

# Type Inference for RAJA

A Static Heap Space Analysis of OO-Programs.

**Dulma Rodriguez**, Martin Hofmann

Department of Computer Science  
Ludwig-Maximilians-University Munich

Dagstuhl Seminar 10351

Modelling, Controlling and Reasoning About State  
August 31th, 2010



# Introduction

- System RAJA (Hofmann and Jost, ESOP 2006)
  - Type system for Java-like programs.
  - Compile-time analysis of heap-space requirements.
  - Amortised complexity analysis.
- Polymorphic types
  - RAJA method types are polymorphic.
  - this enables a local analysis.
- Type inference
  - Restricted to monomorphic recursion.
  - System for generating subtyping and linear arithmetic constraints.
  - Algorithm for solving subtyping constraints.
  - Linear arithmetic constraints solved by an LP Solver.

# Introduction

- System RAJA (Hofmann and Jost, ESOP 2006)
  - Type system for Java-like programs.
  - Compile-time analysis of heap-space requirements.
  - Amortised complexity analysis.
- Polymorphic types
  - RAJA method types are polymorphic.
  - this enables a local analysis.
- Type inference
  - Restricted to monomorphic recursion.
  - System for generating subtyping and linear arithmetic constraints.
  - Algorithm for solving subtyping constraints.
  - Linear arithmetic constraints solved by an LP Solver.

# Introduction

- System RAJA (Hofmann and Jost, ESOP 2006)
  - Type system for Java-like programs.
  - Compile-time analysis of heap-space requirements.
  - Amortised complexity analysis.
- Polymorphic types
  - RAJA method types are polymorphic.
  - this enables a local analysis.
- Type inference
  - Restricted to **monomorphic recursion**.
  - System for generating **subtyping** and **linear arithmetic** constraints.
  - Algorithm for solving subtyping constraints.
  - Linear arithmetic constraints solved by an LP Solver.

# Outline

- 1 Introduction to RAJA
- 2 Monomorphic *RAJA* (*RAJA<sup>m</sup>*)
- 3 *RAJA<sup>m</sup>* Type Inference
  - Constraints-generation system
  - Constraints solver
- 4 Conclusions

# Amortised Analysis of Heap-Usage

- Free-list based model
  - Deallocation in C/C++ style with primitive dispose.
  - Object creation: takes memory units from free-list.
  - Object destruction: returns memory units to free-list.
- Goal:
  - Upper bound on the size of the free-list.
  - As function of the input.
- Amortised analysis
  - Types assign each heap configuration statically a potential.
  - Object creation: must pay using potential from input.
  - Potential of consumed input: upper bound on heap space consumption.

# Amortised Analysis of Heap-Usage

- Free-list based model
  - Deallocation in C/C++ style with primitive dispose.
  - Object creation: takes memory units from free-list.
  - Object destruction: returns memory units to free-list.
- Goal:
  - Upper bound on the size of the free-list.
  - As function of the input.
- Amortised analysis
  - Types assign each heap configuration statically a potential.
  - Object creation: must pay using potential from input.
  - Potential of consumed input: upper bound on heap space consumption.

# Amortised Analysis of Heap-Usage

- Free-list based model
  - Deallocation in C/C++ style with primitive dispose.
  - Object creation: takes memory units from free-list.
  - Object destruction: returns memory units to free-list.
- Goal:
  - Upper bound on the size of the free-list.
  - As function of the input.
- Amortised analysis
  - Types assign each heap configuration statically a potential.
  - Object creation: must pay using potential from input.
  - Potential of consumed input: upper bound on heap space consumption.

## Object-Oriented Language: FJEU

$c ::=$	class $C$ [extends $D$ ] { $A_1; \dots; A_k; M_1 \dots M_j$ }	
$A ::=$	$C$ $a$	
$M ::=$	$C_0$ $m(C_1$ $x_1, \dots, C_j$ $x_j)$ {return $e$ ; }	
$e ::=$	$x$	(Variable)
	null	(Constant)
	new $C$	(Construction)
	free ( $x$ )	(Destruction)
	( $C$ ) $x$	(Cast)
	$x.a_i$	(Access)
	$x.a_i \leftarrow x$	(Update)
	$x.m(x_1, \dots, x_j)$	(Invocation)
	if $x$ instanceof $C$ then $e_1$ else $e_2$	(Conditional)
	let $D$ $x = e_1$ in $e_2$	(Let)

- $\approx$  Featherweight Java (Igarashi, Pierce, Wadler, OOPSLA'99) + imperative field update.

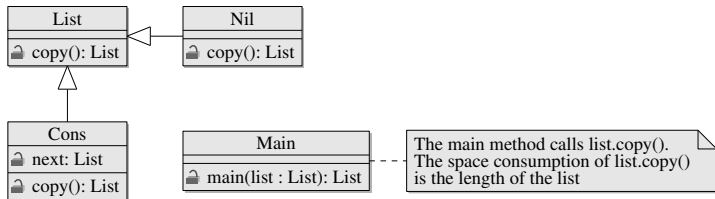
## Lists in FJEU with copy method

```
class List {
  List copy() {
    return null;
  }
}
```

```
class Nil extends List {
  List copy() {
    return new Nil;
  }
}
```

```
class Cons extends List {
  int elem;
  List next;

  List copy() {
    let Cons res = new Cons in
    let Cons _ = res.elem ← this.elem in
    let List rnext = this.next.copy() in
    let List _ = res.next ← rnext in
    return res;
  }
}
```



# Sketch of RAJA System

RAJA program  $P = \textit{annotated}$  FJEU program.

- 1 Set of *views*  $\mathcal{V}$ .
- 2 For each class  $C$  and view  $r$  we have an annotated version  $C^r$ .

