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

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

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

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

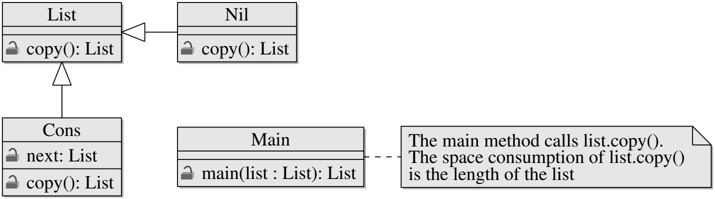
Lists in FJEU with copy method

```
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 = *annotated* FJEU program.

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

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

   - The linear arithmetic constraints are of the following shape:

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

   - The subtyping constraints are of the following shape:

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

Sketch of RAJA System

RAJA program P = *annotated* FJEU program.

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

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

   - The linear arithmetic constraints are of the following shape:

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

   - The subtyping constraints are of the following shape:

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

Sketch of RAJA System

RAJA program P = *annotated* FJEU program.

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

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

   - The linear arithmetic constraints are of the following shape:

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

   - The subtyping constraints are of the following shape:

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

Sketch of RAJA System

RAJA program P = *annotated* FJEU program.

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

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

   - The linear arithmetic constraints are of the following shape:

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

   - The subtyping constraints are of the following shape:

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

Sketch of RAJA System

RAJA program P = *annotated* FJEU program.

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

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

   - The linear arithmetic constraints are of the following shape:

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

   - The subtyping constraints are of the following shape:

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

Sketch of RAJA System

RAJA program P = *annotated* FJEU program.

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

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

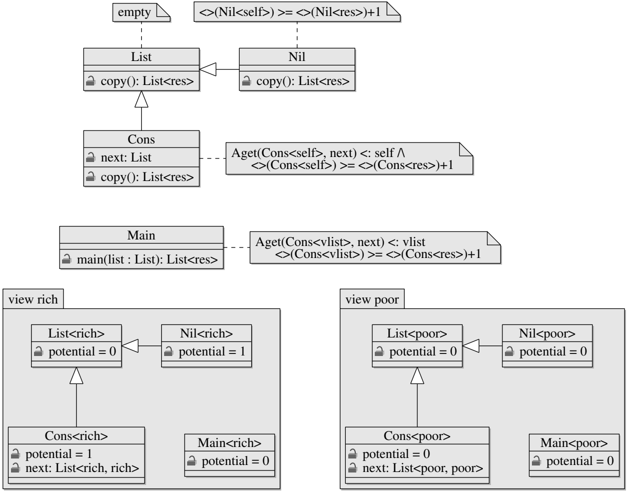
   - The linear arithmetic constraints are of the following shape:

