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 ($\text{RAJA}^m$)

3. $\text{RAJA}^m$ 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

\[
c ::= \text{class } C \text{ [extends } D \text{] } \{ A_1; \ldots; A_k; M_1 \cdots M_j \}
\]

\[
A ::= C \ a
\]

\[
M ::= C_0 \ m(C_1 \ x_1, \ldots, C_j \ x_j) \{ \text{return } e; \}
\]

\[
e ::= x \quad \text{(Variable)}
\]

\[
null \quad \text{(Constant)}
\]

\[
\text{new } C \quad \text{(Construction)}
\]

\[
\text{free } (x) \quad \text{(Destruction)}
\]

\[
(C)x \quad \text{(Cast)}
\]

\[
x.a_i \quad \text{(Access)}
\]

\[
x.a_i \leftarrow x \quad \text{(Update)}
\]

\[
x.m(x_1, \ldots, x_j) \quad \text{(Invocation)}
\]

\[
\text{if } x \text{ instanceof } C \text{ then } e_1 \text{ else } e_2 \quad \text{(Conditional)}
\]

\[
\text{let } D \ x = e_1 \text{ in } e_2 \quad \text{(Let)}
\]

\[ \approx \text{ Featherweight Java (Igarashi, Pierce, Wadler, OOPSLA’99) + impera}
\text{tive field update.} \]
Lists in FJEU with copy method

```java
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;
    }
}
```

The main method calls list.copy(). The space consumption of list.copy() is the length of the list.
Sketch of RAJA System

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

1. Set of \textit{views} $V$.
2. For each class $C$ and view $r$ we have an annotated version $C^r$.
3. Potential:
   \begin{itemize}
   \item $\diamondsuit(C^r) : \text{Class} \times \text{View} \rightarrow \mathbb{Q}^+$
   \end{itemize}
4. (Get- and set-) views for attributes:
   \begin{itemize}
   \item $A^{\text{get}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$ (get-view)
   \item $A^{\text{set}}(C^r, a) : \text{Class} \times \text{View} \times \text{Field} \rightarrow \text{View}$ (set-view)
   \end{itemize}
5. RAJA types for methods:
   \begin{itemize}
   \item $M(\cdot, \cdot) : \text{Class} \times \text{Meth} \rightarrow \text{PolyType}$
     \begin{equation*}
     \phi = \forall v_{\text{self}}, \bar{v}, \bar{q} . \ C^{v_{\text{self}}}; E^1_{v_1}, \ldots, E^j_{v_j} \xrightarrow{q_1/q_2} E^{v_{j+1}}_{v_{j+1}} \& C_m(v_{\text{self}}, \bar{v}, \bar{q})
     \end{equation*}
   \end{itemize}
   - The linear arithmetic constraints are of the following shape:
     \begin{align*}
     p_1 & \geq \diamondsuit(D^v) + 1 \\
     p_2 & \leq p_1 - \diamondsuit(D^v) - 1 \\
     p_2 & \leq q_2 + p_1 - q_1
     \end{align*}
   - The subtyping constraints are of the following shape:
     \begin{align*}
     D^{v_1} & < : C^{v_2} \\
     C.a^{A^{\text{get}}(C^{v_1}, a)} & < : D^{v_2} \\
     E^{v_2} & < : C.a^{A^{\text{set}}(C^{v_1}, a)}
     \end{align*}
Sketch of RAJA System