3 Potential:

$$\diamond(C^r) : \text{Class} \times \text{View} \rightarrow \mathbb{Q}^+$$

4 (Get- and set-) views for attributes:

$$A^{\text{get}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View} \quad (\text{get-view})$$

$$A^{\text{set}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View} \quad (\text{set-view})$$

5 RAJA types for methods:

$$M(\cdot, \cdot) : \text{Class} \times \text{Meth} \rightarrow \text{PolyType}$$

$$\phi = \forall_{v_{\text{self}}, \vec{v}, \vec{q}}. C^{v_{\text{self}}}; E_1^{v_1}, \dots, E_j^{v_j} \xrightarrow{q_1/q_2} E_{j+1}^{v_{j+1}} \ \& \ \mathcal{C}_m(v_{\text{self}}, \vec{v}, \vec{q})$$

• The linear arithmetic constraints are of the following shape:

$$p_1 \geq \diamond(D^v) + 1 \quad p_2 \leq p_1 - \diamond(D^v) - 1 \quad p_2 \leq q_2 + p_1 - q_1$$

• The subtyping constraints are of the following shape:

$$D^{v_1} <: C^{v_2} \quad C.a^{A^{\text{get}}(C^{v_1}, a)} <: D^{v_2} \quad E^{v_2} <: C.a^{A^{\text{set}}(C^{v_1}, a)}$$

# Sketch of RAJA System

RAJA program  $P = \textit{annotated}$  FJEU program.

- 1 Set of *views*  $\mathcal{V}$ .
- 2 For each class  $C$  and view  $r$  we have an annotated version  $C^r$ .
- 3 Potential:
  - $\diamond(C^r) : \text{Class} \times \text{View} \rightarrow \mathbb{Q}^+$
- 4 (Get- and set-) views for attributes:
  - $A^{\text{get}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$  (get-view)
  - $A^{\text{set}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$  (set-view)
- 5 RAJA types for methods:
  - $M(\cdot, \cdot) : \text{Class} \times \text{Meth} \rightarrow \text{PolyType}$ 

$$\phi = \forall v_{\text{self}}, \vec{v}, \vec{q}. C^{v_{\text{self}}}; E_1^{v_1}, \dots, E_j^{v_j} \xrightarrow{q_1/q_2} E_{j+1}^{v_{j+1}} \ \& \ \mathcal{C}_m(v_{\text{self}}, \vec{v}, \vec{q})$$
  - The linear arithmetic constraints are of the following shape:
 
$$p_1 \geq \diamond(D^v) + 1 \quad p_2 \leq p_1 - \diamond(D^v) - 1 \quad p_2 \leq q_2 + p_1 - q_1$$
  - The subtyping constraints are of the following shape:

$$D^{v_1} <: C^{v_2} \quad C.a^{A^{\text{get}}(C^{v_1}, a)} <: D^{v_2} \quad E^{v_2} <: C.a^{A^{\text{set}}(C^{v_1}, a)}$$

# Sketch of RAJA System

RAJA program  $P = \textit{annotated}$  FJEU program.

- 1 Set of *views*  $\mathcal{V}$ .
- 2 For each class  $C$  and view  $r$  we have an annotated version  $C^r$ .
- 3 Potential:
  - $\diamond(C^r) : \text{Class} \times \text{View} \rightarrow \mathbb{Q}^+$
- 4 (Get- and set-) views for attributes:
  - $A^{\text{get}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$  (get-view)
  - $A^{\text{set}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$  (set-view)
- 5 RAJA types for methods:
  - $M(\cdot, \cdot) : \text{Class} \times \text{Meth} \rightarrow \text{PolyType}$ 

$$\phi = \forall v_{\text{self}}, \vec{v}, \vec{q}. C^{v_{\text{self}}}; E_1^{v_1}, \dots, E_j^{v_j} \xrightarrow{q_1/q_2} E_{j+1}^{v_{j+1}} \ \& \ \mathcal{C}_m(v_{\text{self}}, \vec{v}, \vec{q})$$
  - The linear arithmetic constraints are of the following shape:
 
$$p_1 \geq \diamond(D^v) + 1 \quad p_2 \leq p_1 - \diamond(D^v) - 1 \quad p_2 \leq q_2 + p_1 - q_1$$
  - The subtyping constraints are of the following shape:

$$D^{v_1} <: C^{v_2} \quad C.a^{A^{\text{get}}(C^{v_1}, a)} <: D^{v_2} \quad E^{v_2} <: C.a^{A^{\text{set}}(C^{v_1}, a)}$$

# Sketch of RAJA System

RAJA program  $P = \textit{annotated}$  FJEU program.

- 1 Set of *views*  $\mathcal{V}$ .
- 2 For each class  $C$  and view  $r$  we have an annotated version  $C^r$ .
- 3 Potential:
  - $\diamond(C^r) : \text{Class} \times \text{View} \rightarrow \mathbb{Q}^+$
- 4 (Get- and set-) views for attributes:
  - $A^{\text{get}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$  (get-view)
  - $A^{\text{set}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$  (set-view)
- 5 RAJA types for methods:
  - $M(\cdot, \cdot) : \text{Class} \times \text{Meth} \rightarrow \text{PolyType}$

$$\phi = \forall v_{\text{self}}, \vec{v}, \vec{q}. C^{v_{\text{self}}}; E_1^{v_1}, \dots, E_j^{v_j} \xrightarrow{q_1/q_2} E_{j+1}^{v_{j+1}} \ \& \ \mathcal{C}_m(v_{\text{self}}, \vec{v}, \vec{q})$$

- The linear arithmetic constraints are of the following shape:

$$p_1 \geq \diamond(D^v) + 1 \quad p_2 \leq p_1 - \diamond(D^v) - 1 \quad p_2 \leq q_2 + p_1 - q_1$$

- The subtyping constraints are of the following shape:

$$D^{v_1} <: C^{v_2} \quad C.a^{A^{\text{get}}(C^{v_1}, a)} <: D^{v_2} \quad E^{v_2} <: C.a^{A^{\text{set}}(C^{v_1}, a)}$$

# Sketch of RAJA System

RAJA program  $P = \textit{annotated}$  FJEU program.

