Infrastructure for Program Transformation Systems
Master Course Program Transformation 2004-2005

Martin Bravenboer

Institute of Information & Computing Sciences
Utrecht University,
The Netherlands

February 10, 2005
Stratego
- Language for program transformation
- Suitable for implementing complete programs

XT
- Collection of Transformation (X) 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 Infrastructure for Program Transformation Systems
Program Transformation Pipeline

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

Transformation tools
- Input → output
- Executable files

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

http://www.strategoxt.org

Infrastructure for Program Transformation Systems
Infrastructure for Program Transformation Systems
Architecture of Stratego/XT

Parser 100% generated
Language specific support for transformations generated
Basic pretty-printer 100% generated

http://www.strategoxt.org Infrastructure for Program Transformation Systems
Trees are represented as terms in the ATerm format

```
Plus(Int("4"), Call("f", [Mul(Int("5"), Var("x"))]))
```
### ATerm Format

| Application | Void(), Call(t, t) |
| List        | [], [t, t, t]     |
| Tuple       | (t, t), (t, t, t) |
| Integer     | 25                |
| Real        | 38.87             |
| String      | "Hello world"    |
| Annotated term | t{t, t, t} |

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

*Structured Data*: comparable to XML  
*Stratego*: internal is external representation
Simple Expression Language

\[
\begin{align*}
Id & \rightarrow [a-z]+ \\
IntConst & \rightarrow [0-9]+ \\
Exp & \rightarrow Id \\
& \quad IntConst \\
& \quad Exp + Exp \\
& \quad Exp - Exp \\
& \quad Exp * Exp \\
& \quad Exp / Exp \\
& \quad ( Exp )
\end{align*}
\]
(a + n) / 2

Parse Trees

http://www.strategoxt.org  Infrastructure for Program Transformation Systems
Parse Trees and Abstract Syntax Trees

Text → Parse Tree → Abstract Syntax Tree

AsFix: ATerm language for parse trees
- Describes Applications of productions
- All characters of the input
  - Even whitespace and comments!
- Yield parse tree to text
  $ asfix-yield -i exp.asfix
  (a + n) / 2

Abstract Syntax Trees
- Remove literals, whitespace, comments
  $ implode-asfix -i exp.asfix
  Div(Plus(Var("a"), Var("n")), Int("2"))

http://www.strategoxt.org Infrastructure for Program Transformation Systems
Pretty-print an ATerm in a nice layout

$ pp-aterm -i foo.aterm

Usually applied at the end of a pipeline:

$ echo "foo([0], bar(1, 2), fred(3,4))" | ... | pp-aterm
foo(
    [0]
    , bar(1, 2)
    , fred(3, 4)
)
Ambiguity in Context-Free Grammars

- \( e_1 + e_2 * e_3 \)
  - \((e_1 + e_2) * e_3\)
  - \(e_1 + (e_2 * e_3)\)

- \( e_1 + e_2 + e_3 \)
  - \((e_1 + e_2) + e_3\)
  - \(e_1 + (e_2 + e_3)\)

- ++a
  - +(a)
  - ++ a?

- null
  - Keyword or identifier?

- if e1 then if e2 then e3 else e4
  - if e1 then (if e2 then e3) else e4
  - if e1 then (if e2 then e3 else e4)
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
module Lexical
exports
  lexical syntax
  ...

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

module Main
imports Expressions
exports
  context-free start-symbols Exp
## SDF: Parser Generation

### Modules and Definitions

- **SDF Module** (`.sdf`)
- **SDF Definition** (`.def`)

### Generating a parser

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

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

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

- Parse an input file (alternative)
  
  ```bash
  $ sglr -2 -i foo.exp -p Example.tbl | implode-asfix
  ```

[http://www.strategoxt.org](http://www.strategoxt.org)  
Infrastructure for Program Transformation Systems
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
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

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

Longest match: follow restriction

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

Require layout after a keyword

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

Rejects unintended split of keyword

```
$ echo "ifx then y" | sglri
IfThen(Var("x"),Var("y"))
```
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
$ 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")))
    ])
SDF: Priority of Operators

```bash
$ 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")))
])
```

classification

```plaintext
context-free priorities

 "!"  Exp -> Exp
 >  {  
     Exp "*" Exp -> Exp
     Exp "/" Exp -> Exp
   }
 >  {  
     Exp "+" Exp -> Exp
     Exp "-" Exp -> Exp
   }
 >  Exp "&" Exp -> Exp
 >  Exp "|" Exp -> Exp
```
$\text{echo } "1 + 2 - 3" \mid \text{sglri} \ -p \ \text{Example.tbl}$

```plaintext
amb(
    [Min(Plus(Int("1"), Int("2")), Int("3")),
    Plus(Int("1"), Min(Int("2"), Int("3")))
]
)
```

