Embeddable Framework for Syntax-Safe Source Code Generation

Kotelnikov Evgenii

Saint-Petersburg State University,
Faculty of Applied Mathematics and Control Processes

8 March, 2012

Source Code Generation

Simplest example

```
String.format('SELECT * FROM %s
               WHERE first_name = "%s"
                 AND last_name  = "%s"',
              tableName, firstName, lastName);
```

Others

- Source-to-source compilers, preprocessors
- Parser and lexer generators
- Generally, any application involving XML, JSON, etc.

Approaches to Source Code Generation

Code templates

- Naturally suited for any input language
- Embedded in every programming language
- Hard to provide syntactical correctness
- Mix the logic and syntactical issues of produced language

```
String.format('SELECT * FROM %s
               WHERE first_name = "%s"
                 AND last_name  = "%s"',
              tableName, firstName, lastName);
```

Approaches to Source Code Generation

Source code generation framework

- Each input language requires it's own framework
- Tedious and error-prone to implement
- Easier to provide syntactical correctness
- Isolates concerns of concrete syntax

```
selectAll(tableName,
          and(equal("first_name", firstName),
              equal("last_name", lastName)));
```

Existing solutions

Restricted to

- Set of supported input languages

  (`System.CodeDOM`, `JavaGen`, ORM systems)
- Set of target programming languages

  (`System.CodeDOM`, `qretty`, `ASF+SDF`)

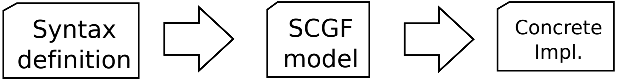
Objective

Design and implementation of a translator

- Takes arbitrary syntax description
- Produce implementation of source code generation framework
- That guarantees syntactical correctness
- In unrestricted set of target programming languages

Design concept

Tranfsormation to a formal model



Design concept

Language definition

- Is given as context-free grammar

Source code generation framework

- Set of algebraic data types definitions
- Printer function

Transformation

- To every non-terminal T associate type $\tau(T)$
- Relations between non-terminals are mapped to correspondent relations between types

Implementation

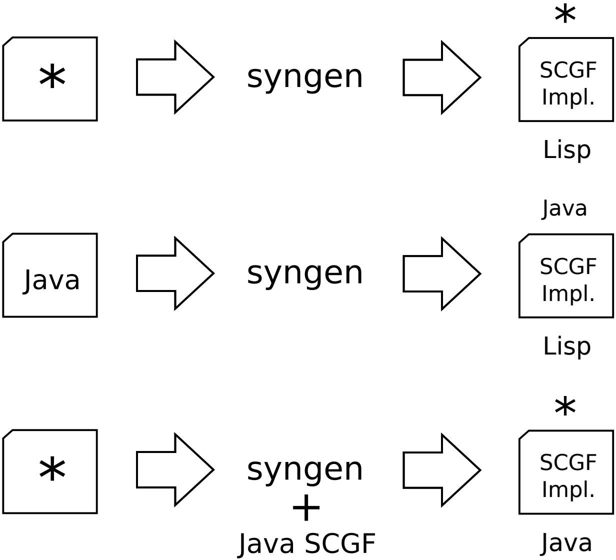
Problem

In order to produce SCGF implementation, `syngen` should itself use SCGF for target programming language.

Solution

Implement `syngen` in Lisp programming language.

Bootstrapping of syngen



Examples of syngen definitions

JSON

```
value : (nothing) "null"
      | (btrue) "true"
      | (bfalse) "false"
      | string
      | integer
      | (array)  "[" . value^","* . "]"
      | (object) "{" . entry^","* . "}"
      ;

entry : string . ":" value ;
```

Usage

```
pr(object(entry("name", string("John Doe")),
          entry("age", integer(32))
          entry("children", array(string("Bob"),
                                  string("Alisa")))))
```

Examples of syngen definitions

Python-like programming language

```
{ : . ":\n" . ;
} : . ;

$( : . "(" . ;
$) : . ")" ;

id : /[a-zA-Z_][a-zA-Z_0-9]*/ ;

unit : stmnt+ ;

_ stmnt \
    : "def" id $( id^", "* $) { stmnt+ }
    | "return" expr
    | (var) id "=" expr
    | "if" expr { stmnt+ }
    ;

expr : (ref) id
     | (int) integer
     | (op) expr op expr
     | (call) id $( expr^", "* $)
     ;

op : (equal) "=="
   | (plus)  "+"
   | (minus) "-"
   | (mult)  "*"
   ;
```

Thank you for attention

Kotelnikov Evgenii

St. Petersburg State University

evgenii.kotelnikov@student.spbu.ru

`http://apmath.spbu.ru/users/students/kotelnikov/`