Spoofax: A Development Environment for Software Transformations

Karl Trygve Kalleberg
University of Bergen
Prerequisites

- Rudimentary programming skills
- Some familiarity with non-linear storytelling
- Basic knowledge of compilers is a plus
Act I

– wherein introductions are made
What is Spoofax?

• An interactive development environment
  – supports development of program transformations
  – based on the Stratego transformation language

• A set of plugins for Eclipse
  – editor – for writing Stratego programs
  – help system – for reading the Stratego manual (!)
  – interpreter – for executing Stratego programs
  – parser – for parsing any language

• An extensible system
  – Stratego scripts may extend the environment
What is Eclipse?

• A framework for application development
  – based on Java
  – open-source

• A development platform
  – for Java, C/C++, Python, PHP, ...

• A pluggable framework
  – everything is a plugin
    • (including the plugin loader)

• Heavily sponsored
  – BEA, Borland, IBM, Oracle, Nokia, Sybase, Intel, Motorola, ...
What is program transformation?

• “The act of changing one program into another”
  – using transformation programs
  – interactively or (fully) automatically
  – often organised as a compiler pipeline

• Useful for
  – language implementation
    • compilers, interpreters, extensible languages, embedded languages
  – program analysis
    • defect checking, code smell sniffing, security analysis, program validation
  – program transformation
    • refactoring, reengineering, language translation, optimisation
  – code generation
    • from UML models, from specifications, in software factories
A transformation pipeline
Why abstraction?

\[
x := 1 + 2
\]

Concrete Syntax Tree
Why abstraction?

\[
x := 1 + 2
\]

\[\downarrow\]

Abstract Syntax Tree

\[
\text{Assign(Var(“x”), Plus(Int(“1”), Int(“2”)))}
\]
A transformation pipeline
Interlude
– an example application
Example language extension: JSwul

class SWULMe {
    public static void main(String[] args) {
        JFrame frame = frame {
            title = "Welcome!"
            content = panel of border layout {
                center = label { text = "Hello World" }
                south = panel of grid layout {
                    row = {
                        button { text = "cancel" }
                        button { text = "ok" }
                    }
                }
            }
        }
    }
}
Step 0: What we have

- SWULMe program
- Syntax definition
  - Parser generator
    - Parse table
    - Parse
  - Signature generator
    - Signature
    - Transformation
    - Pretty-print
  - Pretty-print generator
    - Pretty-print table
    - Program
Step 1: What we want

- Syntax definition
  - Parser generator
    - Parse table
      - Parse
  - Signature generator
    - Signature
    - Transformation
  - Pretty-print generator
    - Pretty-print table
    - Pretty-print

- SWULMe program
  - SwulMe program
A pure Java program

class SWULMe {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Welcome!");
        JPanel mainPanel = new JPanel();
        BorderLayout borderLayout = new BorderLayout();
        JLabel helloLabel = new JLabel("Hello World");
        JPanel southPanel = new JPanel();
        JButton cancelButton = new JButton("cancel");
        JButton okButton = new JButton("ok");

        southPanel.setLayout(new GridLayout(1,2));
        southPanel.add(cancelButton);
        southPanel.add(okButton);

        mainPanel.setLayout(borderLayout);
        mainPanel.add(southPanel, BorderLayout.SOUTH);
        mainPanel.add(helloLabel, BorderLayout.CENTER);

        frame.setContentPane(mainPanel);
    }
}
Step 2: We need a JSwul grammar

- JSwul.sdf
  - syntax definition
  - parser generator
  - parser
  - SWULMe program

- JSwul.tbl
  - parse table

- signature generator
  - transformation
  - pretty-print generator
  - pretty-print table
  - SwulMe program

- pretty-print generator
  - pretty-print

- signature
  - JSwul.tbl
SWUL grammar definition

module Swul
exports context-free start-symbols Component

sorts Component ComponentType ComponentProps ComponentPropValues ComponentPropType
context-free syntax

ComponentType ComponentProps? -> Component

"panel" -> ComponentType
"button" -> ComponentType
"border" "layout" -> ComponentType
"grid" "layout" -> ComponentType
"frame" -> ComponentType

{" ComponentProp* "}" -> ComponentProps
"of" Component -> ComponentProps

ComponentPropType "=" ComponentPropValues -> ComponentProp
{" Component* "}" -> ComponentPropValues