RAJA program $P = \text{annotated}$ 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:
   - $M(\cdot, \cdot) : \text{Class} \times \text{Meth} \rightarrow \text{PolyType}$
     $$\phi = \forall v_{\text{self}}, \bar{v}, \bar{q}. \ C^{v_{\text{self}}}; E_1^{v_1}, \ldots, E_j^{v_j} \frac{q_1/q_2}{\rightarrow} E_{j+1}^{v_{j+1}} \& C_m(v_{\text{self}}, \bar{v}, \bar{q})$$
   - 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 $P = \text{annotated}$ 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:
   \[ \diamondsuit(C^r) : \text{Class} \times \text{View} \rightarrow \mathbb{Q}^+ \]
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:
   \[ M(\cdot, \cdot) : \text{Class} \times \text{Meth} \rightarrow \text{PolyType} \]
   \[ \phi = \forall v_{\text{self}}, \bar{v}, \bar{q}. C^{v_{\text{self}}}; E_1^{v_1}, \ldots, E_j^{v_j} \xrightarrow{q_1/q_2} E_{j+1}^{v_{j+1}} \& C_{m}(v_{\text{self}}, \bar{v}, \bar{q}) \]
   The linear arithmetic constraints are of the following shape:
   \[ p_1 \geq \diamondsuit(D^{v}) + 1 \quad p_2 \leq p_1 - \diamondsuit(D^{v}) - 1 \quad p_2 \leq q_2 + p_1 - q_1 \]
   The subtyping constraints are of the following shape:
   \[ D^{v_1} <: C^{v_2} \quad C.a^{A_{\text{get}}(C^{v_1}, a)} <: D^{v_2} \quad E^{v_2} <: C.a^{A_{\text{set}}(C^{v_1}, a)} \]
Introduction to RAJA

Sketch of RAJA System

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

1. Set of \textit{views} $\mathcal{V}$.
2. For each class $C$ and view $r$ we have an annotated version $C^r$.
3. Potential:
   \begin{align*}
   \diamondsuit (C^r) : \text{Class} \times \text{View} & \rightarrow \mathbb{Q}^+ \\
   \end{align*}
4. (Get- and set-) views for attributes:
   \begin{align*}
   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)} \\
   \end{align*}
5. RAJA types for methods:
   \begin{align*}
   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}, \ldots, E_j^{v_j} & \xrightarrow{q_1/q_2} E_{j+1}^{v_{j+1}} \ & \& C_m(v_{\text{self}}, \vec{v}, \vec{q}) \\
   \end{align*}
   - The linear arithmetic constraints are of the following shape:
     \begin{align*}
     p_1 & \geq \diamondsuit (D^v) + 1 \\
     p_2 & \leq p_1 - \diamondsuit (D^v) - 1 \\
     p_2 & \leq q_2 + p_1 - q_1 \\
     \end{align*}
   - The subtyping constraints are of the following shape:
     \begin{align*}
     D^{v_1} & <: C^{v_2} \\
     C.a^{A^{\text{get}}(C^{v_1}, a)} & <: D^{v_2} \\
     E^{v_2} & <: C.a^{A^{\text{set}}(C^{v_1}, a)} \\
     \end{align*}
Sketch of RAJA System

RAJA program $P = \text{annotated}$ 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:
   - $M(\cdot, \cdot) : \text{Class} \times \text{Meth} \rightarrow \text{PolyType}$
     $$\phi = \forall v_{\text{self}}, \vec{v}, \vec{q}. C_{v_{\text{self}}} ; E_1^{v_1}, \ldots, E_j^{v_j} \frac{q_1/q_2}{q_2} E_{j+1}^{v_j+1} \& C_m(v_{\text{self}}, \vec{v}, \vec{q})$$
   - 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:
   $\diamondsuit(C^r) : \text{Class} \times \text{View} \rightarrow \mathbb{Q}^+$
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:
   - $M(\cdot, \cdot) : \text{Class} \times \text{Meth} \rightarrow \text{PolyType}$
     $\phi = \forall v_{\text{self}}, \vec{v}, \vec{q} \cdot C^{v_{\text{self}}}; E_1^{v_1}, \ldots, E_j^{v_j} \xrightarrow{q_1/q_2} E_{j+1}^{v_{j+1}} \& C_m(v_{\text{self}}, \vec{v}, \vec{q})$
   - The linear arithmetic constraints are of the following shape:
     $p_1 \geq \diamondsuit(D^v) + 1 \quad p_2 \leq p_1 - \diamondsuit(D^v) - 1 \quad p_2 \leq q_2 + p_1 - q_1$
   - 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