context-free priorities

```plaintext
"!"  Exp -> Exp
>
\{\texttt{left}:
    Exp "*"  Exp -> Exp
    Exp "/"  Exp -> Exp
}\n>
\{\texttt{left}:
    Exp "+"  Exp -> Exp
    Exp "-"  Exp -> Exp
}\n>
Exp "&"  Exp -> Exp
>
Exp "|"  Exp -> Exp
```
Parse-unit: Testing SDF Syntax Definitions

testsuite Expressions
topsort Exp

test simple addition
  "2 + 3" \rightarrow \text{Plus}(\text{Int}("2"), \text{Int}("3"))

test addition is left associative
  "1 + 2 + 3" \rightarrow \text{Plus}(\text{Plus}(\_\_, \_\_), \_\_)

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

test
  file foo.exp succeeds

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

- SDF requires an extraordinary general parsing algorithm.
- SDF relies on SGLR parsing
- **Scannerless**: no separate lexical analysis
  - Every character is a token
  - Context-dependent lexical syntax
- **Generalized LR**: allows ambiguities
  - All derivations
  - Produces a parse forest
  - Technique: fork LR parsers
- Advantage: **declarative** syntax definition
  - Excellent for code generation

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

Compare to DTD, W3C XML Schema, RELAX NG
regular tree grammar

start Exp

productions

Exp \rightarrow \text{Int(IntConst)}
  | \text{Bool(BoolConst)}
  | \text{Not(Exp)}
  | \text{Mul(Exp, Exp)}
  | \text{Plus(Exp, Exp)}
  | \text{Call(Id, Exps)}

Exps \rightarrow \langle\text{nil}\rangle()
  | \langle\text{cons}\rangle(Exp, Exps)

BoolConst \rightarrow \langle\text{string}\rangle
IntConst \rightarrow \langle\text{string}\rangle
Id \rightarrow \langle\text{string}\rangle
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

```
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")))
)
```

- Generate tools and libraries

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

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

Code generators and source to source transformation systems need support for pretty-printing.

If (Var("b"), ..., ...)

Stratego/XT: GPP (Generic Pretty-Printing)

- Box language
- Pretty-printer generation
- Different back-ends: abox2text, abox2html, abox2latex

http://www.strategoxt.org

Infrastructure for Program Transformation Systems
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 ]
```

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

http://www.strategoxt.org Infrastructure for Program Transformation Systems
while a do
  if b then
    foo();
  else
    {
      ...
    

Pretty-print Tables

- List of pretty-print rules
- Applied by constructor name (\texttt{cons} attribute)

Example Pretty-Print Table

\[
\begin{array}{l}
\text{Var} \quad \_1, \\
\text{Bool} \quad \_1, \\
\text{Int} \quad \_1, \\
\text{Mul} \quad \_1 \text{KW["*"]} \_2, \\
\text{Plus} \quad \_1 \text{KW["+"]} \_2, \\
\text{Min} \quad \_1 \text{KW["-"]} \_2, \\
\text{Call} \quad \_1 \text{KW["("] \_2 \text{KW["]")]}, \\
\text{Call.2:iter-star-sep} \quad \_1 \text{KW["","]}
\end{array}
\]

\texttt{ast2abox} accepts sequence of pretty-print tables

- Tables can be combined and reused

$\ echo \ "1 + 2" \mid \texttt{sglri} \ -p \ \texttt{Ex.tbl} \mid \texttt{ast2abox} \ -p \ \texttt{Ex.pp} \mid \texttt{abox2text}$
Pretty-printer Generation

- 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 (`sdf2parenthesizer`).

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

- Syntax definition
  - Parser generator
    - Parse table
      - Parse
        - Program
  - Tree grammar generator
    - Tree grammar
      - Transform
        - Tree
          - Program
  - Pretty-printer generator
    - Pretty-print table
      - Pretty-print
        - Program
XT Orbit

- **Java**
  - High-quality syntax definition (1.5)
  - Handcrafted pretty-printer (1.5)
  - Disambiguation
  - Type-checker
- **C (EPITA, France)**
  - Syntax definition (C99)
  - Disambiguation
- **Octave**
  - Parser
  - Type-checker
  - Compiler
- **Prolog**
  - Syntax definition
  - Embedding of object languages
- **BibTeX**
  - Syntax definition
  - Web services

http://www.strategoxt.org
Infrastructure for Program Transformation Systems