- 1 Set of *views*  $\mathcal{V}$ .
- 2 For each class  $C$  and view  $r$  we have an annotated version  $C^r$ .
- 3 Potential:
  - $\diamond(C^r) : \text{Class} \times \text{View} \rightarrow \mathbb{Q}^+$
- 4 (Get- and set-) views for attributes:
  - $A^{\text{get}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$  (get-view)
  - $A^{\text{set}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$  (set-view)
- 5 RAJA types for methods:
  - $M(\cdot, \cdot) : \text{Class} \times \text{Meth} \rightarrow \text{PolyType}$ 

$$\phi = \forall v_{\text{self}}, \vec{v}, \vec{q}. C^{v_{\text{self}}}; E_1^{v_1}, \dots, E_j^{v_j} \xrightarrow{q_1/q_2} E_{j+1}^{v_{j+1}} \ \& \ \mathcal{C}_m(v_{\text{self}}, \vec{v}, \vec{q})$$
  - The linear arithmetic constraints are of the following shape:
 
$$p_1 \geq \diamond(D^v) + 1 \quad p_2 \leq p_1 - \diamond(D^v) - 1 \quad p_2 \leq q_2 + p_1 - q_1$$
  - The subtyping constraints are of the following shape:

$$D^{v_1} <: C^{v_2} \quad C.a^{A^{\text{get}}(C^{v_1}, a)} <: D^{v_2} \quad E^{v_2} <: C.a^{A^{\text{set}}(C^{v_1}, a)}$$

# Sketch of RAJA System

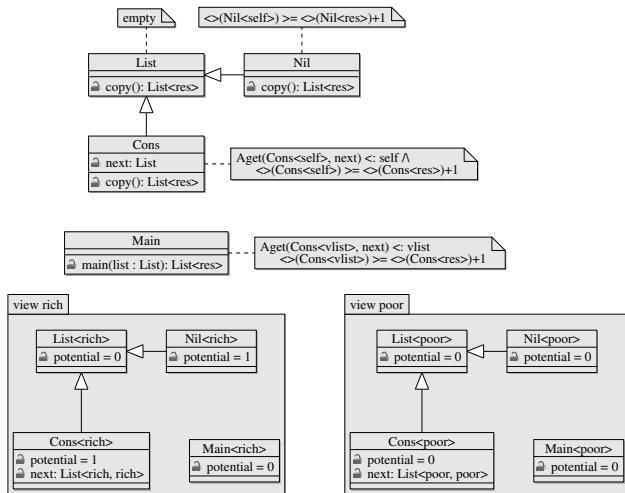
RAJA program  $P = \textit{annotated}$  FJEU program.

- ① Set of *views*  $\mathcal{V}$ .
- ② For each class  $C$  and view  $r$  we have an annotated version  $C^r$ .
- ③ Potential:
  - $\diamond(C^r) : \text{Class} \times \text{View} \rightarrow \mathbb{Q}^+$
- ④ (Get- and set-) views for attributes:
  - $A^{\text{get}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$  (get-view)
  - $A^{\text{set}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$  (set-view)
- ⑤ RAJA types for methods:
  - $M(\cdot, \cdot) : \text{Class} \times \text{Meth} \rightarrow \text{PolyType}$ 

$$\phi = \forall v_{\text{self}}, \vec{v}, \vec{q}. C^{v_{\text{self}}}; E_1^{v_1}, \dots, E_j^{v_j} \xrightarrow{q_1/q_2} E_{j+1}^{v_{j+1}} \ \& \ \mathcal{C}_m(v_{\text{self}}, \vec{v}, \vec{q})$$
  - The linear arithmetic constraints are of the following shape:
 
$$p_1 \geq \diamond(D^v) + 1 \quad p_2 \leq p_1 - \diamond(D^v) - 1 \quad p_2 \leq q_2 + p_1 - q_1$$
  - The subtyping constraints are of the following shape:

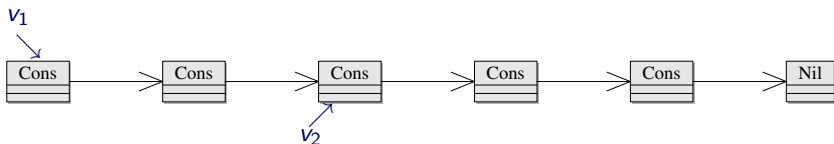
$$D^{v_1} <: C^{v_2} \quad C.a^{A^{\text{get}}(C^{v_1}, a)} <: D^{v_2} \quad E^{v_2} <: C.a^{A^{\text{set}}(C^{v_1}, a)}$$

## Lists with copy method in RAJA



## Potential

- Let  $\Gamma = l_1 : \text{List}^{\text{rich}}, l_2 : \text{List}^{\text{poor}}$  and  $\eta = [l_1 \mapsto v_1, l_2 \mapsto v_2]$  and  $\sigma$  be the following heap:



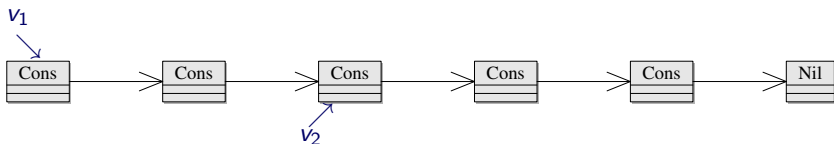
$$\text{pot}_\sigma(v_1 : \text{rich}) = 6$$

$$\text{pot}_\sigma(v_2 : \text{poor}) = 0$$

$$\text{pot}_\sigma(\eta : \Gamma) = \underbrace{\text{pot}_\sigma(v_1 : \text{rich})}_6 + \underbrace{\text{pot}_\sigma(v_2 : \text{poor})}_0 = 6$$