Aget(Cons<self>, next) <: self /
<>(Cons<self>) >= <>(Cons<res>)+1
<>(Nil<self>) >= <>(Nil<res>)+1

List

Cons

next: List

copy(): List<res>

Nil

copy(): List<res>

empty

Aget(Cons<vlist>, next) <: vlist
<>(Cons<vlist>) >= <>(Cons<res>)+1

Main

main(list : List): List<res>

view rich

List<rich>

potential = 0

Cons<rich>

potential = 1

next: List<rich, rich>

Main<rich>

potential = 0

view poor

List<poor>

potential = 0

Cons<poor>

potential = 0

next: List<poor, poor>

Main<poor>

potential = 0

Dulma Rodriguez, Martin Hofmann: Type Inference f. 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:

$\begin{align*}
\text{pot}_\sigma(v_1 : \text{rich}) &= 6 \\
\text{pot}_\sigma(v_2 : \text{poor}) &= 0
\end{align*}$

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

$v_1$

\begin{align*}
\text{Cons} & \rightarrow \text{Cons} & \rightarrow \text{Cons} & \rightarrow \text{Cons} & \rightarrow \text{Cons} & \rightarrow \text{Nil} \\
\downarrow & & & & & \\
\text{Cons} & & & & & \\
\uparrow & & & & & \\
v_2 & & & & & \\
\end{align*}

\begin{align*}
\text{pot}_\sigma(v_1: \text{rich}) &= 6 & \text{pot}_\sigma(v_2: \text{poor}) &= 0 \\
\end{align*}

\begin{align*}
\text{pot}_\sigma(\eta: \Gamma) &= \text{pot}_\sigma(v_1: \text{rich}) + \text{pot}_\sigma(v_2: \text{poor}) \\
&= 6 + 0 \\
&= 6
\end{align*}
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:

\[
pot_\sigma(v_1 : \text{rich}) = 6 \quad \text{pot}_\sigma(v_2 : \text{poor}) = 0
\]

\[
pot_\sigma(\eta : \Gamma) = \underbrace{\text{pot}_\sigma(v_1 : \text{rich})}_6 + \underbrace{\text{pot}_\sigma(v_2 : \text{poor})}_0 = 6
\]
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 \( \Gamma \mid n \vdash_{n'} e : C^r \) so that:

- if \( \Gamma \mid n \vdash_{n'} e : C^r \) and \( \eta, \sigma \vdash e \rightsquigarrow v, \tau \) then

\[
e \text{ evaluates } \checkmark \quad \text{if} \quad |\text{free-list}| \geq n + \text{pot}_\sigma(\eta : \Gamma)
\]
RAJA typing

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

\[
\begin{align*}
\emptyset \vdash_{0}^1 1 + \Diamond(C^r) & \quad \text{(\Diamond New)} \\
\quad \quad \text{new } C : C^r \\
\hline
\Gamma_1 \vdash_{n}^{n'} e_1 : D^s & \quad \Gamma_2, x : D^s \vdash_{n'}^{n''} e_2 : C^r \\
\hline
\Gamma_1, \Gamma_2 \vdash_{n''}^{n'} \text{let } D x = e_1 \text{ in } e_2 : C^r & \quad \text{(\Diamond Let)}
\end{align*}
\]

\[
\begin{align*}
x : C^r \vdash_0^1 1 + \Diamond(C^r) & \quad \text{(\Diamond Free)} \\
\quad \quad \text{free (x) : } E^s
\end{align*}
\]
Sharing relation

- $\forall (r | s_1, \ldots, s_i)$: coinductively defined relation with:
  
  for all $C$, $\Diamond(C^r) \geq \Diamond(C^{s_1}) + \ldots + \Diamond(C^{s_i})$, etc.

- If this were allowed ...