"content" -> ComponentPropType
"title" -> ComponentPropType
"row" -> ComponentPropType
"south" -> ComponentPropType
"center" -> ComponentPropType
"border" -> ComponentPropType
Java + SWUL = JSwul

module JSwul
imports Java-15-Prefixixed Swul-Prefixixed
exports
  sorts JavaExpr SwulComponent JavaBlock
  context-free syntax
  SwulComponent -> JavaExpr
Java + SWUL = JSwul

module JSwul
imports Java-15-Prefixed Swul-Prefixed

exports
sorts JavaExpr SwulComponent JavaBlock
context-free syntax

SwulComponent -> JavaExpr

JavaExpr       -> SwulComponent
JavaBlock      -> SwulComponent
Step 3: Parsing the SWUL language
Step 4: Defining JSwul to Stratego

- **JSwul.sdf** (syntax definition)
  - parser generator
  - **JSwul.tbl** (parse table)
  - parse
  - SWULMe program

- **signature**
  - signature generator
  - transformation

- **pretty-print table**
  - pretty-print
  - SwulMe program

- **pretty-printer generator**
Step 5: Transformation!

- **SWULMe program**
- **JSwul.sdf (syntax definition)**
- **JSwul.tbl (parse table)**
- **SWULMe program**
- **signature generator**
- **signature**
- **parse**
- **pretty-print table**
- **pretty-print**
- **pretty-printer generator**

Diagram: Flowchart showing the transformation process from SWULMe program to SwulMe program, involving parser generator, signature generator, and pretty-printer generator.
Step 6: Transformation!

- JSwul.sdf syntax definition
  - parser generator
  - JSwul.tbl parse table
  - parse
  - SWULMe program

- signature generator
  - transformation

- pretty-printer generator
  - pretty-print table
  - pretty-print
  - SwulMe program
Step 7: Deriving a pretty printer

- **SWULMe program**
- **JSwul.sdf** (syntax definition)
- **JSwul.tbl** (parse table)
- **parser generator**
  - **parse**
  - **SWULMe program**
- **signature generator**
  - **signature**
  - **JSwul.tbl** (parse table)
- **pretty-print generator**
  - **Swul.pp** (prettyprint table)
  - **SwulMe program**
- **transformation**
  - **JSwul.sdf** (syntax definition)
  - **signature generator**
Step 8: Are we there yet? – Yes!

SWULMe program

JSwul.sdf syntax definition

parser generator

JSwul.tbl parse table

signature generator

signature

transform-ation

pretty-print

JSwul.pp prettyprint table

pretty-printer generator

SwulMe program
class SWULMe {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Welcome!");
        JPanel mainPanel = new JPanel();
        JLabel helloLabel = new JLabel("Hello World");
        JButton cancelButton = new JButton("cancel");
        JButton okButton = new JButton("ok");

        southPanel.setLayout(new GridLayout(1,2));
        southPanel.add(cancelButton);
        southPanel.add(okButton);

        mainPanel.setLayout(borderLayout);
        mainPanel.add(southPanel, BorderLayout.SOUTH);
        mainPanel.add(helloLabel, BorderLayout.CENTER);
        frame.setContentPane(mainPanel);
    }
}
Meanwhilst, back at the ranch...
What is Stratego?

• A scripting language for program transformation
  – based on term rewriting...
    • express small, incremental changes to the AST
  – .. and strategic programming
    • separate data processing from data traversal

• Provides powerful domain-specific features
  – rewrite rules
  – application strategies
  – concrete syntax patterns
  – XT – a collection of reusable transformation components
rules

Simplify:
   If(Boolean(True), th, ls) -> th

Simplify:
   Plus(e, Int("0")) -> e

Simplify:
   Plus(Int(x), Int(y)) -> Int(z)
   where <addS> (x, y) => z

Simplify:
   Plus(Int(x), Int(y)) -> Int(<addS> (x, y))
Matching of patterns

- Pattern

\(?Plus(x, y)\)

- Tree (term)

\(Plus(\text{Int}("1"), \text{Int}("2"))\)
Matching of patterns

- Pattern

\(?\text{Plus}(x, y)\)

- Tree (term)

\(\text{Plus}(\text{Int}("1"), \text{Int}("2"))\)

\[x = \text{Int}("1"), \ y = \text{Int}("2")\]
Rewrite rules with concrete syntax


cases

Simplify:

\[
\begin{align*}
&\text{if (true) then } \neg th \text{ else } \neg ls/ ]\to th \\
&\neg e + 0 \to e \\
&\neg x + \neg y \to \neg z \\
\text{where } <addS> (x, y) \to z \\
&\neg x + \neg y \to \text{Int}(<addS> (x, y))
\end{align*}
\]
Application Strategies

