикой и может быть использована в грамматике для выделения городских происшествий. Подробнее об этом будет в описании механизма каскадов.

**Грамматика** — множество правил на языке КС-грамматик, описывающих синтаксическую структуру выделяемых цепочек. Грамматический парсер запускается всегда на одном предложении. Перед запуском терминалы грамматики отображаются на слова предложения.

Одному слову может соответствовать много терминальных символов. Таким образом, парсер получает на вход последовательность множеств терминальных символов. Например, в нашей грамматике есть всего два термина Verb и Noun, а входное предложение «Мама мыла стекло.». Тогда парсер получит на вход такую последовательность: {Noun}, {Verb, Noun}, {Verb, Noun}. На выходе получаются цепочки слов, распознанные этой грамматикой.

**Факты** — таблицы с колонками, которые называются полями фактов. Факты заполняются во время анализа парсером предложения. Как и чем заполнять поля фактов указывается в каждой конкретной грамматике. Это называется интерпретацией. Типы факты описываются на специальном языке в отдельном файле.

Парсер может порождать много вариантов разбора одной и той же цепочки, которые отличаются только деревом разбора. Различные деревья могут порождать разные варианты заполнения полей фактов. Для этой цели правилу можно приписывать вес. По умолчанию вес у всех правил равен 1. Но его можно искусственно уменьшить, чтобы парсер выбрал вариант с большим весом, если такой построился.

Программный код вынесен в приложение А.

# Взаимодействие с Томита-парсером

Взаимодействие PHP и Томита-парсера будет построено на ожидании завершения процесса. Когда создается процесс, он получает уникальный индекс в бытовых ОС, в случае с RTOS [Приложение Б] будем перехватывать указатель. Возможны ситуации, когда Томита-парсер не сможет уложиться в отложенное для него время.

Например, прислали обработчику слишком много данных. Тогда у пользователя должно появиться диалоговое окошко, см. Рисунок 1, в котором предоставляется выбор пользователю. Закрыть сессию (Отменить) и продолжить работать с данными, или ждать ответ.

В случае, если в течение пяти секунд ответа от сервера не поступило, автоматически соединение с пользователем закрывается, но процесс продолжает работать. Если пользователь снова отправит эти данные, он получит индекс старого процесса, если он еще не завершился.

Если процесс закончил работу, данные кэшируются и в течение суток получить эти данные можно мгновенно, без обработки и Томита-парсера. Если в течение часа преобладают некоторые запросы, они кэшируются на сутки.

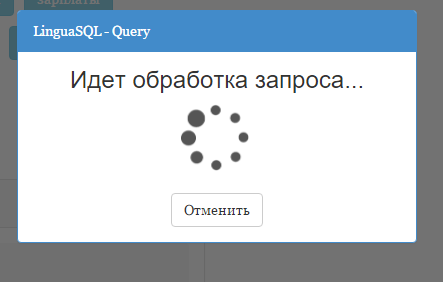


Рисунок 1 – Ожидание запроса

Параллельная отправка одинакового запроса от разных пользователей будет одним процессом. Два разных пользователя получат один уникальный индекс, который скажет им о завершении процесса и отправит им данные. Тем самым происходит экономия ресурсов сервера.

Код программы вынесен в приложение В.

Данный метод взаимодействия увеличил скорость в ~70 раз, рисунок 2 и рисунок 3. До этого метода использовалось проверка на существование файла результатов. То есть в PHP был бесконечный цикл, который проверял наличие(существование) файла. Если в течение 5 секунд файл не появлялся, цикл завершался и возвращал exception.

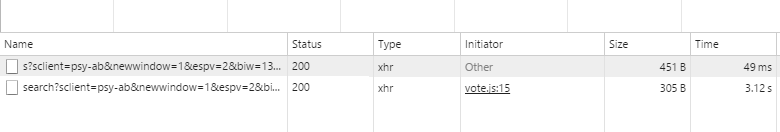


Рисунок 2 – Запрос к Томита-парсеру

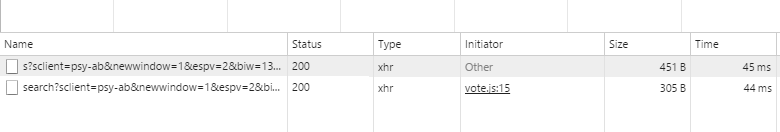


Рисунок 3 – Запрос к Томита-парсеру, улучшенный алгоритм

Скорость увеличилась потому что скрипту не нужно индексировать папку и проверять наличие файла в папке, что очень ресурсоемкая задача, ему достаточно дожидаться завершения процесса по его уникальному индексу, благодаря этому получение ответа от Томита-парсера происходит мгновенно. Мы уже уверены, что по завершению процесса будет файл. Но если файл не был создан, возвратиться Exception, который PHP нормально обработает и вернет соответствующий ответ.

