Project Fortress:

A Multicore Language for Scientists and Engineers

Sukyoung Ryu

Department of Computer Science
Korea Advanced Institute of Science and Technology

December 13, 2009

Project Fortress

- A multicore language for scientists and engineers

Project Fortress

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

Project Fortress

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

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

$$\sum_{k \leftarrow 1:n} a_k \, x^k$$

$$C = A \cup B$$

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

Project Fortress

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

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

$$\sum_{k \leftarrow 1:n} a_k \, x^k$$

$$C = A \cup B$$

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

Project Fortress

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

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

$$\sum_{k \leftarrow 1:n} a_k \, x^k$$

$$C = A \cup B$$

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

- "Growing a Language"

  Guy L. Steele Jr., keynote talk, OOPSLA 1998

Project Fortress

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

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

$$\sum_{k \leftarrow 1:n} a_k \, x^k$$

$$C = A \cup B$$

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

- "Growing a Language"

  Guy L. Steele Jr., keynote talk, OOPSLA 1998

- Parsing syntax, formal semantics, and safety

Project Fortress

- Fortress is a growable, mathematically oriented, parallel programming language for scientific applications.
- Started under Sun/DARPA HPCS program, 2003–2006.
- Fortress is now an open-source project with international participation.
- The Fortress 1.0 release (March 2008) synchronized the specification and implementation.
- Moving forward, we are growing the language and libraries and developing a compiler.

Mathematical Syntax

Mathematical Syntax

Integrated mathematical and object-oriented notation

- Supports a stylistic spectrum that runs from Fortran to Java™—and sticks out at both ends!
  - More conventionally mathematical than Fortran
    - Compare `a*x**2+b*x+c` and $a\,x^2 + b\,x + c$
  - More object-oriented than Java
    - Multiple inheritance
    - Numbers, booleans, and characters are objects
  - To find the size of a set S: either $|S|$ or $S.size$
    - If you prefer $\#S$, defining it is a one-liner.

Mathematical Syntax Using Unicode

- Full Unicode character set available for use, including mathematical operators and Greek letters:

$\times \quad \div \quad \oplus \quad \ominus \quad \otimes \quad \oslash \quad \odot \quad \approx \quad \alpha \quad \beta \quad \gamma \quad \delta$

$\boxplus \quad \boxminus \quad \boxtimes \quad \leftrightarrow \quad \wedge \quad \vee \quad \equiv \quad \not\equiv \quad \epsilon \quad \zeta \quad \eta \quad \theta$

$\leq \quad \geq \quad \sum \quad \prod \quad \prec \quad \preccurlyeq \quad \succcurlyeq \quad \succ \quad \iota \quad \kappa \quad \lambda \quad \mu$

$\cap \quad \cup \quad \uplus \quad \subset \quad \subseteq \quad \supseteq \quad \supset \quad \in \quad \xi \quad \pi \quad \rho \quad \sigma$

$\sqcap \quad \sqcup \quad \sqsubset \quad \sqsubseteq \quad \sqsupseteq \quad \sqsupset \quad \neg \quad \notin \quad \phi \quad \chi \quad \psi \quad \omega$

$\lfloor \quad \rfloor \quad \lceil \quad \rceil \quad \langle \quad \rangle \quad \curlywedge \quad \curlyvee \quad \Gamma \quad \Theta$ and so on

- Use of "funny characters" is under the control of libraries (and therefore users)

Mathematical Syntax Example

NAS NPB 1 Specification

$z = 0$
$r = x$
$\rho = r^T r$
$p = r$
DO $i = 1,25$
$q = A\,p$
$\alpha = \rho / (p^T q)$
$z = z + \alpha\,p$
$\rho_0 = \rho$
$r = r - \alpha\,q$
$\rho = r^T r$
$\beta = \rho / \rho_0$
$p = r + \beta\,p$
ENDDO
compute residual norm explicitly: $\|r\| = \|x - A\,z\|$