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

   - The subtyping constraints are of the following shape:

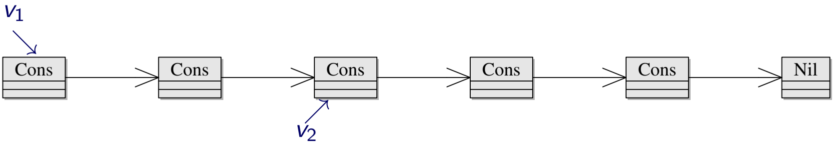
$$D^{v_1} <: C^{v_2} \qquad C.a^{\mathsf{A}^{\mathrm{get}}(C^{v_1},a)} <: D^{v_2} \qquad E^{v_2} <: C.a^{\mathsf{A}^{\mathrm{set}}(C^{v_1},a)}$$

Lists with copy method in RAJA



Potential

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

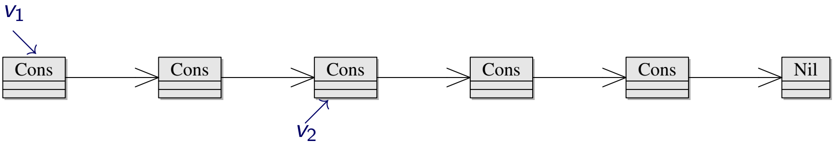


$\mathsf{pot}_\sigma(v_1 : \mathbf{rich}) = 6$ $\qquad$ $\mathsf{pot}_\sigma(v_2 : \mathbf{poor}) = 0$

$$\mathsf{pot}_\sigma(\eta : \Gamma) = \underbrace{\mathsf{pot}_\sigma(v_1 : \mathbf{rich})}_{6} + \underbrace{\mathsf{pot}_\sigma(v_2 : \mathbf{poor})}_{0} = 6$$

Potential

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

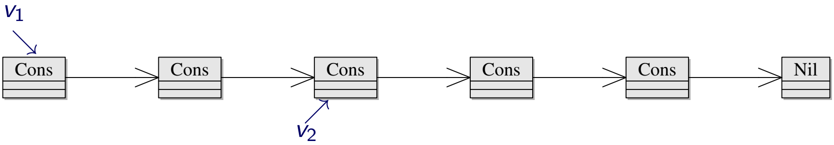


$\mathsf{pot}_\sigma(v_1 : \mathbf{rich}) = 6$ $\qquad$ $\mathsf{pot}_\sigma(v_2 : \mathbf{poor}) = 0$

$$\mathsf{pot}_\sigma(\eta : \Gamma) = \underbrace{\mathsf{pot}_\sigma(v_1 : \mathbf{rich})}_{6} + \underbrace{\mathsf{pot}_\sigma(v_2 : \mathbf{poor})}_{0} = 6$$

Potential

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



$\mathsf{pot}_\sigma(v_1 : \mathbf{rich}) = 6$ $\mathsf{pot}_\sigma(v_2 : \mathbf{poor}) = 0$

$$\mathsf{pot}_\sigma(\eta : \Gamma) = \underbrace{\mathsf{pot}_\sigma(v_1 : \mathbf{rich})}_{6} + \underbrace{\mathsf{pot}_\sigma(v_2 : \mathbf{poor})}_{0} = 6$$

Main result

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

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

RAJA typing

- Typing judgement: $\Gamma \vdash^{n_1}_{n_2} e : C^{\mathsf{r}}$

$$\frac{}{\emptyset \vdash^{1+\Diamond(C^r)}_{0} \mathsf{new}\ C : C^{\mathsf{r}}}\ (\Diamond New) \qquad \frac{}{x : C^{\mathsf{r}} \vdash^{0}_{1+\Diamond(C^r)} \mathsf{free}\,(x) : E^{\mathsf{s}}}\ (\Diamond Free)$$

$$\frac{\Gamma_1 \vdash^{n}_{n'} e_1 : D^{\mathsf{s}} \qquad \Gamma_2, x : D^{\mathsf{s}} \vdash^{n'}_{n''} e_2 : C^{\mathsf{r}}}{\Gamma_1, \Gamma_2 \vdash^{n}_{n''} \mathit{let}\ D\, x = e_1\ \mathit{in}\ e_2 : C^{\mathsf{r}}}\ (\Diamond Let)$$

Sharing relation

- $\curlyvee(\mathsf{r} \,|\, \mathsf{s}_1, \ldots, \mathsf{s}_i)$: coinductively defined relation with:

$$\text{for all } C, \quad \Diamond(C^{\mathsf{r}}) \geq \Diamond(C^{\mathsf{s}_1}) + \ldots + \Diamond(C^{\mathsf{s}_i}), \quad \text{etc.}$$

- If this were allowed ...

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

- ... heap consumption would be $2|l|$, but prediction would be $\mathsf{pot}_\sigma(l : \mathbf{rich}) = |l|$. $\Rightarrow$ unsound!

- The rule $(\Diamond \mathit{Share})$ enables multiple (sound) uses of a variable.

