

# Синтаксический анализ для «встроенных» языков

Андрей Бреслав

Соавторы: A. Annamaa, V. Vene  
(University of Tartu, Estonia)

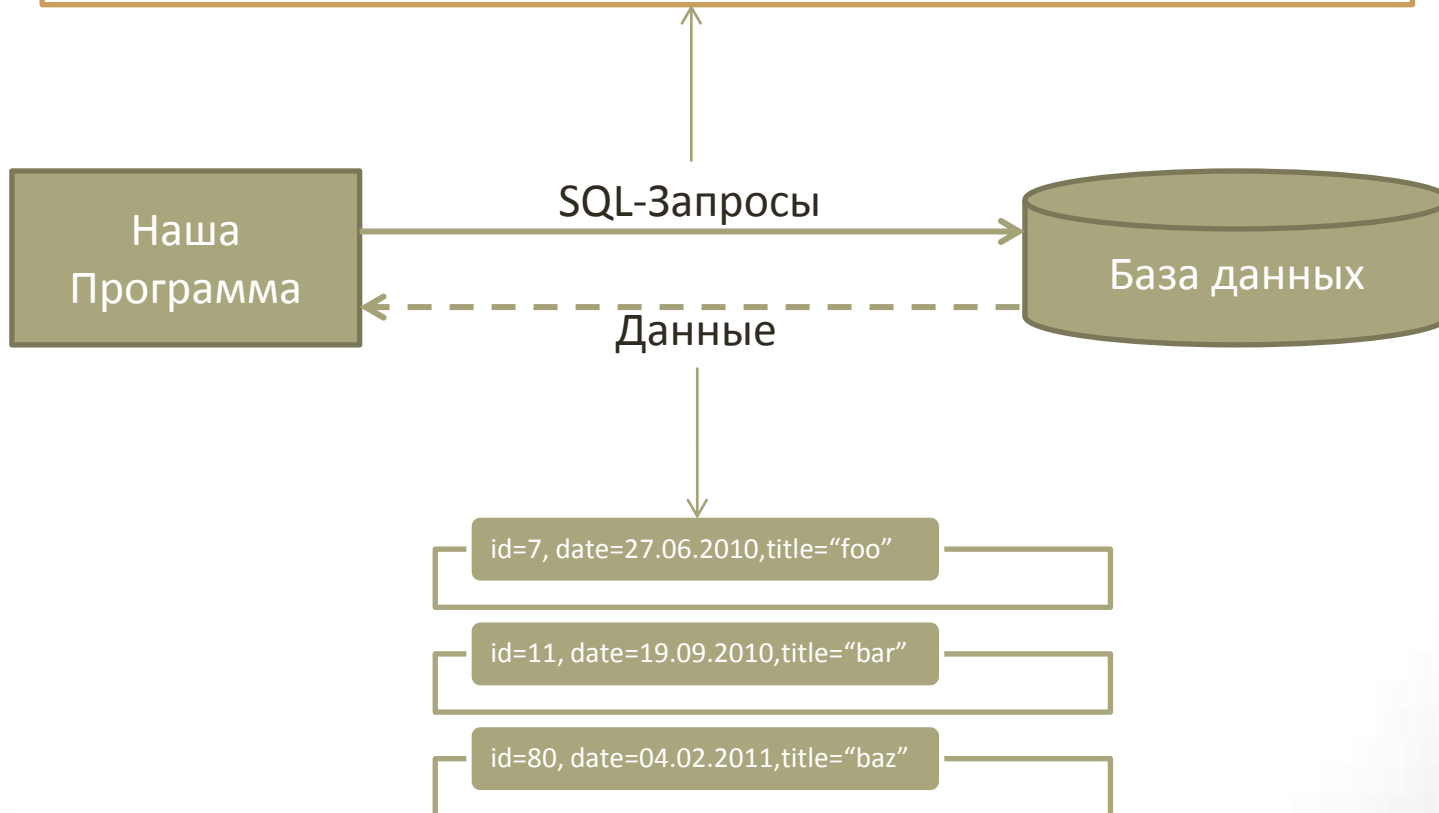


STACC

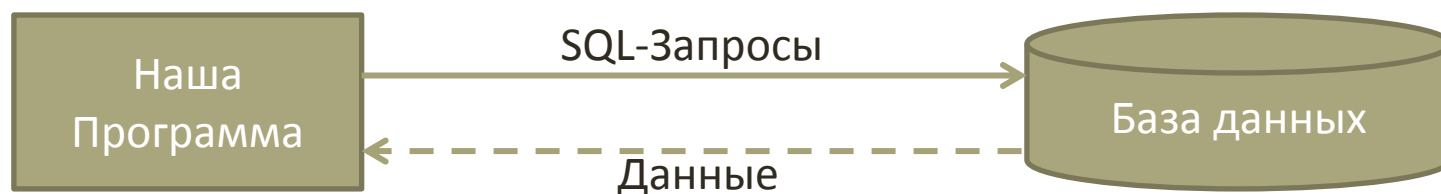
Software Technology and  
Applications Competence Center