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

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

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

$$\Gamma, y_1 : D^{s_1}, y_2 : D^{s_2} \vdash_{n'}^{n} e : C^r \quad \forall (s | s_1, s_2) \quad (\Diamond Share)$$
Introduction to RAJA

Sharing relation

- \( \forall (r \mid s_1, \ldots, s_i) \): coinductively defined relation with:
  
  \[
  \text{for all } C, \quad \Diamond(C^r) \geq \Diamond(C^{s_1}) + \ldots + \Diamond(C^{s_i}), \quad \text{etc.}
  \]

- \( \forall (\text{rich} \mid \text{rich, poor}) \)

  If this were allowed ...

  \[
  l : \text{List}^{\text{rich}} \vdash_0 \quad \text{let } _\_ = l.\text{copy()} \text{ in } l.\text{copy()} : \text{List}^{\text{poor}}
  \]

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

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

\[
\begin{align*}
\Gamma, y_1 : D^{s_1}, y_2 : D^{s_2} \vdash^n e : C^r & \quad \forall (s \mid s_1, s_2) \quad (\Diamond \text{Share}) \\
\Gamma, x : D^s \vdash^n e[x/y_1, x/y_2] : C^r
\end{align*}
\]
Sharing relation

- \( \forall (r \mid s_1, \ldots, s_i) \): coinductively defined relation with:
  
  for all \( C \), \( \diamond(C^r) \geq \diamond(C^{s_1}) + \ldots + \diamond(C^{s_i}) \), etc.

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

- If this were allowed...

  \( l : \text{List}^{\text{rich}} \vdash^0_0 \text{let } _ = l.\text{copy()} \text{ in } l.\text{copy()} : \text{List}^{\text{poor}} \)

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

- The rule \( (\diamond \text{Share}) \) enables multiple (sound) uses of a variable.

\[
\Gamma, y_1 : D^{s_1}, y_2 : D^{s_2} \vdash^n_{n'} e : C^r \quad \forall(s \mid s_1, s_2) \quad (\diamond \text{Share})
\]

\[
\Gamma, x : D^s \vdash^n_{n'} e[x/y_1, x/y_2] : C^r
\]
Sharing relation

- $\forall (r | s_1, \ldots, s_i)$: coinductively defined relation with:
  
  $\forall (r | s_1, \ldots, s_i): coinductively defined relation with$:
  
  for all $C$, $\Diamond(C^r) \geq \Diamond(C^{s_1}) + \ldots + \Diamond(C^{s_i})$, etc.

- $\forall (\text{rich} | \text{rich, poor}) \checkmark \quad \forall (\text{rich} | \text{rich, rich}) \times$

- If this were allowed ...
  
  $l : \text{List}^{\text{rich}} \vdash_0 \text{let } _ = l.\text{copy()} \text{ in } l.\text{copy()} : \text{List}^{\text{poor}}$

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

- The rule ($\Diamond \text{Share}$) enables multiple (sound) uses of a variable.
  
  $\Gamma, y_1 : D^{s_1}, y_2 : D^{s_2} \vdash_{n,n'} e : C^r \quad \forall (s | s_1, s_2) \quad (\Diamond \text{Share})$

  $\Gamma, x : D^s \vdash_{n,n'} e[x/y_1, x/y_2] : C^r$
Sharing relation

- \( \forall (r \mid s_1, \ldots, s_i) \): coinductively defined relation with:
  
  \[
  \text{for all } C, \quad \diamond (C^r) \geq \diamond (C^{s_1}) + \ldots + \diamond (C^{s_i}), \quad \text{etc.}
  \]

- \( \forall (\text{rich} \mid \text{rich, poor}) \) ✔  \( \forall (\text{rich} \mid \text{rich, rich}) \) ✗

If this were allowed ...