$$\frac{\Gamma, y_1 : D^{\mathsf{s}_1}, y_2 : D^{\mathsf{s}_2} \vdash^{n}_{n'} e : C^{\mathsf{r}} \qquad \curlyvee(\mathsf{s} \,|\, \mathsf{s}_1, \mathsf{s}_2)}{\Gamma, x : D^{\mathsf{s}} \vdash^{n}_{n'} e[x/y_1, x/y_2] : C^{\mathsf{r}}} \ (\Diamond \mathit{Share})$$

Sharing relation

- $\curlyvee(\mathsf{r}\,|\,\mathsf{s_1},\ldots,\mathsf{s_i})$: coinductively defined relation with:

$$\text{for all } C\,,\quad \Diamond(C^{\mathsf{r}}) \geq \Diamond(C^{\mathsf{s_1}}) + \ldots + \Diamond(C^{\mathsf{s_i}}), \quad \text{etc.}$$

- $\curlyvee(\mathbf{rich}\,|\,\mathbf{rich}, \mathbf{poor})$ ✓
- If this were allowed ...

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

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

$$\frac{\Gamma, y_1 : D^{\mathsf{s_1}}, y_2 : D^{\mathsf{s_2}} \vdash^{n}_{n'} e : C^{\mathsf{r}} \qquad \curlyvee(\mathsf{s}\,|\,\mathsf{s_1}, \mathsf{s_2})}{\Gamma, x : D^{\mathsf{s}} \vdash^{n}_{n'} e[x/y_1, x/y_2] : C^{\mathsf{r}}}\ (\Diamond \mathit{Share})$$

Sharing relation

- $\curlyvee(\mathsf{r} \mid \mathsf{s_1}, \ldots, \mathsf{s_i})$: coinductively defined relation with:

  $$\text{for all } C, \quad \Diamond(C^{\mathsf{r}}) \geq \Diamond(C^{\mathsf{s_1}}) + \ldots + \Diamond(C^{\mathsf{s_i}}), \quad \text{etc.}$$

- $\curlyvee(\mathsf{rich} \mid \mathsf{rich}, \mathsf{poor})$ ✓ $\qquad \curlyvee(\mathsf{rich} \mid \mathsf{rich}, \mathsf{rich})$ ✗
- If this were allowed ...

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

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

  $$\frac{\Gamma, y_1 : D^{\mathsf{s_1}}, y_2 : D^{\mathsf{s_2}} \vdash^{n}_{n'} e : C^{\mathsf{r}} \qquad \curlyvee(\mathsf{s} \mid \mathsf{s_1}, \mathsf{s_2})}{\Gamma, x : D^{\mathsf{s}} \vdash^{n}_{n'} e[x/y_1, x/y_2] : C^{\mathsf{r}}} \ (\Diamond \mathit{Share})$$

Sharing relation

- $\curlyvee(\mathsf{r}\,|\,\mathsf{s}_1, \ldots, \mathsf{s}_i)$: coinductively defined relation with:

  for all C, $\quad \Diamond(C^{\mathsf{r}}) \geq \Diamond(C^{\mathsf{s}_1}) + \ldots + \Diamond(C^{\mathsf{s}_i})$, $\quad$ etc.

- $\curlyvee(\mathbf{rich}\,|\,\mathbf{rich}, \mathbf{poor})$ ✓ $\qquad \curlyvee(\mathbf{rich}\,|\,\mathbf{rich}, \mathbf{rich})$ ✗
- If this were allowed ...

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

- ... heap consumption would be $2|l|$, but prediction would be $\mathsf{pot}_\sigma(l : \mathbf{rich}) = |l|$. $\Rightarrow$ unsound!

- The rule $(\Diamond Share)$ enables multiple (sound) uses of a variable.