# Potential

- Let  $\Gamma = l_1 : \text{List}^{\text{rich}}, l_2 : \text{List}^{\text{poor}}$  and  $\eta = [l_1 \mapsto v_1, l_2 \mapsto v_2]$  and  $\sigma$  be the following heap:



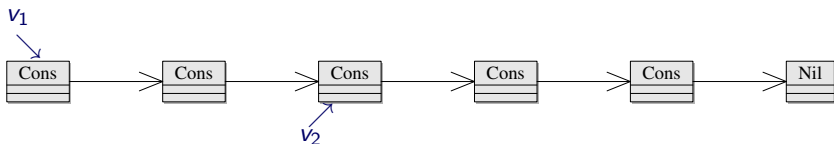
$$\text{pot}_\sigma(v_1 : \text{rich}) = 6$$

$$\text{pot}_\sigma(v_2 : \text{poor}) = 0$$

$$\text{pot}_\sigma(\eta : \Gamma) = \underbrace{\text{pot}_\sigma(v_1 : \text{rich})}_6 + \underbrace{\text{pot}_\sigma(v_2 : \text{poor})}_0 = 6$$

## Potential

- Let  $\Gamma = l_1 : \text{List}^{\text{rich}}, l_2 : \text{List}^{\text{poor}}$  and  $\eta = [l_1 \mapsto v_1, l_2 \mapsto v_2]$  and  $\sigma$  be the following heap:



$$\text{pot}_\sigma(v_1 : \text{rich}) = 6$$

$$\text{pot}_\sigma(v_2 : \text{poor}) = 0$$

$$\text{pot}_\sigma(\eta : \Gamma) = \underbrace{\text{pot}_\sigma(v_1 : \text{rich})}_6 + \underbrace{\text{pot}_\sigma(v_2 : \text{poor})}_0 = 6$$

# Main result

- $\eta, \sigma \vdash e \rightsquigarrow v, \tau$  means:
  - expression  $e$  evaluates successfully to value  $v$
  - beginning with stack  $\eta$  and heap  $\sigma$
  - and ending with heap  $\tau$
  - (with an unbounded free-list).
- We define a typing judgment  $\Gamma \frac{n}{n'} e : C^r$  so that:
  - if  $\Gamma \frac{n}{n'} e : C^r$  and  $\eta, \sigma \vdash e \rightsquigarrow v, \tau$  then

$e$  evaluates  if  $|\text{free-list}| \geq n + \text{pot}_\sigma(\eta : \Gamma)$

## RAJA typing

- Typing judgement:  $\Gamma \frac{n_1}{n_2} e : C^r$

$$\frac{}{\emptyset \frac{1 + \diamond(C^r)}{0} \text{ new } C : C^r} \quad (\diamond \text{New}) \qquad \frac{}{x : C^r \frac{0}{1 + \diamond(C^r)} \text{ free } (x) : E^s} \quad (\diamond \text{Free})$$

$$\frac{\Gamma_1 \frac{n}{n'} e_1 : D^s \quad \Gamma_2, x : D^s \frac{n'}{n''} e_2 : C^r}{\Gamma_1, \Gamma_2 \frac{n}{n''} \text{ let } D x = e_1 \text{ in } e_2 : C^r} \quad (\diamond \text{Let})$$

## Sharing relation

- $\forall(\mathbf{r} \mid \mathbf{s}_1, \dots, \mathbf{s}_i)$ : coinductively defined relation with:  
for all  $C$ ,  $\diamond(C^{\mathbf{r}}) \geq \diamond(C^{\mathbf{s}_1}) + \dots + \diamond(C^{\mathbf{s}_i})$ , etc.

- If this were allowed ...

$$l : \text{List}^{\text{rich}} \stackrel{0}{\vdash} \text{let } \_ = l.\text{copy}() \text{ in } l.\text{copy}() : \text{List}^{\text{poor}}$$

- ... heap consumption would be  $2|l|$ , but prediction would be  $\text{pot}_\sigma(l : \text{rich}) = |l|$ .  $\Rightarrow$  unsound!
- The rule ( $\diamond\text{Share}$ ) enables multiple (sound) uses of a variable.

$$\frac{\Gamma, y_1 : D^{\mathbf{s}_1}, y_2 : D^{\mathbf{s}_2} \stackrel{n}{\vdash}_{n'} e : C^{\mathbf{r}} \quad \forall(\mathbf{s} \mid \mathbf{s}_1, \mathbf{s}_2)}{\Gamma, x : D^{\mathbf{s}} \stackrel{n}{\vdash}_{n'} e[x/y_1, x/y_2] : C^{\mathbf{r}}} \quad (\diamond\text{Share})$$

# Sharing relation

- $\forall(\mathbf{r} \mid \mathbf{s}_1, \dots, \mathbf{s}_i)$ : coinductively defined relation with:  
for all  $C$ ,  $\diamond(C^{\mathbf{r}}) \geq \diamond(C^{\mathbf{s}_1}) + \dots + \diamond(C^{\mathbf{s}_i})$ , etc.

