New Results on the Implementation of Resource Aware JAva (RAJA)

Dulma Rodriguez

Department of Computer Science
Ludwig-Maximilians-University Munich

4th Annual MOBIUS Meeting
June 18th, 2009

Introduction

- System RAJA (Hofmann and Jost, ESOP 2006)
  - Type system for Java-like programs.
  - Compile-time analysis of heap-space requirements.
  - Amortised complexity analysis.

- New results on the implementation
  - Elimination of some annotations in the programs.
  - Algorithmic typing rules (with correctness proof).
  - Algorithmic subtyping (with correctness proof).
  - Extension of the implementation with more practical features.

Introduction

- System RAJA (Hofmann and Jost, ESOP 2006)
  - Type system for Java-like programs.
  - Compile-time analysis of heap-space requirements.
  - Amortised complexity analysis.

- New results on the implementation
  - Elimination of some annotations in the programs.
  - Algorithmic typing rules (with correctness proof).
  - Algorithmic subtyping (with correctness proof).
  - Extension of the implementation with more practical features.

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

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

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

Copy example in RAJA

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

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

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

Copy example in RAJA

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

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

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

- Space consumption of $l.copy() : n = \mathsf{length}(l)$.

Copy example in RAJA

```
views  rich, poor
class List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return null;
    }
}
class Nil extends List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return this;
    } }
```

```
class Cons extends List {
    rich: pot = 1;
    poor: pot = 0;
    rich, poor: int elem;
    rich: List<rich,rich> next;
    poor: List<poor,poor> next;

    rich: List<poor>,0 copy(0) {
        let res = new Cons in
        let _ = res.elem ← this.elem in
        let rnext = this.next.copy() in
        let _ = res.next ← rnext in
        return res;
    } }
```

Copy example in RAJA

```
views  rich, poor
class List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return null;
    }
}
class Nil extends List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return this;
    } }
```

```
class Cons extends List {
    rich: pot = 1;
    poor: pot = 0;
    rich, poor: int elem;
    rich: List<rich,rich> next;
    poor: List<poor,poor> next;

    rich: List<poor>,0 copy(0) {
        let res = new Cons in
        let _ = res.elem ← this.elem in
        let rnext = this.next.copy() in
        let _ = res.next ← rnext in
        return res;
    } }
```

- $\Diamond(List^{\mathsf{rich}}) = \Diamond(List^{\mathsf{poor}}) = \Diamond(Nil^{\mathsf{rich}}) = \Diamond(Nil^{\mathsf{poor}}) = \Diamond(Cons^{\mathsf{poor}}) = 0$

Copy example in RAJA

```
views  rich, poor
class List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return null;
    }
}
class Nil extends List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return this;
    } }
```

```
class Cons extends List {
    rich: pot = 1;
    poor: pot = 0;
    rich, poor: int elem;
    rich: List<rich,rich> next;
    poor: List<poor,poor> next;

    rich: List<poor>,0 copy(0) {
        let res = new Cons in
        let _ = res.elem ← this.elem in
        let rnext = this.next.copy() in
        let _ = res.next ← rnext in
        return res;
    } }
```

- $\Diamond(List^{rich}) = \Diamond(List^{poor}) = \Diamond(Nil^{rich}) = \Diamond(Nil^{poor}) = \Diamond(Cons^{poor}) = 0 \quad \Diamond(Cons^{rich}) = 1$

Copy example in RAJA

```
views  rich, poor
class List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return null;
    }
}
class Nil extends List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return this;
    } }
```

```
class Cons extends List {
    rich: pot = 1;
    poor: pot = 0;
    rich, poor: int elem;
    rich: List<rich,rich> next;
    poor: List<poor,poor> next;

    rich: List<poor>,0 copy(0) {
        let res = new Cons in
        let _ = res.elem ← this.elem in
        let rnext = this.next.copy() in
        let _ = res.next ← rnext in
        return res;
    } }
```

- $\Diamond(\mathit{List}^{\mathsf{rich}}) = \Diamond(\mathit{List}^{\mathsf{poor}}) = \Diamond(\mathit{Nil}^{\mathsf{rich}}) = \Diamond(\mathit{Nil}^{\mathsf{poor}}) = \Diamond(\mathit{Cons}^{\mathsf{poor}}) = 0 \quad \Diamond(\mathit{Cons}^{\mathsf{rich}}) = 1$
- $\mathsf{A}\big(\mathit{Cons}^{\mathsf{rich}}, \mathsf{next}\big) = (\mathbf{rich}, \mathbf{rich})$

Copy example in RAJA

```
views  rich, poor
class List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return null;
    }
}
class Nil extends List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return this;
    } }
```

```
class Cons extends List {
    rich: pot = 1;
    poor: pot = 0;
    rich, poor: int elem;
    rich: List<rich,rich> next;
    poor: List<poor,poor> next;

    rich: List<poor>,0 copy(0) {
        let res = new Cons in
        let _ = res.elem ← this.elem in
        let rnext = this.next.copy() in
        let _ = res.next ← rnext in
        return res;
    } }
```

