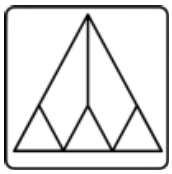

Home
Documentation
Language
Research Papers
Applications

Download
Continuous build
Extensions
Ftp

Support
Mailing lists
IRC
Users Days
Bug Reports

Developers
Subversion
Buildfarm results
Planet Stratego

Stratego Language

Stratego -- Strategies for Program Transformation

Stratego is a small and efficient domain-specific language for program transformation. It is based on the paradigm of programmable rewriting strategies, which makes it extremely well-suited for traversing and transforming all kinds of tree structures. Its library and supporting tools are geared towards processing computer programs in the form of concrete or abstract syntax trees, as produced by text parsers.

Terms

In Stratego, programs are represented by means of terms. Terms are essentially trees. The abstract syntax of a programming language is described by means of a signature.

A signature describing the four basic arithmetic operators (+, -, *, /) might look like this:

```
signature
  sorts Expr
  constructors
    Add : Expr * Expr -> Expr
    Sub : Expr * Expr -> Expr
    Mul : Expr * Expr -> Expr
    Div : Expr * Expr -> Expr
```

A term representing the expression `1 + 2 * 3` would read `Add(1, Mul(2, 3))`.

Conditional Rewrite Rules

Basic transformation steps are specified by means of *conditional rewrite rules*. A rule recognizes a subterm to be transformed by *pattern matching* and replaces it with a pattern instance.

The following rule, called `Eval`, can be applied to an `Add` tree where both subtrees are (integer) constants. It will replace the `Add` tree with the value of the left child added to the right.

```
Eval: Add(l, r) -> Int(n) where n := <add> (l, r)
```

Rewriting Strategies

Rewrite rules are a natural formalism for expressing single program transformations. However, using a standard strategy for normalizing a program with a set of rewrite rules is not adequate for implementing program transformation systems. It may be necessary to apply a rule only in some phase of a transformation, to apply rules in some order, or to apply a rule only to part of a program. These restrictions may be necessary to avoid non-termination or to choose a specific path in a non-confluent rewrite system.

Stratego supports the separation of strategies from transformation rules, thus allowing careful control over the application of these rules. As a result of this separation, transformation rules are reusable in multiple different transformations and generic strategies capturing patterns of control can be described independently of the transformation rules they apply.

The following strategy may be used to attempt and application of a rule `s`. If the

rule is not applicable, e.g., its pattern does not match, the `<+` operator will ensure that the identity, `id`, is applied (which always succeeds, but leaves the term unchanged):

```
try(s) = s <+ id
```

Generic Traversal

Instead of spelling out the details of traversals over trees for a specific abstract syntax, Stratego supports *generic traversal* by means of a few traversal primitives. These primitives can be combined to define a wide range of generic traversals, such as `topdown`, `bottomup`, `once`, `topdown`, and `innermost`.

The above `Eval` rule might be applied across a larger tree using the `bottomup` strategy:

```
bottomup(try(Eval))
```

This will apply the `Eval` rule to each subterm. If `Eval` succeeds, the old subterm is replaced with the result. If it fails, the old subterm is kept.

Dynamic Rules

Normal rewrite rules are *context-free*, that is, they use information only from the pattern that is matched, and possibly from subterms. The *dynamic rules* of Stratego allow the run-time generation of rewrite rules that inherit context information. For example, generation of an inlining rule at the declaration site.

Concrete Syntax

Specification of transformations in abstract syntax becomes cumbersome when the patterns become large. Therefore, Stratego supports specification of patterns in the *concrete syntax* of the *object* language. This is especially useful in template-based program generation, where large fragments of object language programs are instantiated.

The following rule,

```
desugar : While(e, stm) -> If(e, DoWhile(stm, e))
```

may alternatively be written using concrete syntax:

```
desugar : |[ while (e) stm; ]| -> |[ if (e) do stm while(e); ]|
```

The syntax inside the `|[ ]|` brackets is in the subject language, this case Java.

Annotations

The signature of an abstract syntax corresponds to the syntax of the object language. When performing transformations or analyses it is sometimes necessary to attach additional information to nodes. This is supported in Stratego by means of term annotations.

Overlays

Transformations are usually specified in terms of the abstract syntax constructors of the object language. Stratego *overlays* allow the specification of pattern abstractions. This can be used, for example, in defining application specific transformations.

Given the following overlays,

```
overlays
```

```
MethodCall(x) = MethodInvocation(None(), SimpleName(x), [], [])
RetNum(x) = ReturnStatement(NumberLiteral(x))
```

one is able to shorten (abstract syntax) patterns. The following pattern

```
?Block([MethodInvocation(None(), SimpleName("foo"), [], []),
        ReturnStatement(NumberLiteral("0"))
```

might be written as:

```
?Block([MethodCall("foo"), Ret("0")])
```

Stratego Library

The Stratego Library is a collection of modules with reusable rules and strategies. Included in the library are strategies for sequential control, generic traversal, built-in data type manipulation (numbers and strings), standard data type manipulation (lists, tuples, optionals), generic language processing, and system interfacing (I/O, process control, association tables).

In addition, the library contains functionality for dealing with control flow in programs, parsing, pretty-printing and checking that terms adhere to specific structural definitions.

Stratego Compiler

The Stratego Compiler translates Stratego specifications either into a format suitable for interpretation, or into high-level C code. The C-based run-time system is built around the ATerm library, which supports a very efficient in-memory representation of terms. This allows extremely fast pattern matching. There is also an experimental Java-based run-time system which abstracts over the actual term implementation. This has been used to plug Stratego into Eclipse and rewrite the Java abstract syntax trees provided by the Eclipse JDT.

Applications

Stratego is being applied in a wide variety of program transformation projects including a software improvement tool for Cobol programs (CobolX), a compiler for Tiger (Tiger Compiler), a documentation generation tool for SDL, tools for grammar transformation (GrammarTools), a tool for deforestation of functional programs through the warm fusion algorithm (HSX), and a tool for domain-specific optimization of C++ programs in the domain of numeric (coordinate-free) programming (CodeBoost). The StrategoCompiler is bootstrapped, i.e., implemented in Stratego itself.

Distribution

The Stratego programming language is distributed as part of Stratego/XT. This package combines Stratego language with XT, a collection free transformation tools and domain-specific languages that support the implementation of program transformations. The package includes a collection of syntax definitions for programming languages and tools for generating tools such as pretty-printers from syntax definitions.

Stratego/XT is distributed under the GNU Lesser General Public License. The distribution of and publications about Stratego are available from http://www.strategoxt.org.

History

Some notes on this history of Stratego can be found on the StrategoHistory page.