Mathematical Syntax Example

NAS NPB 2.3 Serial Code in Fortran

```
do j=1,naa+1
   q(j) = 0.0d0
   z(j) = 0.0d0
   r(j) = x(j)
   p(j) = r(j)
   w(j) = 0.0d0
enddo
sum = 0.0d0
do j=1,lastcol-firstcol+1
   sum = sum + r(j)*r(j)
enddo
rho = sum
do cgit = 1,cgitmax
   do j=1,lastrow-firstrow+1
      sum = 0.d0
      do k=rowstr(j),rowstr(j+1)-1
         sum = sum + a(k)*p(colidx(k))
      enddo
      w(j) = sum
   enddo
   do j=1,lastcol-firstcol+1
      q(j) = w(j)
   enddo
   do j=1,lastcol-firstcol+1
      w(j) = 0.0d0
   enddo
   sum = 0.0d0
   do j=1,lastcol-firstcol+1
      sum = sum + p(j)*q(j)
   enddo
   d = sum
   alpha = rho / d
   rho0 = rho
   do j=1,lastcol-firstcol+1
      z(j) = z(j) + alpha*p(j)
      r(j) = r(j) - alpha*q(j)
   enddo
   sum = 0.0d0
   do j=1,lastcol-firstcol+1
      sum = sum + r(j)*r(j)
   enddo
   rho = sum
   beta = rho / rho0
   do j=1,lastcol-firstcol+1
      p(j) = r(j) + beta*p(j)
   enddo
enddo
do j=1,lastrow-firstrow+1
   sum = 0.d0
   do k=rowstr(j),rowstr(j+1)-1
      sum = sum + a(k)*z(colidx(k))
   enddo
   w(j) = sum
enddo
do j=1,lastcol-firstcol+1
   r(j) = w(j)
enddo
sum = 0.0d0
do j=1,lastcol-firstcol+1
   d   = x(j) - r(j)
   sum = sum + d*d
enddo
d = sum
rnorm = sqrt( d )
```

Mathematical Syntax Example

NAS NPB 1 Specification

$z = 0$
$r = x$
$\rho = r^T r$
$p = r$
DO $i = 1, 25$
$q = A\,p$
$\alpha = \rho / (p^T q)$
$z = z + \alpha\,p$
$\rho_0 = \rho$
$r = r - \alpha\,q$
$\rho = r^T r$
$\beta = \rho / \rho_0$
$p = r + \beta\,p$
ENDDO
compute residual norm explicitly: $\|r\| = \|x - A\,z\|$

in Fortress

$z : \mathrm{Vec} := 0$
$r : \mathrm{Vec} := x$
$p : \mathrm{Vec} := r$
$\rho : \mathrm{Elt} := r^{\mathrm{T}} r$
for $j \leftarrow$ **seq**$(1 : cgit_{\max})$ **do**
$q = A\,p$
$\alpha = \dfrac{\rho}{p^{\mathrm{T}} q}$
$z := z + \alpha\,p$
$r := r - \alpha\,q$
$\rho_0 = \rho$
$\rho := r^{\mathrm{T}} r$
$\beta = \dfrac{\rho}{\rho_0}$
$p := r + \beta\,p$
end
$(z\,,\ \|x - A\,z\|)$

Mathematical Syntax by 'Fortify'

- Emacs-based code formatter
- Fortress programs can be typed on ASCII keyboards.
- Code automatically formatted for processing by LaTeX

$sum: \mathbb{R}64 := 0$
`for` $k \leftarrow 1:n$ `do`
  $a_k := (1-\alpha)b_k$
  $sum \mathrel{+}= c_k\, x^k$
`end`

All code on these slides was formatted by this tool.

Mathematical Syntax by 'Fortify'

- Emacs-based code formatter
- Fortress programs can be typed on ASCII keyboards.
- Code automatically formatted for processing by LaTeX

sum: $\mathbb{R}64$:= 0
for $k \leftarrow 1 : n$ do
  $a_k := (1 - \alpha) b_k$
  sum += $c_k\, x^k$
end

```
sum: RR64 := 0
for k<-1:n do
  a[k] := (1-alpha)b[k]
  sum += c[k] x^k
end
```

All code on these slides was formatted by this tool.