# Немного о предмете

```
SELECT id, date, title  
FROM Orders  
WHERE (user_id=239) AND (completed=FALSE)  
ORDER BY date ASC
```

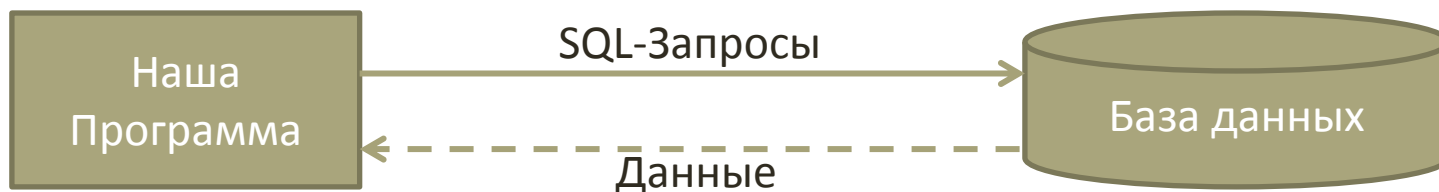


# Как работает «Наша программа»



```
public PreparedStatement selectOrders(  
    int userId, boolean completedOnly,  
    boolean ascOrder) {  
    String sql = "SELECT id, date, title," +  
        "FROM Orders" +  
        "WHERE (user_id=" + userId;  
    if (completedOnly)  
        sql += "AND (completed=FALSE)";  
    sql += "ORDER BY date";  
    sql += (ascOrder) ? "ASC" : "DESC";  
  
    return ConnectionProvider.conn.prepareStatement(sql);  
}
```

# Кто заметил ошибки?



```
public PreparedStatement selectOrders(  
    int userId, boolean completedOnly,  
    boolean ascOrder) {  
    String sql = "SELECT id, date, title,"  
        ➡ "FROM Orders" +  
        ➡ "WHERE (user_id=" + userId;  
    if (completedOnly)  
        sql += "AND (completed=FALSE)";  
    sql += "ORDER BY date";  
    sql += (ascOrder) ➡ "ASC" ➡ "DESC";  
  
    return ConnectionProvider.conn.prepareStatement(sql);  
}
```

# Постановка задачи

- **Статически** обнаруживать синтаксические **ошибки** в SQL-запросах **внутри Java-строк** и сообщать о них пользователю, **не запуская** его программу

# Что кроме SQL?

**<?xml?>**



URI:  
git+ssh://foo.bar.com:foo.git



```
String.format("Foo %s, %d bar!", s, x);
```

# Общая схема решения

Какие строковые выражения нужно проверить?

- `prepareStatement (sql)`
- `prepareCall (sql)`
- `executeQuery (sql)`
- `executeUpdate (sql)`

Каковы возможные значения данного выражения?

- Задача **алгоритмически неразрешима** в общем случае
- Как представить результат?
  - Множества бывают **бесконечными**

Удовлетворяют ли эти значения требованиям синтаксиса встроенного языка?

- $CF \subseteq CF$  – неразрешима
- $REG \subseteq CF$  – неразрешима
- $REG \subseteq REG$  – разрешима
  - но SQL – не регулярный

# Общая схема решения

Какие строковые выражения нужно проверить?

- `prepareStatement (sql)`
- `prepareCall (sql)`
- `executeQuery (sql)`
- `executeUpdate (sql)`

Каковы возможные значения данного выражения?

- **Аппроксимация:**  
Построим регулярное множество, содержащее все возможные значения выражения

Удовлетворяют ли эти значения требованиям синтаксиса встроенного языка?

- **Аппроксимация:**  
Найдем регулярный язык, содержащийся в SQL, и будем проверять включение вида  **$REG \subseteq REG$**  – разрешимая задача



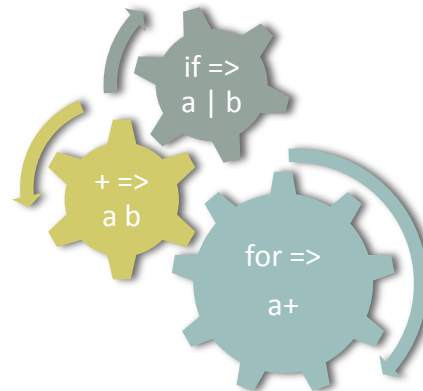
# Abstract Strings

```
String sql = "SELECT id";  
if (needNames)  
    sql += ", name";  
sql += "FROM People WHERE age <= " + maxAge;  
if (minAge >= 0)  
    sql += "AND age >= " + minAge;  
connection.prepareStatement(sql);
```

Где аппроксимация?

