Infrastructure for Program Transformation Systems

Stratego/XT Tutorial

Martin Bravenboer

Institute of Information & Computing Sciences
Utrecht University,
The Netherlands

January 24, 2005
Stratego
- Language for program transformation
- Suitable for implementing complete programs

XT
- Collection of Transformation Tools
- Infrastructure for implementing transformation systems
- Parsing, pretty-printing, interoperability

XT Orbit
- Language specific tools
- Java, C, C++, Octave, ...
Program Transformation Pipeline

http://www.strategoxt.org
Program Transformation Pipeline

Transformation systems
- Composition of tools
- Source → source
- Anything → anything

Transformation tools
- Input → output
- Executable files

Program

http://www.strategoxt.org
Infrastructure for Program Transformation Systems
Architecture of Stratego/XT

Syntax definition

Parse -> Tree

Transform -> Tree

Pretty-print -> Program

Program

http://www.strategoxt.org

Infrastructure for Program Transformation Systems
Architecture of Stratego/XT

Infrastructure for Program Transformation Systems

http://www.strategoxt.org
Architecture of Stratego/XT

Parser 100% generated

Language specific support for transformations generated

Basic pretty-printer 100% generated
Abstract Syntax of Programs

Parser: textual representation → abstract syntax tree

4 + f(5 * x)

Trees are represented as terms in the ATerm format

Plus(Int("4"), Call("f", [Mul(Int("5"), Var("x"))])

http://www.strategoxt.org
### ATerm Format

<table>
<thead>
<tr>
<th>Application</th>
<th>Void(), Call(t, t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>[], [t, t, t]</td>
</tr>
<tr>
<td>Tuple</td>
<td>(t, t), (t, t, t)</td>
</tr>
<tr>
<td>Integer</td>
<td>25</td>
</tr>
<tr>
<td>Real</td>
<td>38.87</td>
</tr>
<tr>
<td>String</td>
<td>&quot;Hello world&quot;</td>
</tr>
<tr>
<td>Annotated term</td>
<td>t{t, t, t}</td>
</tr>
</tbody>
</table>

- Exchange of structured data
- Efficiency through maximal sharing
- Binary encoding

**Stratego:** internal is external representation
⇒ divide systems into smaller, reusable tools
Syntax Definition in Stratego/XT

SDF – Syntax Definition Formalism

1. Declarative
   ▶ Important for code generation
   ▶ Completely define the syntax of a language

2. Modular
   ▶ Syntax definitions can be composed!

3. Context-free and lexical syntax
   ▶ No separate specification of tokens for scanner

4. Declarative disambiguation
   ▶ Priorities, associativity, follow restrictions

5. All context-free grammars
   ▶ Beyond LALR, LR, LL

http://www.strategoxt.org Infrastructure for Program Transformation Systems
Syntax Definition in Stratego/XT

SDF – Syntax Definition Formalism

1. Declarative
   ▶ Important for code generation
   ▶ Completely define the syntax of a language

2. Modular
   ▶ Syntax definitions can be composed!

3. Context-free and lexical syntax
   ▶ No separate specification of tokens for scanner

4. Declarative disambiguation
   ▶ Priorities, associativity, follow restrictions

5. All context-free grammars
   ▶ Beyond LALR, LR, LL

http://www.strategoxt.org
Infrastructure for Program Transformation Systems
Syntax Definition in Stratego/XT

SDF – Syntax Definition Formalism

1. **Declarative**
   - Important for code generation
   - Completely define the syntax of a language

2. **Modular**
   - Syntax definitions can be composed!

3. **Context-free and lexical syntax**
   - No separate specification of tokens for scanner

4. **Declarative disambiguation**
   - Priorities, associativity, follow restrictions

5. **All context-free grammars**
   - Beyond LALR, LR, LL

http://www.strategoxt.org

Infrastructure for Program Transformation Systems
1. **Declarative**
   - Important for code generation
   - Completely define the syntax of a language

2. **Modular**
   - Syntax definitions can be composed!

3. **Context-free and lexical syntax**
   - No separate specification of tokens for scanner

4. **Declarative disambiguation**
   - Priorities, associativity, follow restrictions

5. **All context-free grammars**
   - Beyond LALR, LR, LL

http://www.strategoxt.org - Infrastructure for Program Transformation Systems
1. **Declarative**
   - Important for code generation
   - Completely define the syntax of a language

2. **Modular**
   - Syntax definitions can be composed!