# Запросы с подзапросами

С помощью SQL вы можете вкладывать запросы внутрь друга друга. Обычно, внутренний запрос генерирует значение, которое проверяется в предикате внешнего запроса, определяющего верно оно или, нет. Например, предположим, что мы знаем имя продавца: Kira, но не знаем значение его поля num, и хотим извлечь все порядки из таблицы заказов. Имеется способ чтобы сделать это:

SELECT \*

FROM Orders

WHERE num = (

SELECT num

FROM Salespeople

WHERE name = Kira

);

Чтобы оценить внешний (основной) запрос, SQL сначала должен оценить внутренний запрос (или подзапрос) внутри предложения WHERE. Он делает это так, как и должен делать запрос, имеющий единственную цель - отыскать через таблицу Продавцов все строки, где поле name равно значению Kira, и затем извлечь значения поля num этих строк.

Единственной найденной строкой естественно будет num = 1004. Однако SQL, не просто выдает это значение, а помещает его в предикат основного запроса вместо самого подзапроса, так чтобы предиката прочитал что WHERE num = 1004.

В сервисе же, создание запросов с подзапросами реализуется с помощью дерева. Пользователь, в ручном режиме, с помощью интерфейса собирает дерево, которое отправляет в обработчик. Используя SQL схему, обработчик, осуществляет трансляцию из ЕЯ в язык SQL.

Пользователь может выбрать какие таблицы и по каким колонкам ему нужно соединить. Первое, что делает анализатор, написанный на JavaScript, разбивает текст, условие, на отдельные слова. Тем самым пользователь сам сортирует данные и отправляет их. Дальше анализатор, если обнаружил возможный подзапрос, отправляет условие с помощью ajax в алгоритм Стеммер Портера, который в словарике проверяет наличие таблиц. Если таблицы были найдены, подключается словарь схем таблиц и они отправляются пользователю, который уже решает правильно ли система определила возможные соединения, рисунок 4. Так же пользователь может сам соединить таблицы в ручном режиме.

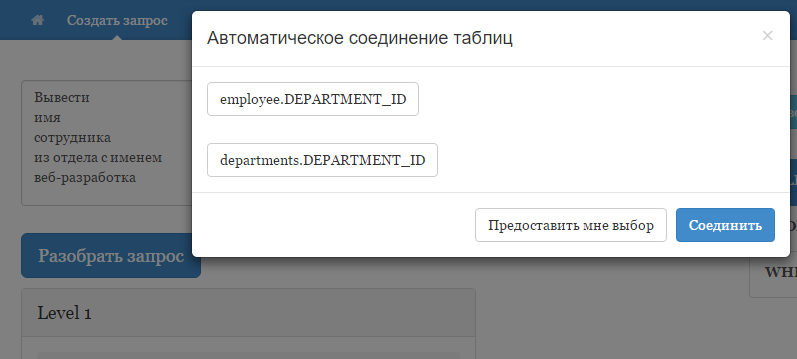


Рисунок 4 – Автоматическое соединение таблиц

Код программы вынесен в приложение Г.

# ЗАКЛЮЧЕНИЕ

Результатом данной работы является программное обеспечение трансляции с фрагмента естественного языка в язык SQL. Разработан алгоритм доступа для взаимодействия с Томита-парсером, по средствам Ajax. Существует возможность создавать запросы с подзапросами. Система может быть применена как в обучении, так и в профессиональной сфере.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Найханова, Л.В. Методы и алгоритмы трансляции естественно–языковых запросов к базе данных в SQL-запросы / Л.В. Найханова, И.С. Евдокимова. – Улан-Удэ: ВСГТУ, 2004. – 148 с.
2. Бессарабов, Н.В. Реализация графовых моделей данных и знаний в Cache / Н.В. Бессарабов, А.Г. Коблов // Экологический вестник научных центров ЧЭС, 2006, Приложение. с. 64–68.
3. Бессарабов, Н. В. Гипотеза Сепира-Уорфа и инвариантные структуры данных // Обозрение прикладной и промышленной математики, 2002, т. 9. в. 2. с. 336-337.
4. Документация к фреймворку PHPixie [электронный ресурс] URL: http://phpixie.com/2.x.html (дата обращения 11.03.2015).
5. Документация к фреймворку jVision [электронный ресурс] URL: http://rez1dent3.github.io/jVision/ (дата обращения 17.05.2015).