Mathematical Syntax Using Editors

- Fortress mode for Emacs
  - Provides syntax coloring
  - Some automatic formatting
  - Unicode font conversion
- Fortress NetBeans™ plug-in
  - Syntax highlighting
  - Mark occurrences
  - Instant rename
- These tools were contributed by people outside Sun.

Parallelism by Default

A Parallel Language

High productivity for multicore, SMP, and cluster computing

- Hard to write a program that isn't potentially parallel
- Support for parallelism at several levels
  - Expressions
  - Loops, reductions, and comprehensions
  - Parallel code regions
  - Explicit multithreading
- Shared global address space model with shared data
- Thread synchronization through atomic blocks and transactional memory

Implicit Parallelism

- Tuples $(a, b, c) = (f(x), g(y), h(z))$
- Functions, operators, method call recipients, and their arguments

$e_1\, e_2$ $\qquad$ $e_1 + e_2$

$e_1(e_2)$ $\qquad$ $e_1.method(e_2)$

- Expressions with generators

$$s = \sum_{k \leftarrow 1:n} c_k\, x^k$$

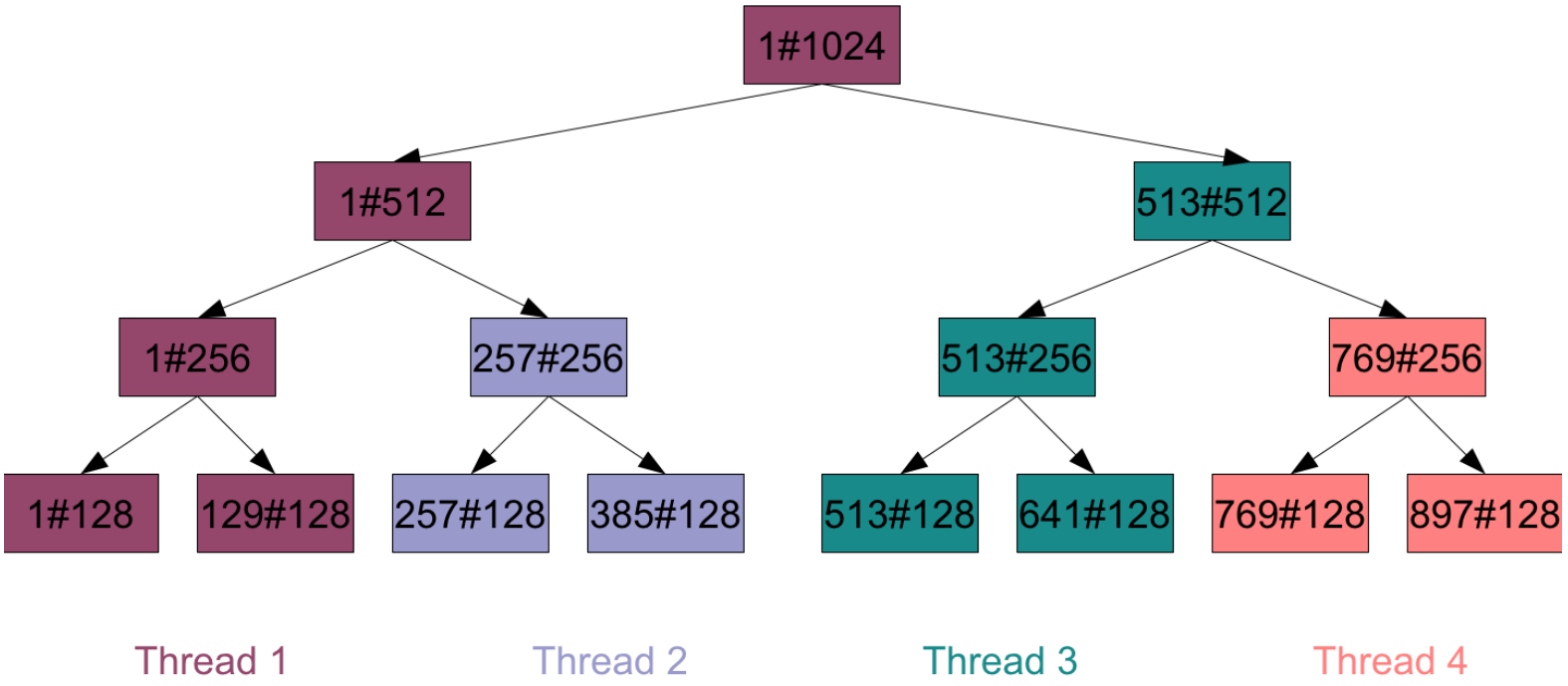
$$\{\, x^2 \mid x \leftarrow xs, x > 43 \,\}$$

```
for k ← 1:n do
  sum += c_k x^k
end
```