Сокращение:

AS? := (AS | "")

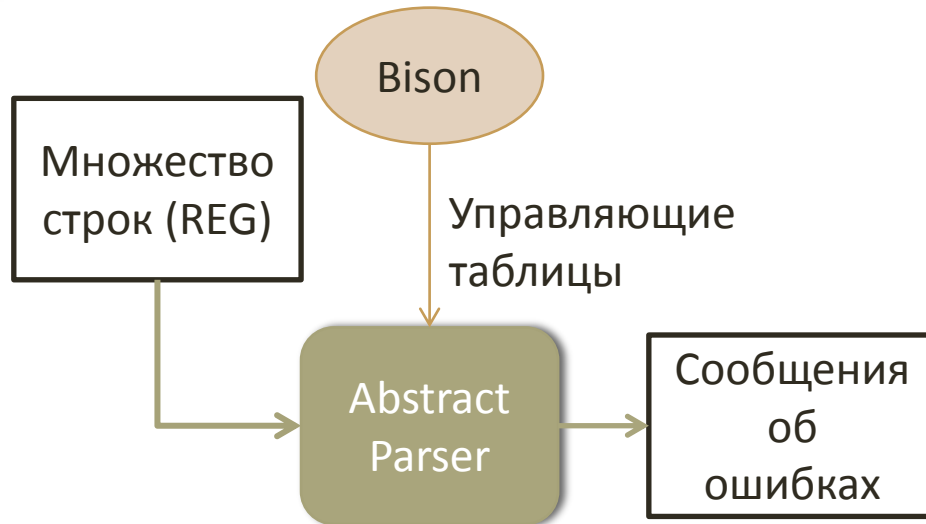


```
"SELECT id ", name? "FROM People WHERE age <= "  
age ("AND age >= minAge)?
```

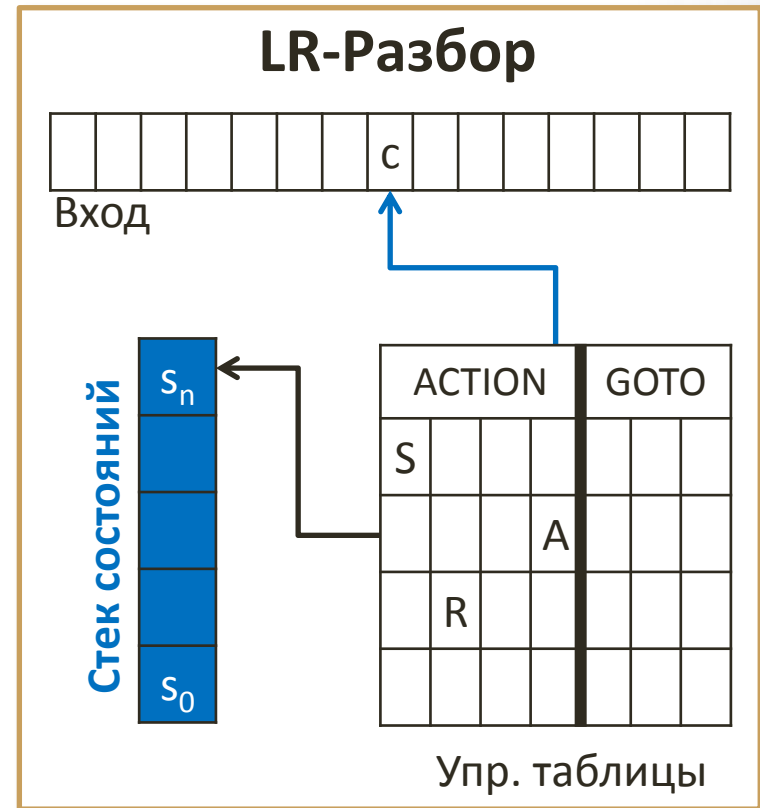
# Время работы

Проект	Размер LOC	# Выражений		Время (сек)			Память
		всего	ошибок	общее	AStrings	кэш	
Plazma	48'520	94	4	6	<b>3.8</b>	<b>0.4</b>	65
Compiere	319'570	1343	12	138	<b>120</b>	<b>0.5</b>	445