# ПРИЛОЖЕНИЕ А

ВЫЧИСЛЕНИЕ ВЕСА СЛОВА, И РАНЖИРОВАНИЕ

public function weight($word)

{

$profile = [

'С' => 3, 'Г' => 3,

'П' => 2, 'Н' => 2,

'ПРЕДЛ' => 0, 'СОЮЗ' => 0,

'МЕЖД' => 0, 'ВВОДН' => 0,

'ЧАСТ' => 0, 'МС' => 0

];

$parts\_of\_speech = $this->morphy->get('ru')->getPartOfSpeech($word);

if (!$parts\_of\_speech)

return 1;

$range = [];

$index = 0;

foreach($parts\_of\_speech as $word => $speech) {

$range[$index] = 1;

if (is\_array($speech)) {

if (count($speech)) {

$temp\_speech = [current($speech)];

while (next($speech)) {

$key = current($speech);

if (isset($profile[$key]))

$temp\_speech[] = $profile[$key];

}

$range[$index] = max($temp\_speech);

}

}

else {

if (isset($profile[$speech])) {

$range[$index] = $profile[$speech];

}

}

}

return max($range);

}

# ПРИЛОЖЕНИЕ Б

СПИСОК ТЕРМИНОВ, ВСТРЕЧАЮЩИХСЯ В РАБОТЕ

Система реального времени (СРВ) — это система, которая должна реагировать на события во внешней по отношению к системе среде или воздействовать на среду в рамках требуемых временных ограничений. Оксфордский словарь английского языка говорит об СРВ как о системе, для которой важно время получения результата. Другими словами, обработка информации системой должна производиться за определённый конечный период времени, чтобы поддерживать постоянное и своевременное взаимодействие со средой. Естественно, что масштаб времени контролирующей системы и контролируемой ей среды должен совпадать.

Под реальным временем понимается количественная характеристика, которая может быть измерена реальными физическими часами, в отличие от логического времени, определяющего лишь качественную характеристику, выражаемую относительным порядком следования событий. Говорят, что система работает в режиме реального времени, если для описания работы этой системы требуются количественные временные характеристики.

Операционная система, ОС — комплекс взаимосвязанных программ, предназначенных для управления ресурсами вычислительного устройства и организации взаимодействия с пользователем.

В логической структуре типичной вычислительной системы операционная система занимает положение между устройствами с их микроархитектурой, машинным языком и, возможно, собственными (встроенными) микропрограммами (драйверами) — с одной стороны — и прикладными программами с другой.

Разработчикам программного обеспечения операционная система позволяет абстрагироваться от деталей реализации и функционирования устройств, предоставляя минимально необходимый набор.

Операционная система реального времени, ОСРВ — тип операционной системы, основное назначение которой — предоставление необходимого и достаточного набора функций, обеспечивающих разработку программными средствами систем реального времени на конкретном аппаратном оборудовании.

# ПРИЛОЖЕНИЕ В

ВЗАИМОДЕЙСТВИЕ С ТОМИТА ПАРСЕРОМ

public static function path\_config()

{

$path = self::\_path\_io() . self::config;

$p = self::\_path\_io() . self::dconfig;

$change = false;

if (file\_exists($\_p = self::path\_path()))

$change = md5\_file($\_p) != md5(self::\_path\_io());

else

file\_put\_contents($\_p, self::\_path\_io());

if ($change) {

if (file\_exists($path))

unlink($path);

file\_put\_contents($\_p, self::\_path\_io());

}

if (!file\_exists($path) || (filectime($path) > filectime($p))) {

$content = file\_get\_contents($p);

$content = strtr($content, array(

'{PATH\_PRETTY}' => self::change\_ds(self::path\_pretty()),

'{PATH\_INPUT}' => self::change\_ds(self::path\_input()),

'{PATH\_OUTPUT}' => self::change\_ds(self::path\_output()),

'{PATH\_DICTIONARY}' => self::change\_ds(self::path\_mydic()),

'{TOMITA\_TYPE}' => self::get\_type\_config()

));

file\_put\_contents($path, $content);

}

return $path;

}

public static function path\_Tomita()

{

$Tomita = 'tomita';

switch (\App\Config::OS) {

case 'Darwin':