$$\frac{\Gamma, y_1 : D^{\mathsf{s}_1}, y_2 : D^{\mathsf{s}_2} \vdash^{n}_{n'} e : C^{\mathsf{r}} \qquad \curlyvee(\mathsf{s}\,|\,\mathsf{s}_1, \mathsf{s}_2)}{\Gamma, x : D^{\mathsf{s}} \vdash^{n}_{n'} e[x/y_1, x/y_2] : C^{\mathsf{r}}}\ (\Diamond Share)$$

Sharing relation

- $\curlyvee(\mathsf{r}\,|\,\mathsf{s}_1,\ldots,\mathsf{s}_i)$: coinductively defined relation with:

$$\text{for all } C, \quad \Diamond(C^{\mathsf{r}}) \geq \Diamond(C^{\mathsf{s}_1}) + \ldots + \Diamond(C^{\mathsf{s}_i}), \quad \text{etc.}$$

- $\curlyvee(\mathsf{rich}\,|\,\mathsf{rich},\mathsf{poor})$ ✓ $\quad \curlyvee(\mathsf{rich}\,|\,\mathsf{rich},\mathsf{rich})$ ✗
- If this were allowed ...

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

- ... heap consumption would be $2|l|$, but prediction would be $\mathsf{pot}_\sigma(l : \mathsf{rich}) = |l|$. $\Rightarrow$ unsound!

- The rule $(\Diamond \mathit{Share})$ enables multiple (sound) uses of a variable.

$$\frac{\Gamma, y_1 : D^{\mathsf{s}_1}, y_2 : D^{\mathsf{s}_2} \vdash^{n}_{n'} e : C^{\mathsf{r}} \qquad \curlyvee(\mathsf{s}\,|\,\mathsf{s}_1,\mathsf{s}_2)}{\Gamma, x : D^{\mathsf{s}} \vdash^{n}_{n'} e[x/y_1, x/y_2] : C^{\mathsf{r}}}\ (\Diamond \mathit{Share})$$

Sharing relation

- $\curlyvee(\mathsf{r} \mid \mathsf{s_1}, \ldots, \mathsf{s_i})$: coinductively defined relation with:

$$\text{for all } C, \quad \Diamond(C^{\mathsf{r}}) \geq \Diamond(C^{\mathsf{s_1}}) + \ldots + \Diamond(C^{\mathsf{s_i}}), \quad \text{etc.}$$

- $\curlyvee(\mathsf{rich} \mid \mathsf{rich}, \mathsf{poor})$ ✓ $\quad \curlyvee(\mathsf{rich} \mid \mathsf{rich}, \mathsf{rich})$ ✗
- If this were allowed ...

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

- ... heap consumption would be $2|l|$, but prediction would be $\mathsf{pot}_\sigma(l : \mathsf{rich}) = |l|$. $\Rightarrow$ unsound!

- The rule $(\Diamond \mathit{Share})$ enables multiple (sound) uses of a variable.