- $\forall(\mathbf{rich} \mid \mathbf{rich}, \mathbf{poor})$  ✓

- If this were allowed ...

$$l : \text{List}^{\mathbf{rich}} \vdash_0^0 \text{ let } \_ = l.\text{copy}() \text{ in } l.\text{copy}() : \text{List}^{\mathbf{poor}}$$

- ... heap consumption would be  $2|l|$ , but prediction would be  $\text{pot}_\sigma(l : \mathbf{rich}) = |l|$ .  $\Rightarrow$  unsound!
- The rule ( $\diamond\text{Share}$ ) enables multiple (sound) uses of a variable.

$$\frac{\Gamma, y_1 : D^{\mathbf{s}_1}, y_2 : D^{\mathbf{s}_2} \vdash_{n'}^n e : C^{\mathbf{r}} \quad \forall(\mathbf{s} \mid \mathbf{s}_1, \mathbf{s}_2)}{\Gamma, x : D^{\mathbf{s}} \vdash_{n'}^n e[x/y_1, x/y_2] : C^{\mathbf{r}}} \quad (\diamond\text{Share})$$

# Sharing relation

- $\forall(\mathbf{r} \mid \mathbf{s}_1, \dots, \mathbf{s}_i)$ : coinductively defined relation with:  
for all  $C$ ,  $\diamond(C^{\mathbf{r}}) \geq \diamond(C^{\mathbf{s}_1}) + \dots + \diamond(C^{\mathbf{s}_i})$ , etc.

- $\forall(\mathbf{rich} \mid \mathbf{rich}, \mathbf{poor})$  ✓       $\forall(\mathbf{rich} \mid \mathbf{rich}, \mathbf{rich})$  ✗

- If this were allowed ...

$l : \text{List}^{\mathbf{rich}} \vdash_0^0 \text{ let } \_ = l.\text{copy}() \text{ in } l.\text{copy}() : \text{List}^{\mathbf{poor}}$

- ... heap consumption would be  $2|l|$ , but prediction would be  $\text{pot}_\sigma(l : \mathbf{rich}) = |l|$ .  $\Rightarrow$  unsound!
- The rule ( $\diamond\text{Share}$ ) enables multiple (sound) uses of a variable.

$$\frac{\Gamma, y_1 : D^{\mathbf{s}_1}, y_2 : D^{\mathbf{s}_2} \vdash_{n'}^n e : C^{\mathbf{r}} \quad \forall(\mathbf{s} \mid \mathbf{s}_1, \mathbf{s}_2)}{\Gamma, x : D^{\mathbf{s}} \vdash_{n'}^n e[x/y_1, x/y_2] : C^{\mathbf{r}}} \quad (\diamond\text{Share})$$

## Sharing relation

- $\forall(\mathbf{r} \mid \mathbf{s}_1, \dots, \mathbf{s}_i)$ : coinductively defined relation with:  
for all  $C$ ,  $\diamond(C^{\mathbf{r}}) \geq \diamond(C^{\mathbf{s}_1}) + \dots + \diamond(C^{\mathbf{s}_i})$ , etc.

- $\forall(\mathbf{rich} \mid \mathbf{rich}, \mathbf{poor})$  ✓  $\forall(\mathbf{rich} \mid \mathbf{rich}, \mathbf{rich})$  ✗

- If this were allowed ...

$l : \text{List}^{\mathbf{rich}} \vdash_0^0$  let  $\_ = l.\text{copy}()$  in  $l.\text{copy}() : \text{List}^{\mathbf{poor}}$

- ... heap consumption would be  $2|l|$ , but prediction would be  $\text{pot}_\sigma(l : \mathbf{rich}) = |l|$ .  $\Rightarrow$  unsound!
- The rule ( $\diamond\text{Share}$ ) enables multiple (sound) uses of a variable.

$$\frac{\Gamma, y_1 : D^{\mathbf{s}_1}, y_2 : D^{\mathbf{s}_2} \vdash_{n'}^n e : C^{\mathbf{r}} \quad \forall(\mathbf{s} \mid \mathbf{s}_1, \mathbf{s}_2)}{\Gamma, x : D^{\mathbf{s}} \vdash_{n'}^n e[x/y_1, x/y_2] : C^{\mathbf{r}}} \quad (\diamond\text{Share})$$

## Sharing relation

- $\forall(\mathbf{r} \mid \mathbf{s}_1, \dots, \mathbf{s}_i)$ : coinductively defined relation with:  
for all  $C$ ,  $\diamond(C^{\mathbf{r}}) \geq \diamond(C^{\mathbf{s}_1}) + \dots + \diamond(C^{\mathbf{s}_i})$ , etc.

- $\forall(\mathbf{rich} \mid \mathbf{rich}, \mathbf{poor})$  ✓       $\forall(\mathbf{rich} \mid \mathbf{rich}, \mathbf{rich})$  ✗

- If this were allowed ...

$l : \text{List}^{\mathbf{rich}} \vdash_0^0$  let  $\_ = l.\text{copy}()$  in  $l.\text{copy}() : \text{List}^{\mathbf{poor}}$

- ... heap consumption would be  $2|l|$ , but prediction would be  $\text{pot}_\sigma(l : \mathbf{rich}) = |l|$ .  $\Rightarrow$  unsound!
- The rule ( $\diamond\text{Share}$ ) enables multiple (sound) uses of a variable.

$$\frac{\Gamma, y_1 : D^{\mathbf{s}_1}, y_2 : D^{\mathbf{s}_2} \vdash_{n'}^n e : C^{\mathbf{r}} \quad \forall(\mathbf{s} \mid \mathbf{s}_1, \mathbf{s}_2)}{\Gamma, x : D^{\mathbf{s}} \vdash_{n'}^n e[x/y_1, x/y_2] : C^{\mathbf{r}}} \quad (\diamond\text{Share})$$

## Sharing relation

- $\forall(\mathbf{r} \mid \mathbf{s}_1, \dots, \mathbf{s}_i)$ : coinductively defined relation with:  
for all  $C$ ,  $\diamond(C^{\mathbf{r}}) \geq \diamond(C^{\mathbf{s}_1}) + \dots + \diamond(C^{\mathbf{s}_i})$ , etc.