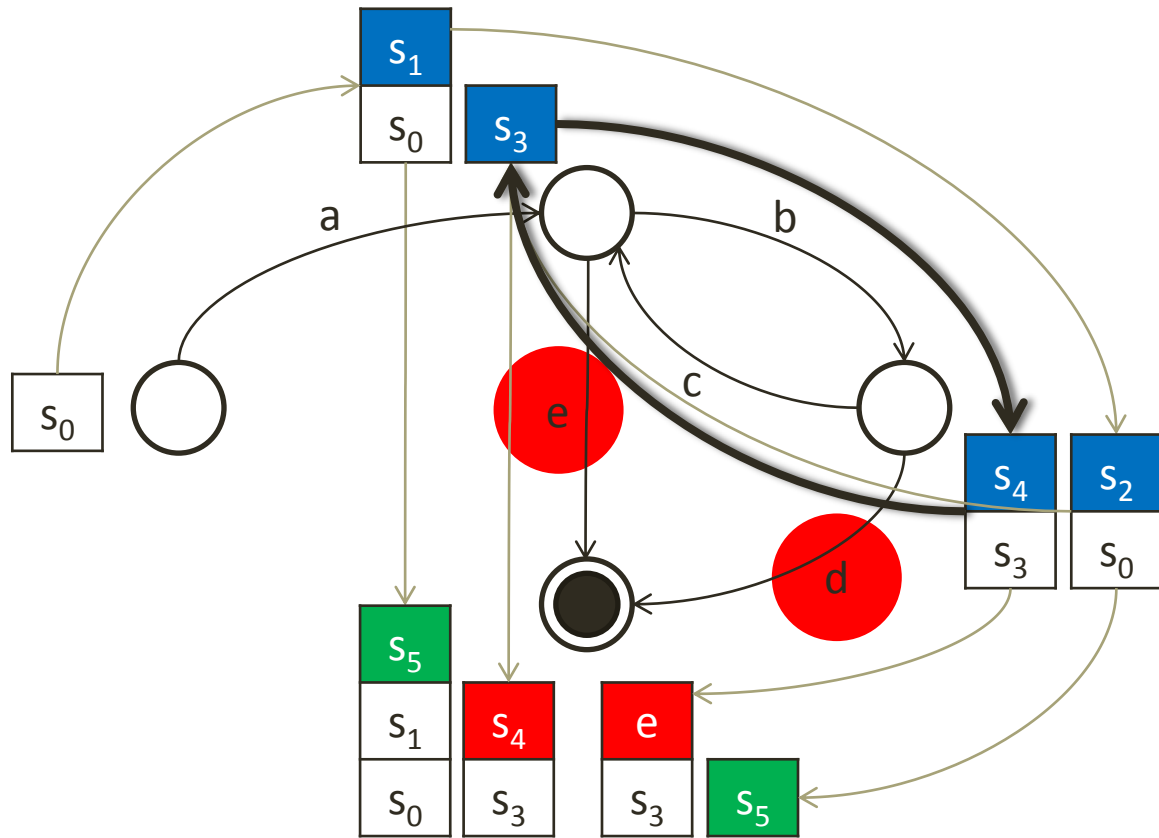
# Abstract Parsing



- Управляющие таблицы **не меняются** (для данного языка)
- **Изменяемое состояние:**
  - **Стек состояний** парсера
  - **Позиция считывающей головки**
- Основная идея абстрактного разбора
  - Для каждой **позиции** во входном **автомате**
    - Вычислить **множество всех** возможных **стеков** состояний парсера