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

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

- The rule \( (\diamond \text{Share}) \) enables multiple (sound) uses of a variable.

\[
\frac{\Gamma, y_1 : D^{s_1}, y_2 : D^{s_2} \vdash_{n'} e : C^r \quad \forall (s | s_1, s_2) \quad (\diamond \text{Share})}{\Gamma, x : D^s \vdash_{n'} e[x/y_1, x/y_2] : C^r}
\]
Sharing relation

\( \forall (r \mid s_1, \ldots, s_i) \): coinductively defined relation with:

\[
\text{for all } C, \quad \diamond (C^r) \geq \diamond (C^{s_1}) + \ldots + \diamond (C^{s_i}), \quad \text{etc.}
\]

\( \forall (\text{rich} \mid \text{rich}, \text{poor}) \) ✔️  \( \forall (\text{rich} \mid \text{rich}, \text{rich}) \) ❌

If this were allowed ...

\[
\Gamma : \text{List}^\text{rich} \vdash_0 0 \quad \text{let } l = l.\text{copy()} \text{ in } l.\text{copy()} : \text{List}^\text{poor}
\]

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

The rule (\( \diamond \text{Share} \)) enables multiple (sound) uses of a variable.

\[
\frac{\Gamma, y_1 : D_{s_1}, y_2 : D_{s_2} \vdash_0 e : C^r \quad \forall (s \mid s_1, s_2) \quad (\diamond \text{Share})}{\Gamma, x : D^s \vdash_{n/n'} e[x/y_1, x/y_2] : C^r}
\]
Method invocation

- RAJA method types are polymorphic
  \[ M(C, m) = \phi \]
  \[ \phi = \forall v_{\text{self}}, v, q \cdot C^{v_{\text{self}}}; E_1^{v_1}, \ldots, E_j^{v_j \xrightarrow{q_1/q_2} E_j^{v_{j+1}}} & C_m(v_{\text{self}}, v, q) \]

- \((C^{s_{\text{self}}}; E_1^{s_1}, \ldots, E_j^{s_j \xrightarrow{n_1/n_2} E_j^{s_{j+1}}})\) instanceof \(\phi\) means
  \[ \pi = \{ v_{\text{self}} \mapsto s_{\text{self}}, v_i \mapsto s_i, q_i \mapsto n_i \} \models C_m \]

\[ M(C, m) = \phi \quad (C^{s_{\text{self}}}; E_1^{s_1}, \ldots, E_j^{s_j \xrightarrow{n_1/n_2} E_j^{s_{j+1}}}) \text{ instanceof } \phi \]

\[ x : C^{s_{\text{self}}}, y_1 : E_1^{s_1}, \ldots, y_j : E_j^{s_j \xrightarrow{n_1/n_2} E_j^{s_{j+1}}} \]
\[ x.\text{m}(y_1, \ldots, y_j) : E_j^{s_{j+1}} \] (\(\Diamond\) PolyInv.)

- RAJA Method Typing \(\vdash m : \phi\) ok
  \[ \exists T' \text{ instanceof } \phi \]
  \[ \forall T \text{ instanceof } \phi \quad T = C^{s_{\text{self}}}; E_1^{s_1}, \ldots, E_j^{s_j \xrightarrow{n_1/n_2} E_j^{s_{j+1}}} \]
  \[ \text{this} : C^{s_{\text{self}}}, x_1 : E_1^{s_1}, \ldots, x_j : E_j^{s_j \xrightarrow{n_1/n_2} M_{\text{body}}(C, m) : E_j^{s_{j+1}}} \]

\(\vdash m : \phi\) ok (\(\Diamond\) MBody)
Method invocation

- RAJA method types are polymorphic
  \[ \text{M}(C, m) = \phi \]
  \[
  \phi = \forall v_{\text{self}}, \vec{v}, \vec{q}. \ C^{v_{\text{self}}}; E_1^{v_1}, \ldots, E_j^{v_j \rightarrow q_1/q_2} E_{j+1}^{v_{j+1}} \quad \& \quad C_m(v_{\text{self}}, \vec{v}, \vec{q})
  \]

