ERC 4th Workshop

PLRG @ KAIST

Sukyoung Ryu

PLRG @ KAIST

August 26, 2010

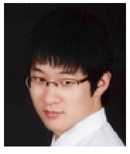
PLRG @ KAIST: Members

Professor

Sukyoung Ryu

E-mail: sryu [at] cs.kaist.ac.kr
http://plrg.kaist.ac.kr/ryu

Students



Jieung Kim

E-mail: gbali [at] kaist.ac.kr
http://plrg.kaist.ac.kr/kje

Coq Mechanization of Basic Core Fortress for Type Soundness

Changhee Park

E-mail: changhee.park [at] kaist.ac.kr
http://plrg.kaist.ac.kr/pch

Adding Pattern Matching to Existing Object-Oriented Languages

Seonghoon Kang

E-mail: kang.seonghoon [at] mearie.org
http://plrg.kaist.ac.kr/ksh

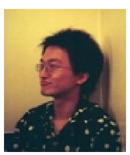
FortressCheck: Automatic Testing for Implicit Parallelism and Generic Properties

PLRG @ KAIST: Undergraduates

Jae sung Chung

E-mail: battery [at] kaist.ac.kr
http://plrg.kaist.ac.kr/jae-sung-chung

TeachScheme / Code-share /
education for everyone



Kyunghun Kim

E-mail: persona [at] chickenjuice.net
http://plrg.kaist.ac.kr/khkim

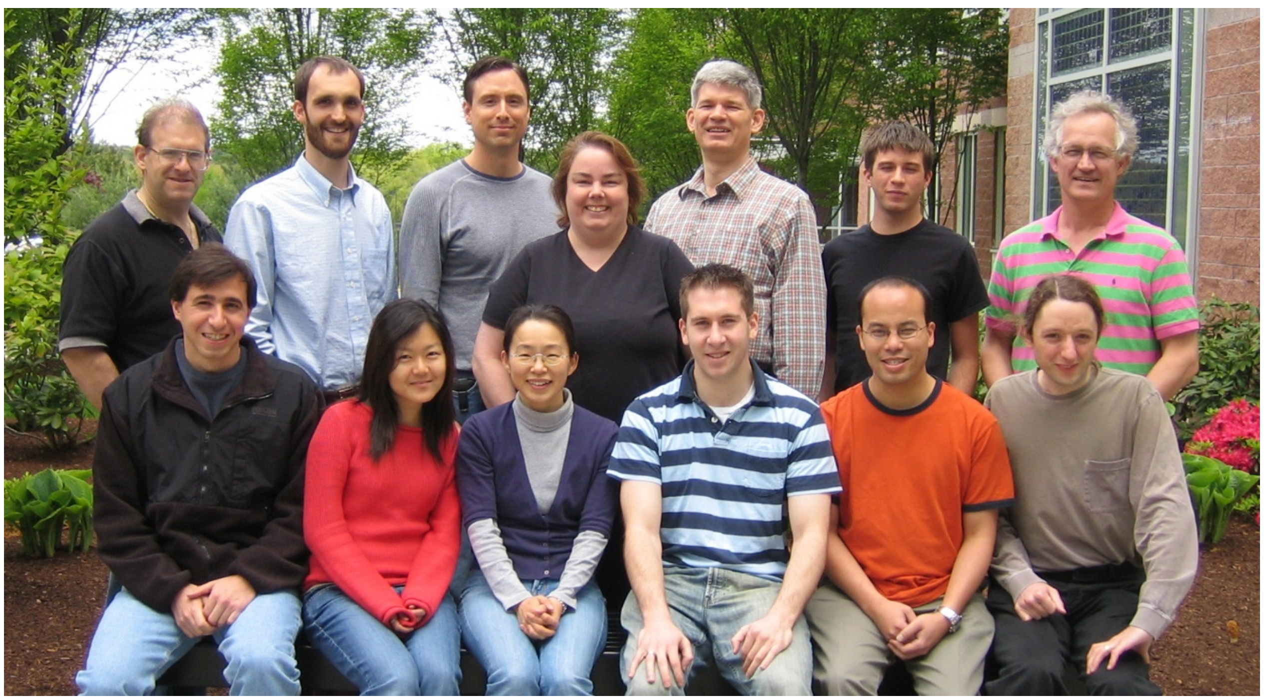
Data Processing and 3D Visualization of
High-Speed Medical Imaging System