# Алгоритм

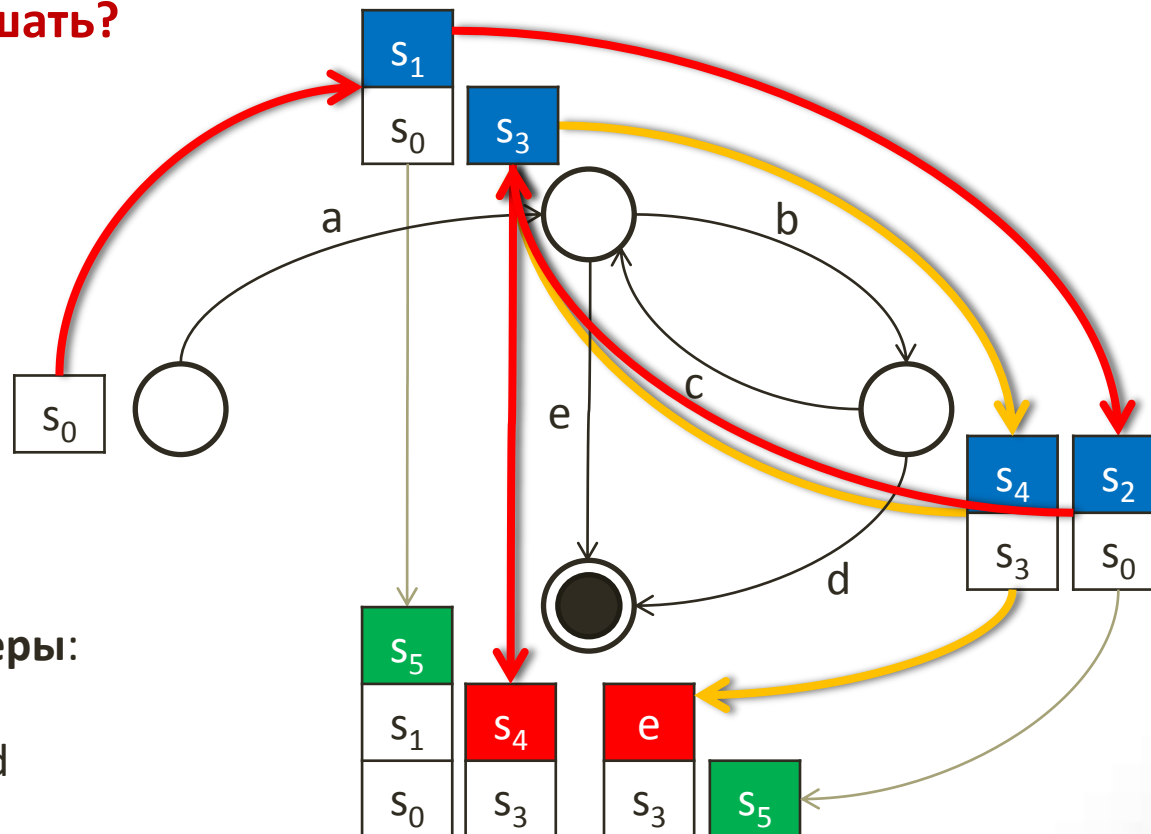


# Время работы

- $S_{in}$  – множество состояний входного автомата
- $S_p$  – множество управляющих состояний парсера
- $Stacks(S_p)$  – множество стеков состояний парсера
- Время работы алгоритма
  - $O(|S_{in}| * |Stacks(S_p)|) = \infty$ , задача неразрешима!
- Регулярная аппроксимация
  - **Ограничим глубину стеков** состояний парсера числом  $D$
  - $O(|S_{in}| * |Stacks(S_p)|) = O(|S_{in}| * |S_p|^D) < \infty$

# Поиск контрпримеров

- Пользователю нужно показать, **какую именно** неправильную строку может сформировать его программа
  - Контрпример** – путь в **графе стеков**, заканчивающийся **ошибочным** состоянием
  - Обычно нас интересует **самый короткий** контрпример
- Как решать?**



Контрпримеры:

→ a(bc)+e

→ ab(cb)+d

# Технические требования

- Нужно сообщать об ошибках в процессе написания кода
  - **нужны инкрементальные алгоритмы**
- Подход должен единообразно поддерживать разные встроенные языки

<?xml?>



SQL



- **Хотелось бы** просто описывать синтаксис (контекстно-свободной) **грамматикой**
- **Не хотелось бы** создавать такие грамматики вручную. Лучше взять готовую, например, из **стандарта языка**.
- Грамматики, приводимые в стандартах, **содержат множество неоднозначностей**

# Что я не рассказал

- Abstract GLR-Parsing
  - Возможность работать с неоднозначными грамматиками
- Abstract Lexical Analysis
  - Входной алфавит парсера на самом деле не Unicode, а алфавит лексем: ключевых слов, идентификаторов, констант...
    - Как сконвертировать один автомат в другой?
- Как проверять отсутствие опечаток в идентификаторах
  - Автоматические тесты
  - **[Открытый вопрос]** Булевские грамматики
- Метод можно **обобщить** так, что для любого ***автоматного предиката*** можно
  - Проверить его истинность на регулярном множестве строк
  - Найти кратчайший контрпример



# Темы дипломных работ

1. [М] Использование **булевских грамматик** и **Abstract Parsing** для обнаружения опечаток в **идентификаторах**
2. [М] Использование **булевских грамматик** для реализации **автодополнения** идентификаторов в SQL-запросах
3. [Б] Оптимизация **регулярной абстракции**: ограничивать глубину стека только там и так, где и как это необходимо
4. [Б] Оптимизация потребления памяти **GLR Abstract Parsing** – разработка и реализация эффективных структур данных для хранения множества множеств стеков