$$\frac{\Gamma, y_1 : D^{\mathsf{s_1}}, y_2 : D^{\mathsf{s_2}} \vdash^{n}_{n'} e : C^{\mathsf{r}} \qquad \curlyvee(\mathsf{s} \mid \mathsf{s_1}, \mathsf{s_2})}{\Gamma, x : D^{\mathsf{s}} \vdash^{n}_{n'} e[x/y_1, x/y_2] : C^{\mathsf{r}}} \ (\Diamond \mathit{Share})$$

Method invocation

- RAJA method types are polymorphic
- $\mathsf{M}(C, m) = \phi$

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

- $(C^{\mathsf{s_{self}}} ; E_1^{\mathsf{s_1}}, \ldots, E_j^{\mathsf{s_j}} \xrightarrow{n_1/n_2} E_{j+1}^{\mathsf{s_{j+1}}})$ $\mathsf{instanceof}\ \phi$ means

$$\pi = \{v_{\mathsf{self}} \mapsto \mathsf{s_{self}}, v_i \mapsto \mathsf{s}_i, q_i \mapsto n_i\} \models \mathscr{C}_m$$

$$\frac{\mathsf{M}(C, m) = \phi \qquad (C^{\mathsf{s_{self}}} ; E_1^{\mathsf{s_1}}, \ldots, E_j^{\mathsf{s_j}} \xrightarrow{n_1/n_2} E_{j+1}^{\mathsf{s_{j+1}}})\ \mathsf{instanceof}\ \phi}{x : C^{\mathsf{s_{self}}}, y_1 : E_1^{\mathsf{s_1}}, \ldots, y_j : E_j^{\mathsf{s_j}} \vdash^{n_1}_{n_2} x.m(y_1, \ldots, y_j) : E_{j+1}^{\mathsf{s_{j+1}}}} \ (\Diamond \mathit{PolyInv.})$$

- RAJA Method Typing $\vdash m : \phi\ \ \mathsf{ok}$

$$\frac{\exists\, T'\ \mathsf{instanceof}\ \phi \qquad \forall\, T\ \mathsf{instanceof}\ \phi \quad T = C^{\mathsf{s_{self}}} ; E_1^{\mathsf{s_1}}, \ldots, E_j^{\mathsf{s_j}} \xrightarrow{n_1/n_2} E_{j+1}^{\mathsf{s_{j+1}}} \qquad \mathtt{this} : C^{\mathsf{s_{self}}}, x_1 : E_1^{\mathsf{s_1}}, \ldots, x_j : E_j^{\mathsf{s_j}} \vdash^{n_1}_{n_2} \mathsf{M_{body}}(C, m) : E_{j+1}^{\mathsf{s_{j+1}}}}{\vdash m : \phi\ \ \mathsf{ok}} \ (\Diamond \mathit{MBody})$$

Method invocation

- RAJA method types are polymorphic
- $\mathsf{M}(C, m) = \phi$

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

- $(C^{\mathsf{s}_{\mathsf{self}}}; E_1^{\mathsf{s}_1}, \ldots, E_j^{\mathsf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathsf{s}_{j+1}})$ $\mathsf{instanceof}$ ϕ means

$$\pi = \{v_{\mathsf{self}} \mapsto \mathsf{s}_{\mathsf{self}}, v_i \mapsto \mathsf{s}_i, q_i \mapsto n_i\} \models \mathscr{C}_m$$

$$\frac{\mathsf{M}(C, m) = \phi \qquad (C^{\mathsf{s}_{\mathsf{self}}}; E_1^{\mathsf{s}_1}, \ldots, E_j^{\mathsf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathsf{s}_{j+1}}) \;\mathsf{instanceof}\; \phi}{x : C^{\mathsf{s}_{\mathsf{self}}}, y_1 : E_1^{\mathsf{s}_1}, \ldots, y_j : E_j^{\mathsf{s}_j} \vdash^{n_1}_{n_2} x.m(y_1, \ldots, y_j) : E_{j+1}^{\mathsf{s}_{j+1}}} \;(\Diamond PolyInv.)$$

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

$$\frac{\begin{array}{c} \exists\, T' \;\mathsf{instanceof}\; \phi \\ \forall\, T \;\mathsf{instanceof}\; \phi \qquad T = C^{\mathsf{s}_{\mathsf{self}}}; E_1^{\mathsf{s}_1}, \ldots, E_j^{\mathsf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathsf{s}_{j+1}} \\ \mathtt{this} : C^{\mathsf{s}_{\mathsf{self}}}, x_1 : E_1^{\mathsf{s}_1}, \ldots, x_j : E_j^{\mathsf{s}_j} \vdash^{n_1}_{n_2} \mathsf{M}_{\mathsf{body}}(C, m) : E_{j+1}^{\mathsf{s}_{j+1}} \end{array}}{\vdash m : \phi \;\; \mathsf{ok}} \;(\Diamond MBody)$$

Monomorphic Recursion in RAJA$^{\mathrm{m}}$

- RAJA Method Typing $\vdash m : \phi\ \ \mathsf{ok}$

$$\frac{\forall\, T \text{ instanceof } \phi \qquad \begin{array}{c}\exists\, T' \text{ instanceof } \phi \\ T = C^{\mathsf{s}_{\mathsf{self}}}; E_1^{\mathsf{s}_1}, \ldots, E_j^{\mathsf{s}_j} \xrightarrow{n_1 / n_2} E_{j+1}^{\mathsf{s}_{j+1}}\end{array} \qquad \Xi(C, m) = T \qquad \\ \mathtt{this} : C^{\mathsf{s}_{\mathsf{self}}}, x_1 : E_1^{\mathsf{s}_1}, \ldots, x_j : E_j^{\mathsf{s}_j} \vdash^{n_1}_{n_2} \mathsf{M}_{\mathsf{body}}(C, m) : E_{j+1}^{\mathsf{s}_{j+1}} \mid \Xi}{\vdash m : \phi\ \ \mathsf{ok}} \ (\Diamond MBody)$$

- Ξ records the instance of ϕ that is used for typechecking the method.