- \((C^{s_{\text{self}}}; E_1^{s_1}, \ldots, E_j^{s_j \rightarrow E_{j+1}^{s_{j+1}}})\) instanceof \(\phi\) means
  \[
  \pi = \{ v_{\text{self}} \mapsto s_{\text{self}}, v_i \mapsto s_i, q_i \mapsto n_i \} \models C_m
  \]

\[
\text{M}(C, m) = \phi \quad (C^{s_{\text{self}}}; E_1^{s_1}, \ldots, E_j^{s_j \rightarrow E_{j+1}^{s_{j+1}}})\text{ instanceof } \phi
\]

- RAJA Method Typing \(\vdash m : \phi \text{ ok}\)
  \[
  \exists T' \text{ instanceof } \phi
  \]
  \[
  \forall T \text{ instanceof } \phi \quad T = C^{s_{\text{self}}}; E_1^{s_1}, \ldots, E_j^{s_j \rightarrow E_{j+1}^{s_{j+1}}}
  \]

  \[
  \text{this}: C^{s_{\text{self}}}, x_1:E_1^{s_1}, \ldots, x_j:E_j^{s_j \rightarrow E_{j+1}^{s_{j+1}}} \quad \vdash m : \phi \text{ ok}
  \]
Monomorphically RAJA Method Typing $\vdash m : \phi \ ok$

$$\exists T' \text{ instanceof } \phi$$

$$\forall T \text{ instanceof } \phi \quad T = C^{s_\text{self}} ; E^s_1 ; \ldots , E^s_j \xrightarrow{n_1/n_2} E^s_{j+1} \quad \Xi(C, m) = T$$

this $: C^{s_\text{self}} , x_1 : E^s_1 , \ldots , x_j : E^s_j \mid \frac{n_1}{n_2} M_{\text{body}}(C, m) : E^s_{j+1} \mid \Xi $$

(◊MBody)

$\Xi$ records the instance of $\phi$ that is used for typechecking the method.
Monomorphic Recursion in RAJA\textsuperscript{m}

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

$$\exists T' \text{ instanceof } \phi$$

$$\forall T \text{ instanceof } \phi \quad T = C^s_{\text{self}}; E^s_1, \ldots, E^s_j \xrightarrow{n_1/n_2} E^s_{j+1} \quad \Xi(C, m) = T$$

\[\text{this} : C^s_{\text{self}}, x_1 : E^s_1, \ldots, x_j : E^s_j \xrightarrow{n_1/n_2} M_{\text{body}}(C, m) : E^s_{j+1} | \Xi\]

$\vdash m : \phi$ ok

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

- Typing judgement: $\Gamma \xrightarrow{n_1/n_2} e : C^r | \Xi$

- New rule in RAJA\textsuperscript{m}

\[M(C, m) = \phi \quad (C^s_{\text{self}}; E^s_1, \ldots, E^s_j \xrightarrow{n_1/n_2} E^s_{j+1}) \text{ instanceof } \phi\]

$$\Xi(C, m) = C^s_{\text{self}}; E^s_1, \ldots, E^s_j \xrightarrow{n_1/n_2} E^s_{j+1}$$

\[x : C^s_{\text{self}}, y_1 : E^s_1, \ldots, y_j : E^s_j \xrightarrow{n_1/n_2} x.m(y_1, \ldots, y_j) : E^s_{j+1} | \Xi\]
Subtyping of RAJA types

- We extend subtyping to RAJA-classes by

\[ C' <: D^s \iff C <: D \text{ and } r \sqsubseteq s \]  \hspace{1cm} (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\textsuperscript{m}

Input: FJEU Method

RAJA Type Inference

Constraint generation

Elimination of variables

Elimination of view variables

Views creation

LP-solver

Output: RAJA Method type + Views
Rules for collecting the constraints

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

$$AC = p_1 \geq \Diamond (D^{v_1}) + 1 \land p_2 \leq p_1 - \Diamond (D^{v_1}) - 1$$

$$\emptyset \vdash_{p_1 \leq p_2} \text{new } D : C^{v_2} \& (D^{v_1} <: C^{v_2} \land AC) | M; \Xi \tag{\vdash \text{New}}$$