PLRG @ KAIST: Collaborators

나현익: Binary Methods Support Using Self-Type Idiom in Fortress

김준범: Cloud Computing for Large-Scale Scientific Data Analysis

PLRG @ Sun Labs, Oracle: Type-checking Modular Multiple Dispatch with Parametric Polymorphism and Multiple Inheritance



Project Fortress

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

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

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

$$C = A \cup B$$

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

- "Growing a Language"

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

Formalism for Fortress

- Fortress calculi
  - Basic core Fortress
  - Core Fortress with where clauses
  - Core Fortress with overloading
  - Acyclic core Fortress with field definitions
- For each Fortress calculus
  - Syntax
  - Static semantics
  - Dynamic semantics
  - Type soundness proof

Fortress 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

Example: Binary Tree in Fortress

```
trait Tree
  getter item(): ℤ32
  getter depth(): ℤ32
end
object Node(left: Tree, item: ℤ32, right: Tree) extends Tree
  getter depth(): ℤ32 = 1 + (left.depth MAX right.depth)
end
object Leaf(item: Z32) extends Tree
  getter depth(): Z32 = 1
end
```

Basic Core Fortress (BCF)

α, β					type variables
τ, τ', τ''	$::=$	α	$\vert$	σ	type
σ	$::=$	N	$\vert$	$O[\![\overrightarrow{\tau}]\!]$	named type
N, M, L	$::=$	$T[\![\overrightarrow{\tau}]\!]$	$\vert$	Object	trait type
p	$::=$	$\overrightarrow{d}\ e$			program
d	$::=$	td	$\vert$	od	definition
td	$::=$	$\texttt{trait}\ T[\![\overrightarrow{\alpha\ \texttt{extends}\ N}]\!]\ \texttt{extends}\ \{\overrightarrow{N}\}\ \overrightarrow{fd}\ \texttt{end}$			trait definition
od	$::=$	$\texttt{object}\ O[\![\overrightarrow{\alpha\ \texttt{extends}\ N}]\!](\overrightarrow{x{:}\tau})\ \texttt{extends}\ \{\overrightarrow{N}\}\ \overrightarrow{fd}\ \texttt{end}$			object definition
fd	$::=$	$f[\![\overrightarrow{\alpha\ \texttt{extends}\ N}]\!](\overrightarrow{x{:}\tau}){:}\tau = e$			method definition
e	$::=$	x			expression
	$\vert$	$\texttt{self}$			
	$\vert$	$O[\![\overrightarrow{\tau}]\!](\overrightarrow{e})$			
	$\vert$	$e.x$			
	$\vert$	$e.f[\![\overrightarrow{\tau}]\!](\overrightarrow{e})$			

Core Fortress with Overloading

- BCF + overloading
- Overloading
  - > Multiple declarations for the same functional name
  - > Several of the overloaded declarations may be applicable to any particular functional call

Functionals in Fortress

- Functionals
  - > Functions
    - * Top-level functions
    - * Local functions
  - > Methods
    - * Dotted methods
    - * Functional methods
- Special functionals
  - > Operators
  - > Coercions

Functionals in Fortress

- Functionals
  > Functions — first-class values
    * Top-level functions — top-level in components or APIs
    * Local functions — within blocks
  > Methods — have owners (traits or objects)
    * Dotted methods — implicit `self`
    * Functional methods — explicit `self`
- Special functionals
  > Operators — top-level functions or functional methods
  > Coercions — special dotted methods

Why Functional Methods?

- For data extensibility and encapsulation
- For function extensibility even with top-level functions
- For mathematical syntax with overloaded operators

$\texttt{trait}\ \mathrm{Matrix}\ \texttt{excludes}\ \mathrm{Vector}$

$\quad \texttt{opr}\ \cdot(\texttt{self}, \mathit{other}\colon \mathrm{Vector})\colon \mathrm{Matrix}$

$\quad \texttt{opr}\ \cdot(\mathit{other}\colon \mathrm{Vector}, \texttt{self})\colon \mathrm{Matrix}$