$$sum \mathrel{+}= c_k\, x^k, k \leftarrow 1:n$$

Recursive Subdivision and Work Stealing

$$\sum_{k \leftarrow 1\#1024} c_k \, x^k$$



Growable Language

Growing a Language

- Languages have gotten much bigger.
- You can't build one all at once.
- Therefore it must grow over time.
- What happens if you design it to grow?
- How does the need to grow affect the design?
- Need to grow a user community, too.

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.

`for` Loops in Fortress

```
for generators do
    body
end
```

- Generator list produces data items (usually in parallel).
- Body computation is executed for each data item.
- Whether/how they run in parallel is defined in the libraries.

`for` Loops in Fortress

```
for generators do
    body
end
```

```
for i ← {1, 2}, j ← {3, 4} do
    println "(" i ", " j ")"
end
```

- Generator list produces data items (usually in parallel).
- Body computation is executed for each data item.
- Whether/how they run in parallel is defined in the libraries.
- In Fortress, `for` loops are parallel by default.

$(2, 3)$
$(1, 3)$
$(2, 4)$
$(1, 4)$

Desugaring `for` Loops

$g_1 = \{1, 2\}$

$g_2 = \{3, 4\}$

```
for i ← g1, j ← g2 do
    println "(" i ", " j ")"
end
```

Desugaring `for` Loops

$g_1 = \{1, 2\}$
$g_2 = \{3, 4\}$
`for` $i \leftarrow g_1, j \leftarrow g_2$ `do`
    $println$ “(” i “, ” j “)”
`end`

desugars to

$g_1.loop($`fn` $i \Rightarrow$
    $g_2.loop($`fn` $j \Rightarrow$
        $println$ “(” i “, ” j “)” $))$

Desugaring `for` Loops

- Design strategy

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

Desugaring `for` Loops

- Design strategy

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

- `for` loops desugaring in the current implementation is built directly into the interpreter.

Desugaring `for` Loops

- Design strategy

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

- `for` loops desugaring in the current implementation

  is built directly into the interpreter.

- `for` loops desugaring by syntactic abstraction

  is provided by a library.

Syntactic Abstraction[a]

- New syntax indistinguishable from the 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 a language extension

[a] Growing a Syntax, Eric Allen, Ryan Culpepper, Janus Dam Nielsen, Jon Rafkind, and Sukyoung Ryu. The Foundations of Object-Oriented Languages 2009

Syntactic Abstraction Example

- Use of a new syntax

$xml_2 =$ `<note><to>Tove</to></eton>`

$println$ “`Xml: `” xml_2

Syntactic Abstraction Example

- Use of a new syntax

$xml_2 =$ `<note><to>Tove</to></eton>`

$println$ "`Xml: `" xml_2

- Definition of a new syntax

grammar *xml* **extends** {Expression, Symbols}

Expr |:=

x: XExpr $\Rightarrow\lhd\lfloor\, x \,\rfloor\rhd$

XExpr: Element : Expr :=

b: XmlStart c: XmlContent e: XmlEnd $\Rightarrow\lhd\lfloor$ Element(b, c, e) **asif** Content $\rfloor\rhd$

| b: XmlStart e: XmlEnd $\Rightarrow\lhd\lfloor$ Element(b, e) **asif** Content $\rfloor\rhd$

| x: XmlComplete $\Rightarrow\lhd\lfloor$ Element(x) **asif** Content $\rfloor\rhd$

...

end

Parsing Fortress Syntax

Challenges in Parsing

- Parsing juxtaposition requires types of juxtaposed items.
- Operator fixity is dependent on whitespace context.
- Program itself defines how it is parsed.
- Language syntax changes constantly.