$Tomita .= '-mac';

break;

case 'Windows':

case 'WIN32':

case 'WINNT':

$Tomita .= 'parser.exe';

break;

case 'FreeBSD':

$Tomita .= '-' . (\App\Config::bit\_os\_64() ? 'freebsd64' : 'linux32');

break;

default:

$Tomita .= '-linux' . (\App\Config::bit\_os\_32() ? '32' : '64');

break;

}

return self::$root . self::Tomita . \App\Config::DS . $Tomita;

}

public static function exec()

{

chdir(self::$root . self::Tomita);

try {

llexec(self::path\_Tomita() . " " . self::path\_config());

}

catch (\Exception $e) {

llpassthru(self::path\_Tomita() . " " . self::path\_config());

}

}

# ПРИЛОЖЕНИЕ Г

ОБРАБОТКА ДЕРЕВЬЕВ

public function select(\SimpleXMLElement $xml, $get)

{

$obj = json\_decode(json\_encode($xml));

if (!isset($obj->document))

return ['Error' => '333u'];

$obj = $obj->document->facts->Select;

$array\_result = [];

$from = [];

foreach ($obj as $key => $c) {

if (isset($c->What)) {

$array\_result[$key]['What'] = $this->syntax\_sql($get($c, 'What'));

}

if (isset($array\_result[$key]['What'])) {

preg\_match('/[\(]{0,}([A-Z\_\d]+)[\.]/i', $array\_result[$key]['What'], $out);

if (isset($out[1])) {

$from[] = $out[1];

}

}

}

$result = array\_unique(

array\_map(function ($array) {

return $array['What'];

}, $array\_result)

);

$from = array\_unique($from);

return array\_merge(

$array\_result,

array(

'From' => $from,

'Result' => $result,

'Error' => count($array\_result) - count($result)

)

);

}

public function join(\SimpleXMLElement $xml, $get)

{

$obj = json\_decode(json\_encode($xml));

if (!isset($obj->document))

return ['Error' => '333u'];

$obj = $obj->document->facts->Clause;

$array\_result = [];

foreach ($obj as $key => &$c) {

$array\_result[$key]['What'] = '';

$array\_result[$key]['AndOr'] = '';

$array\_result[$key]['Is'] = '';

$array\_result[$key]['Then'] = '';

$array\_result[$key]['Result'] = '';

$array\_result[$key]['Error'] = false;

if (isset($c->What)) {

$array\_result[$key]['What'] = $get($c, 'What');

$array\_result[$key]['Is'] = $get($c, 'Is');

$array\_result[$key]['Then'] = $get($c, 'Then');

$array\_result[$key]['Result'] =

$array\_result[$key]['What'] . ' ' .

$array\_result[$key]['Is'] . ' ' .

$array\_result[$key]['Then'];

$array\_result[$key]['Error'] = $array\_result[$key]['What'] == $array\_result[$key]['Then'];

}

if ((count($array\_result) > 1) && isset($c->AndOr)) {

if (empty($array\_result[$key - 1]['What']))

return ['Error' => '444u'];

$array\_result[$key]['What'] = &$array\_result[$key - 1]['What'];

$array\_result[$key]['AndOr'] = $get($c, 'AndOr');

$array\_result[$key]['Is'] = $get($c, 'Is');

$array\_result[$key]['Then'] = $get($c, 'Then');

$array\_result[$key]['Result'] =

$array\_result[$key]['What'] . ' ' .

$array\_result[$key]['Is'] . ' ' .

$array\_result[$key]['Then'];

$array\_result[$key]['Error'] = $array\_result[$key]['What'] == $array\_result[$key]['Then'];

}

}

$str = [];

$index = 0;

$errors = 0;

$skb = [];

foreach ($array\_result as &$w) {

if (!empty($w['AndOr'])) {

if ($w['AndOr'] == 'AND') {

$str[$index] = $str[$index - 1];

$str[$index - 1] = '(';

$index++;

$skb[] = 1;

}

$str[$index++] = $w['AndOr'];

}

else if ($index) {

foreach ($skb as $s) {

$str[$index++] = ')';

}

$w['AndOr'] = 'OR';

$str[$index++] = $w['AndOr'];

}

$str[$index++] = $w['Result'];

$errors += (int)$w['Error'];

}

foreach ($skb as $s)

$str[$index++] = ')';

return array\_merge(

$array\_result,

array(

'Result' => $str,

'Error' => $errors

)

);

}