$\texttt{end}$

$v \cdot M + M \cdot v$

Fortress Overloading

- Goal

  No ambiguous nor undefined calls at run time

- Challenges

  Modular Multiple dispatch & Multiple inheritance[a]

- Solution

  Static overloading rules to guarantee the goal

[a] "Modular Multiple Dispatch with Multiple Inheritance," Eric Allen, J.J. Hallett, Victor Luchangco, Sukyoung Ryu, and Guy L. Steele Jr. SAC 2007: 22nd Annual ACM Symposium on Applied Computing

Overloading Rules

- Compare overloaded declarations pairwise.
- If any rule holds then a valid overloading:
  - Exclusion Rule
    - Parameter types exclude each other.
  - Subtype Rule
    - Parameter type of one declaration is a subtype of the other.
    - Return types must also be in subtype relation.
  - Meet Rule
    - Exists a declaration that is more specific than both.

Overloading Resolution Proof

Theorem 1. If all the overloaded declarations satisfy the static overloading rules, there are no ambiguous nor undefined calls at run time.

How about Generic Functionals?

$$size[\![\alpha\ \texttt{extends}\ \mathrm{Any}]\!](l\colon \mathrm{ArrayList}[\![\alpha]\!])\colon \mathbb{Z}$$

$$size[\![\beta\ \texttt{extends}\ \mathbb{Z}]\!](l\colon \mathrm{List}[\![\beta]\!])\colon \mathbb{Z}$$

where $\mathrm{ArrayList}[\![T]\!] <: \mathrm{List}[\![T]\!]$ for all types T

How about Generic Functionals?

$size[\![\alpha \texttt{ extends } \mathrm{Any}]\!](l\colon \mathrm{ArrayList}[\![\alpha]\!])\colon \mathbb{Z}$

$size[\![\beta \texttt{ extends } \mathbb{Z}]\!](l\colon \mathrm{List}[\![\beta]\!])\colon \mathbb{Z}$

where $\mathrm{ArrayList}[\![T]\!] <: \mathrm{List}[\![T]\!]$ for all types T

$size[\![\gamma \texttt{ extends } \mathbb{Z}]\!](l\colon \mathrm{ArrayList}[\![\gamma]\!])\colon \mathbb{Z}$

How about Generic Functionals?

$size[\![\alpha \ \texttt{extends}\ \mathrm{Any}]\!](l\colon \mathrm{ArrayList}[\![\alpha]\!])\colon \mathbb{Z}$

$size[\![\beta \ \texttt{extends}\ \mathbb{Z}]\!](l\colon \mathrm{List}[\![\beta]\!])\colon \mathbb{Z}$

where $\mathrm{ArrayList}[\![T]\!] <: \mathrm{List}[\![T]\!]$ for all types T

$size[\![\gamma \ \texttt{extends}\ \mathbb{Z}]\!](l\colon \mathrm{ArrayList}[\![\gamma]\!])\colon \mathbb{Z}$

$\mathrm{BadList} <: \{\ \mathrm{ArrayList}[\![\mathrm{String}]\!], \mathrm{List}[\![\mathbb{Z}]\!]\ \}$

The first two are applicable to $\mathrm{BadList}$ but not the third.

Generic Overloading Rules[a]

- No Duplicates Rule
- Meet Rule
- Return Type Rule
- Polymorphic Exclusion

  A type (other than BottomType) must not be a subtype of multiple distinct instantiations of a type constructor.

[a] "Type-checking Modular Multiple Dispatch with Parametric Polymorphism and Multiple Inheritance," Eric Allen, Justin Hilburn, Scott Kilpatrick, Sukyoung Ryu, David Chase, Victor Luchangco, and Guy L. Steele Jr. (submitted)

More Features to Prove

- Pattern matching (박창희 & Guy Steele)
- Self-type idiom (나현익 & Victor Luchangco)
- Generic overloaded functionals (Sun Labs, Oracle)
- Where clauses
- Coercions
- Type inferene
- . . .

More Fun to Do

- Fortress
  - > Static checks and analyses
  - > FortressCheck
  - > Fortress calculi mechanization in Coq
- And beyond
  - > Program analyses for medical imaging systems
  - > Teaching material for KSA (Korea Science Academy) & *education for everyone*

Sukyoung Ryu

sryu@cs.kaist.ac.kr
http://plrg.kaist.ac.kr