- $\forall(\mathbf{rich} \mid \mathbf{rich}, \mathbf{poor})$  ✓  $\forall(\mathbf{rich} \mid \mathbf{rich}, \mathbf{rich})$  ✗

- If this were allowed ...

$$l : \text{List}^{\mathbf{rich}} \stackrel{0}{\vdash} \text{ let } \_ = l.\text{copy}() \text{ in } l.\text{copy}() : \text{List}^{\mathbf{poor}}$$

- ... heap consumption would be  $2|l|$ , but prediction would be  $\text{pot}_\sigma(l : \mathbf{rich}) = |l|$ .  $\Rightarrow$  unsound!
- The rule ( $\diamond\text{Share}$ ) enables multiple (sound) uses of a variable.

$$\frac{\Gamma, y_1 : D^{\mathbf{s}_1}, y_2 : D^{\mathbf{s}_2} \stackrel{n}{\vdash}_{n'} e : C^{\mathbf{r}} \quad \forall(\mathbf{s} \mid \mathbf{s}_1, \mathbf{s}_2)}{\Gamma, x : D^{\mathbf{s}} \stackrel{n}{\vdash}_{n'} e[x/y_1, x/y_2] : C^{\mathbf{r}}} \quad (\diamond\text{Share})$$

# Method invocation

- RAJA method types are polymorphic

- $M(C, m) = \phi$

$$\phi = \forall v_{\text{self}}, \vec{v}, \vec{q}. C^{v_{\text{self}}}; E_1^{v_1}, \dots, E_j^{v_j} \xrightarrow{q_1/q_2} E_{j+1}^{v_{j+1}} \ \& \ \mathcal{C}_m(v_{\text{self}}, \vec{v}, \vec{q})$$

- $(C^{\mathbf{s}_{\text{self}}}; E_1^{\mathbf{s}_1}, \dots, E_j^{\mathbf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathbf{s}_{j+1}})$  instance of  $\phi$  means

$$\pi = \{v_{\text{self}} \mapsto \mathbf{s}_{\text{self}}, v_i \mapsto \mathbf{s}_i, q_i \mapsto n_i\} \models \mathcal{C}_m$$

$$\frac{M(C, m) = \phi \quad (C^{\mathbf{s}_{\text{self}}}; E_1^{\mathbf{s}_1}, \dots, E_j^{\mathbf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathbf{s}_{j+1}}) \text{ instance of } \phi}{x : C^{\mathbf{s}_{\text{self}}}, y_1 : E_1^{\mathbf{s}_1}, \dots, y_j : E_j^{\mathbf{s}_j} \vdash_{n_1/n_2}^{\text{Sj}} x.m(y_1, \dots, y_j) : E_{j+1}^{\mathbf{s}_{j+1}}} \quad (\diamond \text{PolyInv.})$$

- RAJA Method Typing  $\vdash m : \phi$  ok

$$\exists T' \text{ instance of } \phi$$

$$\forall T \text{ instance of } \phi \quad T = C^{\mathbf{s}_{\text{self}}}; E_1^{\mathbf{s}_1}, \dots, E_j^{\mathbf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathbf{s}_{j+1}}$$

$$\text{this} : C^{\mathbf{s}_{\text{self}}}, x_1 : E_1^{\mathbf{s}_1}, \dots, x_j : E_j^{\mathbf{s}_j} \vdash_{n_1/n_2}^{\text{Sj}} M_{\text{body}}(C, m) : E_{j+1}^{\mathbf{s}_{j+1}}$$

$$\vdash m : \phi \text{ ok}$$

 $(\diamond \text{MBody})$

# Method invocation

- RAJA method types are polymorphic

- $M(C, m) = \phi$

$$\phi = \forall v_{\text{self}}, \vec{v}, \vec{q}. C^{v_{\text{self}}}; E_1^{v_1}, \dots, E_j^{v_j} \xrightarrow{q_1/q_2} E_{j+1}^{v_{j+1}} \ \& \ \mathcal{C}_m(v_{\text{self}}, \vec{v}, \vec{q})$$

- $(C^{\mathbf{s}_{\text{self}}}; E_1^{\mathbf{s}_1}, \dots, E_j^{\mathbf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathbf{s}_{j+1}})$  instance of  $\phi$  means

$$\pi = \{v_{\text{self}} \mapsto \mathbf{s}_{\text{self}}, v_i \mapsto \mathbf{s}_i, q_i \mapsto n_i\} \models \mathcal{C}_m$$

$$\frac{M(C, m) = \phi \quad (C^{\mathbf{s}_{\text{self}}}; E_1^{\mathbf{s}_1}, \dots, E_j^{\mathbf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathbf{s}_{j+1}}) \text{ instance of } \phi}{x : C^{\mathbf{s}_{\text{self}}}, y_1 : E_1^{\mathbf{s}_1}, \dots, y_j : E_j^{\mathbf{s}_j} \vdash_{\frac{n_1}{n_2}} x.m(y_1, \dots, y_j) : E_{j+1}^{\mathbf{s}_{j+1}}} \quad (\diamond \text{PolyInv.})$$

- RAJA Method Typing  $\vdash m : \phi$  ok

$$\exists T' \text{ instance of } \phi$$

$$\forall T \text{ instance of } \phi \quad T = C^{\mathbf{s}_{\text{self}}}; E_1^{\mathbf{s}_1}, \dots, E_j^{\mathbf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathbf{s}_{j+1}}$$

$$\text{this} : C^{\mathbf{s}_{\text{self}}}, x_1 : E_1^{\mathbf{s}_1}, \dots, x_j : E_j^{\mathbf{s}_j} \vdash_{\frac{n_1}{n_2}} M_{\text{body}}(C, m) : E_{j+1}^{\mathbf{s}_{j+1}}$$

$$\vdash m : \phi \text{ ok}$$

 $(\diamond \text{MBody})$

# Monomorphic Recursion in RAJA<sup>m</sup>

- RAJA Method Typing  $\vdash m : \phi$  ok

$$\begin{array}{c}
 \exists T' \text{ instance of } \phi \\
 \forall T \text{ instance of } \phi \quad T = C^{\text{self}}; E_1^{s_1}, \dots, E_j^{s_j} \xrightarrow{n_1/n_2} E_{j+1}^{s_{j+1}} \quad \Xi(C, m) = T \\
 \text{this} : C^{\text{self}}, x_1 : E_1^{s_1}, \dots, x_j : E_j^{s_j} \vdash_{n_1/n_2} M_{\text{body}}(C, m) : E_{j+1}^{s_{j+1}} \mid \Xi \\
 \hline
 \vdash m : \phi \text{ ok} \quad (\diamond \text{MBody})
 \end{array}$$

- $\Xi$  records the instance of  $\phi$  that is used for typechecking the method.