3. **Context-free and lexical syntax**
   - No separate specification of tokens for scanner

4. **Declarative disambiguation**
   - Priorities, associativity, follow restrictions

5. **All context-free grammars**
   - Beyond LALR, LR, LL
module Lexical
exports
  lexical syntax
  ...

module Expressions
imports Lexical
exports
  context-free syntax
  ...

module Main
imports Expressions
exports
  context-free start-symbols Exp
Lexical syntax is defined with ordinary productions.

```plaintext
module Lexical
exports
sorts Id IntConst BoolConst
lexical syntax
  [A-Za-z][A-Za-z0-9]* -> Id
  [0-9]+ -> IntConst
  "true" -> BoolConst
  "false" -> BoolConst
  [\r\n\t\ ] -> LAYOUT
  "//" ~[\n]* [\n] -> LAYOUT
```

- Even *context-free* lexical syntax is possible
- Avoid complex regular expressions

http://www.strategoxt.org  Infrastructure for Program Transformation Systems
Declaring reserved keywords: reject certain productions

**lexical syntax**
- "true" -> Id {reject}
- "false" -> Id {reject}

Longest match: follow restriction

**lexical restrictions**
- Id -/- [A-Za-z0-9]
- IntConst -/- [0-9]

Require layout after a keyword

**lexical restrictions**
- "if" -/- [A-Za-z0-9]
Declaring reserved keywords: reject certain productions

```
lexical syntax
 "true" -> Id {reject}
 "false" -> Id {reject}
```

Solves ambiguity between variable and boolean constant.

```
$ echo "true" | sglri -p Example.tbl
amb([Bool("true"),Var("true")])
```

Longest match: follow restriction

```
lexical restrictions
 Id   -/- [A-Za-z0-9]
 IntConst -/- [0-9]
```

Require layout after a keyword

```
lexical restrictions
 "if" -/- [A-Za-z0-9]
```

http://www.strategoxt.org
Infrastructure for Program Transformation Systems
Declaring reserved keywords: reject certain productions

**lexical syntax**

```
"true"  ->  Id  {reject}
"false" ->  Id  {reject}
```

Longest match: follow restriction

**lexical restrictions**

```
Id   -/-  [A-Za-z0-9]
IntConst  -/-  [0-9]
```

Rejects unintended split of identifier

```
$ echo "xinstanceof Foo" | sglri
InstanceOf(Var("x"),"Foo")
```

Require layout after a keyword

**lexical restrictions**

```
"if"  -/-  [A-Za-z0-9]
```
SDF: Disambiguation of Lexical Syntax

Declaring reserved keywords: reject certain productions

**lexical syntax**
- "true" -> Id {reject}
- "false" -> Id {reject}

Longest match: follow restriction

**lexical restrictions**
- Id -> [A-Za-z0-9]
- IntConst -> [0-9]

Require layout after a keyword

**lexical restrictions**
- "if" -> [A-Za-z0-9]

Rejects unintended split of keyword

$ echo "ifx then y" | sglri
IfThen(Var("x"), Var("y"))

http://www.strategoxt.org
Infrastructure for Program Transformation Systems
context-free syntax

Id -> Exp {cons("Var")}
IntConst -> Exp {cons("Int")}
BoolConst -> Exp {cons("Bool")}

"(" Exp ")" -> Exp {bracket}

Exp "+" Exp -> Exp {cons("Plus")}
Exp "-" Exp -> Exp {cons("Min")}
Exp "*" Exp -> Exp {cons("Mul")}
Exp "/" Exp -> Exp {cons("Div")}

Exp "&" Exp -> Exp {cons("And")}
Exp "|" Exp -> Exp {cons("Or")}
"!" Exp -> Exp {cons("Not")}

Id "(" {Exp ","}* ")" -> Exp {cons("Call")}

http://www.strategoxt.org
Infrastructure for Program Transformation Systems
SDF: Associativity of Operators

$ echo "1 + 2 + 3" | sglri -p Example.tbl
amb(
    Plus(Plus(Int("1"), Int("2")), Int("3")),
    Plus(Int("1"), Plus(Int("2"), Int("3")))
)

Declare associativity in attribute:

Exp "+" Exp -> Exp {left, cons("Plus")}
Exp ">" Exp -> Exp {non-assoc, cons("Gt")}

▶ left
▶ right
▶ assoc
▶ non-assoc

http://www.strategoxt.org Infrastructure for Program Transformation Systems
$ echo "1 + 2 * 3" | sglri -p Example.tbl
amb([
    Mul(Plus(Int("1"), Int("2")), Int("3")),
    , Plus(Int("1"), Mul(Int("2"), Int("3")))
])

can-free priorities

"!" Exp -> Exp
>
{left:
    Exp "*" Exp -> Exp
    Exp "/" Exp -> Exp
}
>
{left:
    Exp "+" Exp -> Exp
    Exp "-" Exp -> Exp
}
>
Exp "&" Exp -> Exp
>
Exp "|" Exp -> Exp

http://www.strategoxt.org Infrastructure for Program Transformation Systems
testsuite Expressions
topsort Exp

test simple addition
   "2 + 3" -> Plus(Int("2"), Int("3"))

test addition is left associative
   "1 + 2 + 3" -> Plus(Plus(_, _), _)

test > is not associative
   "1 > 2 > 3" fails

test
   file foo.exp succeeds

$ parse-unit -i exp.testsuite -p Example.tbl
  ...
SDF: Parsing

Parsing Technology

SDF relies on SGLR parsing

- **Scannerless**: no separate lexical analysis
- **Generalized LR**: allows ambiguities

SDF requires an extraordinary general parsing algorithm.