Challenges in Parsing

- Parsing juxtaposition requires types of juxtaposed items.

  $3\,x\ \sin\ x\ \cos\ 2\,x\ \log\ \log\ x$

- Operator fixity is dependent on whitespace context.

- Program itself defines how it is parsed.

- Language syntax changes constantly.

Challenges in Parsing

- Parsing juxtaposition requires types of juxtaposed items.

  $$((((3\,x)\,(\sin\ x))\,(\cos\,(2\,x)))\,(\log\,(\log\ x)))$$

- Operator fixity is dependent on whitespace context.

- Program itself defines how it is parsed.

- Language syntax changes constantly.

Challenges in Parsing

- Parsing juxtaposition requires types of juxtaposed items.

  $$((((3\,x)\,(\sin\ x))\,(\cos\,(2\,x)))\,(\log\,(\log\ x)))$$

- Operator fixity is dependent on whitespace context.

  $$\{\,|x|\mid x \leftarrow mySet, 3|x\,\}$$

- Program itself defines how it is parsed.

- Language syntax changes constantly.

Challenges in Parsing

- Parsing juxtaposition requires types of juxtaposed items.

  $$((((3\,x)\,(\sin\ x))\,(\cos\,(2\,x)))\,(\log\,(\log\ x)))$$

- Operator fixity is dependent on whitespace context.

  $$\{\,|x|\mid x \leftarrow mySet, 3|x\,\}$$

- Program itself defines how it is parsed.

- Language syntax changes constantly.

Challenges in Parsing

- Parsing juxtaposition requires types of juxtaposed items.

  $((((3\,x)\,(\sin\ x))\,(\cos\,(2\,x)))\,(\log\,(\log\ x)))$

- Operator fixity is dependent on whitespace context.

  $\{\,|x|\mid x \leftarrow mySet, 3|x\,\}$

- Program itself defines how it is parsed.

  $xml =$ `<to>Tove</to>`

- Language syntax changes constantly.

Challenges in Parsing

- Parsing juxtaposition requires types of juxtaposed items.

  $((((3\,x)\,(\sin\ x))\,(\cos\,(2\,x)))\,(\log\,(\log\ x)))$

- Operator fixity is dependent on whitespace context.

  $\{\,|x|\mid x \leftarrow mySet, 3|x\,\}$

- Program itself defines how it is parsed.

  $xml =$ `<to>Tove</to>` `grammar` xml `extends` $\{\text{Expression}, \text{Symbols}\}\ldots$

- Language syntax changes constantly.

Challenges in Parsing

- Parsing juxtaposition requires types of juxtaposed items.

  $((((3\,x)\,(\sin\ x))\,(\cos\,(2\,x)))\,(\log\,(\log\ x)))$

- Operator fixity is dependent on whitespace context.

  $\{\,|x|\mid x \leftarrow mySet, 3|x\,\}$

- Program itself defines how it is parsed.

  $xml =$ `<to>Tove</to>` `grammar` xml `extends` $\{\text{Expression}, \text{Symbols}\}\ldots$

- Language syntax changes constantly.

  language being developed by a research lab.

Parsing Fortress Syntax

- Parsing juxtaposition requires types of juxtaposed items.

  two-phase parsing: `Juxt(3,x,sin,x,cos,2,x,log,log,x)`
- Operator fixity is dependent on whitespace context.
- Program itself defines how it is parsed.
- Language syntax changes constantly.