Monomorphic Recursion in RAJA$^{\mathrm{m}}$

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

$$\frac{\forall\, T \text{ instanceof } \phi \qquad \begin{array}{c}\exists\, T' \text{ instanceof } \phi\\ T = C^{\mathsf{s}_{\mathsf{self}}}; E_1^{\mathsf{s}_1}, \ldots, E_j^{\mathsf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathsf{s}_{j+1}}\end{array} \qquad \Xi(C, m) = T \qquad \mathtt{this} : C^{\mathsf{s}_{\mathsf{self}}}, x_1 : E_1^{\mathsf{s}_1}, \ldots, x_j : E_j^{\mathsf{s}_j} \vdash^{n_1}_{n_2} \mathsf{M_{body}}(C, m) : E_{j+1}^{\mathsf{s}_{j+1}} \mid \Xi}{\vdash m : \phi \ \text{ ok}} \ (\Diamond MBody)$$

- Ξ records the instance of ϕ that is used for typechecking the method.

- Typing judgement: $\Gamma \vdash^{n_1}_{n_2} e : C^{\mathsf{r}} \mid \Xi$
- New rule in RAJA$^{\mathrm{m}}$

$$\frac{\mathsf{M}(C, m) = \phi \qquad (C^{\mathsf{s}_{\mathsf{self}}}; E_1^{\mathsf{s}_1}, \ldots, E_j^{\mathsf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathsf{s}_{j+1}}) \text{ instanceof } \phi \qquad \Xi(C, m) = C^{\mathsf{s}_{\mathsf{self}}}; E_1^{\mathsf{s}_1}, \ldots, E_j^{\mathsf{s}_j} \xrightarrow{n_1/n_2} E_{j+1}^{\mathsf{s}_{j+1}}}{x : C^{\mathsf{s}_{\mathsf{self}}}, y_1 : E_1^{\mathsf{s}_1}, \ldots, y_j : E_j^{\mathsf{s}_j} \vdash^{n_1}_{n_2} x.m(y_1, \ldots, y_j) : E_{j+1}^{\mathsf{s}_{j+1}} \mid \Xi} \ (\Diamond^m MonInv.)$$

Subtyping of RAJA types

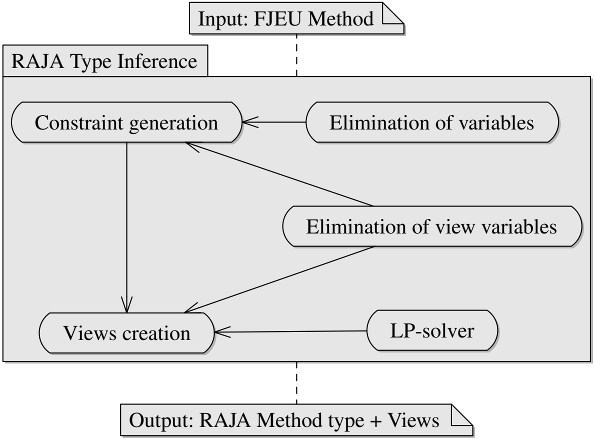
- We extend subtyping to RAJA-classes by

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