Generating a parser

- Collect SDF modules into a single syntax definition
  
  $ pack-sdf -i Example.sdf -o Example.def

- Generate a parse-table
  
  $ sdf2table -i Example.def -o Example.tbl -m Main

- Parse an input file
  
  $ sglri -i foo.exp -p Example.tbl

http://www.strategoxt.org

Infrastructure for Program Transformation Systems
Syntax definitions (grammars) define a set of strings

Transformation tools operate on trees

Tree grammars define the format of trees

Used to generate tools and libraries
regular tree grammar

start Exp

productions

Exp -> Int(IntConst)
    | Bool(BoolConst)
    | Not(Exp)
    | Mul(Exp, Exp)
    | Plus(Exp, Exp)
    | Call(Id, Exps)

Exps -> <nil>()
    | <cons>(Exp, Exps)

BoolConst -> <string>
IntConst -> <string>
Id -> <string>
Tools for Regular Tree Grammars

▶ Derive from SDF syntax definition

$ sdf2rtg -i Example.def -m Example -o Example.rtg

▶ Check the format of a tree

$ format-check --rtg Example.rtg

```
 martin@logistico:~> format-check --rtg Exp.rtg -i exp3.trm --vis
 error: cannot type Int(1)
     inferred types of subterms:
     typed 1 as <int>
 error: cannot type Div(1,Var("c"))
     inferred types of subterms:
     typed 1 as <int>
     typed Var("c") as Exp
Plus(
    Mul(Int(1), Var("a"))
 , Minus(Var("b"), Div(1, Var("c")))
)
 martin@logistico:~>  
```

▶ Generate tools and libraries

$ rtg2sig -i Example.rtg -o Example.str

http://www.strategoxt.org
Infrastructure for Program Transformation Systems
Code generators and source to source transformation systems need support for pretty-printing.

Stratego/XT: GPP (Generic Pretty-Printing)
- Box language
- Pretty-printer generation
- Different back-ends: abox2text, abox2html, abox2latex
Box Language

- Text formatting language
- Options for spacing, indenting
- ‘CSS for plain text’

H hs=x [ B B B ] → B B B

V vs=x is=y [ B B B ] → B

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

Other boxes: HV, ALT, KW, VAR, NUM, C

http://www.strategoxt.org
Example Box

\[
V \text{ is=2 [} \\
\quad H \text{ [KW["while"] } \text{ "a" KW["do"]]} \\
\quad V \text{ [} \\
\qquad V \text{ is=2 [} \\
\qquad\quad H \text{ hs=1 [KW["if"] } \text{ "b" KW["then"]]} \\
\qquad\quad H \text{ hs=0 ["foo()" ";"]} \\
\qquad\] \\
\quad KW["else"] \\
\quad V \text{ [V is=2 ["{" "..."} "]]} \\
\] \\
\]

while a do
  if b then
    foo();
  else
    {
      ...
    }
\]

http://www.strategoxt.org Infrastructure for Program Transformation Systems
Pretty-print Tables

- List of pretty-print rules
- Applied by constructor name (cons attribute)

Example Pretty-Print Table

```plaintext
[
  Var -- _1,
  Bool -- _1,
  Int -- _1,
  Mul -- _1 KW["*"] _2,
  Plus -- _1 KW["+"] _2,
  Min -- _1 KW["-" ] _2,
  Call -- _1 KW["(" ] _2 KW["]" ]
  Call.2:iter-star-sep -- _1 KW[",""]
]
```

- ast2abox accepts sequence of pretty-print tables
- Tables can be combined and reused

```bash
$ echo "1 + 2" | sglri -p Ex.tbl | ast2abox -p Ex.pp | abox2text
```

http://www.strategoxt.org
Infrastructure for Program Transformation Systems
Pretty-print table can be generated from SDF syntax definition (ppgen)
  ▶ Complete and correct (usually)
  ▶ Minimal formatting

Customization by hand for pretty result
  ▶ Tools for consistency checking and patching (pptable-diff)

Parentheses problem: parentheses inserter can be generated from SDF syntax definition (sdf2parenthesize).
Components of transformation system are executable tools
Standardized tool interface
ATerm exchange format

**XTC: Library for Tool Composition**

- Abstraction over *tool location*
  - XTC Repository
- Abstractions for *tool invocation*
  - xtc-transform
- Abstraction from *wiring*
  - Management of intermediate files
- Abstractions for *composing tools*
XT Orbit

- **Java**
  - High-quality syntax definition (1.5)
  - Handcrafted pretty-printer (1.5)
  - Disambiguation
  - Type-checker (work in progress)

- **C** (EPITA, France)
  - Syntax definition (C99)
  - Disambiguation

- **Octave**
  - Talk this afternoon

- **Prolog**
  - Syntax definition
  - Embedding of object languages (talk this afternoon)

- **BibTeX**
  - Syntax definition
  - Web services