Parsing Fortress Syntax

- Parsing juxtaposition requires types of juxtaposed items.

  two-phase parsing: `Juxt(3,x,sin,x,cos,2,x,log,log,x)`
- Operator fixity is dependent on whitespace context.

  space-sensitive parsing: `Expression,NoNewlineExpr,NoSpaceExpr`
- Program itself defines how it is parsed.

- Language syntax changes constantly.

Parsing Fortress Syntax

- Parsing juxtaposition requires types of juxtaposed items.

  two-phase parsing: `Juxt(3,x,sin,x,cos,2,x,log,log,x)`
- Operator fixity is dependent on whitespace context.

  space-sensitive parsing: `Expression,NoNewlineExpr,NoSpaceExpr`
- Program itself defines how it is parsed.

  syntax normalization: parsing $\Rightarrow$ transformation
- Language syntax changes constantly.

Parsing Fortress Syntax

- Parsing juxtaposition requires types of juxtaposed items.

  two-phase parsing: `Juxt(3,x,sin,x,cos,2,x,log,log,x)`
- Operator fixity is dependent on whitespace context.

  space-sensitive parsing: `Expression,NoNewlineExpr,NoSpaceExpr`
- Program itself defines how it is parsed.

  syntax normalization: parsing $\Rightarrow$ transformation
- Language syntax changes constantly.

  automatic generation: ASTGen and Rats!

Parsing Techniques Tried

- JavaCC: LL(k)
- CUP: LALR(1)
- Hand-Written Recursive Descent
- Elkhound: GLR(1)
- Rats!: Packrat

Parsing Techniques Tried

- JavaCC: LL(k)

  requires big lookahead
- CUP: LALR(1)

  requires big lookahead
- Hand-Written Recursive Descent

  hard to understand, implement, adapt to syntax changes
- Elkhound: GLR(1)

  no abstraction mechanism, not scalable
- Rats!: Packrat

Rats! Parser Generator: Pros

- Unlimited lookahead
- Close to EBNF notation
- Module system
- Linear time performance
- Global parsing state

Rats! Parser Generator: Cons

- Syntax error reporting
  - Unlimited lookahead
  - Module system
- Memory consumption
  - Linear time performance
- Left-recursive productions
- Subtle grammatical errors

Rats! Parser Generator: Cons

- Syntax error reporting
  - Delimiter checker: pre-parsing
  - Syntax checker: post-parsing
- Memory consumption
  - `transient` productions
- Left-recursive productions: OMeta parsing algorithm, ...
- Subtle grammatical errors
  - Extensive regression testing

Parsing Fortress Syntax

- Parsing juxtaposition requires types of juxtaposed items.
- Operator fixity is dependent on whitespace context.
- Program itself defines how it is parsed.
- Language syntax changes constantly.

Parsing Fortress Syntax

- Parsing juxtaposition requires types of juxtaposed items.
- Operator fixity is dependent on whitespace context.
- Program itself defines how it is parsed.
- Language syntax changes constantly.
- Fortress "parser" family
  - Abstract Syntax Tree: ASTGen
  - Syntactic abstraction: syntax normalization
  - Delimiter checker: pre-parsing
  - Parser: Rats!
  - Syntax checker: post-parsing
  - Type checker: juxtaposition

Formal Semantics and Safety

Formalism for the Fortress Programming Language

Eric Allen
Eric.Allen@sun.com

Sukyoung Ryu
Sukyoung.Ryu@sun.com

Joe Hallett
Joseph.Hallett@sun.com

The Value of Formal Methods

Ariane 5

A data conversion from 64-bit floating point to 16-bit signed integer value raised an uncaught Overflow exception.

Result: The launcher was destroyed 40 seconds into the flight. The launch cost of an Ariane 5 was $180 million.

Mars Climate Orbiter

Orbiter software represented Force Time in Ns. Ground software represented Force Time in lbf s.

Result: The spacecraft was lost. The project cost was S327.6 million for both orbiter and lander.