# Monomorphic Recursion in RAJA<sup>m</sup>

- RAJA Method Typing  $\vdash m : \phi$  ok

$$\begin{array}{c}
 \exists T' \text{ instanceof } \phi \\
 \forall T \text{ instanceof } \phi \quad T = C^{\text{self}}; E_1^{S_1}, \dots, E_j^{S_j} \xrightarrow{n_1/n_2} E_{j+1}^{S_{j+1}} \quad \Xi(C, m) = T \\
 \text{this} : C^{\text{self}}, x_1 : E_1^{S_1}, \dots, x_j : E_j^{S_j} \vdash_{n_1/n_2} M_{\text{body}}(C, m) : E_{j+1}^{S_{j+1}} \mid \Xi \\
 \hline
 \vdash m : \phi \text{ ok} \quad (\diamond \text{MBody})
 \end{array}$$

- $\Xi$  records the instance of  $\phi$  that is used for typechecking the method.
- Typing judgement:  $\Gamma \vdash_{n_1/n_2} e : C^r \mid \Xi$
- New rule in RAJA<sup>m</sup>

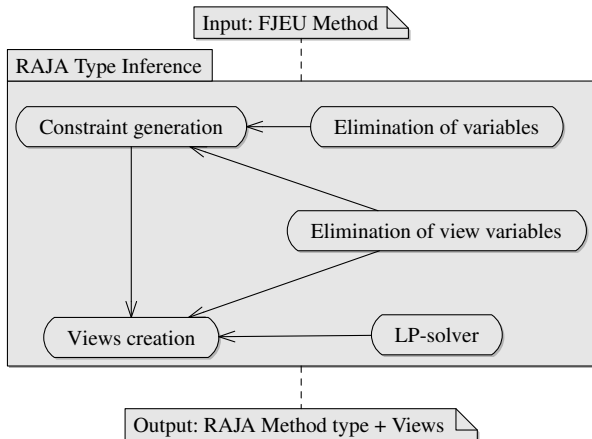
$$\begin{array}{c}
 M(C, m) = \phi \quad (C^{\text{self}}; E_1^{S_1}, \dots, E_j^{S_j} \xrightarrow{n_1/n_2} E_{j+1}^{S_{j+1}}) \text{ instanceof } \phi \\
 \Xi(C, m) = C^{\text{self}}; E_1^{S_1}, \dots, E_j^{S_j} \xrightarrow{n_1/n_2} E_{j+1}^{S_{j+1}} \\
 \hline
 x : C^{\text{self}}, y_1 : E_1^{S_1}, \dots, y_j : E_j^{S_j} \vdash_{n_1/n_2} x.m(y_1, \dots, y_j) : E_{j+1}^{S_{j+1}} \mid \Xi \quad (\diamond^m \text{MonInv.})
 \end{array}$$

## Subtyping of RAJA types

- We extend subtyping to RAJA-classes by

$$C^r <: D^s \iff C <: D \text{ and } r \sqsubseteq s \quad (2.1)$$

- We define a preorder  $r \sqsubseteq s$  on views as a largest fixpoint.
- $\Rightarrow$  Solving the constraint  $C^v <: D^u$  can be reduced to solving  $v \sqsubseteq u$ .

Overview of the Type Inference Algorithm for RAJA<sup>m</sup>

## Rules for collecting the constraints

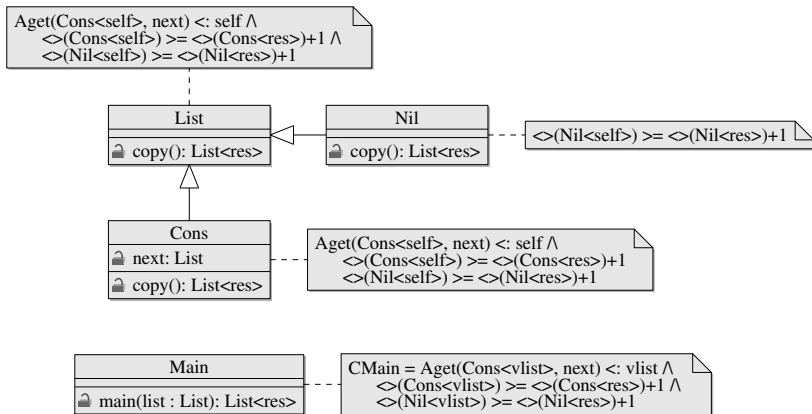
- Judgement  $\Gamma \frac{p_1}{p_2} e : C^v \& \mathcal{C} \mid M; \Xi$
- Syntax directed rules.
- Elimination of rules ( $\diamond$ Share), ( $\diamond$ Waste).
- Subtyping and sharing are integrated in the rules.

$$\frac{AC = p_1 \geq \diamond(D^{v_1}) + 1 \wedge p_2 \leq p_1 - \diamond(D^{v_1}) - 1}{\emptyset \frac{p_1}{p_2} \text{ new } D : C^{v_2} \& (D^{v_1} <: C^{v_2} \wedge AC) \mid M; \Xi} \quad (\vdash \text{New})$$