- **Apply rule to tree (term)**
  - \(<R>\ t\)

- **Left choice**
  - \(s_1 <+ s_2\)

- **Guarded choice**
  - \(s_1 < s_2 + s_3\)

- **Sequence**
  - \(s_1 ; s_2\)

- **Subtree(term) traversal**
  - \(\text{all}(s)\) – all subterms
  - \(\text{one}(s)\) – one subterm

- **Generic traversals**

```plaintext
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))
```
Assimilation of SWUL

\[
\text{swul-assimilate} = \\
\text{class-declaration} \\
\text{<+ class-initializer} \\
\text{<+ class-method} \\
\text{<+ swul-expression} \\
\text{<+ all(swul-assimilate)}
\]
Assimilation of SWUL

```
swul-assimilate =
    class-declaration
  <+ class-initializer
  <+ class-method
  <+ swul-expression
  <+ all(swul-assimilate)
```

```
swul-expression = ... ; SwulAs-Component ... ; ...
```
Assimilation of SWUL

```plaintext
swul-assimilate =
  class-declaration
  <+ class-initializer
  <+ class-method
  <+ swul-expression
  <+ all(swul-assimilate)

swul-expression = ... ; SwulAs-Component ... ; ...

SwulAs-Component = SwulAs-Container <+ SwulAs-JExpr
SwulAs-Container = SwulAs-JComponent
SwulAs-JComponent = SwulAs-JPanel <+ SwulAs-JToolBar <+ SwulAs-JSplitPane <+ ..

SwulAs-JPanel:
  swul c | [ panel { ps* } ] | {x} -> expr | [ { | x = new JPanel(); bstm* | x | } ] |
  where <map(SwulAs-JPanelProp(|x|) > ps* => bstm*

SwulAs-JPanelProp(|x|) = ... <+ SwulAs-ContainerProp(|x|) <+ ...

SwulAs-ContainerProp(|x|):
  swul cp | [ layout = c ]| -> bstm | [ x.setLayout( e ) ; ] |
  where <SwulAs-LayoutManager(|x|) > c => e
```
Act II
– the showdown
Development aids for Stratego

- Syntax highlighting
  - SDF and Stratego
- Content completion
  - rules, strategies, constructors, modules
- Navigation
  - to rule, strategy, constructor, module definitions
- Project building
  - incremental, automatic
- Code outline
  - definitions in module
- Help
  - bundled Stratego/XT manual
- Tasks
  - FIXME, TODO
- Extensible
  - Stratego scripts
Demo
Epilogue
Conclusion

• Program transformation techniques are widely applicable
  – software processing
  – language development
  – program analysis

• Program transformation has never been less difficult!
  – Spoofax + Stratego/XT = transformation development platform
  – www.spoofax.org
  – www.strategoxt.org
    • Works on and for Linux, Unix, OSX and Windows
    • Licensed under the LGPL
    • Tutorials, examples and API docs online
The End.
SWUL grammar definition (with cons)

module Swul
exports context-free start-symbols Component

sorts Component ComponentType ComponentProps ComponentPropValues ComponentPropType

context-free syntax

ComponentType ComponentProps? -> Component {cons("Component")}

"panel" -> ComponentType {cons("JPanel")}
"button" -> ComponentType {cons("JButton")}
"border" "layout" -> ComponentType {cons("BorderLayout")}
"grid" "layout" -> ComponentType {cons("GridLayout")}
"frame" -> ComponentType {cons("JFrame")}

{" ComponentProp* "}" -> ComponentProps {cons("ComponentProps")}
"of" Component -> ComponentProps {cons("DefaultPropsWithComponent")}

ComponentPropType "=" ComponentPropValues -> ComponentProp {cons("ComponentProp")}
{" Component* "}" -> ComponentPropValues {cons("ComponentPropMultiValue")}

"content" -> ComponentPropType {cons("Content")}
"title" -> ComponentPropType {cons("Title")}
"row" -> ComponentPropType {cons("Row")}
"south" -> ComponentPropType {cons("South")}
"center" -> ComponentPropType {cons("Center")}
"border" -> ComponentPropType {cons("Border")}

Spoofax Architecture

```
org.spoofax
  └── org.spoofax.editor
    └── org.spoofax.jsglr
  └── org.spoofax.help
  └── org.spoofax.interpreter
    └── org.spoofax.interpreter.adapter.aterm
  └── org.spoofax.aterm
```