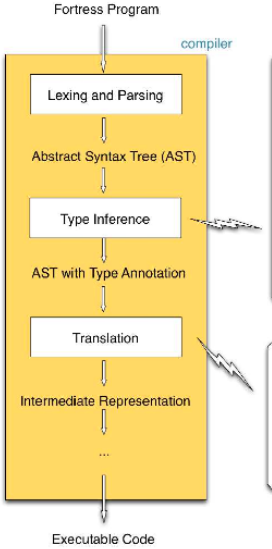
Patriot Missile Failure

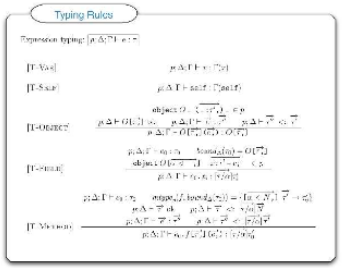
Accumulated rounding error in patriot missile software caused a missile to track its target incorrectly.

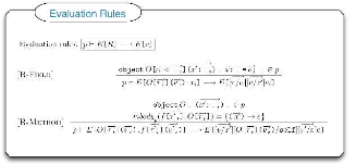
Result: SCUD missile was able to strike an army barrack, resulting in 28 Americans killed.

Formalized Semantics

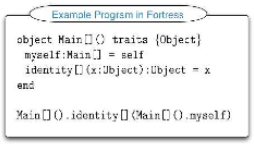
- Provides unambiguous specification for compiler writers
  - Fewer insidious bugs
  - More portable code





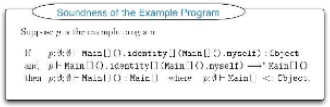


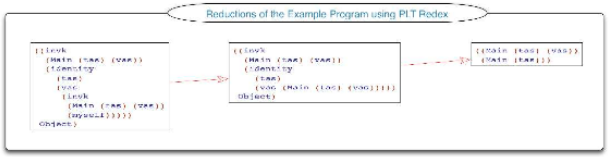
- Allows proofs of soundness and formal analysis



Mechanized Semantics

- Tests soundness of language semantics





Formal Semantics

- Formalized Semantics
  - Provides unambiguous specification for compiler writers
    - Fewer insidious bugs
    - More portable code
  - Allows proofs of soundness and formal analysis
- Mechanized Semantics
  - Tests soundness of language semantics by PLT Redex

Type System

- Traits are like Java™ interfaces, but may contain code
- Objects are like Java™ classes, but may not be extended
- Multiple inheritance of code (but not fields)
  - Objects with fields are the leaves of the hierarchy
- Traits and objects may be parameterized
  - Parameters may be types or compile-time constants
- Primitive types are first-class
  - Booleans, integers, floats, characters are all objects

Safety in Fortress

- Strong static type checking
  - Including dimensions and units
- Bounds checking
  - Enclose bounds in the type, check at compile time
- Atomic transactional synchronization
- Garbage collection
- Design-by-contract
  - requires, ensures, invariant clauses
- Test-driven development
  - properties and tests

Current Task: Compiler Development

- Better syntax error messages
- Full static type checker and various static guarantees
- Static type inference to reduce "visual clutter"
- Code generation
  - Initially targeted to JVM for full multithreaded platform independence
  - After that, VM customization for Fortress-specific optimizations
- Collaboration with universities and individuals

Conclusion

Help scientists and engineers build the programs with higher:

- Productivity
- Performance
- Correctness

Conclusion

Help scientists and engineers build the programs with higher:

- Productivity
  - > Mathematical syntax
  - > Growable language
- Performance
  - > Parallelism by default
- Correctness
  - > Formal semantics and safety

Conclusion

Help scientists and engineers build the programs with higher:

- Productivity
  - > Mathematical syntax
  - > Growable language
- Performance
  - > Parallelism by default
- Correctness
  - > Formal semantics and safety

"...

Thanks again for all the work you guys have put into this, There is fortress code being used in the wild (it is REALLY handy if the code needs to be audited and we have a math based spec.)"

– blair (ProjectFortress community)