Stratego/XT in a Bash Shell

An introduction to Stratego/XT for the CS 846 course

Karl Trygve Kalleberg

Institutt for Informatikk
Universitetet i Bergen

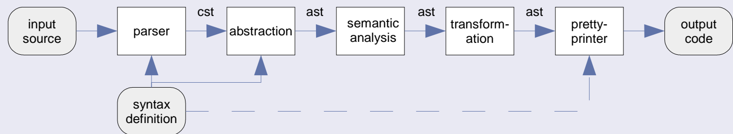
University of Waterloo
14. Feb 2006

What is Stratego/XT?

- **Stratego** – a modular, rule-based rewriting language with congruences, dynamic rewrite rules, embedded concrete syntax and flexible term traversal strategies.
- **XT** – a collection of ready-made transformation components, declarative languages, and generators for producing additional components.

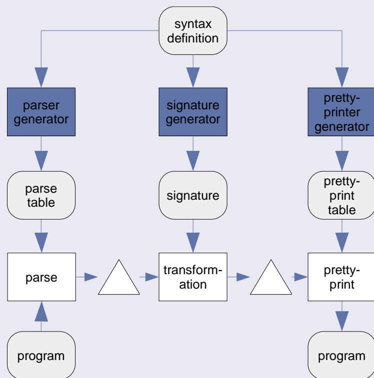
The Stratego/XT Pipeline

A Typical Transformation Pipeline



1. Source code is *parsed* into a *structured* representation, a *concrete syntax tree* (CST).
2. The CST is pruned for non-essential information, to yield an *abstract syntax tree* (AST).
3. Additional information is added, e.g. type information.
4. One or multiple *transformations* are applied to the AST.
5. The result is serialized back to file.

The Stratego/XT Architecture



The *syntax definition* declares the *structure* of the language for the programs we transform and is used in constructing most pipeline stages.

Part I

Elements of XT

Syntax Definitions

Elements

- **Sorts** – declare the names of grammar non-terminals; term comes from algebraic specifictions.
- **Productions** – grammar production rules.
- **Priorities** – ambiguities between productions can be solved by declaring priority.
- **Associativity** – ambiguities of operators can be solved by declaring associativity.
- **AST generation** – an abstract syntax tree will be machine-derived from the parse tree using AST building markers.
- **Restriction** – rules at the lexical level for disallowing possible derivations.

Pretty-Printing

Box-based

- All code elements are divided into *boxes*.
- Boxes can be arranged *horizontally* or *vertically*.
- Spacing between boxes can be controlled horizontally, vertically or “indentally”.

Pretty-Printing

Box-based

- All code elements are divided into *boxes*.
- Boxes can be arranged *horizontally* or *vertically*.
- Spacing between boxes can be controlled horizontally, vertically or “indentally”.

Examples

H hs=x [B B B] → B B B

V vs=x is=y [B B B] → B B B

A hs=x vs=y [
R [B B B]
R [B B B]
] → B B B / B B B

Part II

Elements of Stratego

Terms

- Terms are a *structured representation* of programs.
- Their structure is often derived from the syntax definition of the language.
- The syntax definition gives rules for the structural validation of terms.

Terms are primarily used to encode *abstract syntax trees*.

Structural Definition of Annotated Terms

Approximate Grammar for ATerms

```
t  := bt                     -- basic term
    | bt { t }               -- annotated term

bt := C                      -- constant
    | C(t1,...,tn)           -- n-ary constructor
    | (t1,...,tn)            -- n-ary tuple
    | [t1,...,tn]            -- list
    | "ccc"                  -- quoted string
    | int                    -- integer
    | real                   -- floating point number
    | blob                   -- binary large object
```

Signatures

- In Stratego, signatures are *data declarations* that define the *structure* of terms.
- Terminology comes from the field of *universal algebra*; sorts with operations.
- A *signature* is a set of *constructors* on the form:
  - `name : sort * sort * ... -> sort`

Rewrite Rules

Definition

A rewrite rule `R : l -> r where c` replaces the pattern l, called the *left-hand side*, with r, the *right-hand side* when:

- l matches the *current term*
- c holds

Informally

Rewrite rules are basic units of tranformation, taking one term to another, based on structural pattern matching.

Primitive Strategies

Identity

- Syntax: `id`
- Always succeed
- Some laws
  - `id ;` $s \equiv s$
  - s `; id` $\equiv s$
  - `id <+` $s \equiv$ `id`
  - s `<+ id` $\not\equiv s$

Failure

- Syntax: `fail`
- Always fail
- Some laws
  - `fail <+` $s \equiv s$
  - s `<+ fail` $\equiv s$
  - `fail ;` $s \equiv$ `fail`
  - s `; fail` $\not\equiv$ `fail`

Defined Combinators

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

repeat(s)       = try(s; repeat(s))

while(c, s)     = if c then s; while(c,s) end

do-while(s, c) = s; if c then do-while(s, c) end
```

Traversal Strategies

Visiting All Subterms

- Syntax: `all(`*s*`)`
- Apply strategy s to all direct sub-terms

```
Plus(Int("14"),Int("3"))
stratego> all(!Var("a"))
Plus(Var("a"),Var("a"))
```

Traversal Strategies

Visiting All Subterms

- Syntax: `all(`s`)`
- Apply strategy s to all direct sub-terms

```
Plus(Int("14"),Int("3"))
stratego> all(!Var("a"))
Plus(Var("a"),Var("a"))
```

```
bottomup(s)  = all(bottomup(s)); s
topdown(s)   = s; all(topdown(s))
downup(s)    = s; all(downup(s)); s
alltd(s)     = s <+ all(alltd(s))
innermost(s) = bottomup(try(s; innermost(s)))
```

Traversal Strategies

Visiting All Subterms

- Syntax: `all(s)`
- Apply strategy *s* to all direct sub-terms

```
Plus(Int("14"),Int("3"))
stratego> all(!Var("a"))
Plus(Var("a"),Var("a"))
```

```
bottomup(s)  = all(bottomup(s)); s
topdown(s)   = s; all(topdown(s))
downup(s)    = s; all(downup(s)); s
alltd(s)     = s <+ all(alltd(s))
innermost(s) = bottomup(try(s; innermost(s)))
```

```
for-with-while = topdown(try(ForToWhile))
simple-expressions =
  innermost(Simplify <+ LiftNonAtomicArgument <+ Desugar
            <+ LetSplit)
```

Congruences

Data-type Specific Traversal

- Syntax: $c(s_1, \ldots, s_n)$
  for each n-ary constructor c
- Apply strategies to direct sub-terms of a c term

```
Add(Int("14"),Int("3"))
stratego> Add(!Var("a"), id)
Add(Var("a"),Int("3"))
```

```
mark-procedure-calls =
  rec mark(
    alltd(
      Program(mark)
      <+ For(id, id, id, mark)
      <+ If(id, mark, mark)
      <+ Var(id, id)
      <+ Block(mark)
      <+ !Proc(<Call(id,id)>)))
```

Demonstration

Demonstration

```
# cd tutorial/
```

Conclusion

Resources

- `www.stratego-language.org`
  - compiler, interpreter, documentation, tools
- `www.spoofax.org`
  - Eclipse-based editing environment.
- `www.program-transformation.org`
  - community wiki.
- `planet.stratego.org`
  - aggregation of Stratego developer web journals/logs.