- 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^m



Rules for collecting the constraints

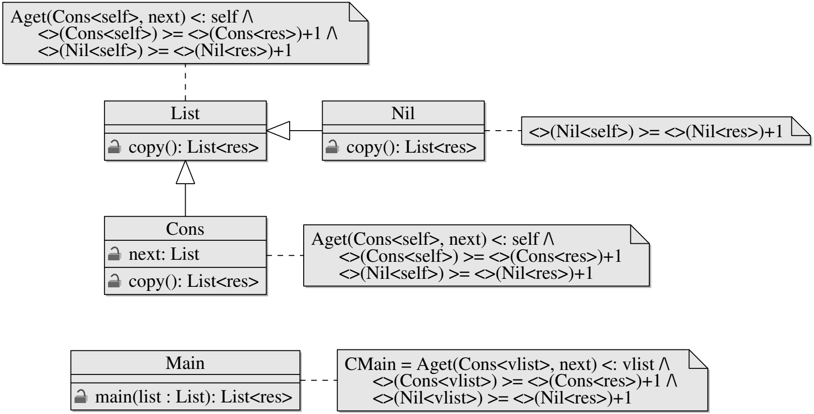
- Judgement $\Gamma \vdash^{p_1}_{p_2} e : C^{\mathsf{v}} \,\&\, \mathscr{C} \mid \mathsf{M}; \Xi$
- Syntax directed rules.
- Elimination of rules $(\Diamond Share)$, $(\Diamond Waste)$.
- Subtyping and sharing are integrated in the rules.

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

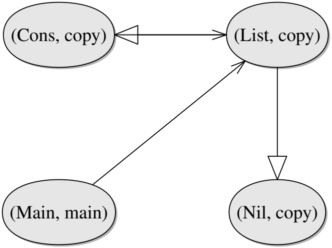
$$\mathcal{D}_i = (\Delta^{u_i}_{x_i} <: \Delta^{v_i + w_i}_{x_i}) \qquad \mathcal{E} = (\mathscr{C}_1 \ \wedge\ \mathscr{C}_2 \ \wedge\ \textstyle\bigwedge_i \mathcal{D}_i) \qquad \mathscr{C} = \mathsf{elim}_{\tilde{\mathsf{v}},\tilde{\mathsf{w}},\mathsf{u},\mathsf{t}}(\mathcal{E})$$

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

Constraints of the copy method

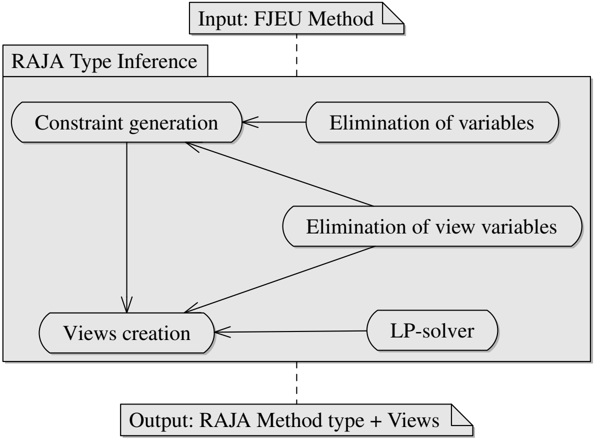


Call graph of the copy example



- 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^m



Algorithmic views

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

Lemma

1. $\mathsf{r} \wedge \mathsf{s}$ *is the greatest lower bound of* r *and* s.
2. $\mathsf{r} \vee \mathsf{s}$ *is the least upper bound of* r *and* 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:

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

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

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

- If $\mathcal{SC} \to_v \mathcal{SC}'$ 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.