Growing a Syntax

Eric Allen[1], Ryan Culpepper[2], **Janus Dam Nielsen**[3], Jon Rafkind[4], and Sukyoung Ryu[1]

[1]Sun Microsystems
[2]Northeastern University
[3]**Aarhus University**
[4]University of Utah

Foundations of Object-Oriented Languages, 2009

Outline

Outline

The Fortress Programming Language

- A multicore language for scientists and engineers

The Fortress Programming Language

- A multicore language for scientists and engineers
- Run your whiteboard in parallel!

The Fortress Programming Language

- A multicore language for scientists and engineers
- Run your whiteboard in parallel!

$$v_{norm} = v / \| v \|$$
$$c_{ij} = \sum_{k \leftarrow 0:n} a_{ik}\, b_{kj}$$
$$C = A \cup B$$
$$y = 3x \sin x \cos 2x \log \log x$$

The Fortress Programming Language

- A multicore language for scientists and engineers
- Run your whiteboard in parallel!

$$v_{norm} = v / \| v \|$$

$$c_{ij} = \sum_{k \leftarrow 0:n} a_{ik} \, b_{kj}$$

$$C = A \cup B$$

$$y = 3x \sin x \cos 2x \log \log x$$

The Fortress Programming Language

- A multicore language for scientists and engineers
- Run your whiteboard in parallel!

$$v_{norm} = v / \| v \|$$
$$c_{ij} = \sum_{k \leftarrow 0:n} a_{ik} b_{kj}$$
$$C = A \cup B$$
$$y = 3x \sin x \cos 2x \log \log x$$

- Growing a Language
  Guy L. Steele Jr., keynote talk OOPSLA 1998
  Higher-Order and Symbolic Computation 12, 221-236 (1999)

Outline

Growing a Language

"So I think the sole way to win is to plan for growth with help from the users."

Guy L. Steele Jr.
keynote talk, OOPSLA 1998;
Higher-Order and Symbolic Computation 12, 221-236 (1999)

Design Strategy

Consider how a proposed language feature might be provided by a library rather than building features directly into the compiler.

This requires control over both syntax and semantics, not just the ability to add new functions and methods.

Outline

XML in Fortress

```
x = <html xmlns = "http://www.w3.org/1999/xhtml">
      <title> Project Fortress </title>
      <body/>
    </html>
```

<body/>.*hasElements*

x.children

x.attributes

- XML literals in Fortress
- Seamless integration
- DOM operations on values

XML in Fortress

$$x = \langle \text{html}\ \ \texttt{xmlns} = \text{“}\texttt{http://www.w3.org/1999/xhtml}\text{”}\rangle$$
$$\langle \text{title}\rangle\ \text{Project Fortress}\ \langle/\text{title}\rangle$$
$$\langle \text{body}/\rangle$$
$$\langle/\text{html}\rangle$$

$$\langle \text{body}/\rangle.\mathit{hasElements}$$
$$x.\mathit{children}$$
$$x.\mathit{attributes}$$

- XML literals in Fortress
- Seamless integration
- DOM operations on values

XML in Fortress

$x = $ <html `xmlns` = “`http://www.w3.org/1999/xhtml`”>
<title> Project Fortress </title>
<body/>
</html>

<body/>.*hasElements*
x.*children*
x.*attributes*

- XML literals in Fortress
- Seamless integration
- DOM operations on values

Desugaring XML

The XML literal:

```
<html xmlns = "http://www.w3.org/1999/xhtml">
  <title> Project Fortress </title>
  <body/>
</html>
```

Desugars to:

```
Element(
 Header("html",
  ⟨Attribute("xmlns", "http://www.w3.org/1999/xhtml")⟩),
 ⟨Element(
   Header("title", ⟨⟩),
   ⟨CData("Project Fortress")⟩, "title"),
  Element(Header("body", ⟨⟩), ⟨⟩, "body")⟩, "html")
```

Basic XML DOM Structure in Fortress

```
object Element(info : Header, contents : List⟦Content⟧,
                    endTag : String) extends Content
  getter name() : String
  getter hasElements() : Boolean
  getter children() : List⟦Element⟧
  getter cdata() : CData
  getter attributes() : List⟦Attribute⟧
  getter toXml() : String
end
trait Content end
object CData(c : String) extends Content end
object Header(startTag : String, attributes : List⟦Attribute⟧) end
object Attribute(key : String, val : String) end
```

