Relaxed Parsing of Regular Approximations of String-Embedded Languages

Author: Ekaterina Verbitskaia

Saint Petersburg State University
JetBrains Programming Languages and Tools Lab

26/08/2015

String embedding

- Dynamic SQL

```
IF @X = @Y
    SET @TBL = ' #table1 '
ELSE
    SET @TBL = ' table2 '
SET @S = 'SELECT x FROM' + @TBL + 'WHERE ISNULL(n,0) > 1'
EXECUTE (@S)
```

- Embedded SQL

```
SqlCommand myCommand = new SqlCommand(
    "SELECT * FROM table WHERE Column = @Param2",
    myConnection);
myCommand.Parameters.Add(myParam2);
```

Problems

- String-embedded code are expressions in some programming language
  - It may be necessary to support them in IDE: code highlighting, autocomplete, refactorings
  - It may be necessary to transform them: migration of legacy software on new platforms
  - It may be necessary to detect vulnerabilities in such code
  - Any other problems of programming languages can occur

Static analysis of string-embedded code

- Performed without programm execution
- Checks that the set of properties holds for each possible expression value

- Undecidable for string-embedded code in the general case
- The set of possible expression values is over approximated and then the approximation is analysed.

Static analysis of string-embedded code: the scheme

- Identification of hotspots: points of interest, where the analysis is desirable
- Approximation construction
- Lexical analysis
- **Syntactic analysis**
- Semantic analysis

Code: hotspot is marked

```
string res = "";
for(i = 0; i < l; i++) {
    res = "()" + res;
}
use(res);
```

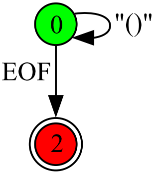
Possible values

```
{"",
 "()",
 "()()",
 ...
 "()"^l,
}
```

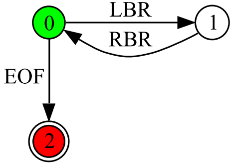
Regular approximation

```
("()")*
```

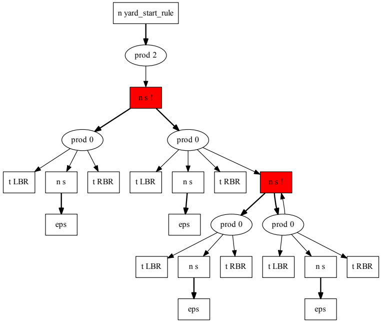
Approximation



lex out



ast



Existing tools

- Java String Analyzer, Alvor
  - Regular approximation
- PHP String Analyzer
  - Context-free approximation
- Kyung-Goo Doh et al.
  - Data flow equations in the domain of LR-stacks

- Flaws
  - Hard to extend them with new features or support new languages
  - Do not create structural representation of code

Problem statement

The aim is to develop the algorithm suitable for syntactic analysis of string-embedded code

Tasks:

- Develop an algorithm for parsing of regular approximation of embedded code which produce a finite parse forest
- Parse forest should contain a parse tree for every correct (w.r.t. reference grammar) string accepted by the input automaton
- Incorrect strings should be omitted: no error detection

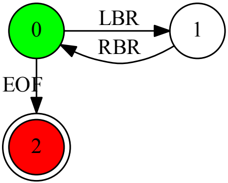
Algorithm

- **Input**: reference DCF-grammar G and DFA graph with no ϵ-transitions over the alphabeth of terminals of G
- **Output**: finite representation of the trees corresponding to all correct string accepted by input automaton

Algorithm

```
string res = "";
for(i = 0; i < l; i++) {
    res = "()" + res;
}
```

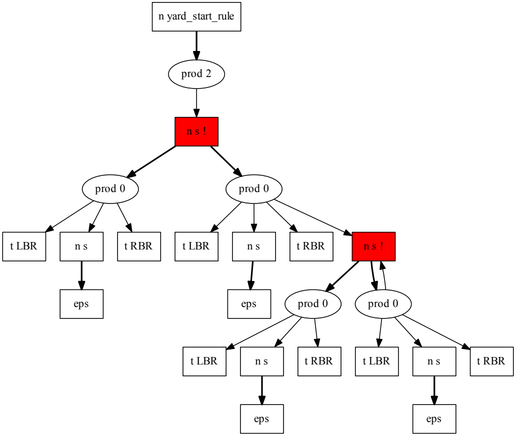
Approximation:



Grammar:

$$\begin{array}{lcl} \mathit{start} & ::= & s \\ s & ::= & \mathtt{LBR}\ s\ \mathtt{RBR}\ s \\ s & ::= & \epsilon \end{array}$$

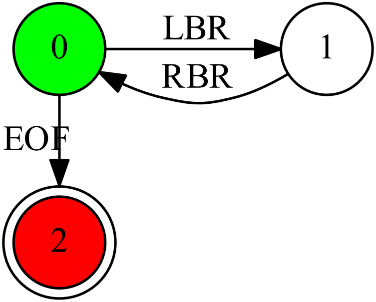
Output (SPPF):

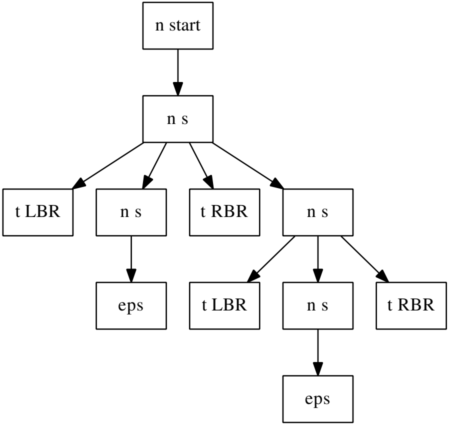


Algorithm

- Traverse the automaton graph and sequentially construct GSS, similarly as in RNGLR
- The set of LR-states is associated with each of input graph vertices
- The order in which the vertices of input graph are traversed is controled with a queue. The vertex is enqueued whenever new edge with the head equal to the vertex is added to the GSS
- The algorithm implements relaxed parsing: errors are not detected, erroneous strings are ignored

Correct tree – derivation tree of some string accumulated along the path in the input graph





Algorithm: correctness

Theorem (Termination)

Algorithm terminates for any input

Theorem (Correctness)

Every tree, generated from SPPF, is correct

Theorem (Correctness)

For every path p in the inner graph, recognized w.r.t. reference grammar, a correct tree corresponding to p can be generated from SPPF

Implementation

- The algorithm is implemented as a part of YaccConstructor project using F# programming language
- The generator of RNGLR parse tables and data structures for GSS and SPPF are reused

Evaluation

- The data is taken from the project of migration from MS-SQL to Oracle Server
- 2,7 lines of code, 2430 queries, 2188 successfully processed
- The number of queries which previously could not be processed because of timeout is decreased from 45 to 1

Conclusion

- The algorithm for parsing of regular approximation of dynamically generated string which constructs the finite representation of parse forest is developped
- Its termination and correctness are proved
- The algorithm is implemented as a part of YaccConstructor project
- The evaluation demonstrated it could be used for complex tasks