$$\mathcal{D}_i = (\Delta^v_{x_i} <: \Delta^{v_i}+w_i)$$

$$\mathcal{E} = (\mathcal{C}_1 \land \mathcal{C}_2 \land \bigwedge_i \mathcal{D}_i)$$

$$\mathcal{C} = \text{elim}_{\bar{v},\bar{w},u,t}(\mathcal{E})$$

$$\Delta^{\bar{v}} \vdash_{p_1} e_1 \Leftarrow D^u \& \mathcal{C}_1 | M; \Xi$$

$$\Delta^{\bar{w}}, x : D^u \vdash_{p_2} e_2 \Leftarrow C^v \& \mathcal{C}_2 | M; \Xi \tag{\vdash \text{Let}}$$

$$\Delta^{\bar{u}} \vdash_{p_1 \leq p_2} \text{let } D x = e_1 \text{ in } e_2 \Leftarrow C^v \& \mathcal{C}(\bar{u}, v, \bar{p}) | M; \Xi$$
Constraints of the copy method

\[
A\text{get}(\text{Cons}<\text{self}>, \text{next}) <: \text{self} \land \\
\langle (\text{Cons}<\text{self}>) >= \langle (\text{Cons}<\text{res}>) + 1 \land \\
\langle (\text{Nil}<\text{self}>) >= \langle (\text{Nil}<\text{res}>) + 1
\]

\[
\text{Main} = A\text{get}(\text{Cons}<\text{vlist}>, \text{next}) <: \text{vlist} \land \\
\langle (\text{Cons}<\text{vlist}>) >= \langle (\text{Cons}<\text{res}>) + 1 \land \\
\langle (\text{Nil}<\text{vlist}>) >= \langle (\text{Nil}<\text{res}>) + 1
\]
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$^m$

Input: FJEU Method

RAJA Type Inference

- Constraint generation
- Elimination of variables
- Elimination of view variables
- Views creation
- LP-solver

Output: RAJA Method type + Views
Algorithmic views

- We add the connectives $\land, \lor, +, \dot{-}$ to the set of views.
  $\Rightarrow (\forall, \sqsubseteq)$ is a partially ordered set.

**Lemma**

1. $r \land s$ is the greatest lower bound of $r$ and $s$.
2. $r \lor s$ 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. $s_1 \lor s_2 \sqsubseteq r \iff s_1 \sqsubseteq r \land s_2 \sqsubseteq r$.
5. $r \sqsubseteq s_1 \land s_2 \iff r \sqsubseteq s_1 \land r \sqsubseteq s_2$. 
Algorithm for eliminating a view variable from a constraint set.

- The constraints can be brought to the following form:

\[
\begin{align*}
    u_i & \quad \sqsubseteq \quad v \\
    v & \quad \sqsubseteq \quad w_j \\
    A^{\text{get}}(C_v, a_r) & \quad \sqsubseteq \quad \hat{u}_r \\
    \hat{w}_t & \quad \sqsubseteq \quad A^{\text{set}}(C_v, a_t) \\
    \bar{w}_l & \quad \sqsubseteq \quad \bar{u}_k
\end{align*}
\]

\[\diamondsuit(C^v) \geq p_m \quad q_n \geq \diamondsuit(C^v) \quad t_s \geq t_o\]

- A step eliminating \( v \) (\( SC \rightarrow_v SC' \)):

\[
\begin{align*}
    u_i & \quad \sqsubseteq \quad w_j \\
    A^{\text{get}}(C^{u_i}, a_r) & \quad \sqsubseteq \quad \hat{u}_r \\
    \hat{w}_t & \quad \sqsubseteq \quad A^{\text{set}}(C^{u_i}, a_t) \\
    \bar{w}_l & \quad \sqsubseteq \quad \bar{u}_k
\end{align*}
\]

\[\diamondsuit(C^{u_i}) \geq p_m \quad q_n \geq \diamondsuit(C^{w_j}) \quad q_n \geq p_m \quad t_s \geq t_o\]

- If \( SC \rightarrow_v SC' \) then \( SC \iff \exists v . SC' \).
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.