Desugaring XML

- Design Strategy
  Consider how a proposed language feature might be provided by a library rather than building features directly into the compiler
- XML desugaring is provided by syntactic abstraction in a library

Outline

Syntactic Abstraction Goals

- New syntax indistinguishable from core syntax
- Similar syntax for definition/use of a language extension
- Composition of independent language extensions
- Expansion into other language extensions
- Mutually recursive definition of language extensions

Outline

Grammar of XML Literals in Fortress

$$
\begin{array}{l}
\texttt{grammar}\ \text{xml}\ \texttt{extends}\ \{\,\text{Expression}, \text{Symbols}\,\} \\
\quad \text{Expr}\ |{:}{=}\ \ x : \text{XExpr}\ \Rightarrow \lhd[\, x \,]\!\rhd \\
\quad \text{XExpr}\ \ ::= \\
\qquad b : \text{startTag}\ c : \text{XmlContent}\ e : \text{endTag}\ \Rightarrow \lhd[\,\text{Element}(b, c, e)\,]\!\rhd \\
\quad \mid\ b : \text{startTag}\ e : \text{endTag}\ \Rightarrow \lhd[\,\text{Element}(b, \langle\,\rangle, e)\,]\!\rhd \\
\quad \text{XmlContent}\ \ ::= \\
\qquad s : \text{XmlIdentifier}\ \Rightarrow \lhd[\,\langle\,\text{CData}(s)\,\rangle\,]\!\rhd \\
\quad \mid\ \{x : \text{XExpr}\ \mathit{SPACE}\}+\ \Rightarrow \lhd[\,\langle\, x^{**}\,\rangle\,]\!\rhd \\
\quad \text{startTag}\ \ ::= \\
\qquad < s : \text{XmlIdentifier}\ \{a : \text{Attribute}\ \mathit{SPACE}\}^{*} > \ \Rightarrow \lhd[\,\mathit{Header}(s, \langle\, a^{**}\,\rangle\,]\!\rhd \\
\quad \text{endTag}\ \ ::= \\
\qquad </\ s : \text{XmlIdentifier}>\ \Rightarrow \lhd[\, s \,]\!\rhd \\
\texttt{end}
\end{array}
$$

Examples of Goals

- New syntax indistinguishable from core syntax

  $<\text{body}\ /> \ .hasElements$

- Similar syntax for definition/use of a language extension

  $< s : \text{XmlIdentifier}\ \{a : \text{Attribute}\ SPACE\}^*\ /> \ \Rightarrow\ \ldots$

  $<\text{html}\ \ \texttt{xmlns} = \text{“}\texttt{http://www.w3.org/1999/xhtml}\text{”}/>$

Examples of Goals

- Composition of independent language extensions

  $\texttt{grammar}\ \mathrm{xml}\ \texttt{extends}\ \{\, \mathrm{Expression}, \mathrm{Symbols} \,\}$

- Expansion into other language extensions

  $\ldots \Rightarrow \lhd[<\mathrm{body} \,/>]\rhd$

- Mutually recursive definition of language extensions

Outline

Syntax Normalization

- Parsing stage - transforms a source program (in text) into a *parsed program* (in node expression)

- Transformation stage - transforms the parsed program into a program in executable core Fortress AST

Syntax Normalization

- Parsing stage - transforms a source program (in text) into a *parsed program* (in node expression)
  - First step
    parses the macro definition into an intermediate form to generate a parser that recognizes the new syntax
  - Second step
    uses the generated parser to parse a source program using the new syntax
- Transformation stage - transforms the parsed program into a program in executable core Fortress AST

Node Expressions - Intermediate Representation

$$
\begin{array}{lcl}
\mathit{NodeExpr} & ::= & \mathit{PatternVar} \\
& | & \mathit{Transformer}\ (\overline{NodeExpr}) \\
& | & \mathit{NodeConstructor}(\overline{NodeExpr}) \\
& | & \mathrm{Ellipses}\ (NodeExpr, \overline{NodeExpr}) \\
& | & \texttt{case}\ \mathit{PatternVar}\ \texttt{of} \\
& & \quad \mathrm{Empty} \Rightarrow \mathit{NodeExpr} \\
& & \quad \mathrm{Cons}(\mathit{PatternVar}, \mathit{PatternVar}) \Rightarrow \mathit{NodeExpr} \\
& & \texttt{end}
\end{array}
$$