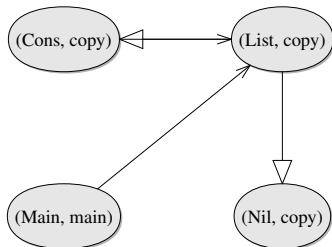
$$\mathcal{D}_i = (\Delta_{x_i}^{u_i} <: \Delta_{x_i}^{v_i + w_i}) \quad \mathcal{E} = (\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \bigwedge_i \mathcal{D}_i) \quad \mathcal{C} = \text{elim}_{\vec{v}, \vec{w}, u, t}(\mathcal{E})$$

$$\frac{\Delta^{\vec{v}} \frac{p_1}{t} e_1 \Leftarrow D^u \& \mathcal{C}_1 \mid M; \Xi \quad \Delta^{\vec{w}, x} : D^u \frac{t}{p_2} e_2 \Leftarrow C^v \& \mathcal{C}_2 \mid M; \Xi}{\Delta^{\vec{u}} \frac{p_1}{p_2} \text{ let } D x = e_1 \text{ in } e_2 \Leftarrow C^v \& \mathcal{C}(\vec{u}, v, \vec{p}) \mid M; \Xi} \quad (\vdash \text{Let})$$

# Constraints of the copy method

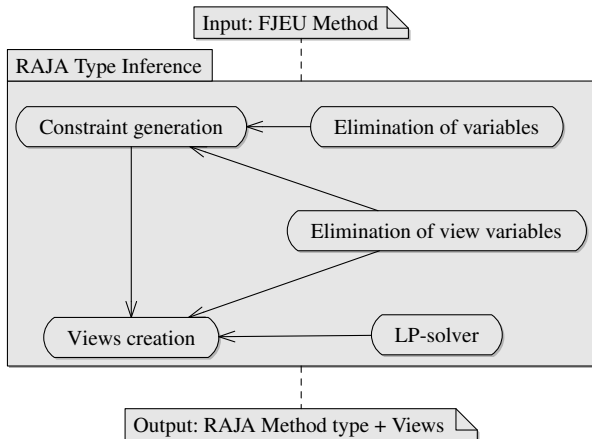


## Call graph of the copy example



- Partition the call graph in SCCs.
- Sort topologically the resulting dag.
- Call the constraint generation algorithm in that order.

# Overview of the Type Inference Algorithm for RAJA<sup>m</sup>



## Algorithmic views

- We add the connectives  $\wedge, \vee, +, \dot{-}$  to the set of views.  
 $\Rightarrow (\mathcal{V}, \sqsubseteq)$  is a partially ordered set.

### Lemma

- 1  $\mathbf{r} \wedge \mathbf{s}$  is the greatest lower bound of  $\mathbf{r}$  and  $\mathbf{s}$ .
- 2  $\mathbf{r} \vee \mathbf{s}$  is the least upper bound of  $\mathbf{r}$  and  $\mathbf{s}$ .
- 3  $s_1 \dot{-} s_2 \sqsubseteq r \iff s_1 \sqsubseteq r + s_2$ .
- 4  $s_1 \vee s_2 \sqsubseteq r \iff s_1 \sqsubseteq r \wedge s_2 \sqsubseteq r$ .
- 5  $r \sqsubseteq s_1 \wedge s_2 \iff r \sqsubseteq s_1 \wedge r \sqsubseteq s_2$ .

# Algorithm for eliminating a view variable from a constraint set.

- The constraints can be brought to the following form:

$$\begin{array}{llll}
 u_i & \sqsubseteq & v & \diamond(C^v) \geq p_m \\
 v & \sqsubseteq & w_j & q_n \geq \diamond(C^v) \\
 A^{\text{get}}(C^v, a_r) & \sqsubseteq & \hat{u}_r & t_s \geq t_o \\
 \hat{w}_t & \sqsubseteq & A^{\text{set}}(C^v, a_t) & \\
 \bar{w}_l & \sqsubseteq & \bar{u}_k & 
 \end{array}$$

- A step eliminating  $v$  ( $\mathcal{SC} \rightarrow_v \mathcal{SC}'$ ):

$$\begin{array}{llll}
 u_i & \sqsubseteq & w_j & \diamond(C^{u_i}) \geq p_m \\
 A^{\text{get}}(C^{u_i}, a_r) & \sqsubseteq & \hat{u}_r & q_n \geq \diamond(C^{w_j}) \\
 \hat{w}_t & \sqsubseteq & A^{\text{set}}(C^{u_i}, a_t) & q_n \geq p_m \\
 \bar{w}_l & \sqsubseteq & \bar{u}_k & t_s \geq t_o
 \end{array}$$

- If  $\mathcal{SC} \rightarrow_v \mathcal{SC}'$  then  $\mathcal{SC} \iff \exists v. \mathcal{SC}'$ .

# Conclusions and further work

- Conclusions
  - Our type-based analysis encompasses:
    - ① Objects
    - ② Inheritance
    - ③ Downcast
    - ④ Imperative update
    - ⑤ Aliasing
    - ⑥ Circular data
  - Type inference allow the analysis of heap-space requirements of FJEU programs without annotations.
- Further work
  - Implementation
  - More examples: Iterators on lists, etc.
  - Making the system more expressive.

# Conclusions and further work

- Conclusions
  - Our type-based analysis encompasses:
    - ① Objects
    - ② Inheritance
    - ③ Downcast
    - ④ Imperative update
    - ⑤ Aliasing
    - ⑥ Circular data
  - Type inference allow the analysis of heap-space requirements of FJEU programs without annotations.
- Further work
  - Implementation
  - More examples: Iterators on lists, etc.
  - Making the system more expressive.