Transformation: Evaluation of Node Expressions

- Pattern variables are substituted by the corresponding values
- Transformers are replaced with their bodies, substituting pattern variables along the way
- Core Fortress AST nodes transform their arguments
- Case expression match its input to constructors Empty and Cons, and invokes the corresponding transformer

Summary

- Fortress is a growable language
- Syntactic abstraction supports language growth
- Implementation is available as part of the open-source Fortress reference implementation at: `http://projectfortress.sun.com`

Evaluation of Node Expressions (1)

[Pattern Variable]

$$\frac{\Gamma(x) = v}{\Upsilon, \Gamma \vdash x \rightarrow \Upsilon, \Gamma \vdash v}$$

[Node Constructor]

$$\frac{\Upsilon, \Gamma \vdash \overline{n} \rightarrow \Upsilon, \Gamma \vdash \overline{n'}}{\Upsilon, \Gamma \vdash c(\overline{n}) \rightarrow \Upsilon, \Gamma \vdash c(\overline{n'})}$$

Evaluation of Node Expressions (2)

[Macro Invocation Arguments]

$$\frac{\Upsilon, \Gamma \vdash \overline{n} \rightarrow \Upsilon, \Gamma \vdash \overline{n'}}{\Upsilon, \Gamma \vdash t(\overline{n}) \rightarrow \Upsilon, \Gamma \vdash t(\overline{n'})}$$

[Macro Invocation]

$$\frac{\Upsilon(t) = t\,\overline{x}.n}{\Upsilon, \Gamma \vdash t(\overline{v}) \rightarrow \Upsilon, \Gamma\left[\overline{x} \mapsto \overline{v}\right] \vdash n}$$

Evaluation of Node Expressions (3)

[Case Empty]

$$\frac{\Upsilon, \Gamma \vdash x \rightarrow \Upsilon, \Gamma \vdash \text{Empty}}{\begin{array}{l}\Upsilon, \Gamma \vdash \texttt{case}\ x\ \texttt{of} \quad \rightarrow \Upsilon, \Gamma \vdash n_1 \\ \qquad \text{Empty} \Rightarrow n_1 \\ \qquad \text{Cons}(x_1, x_2) \Rightarrow n_2 \\ \quad \texttt{end}\end{array}}$$

[Case Cons]

$$\frac{\Upsilon, \Gamma \vdash x \rightarrow \Upsilon, \Gamma \vdash \text{Cons}(v_1, v_2)}{\begin{array}{l}\Upsilon, \Gamma \vdash \texttt{case}\ x\ \texttt{of} \quad \rightarrow \Upsilon, \Gamma\,[x_1 \mapsto v_1]\,[x_2 \mapsto v_2] \vdash n_2 \\ \qquad \text{Empty} \Rightarrow n_1 \\ \qquad \text{Cons}(x_1, x_2) \Rightarrow n_2 \\ \quad \texttt{end}\end{array}}$$

Evaluation of Node Expressions (4)

[Ellipses First]

$$\frac{|\overline{v'}| + 1 = i \leq size(n) \qquad x_j \in PV(n) \qquad |\Gamma(x_j)| > 1 \\ \Gamma' = \Gamma\left[\overline{x_j} \mapsto \overline{\Gamma(x_j).nth(i)}\right] \qquad \Upsilon, \Gamma' \vdash n'' \rightarrow \Upsilon, \Gamma' \vdash n'''}{\Upsilon, \Gamma \vdash \text{Ellipses}(n, \overline{v'}n'') \rightarrow \Upsilon, \Gamma \vdash \text{Ellipses}(n, \overline{v'}n''')}$$

[Ellipses Middle]

$$\frac{|\overline{v'}| + 1 = i - 1 < size(n) \qquad x_j \in PV(n) \qquad |\Gamma(x_j)| > 1 \\ \Gamma' = \Gamma\left[\overline{x_j} \mapsto \overline{\Gamma(x_j).nth(i)}\right] \qquad \Upsilon, \Gamma' \vdash n \rightarrow \Upsilon, \Gamma' \vdash n'''}{\Upsilon, \Gamma \vdash \text{Ellipses}(n, \overline{v'}v'') \rightarrow \Upsilon, \Gamma \vdash \text{Ellipses}(n, \overline{v'}v''n''')}$$

[Ellipses Last]

$$\frac{|\overline{v'}| + 1 = size(n)}{\Upsilon, \Gamma \vdash \text{Ellipses}(n, \overline{v'}v'') \rightarrow \Upsilon, \Gamma \vdash \overline{v'}v''}$$