- $\Diamond(\mathit{List}^{\mathsf{rich}}) = \Diamond(\mathit{List}^{\mathsf{poor}}) = \Diamond(\mathit{Nil}^{\mathsf{rich}}) = \Diamond(\mathit{Nil}^{\mathsf{poor}}) = \Diamond(\mathit{Cons}^{\mathsf{poor}}) = 0 \quad \Diamond(\mathit{Cons}^{\mathsf{rich}}) = 1$
- $\mathsf{A}(\mathit{Cons}^{\mathsf{rich}}, \mathsf{next}) = (\mathbf{rich}, \mathbf{rich}) \qquad \mathsf{A}(\mathit{Cons}^{\mathsf{poor}}, \mathsf{next}) = (\mathbf{poor}, \mathbf{poor})$

Copy example in RAJA

```
views  rich, poor
class List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return null;
    }
}
class Nil extends List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return this;
    } }
```

```
class Cons extends List {
    rich: pot = 1;
    poor: pot = 0;
    rich, poor: int elem;
    rich: List<rich,rich> next;
    poor: List<poor,poor> next;

    rich: List<poor>,0 copy(0) {
        let res = new Cons in
        let _ = res.elem ← this.elem in
        let rnext = this.next.copy() in
        let _ = res.next ← rnext in
        return res;
    } }
```

- $\Diamond(List^{\mathsf{rich}}) = \Diamond(List^{\mathsf{poor}}) = \Diamond(Nil^{\mathsf{rich}}) = \Diamond(Nil^{\mathsf{poor}}) = \Diamond(Cons^{\mathsf{poor}}) = 0 \quad \Diamond(Cons^{\mathsf{rich}}) = 1$
- $\mathsf{A}(Cons^{\mathsf{rich}}, \mathsf{next}) = (\mathbf{rich}, \mathbf{rich}) \qquad \mathsf{A}(Cons^{\mathsf{poor}}, \mathsf{next}) = (\mathbf{poor}, \mathbf{poor})$
- $\mathsf{M}(List^{\mathsf{rich}}; copy) = \mathsf{M}(Nil^{\mathsf{rich}}; copy) = \mathsf{M}(Cons^{\mathsf{rich}}; copy) = () \xrightarrow{0/0} List^{\mathsf{poor}}$

Copy example in RAJA

```
views  rich, poor
class List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return null;
    }
}
class Nil extends List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return this;
    } }
```

```
class Cons extends List {
    rich: pot = 1;
    poor: pot = 0;
    rich, poor: int elem;
    rich: List<rich,rich> next;
    poor: List<poor,poor> next;

    rich: List<poor>,0 copy(0) {
        let res = new Cons in
        let _ = res.elem ← this.elem in
        let rnext = this.next.copy() in
        let _ = res.next ← rnext in
        return res;
    } }
```

- Space consumption of $l.copy() : n = \mathsf{length}(l)$.

Copy example in RAJA

```
views  rich, poor
class List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return null;
    }
}
class Nil extends List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return this;
    } }
```

```
class Cons extends List {
    rich: pot = 1;
    poor: pot = 0;
    rich, poor: int elem;
    rich: List<rich,rich> next;
    poor: List<poor,poor> next;

    rich: List<poor>,0 copy(0) {
        let res = new Cons in
        let _ = res.elem ← this.elem in
        let rnext = this.next.copy() in
        let _ = res.next ← rnext in
        return res;
    } }
```

- Space consumption of $l.copy() : n = \mathsf{length}(l)$.
- The potential of a list l is $\mathsf{length}(l)$ under the view **rich** or 0 under the view **poor**.

Copy example in RAJA

```
views  rich, poor
class List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return null;
    }
}
class Nil extends List {
    rich, poor: pot = 0;
    rich: List<poor>,0 copy(0) {
        return this;
    } }
```

```
class Cons extends List {
    rich: pot = 1;
    poor: pot = 0;
    rich, poor: int elem;
    rich: List<rich,rich> next;
    poor: List<poor,poor> next;

    rich: List<poor>,0 copy(0) {
        let res = new Cons in
        let _ = res.elem ← this.elem in
        let rnext = this.next.copy() in
        let _ = res.next ← rnext in
        return res;
    } }
```

- Space consumption of $l.copy() : n = \mathsf{length}(l)$.
- The potential of a list l is $\mathsf{length}(l)$ under the view **rich** or 0 under the view **poor**.
- We pay the copy of $l : \mathsf{List}^{\mathsf{rich}}$ from its potential but we *cannot* copy $l : \mathsf{List}^{\mathsf{poor}}$.

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:
     - $A^{\text{get}}(C^{\mathbf{r}}, a)$: Class $\times$ View $\times$ Field $\to$ View (get-view)
     - $A^{\text{set}}(C^{\mathbf{r}}, a)$: Class $\times$ View $\times$ Field $\to$ View (set-view)
  5. RAJA types for methods:
     - $M(C^{\mathbf{r}}, m)$: Class $\times$ View $\times$ Method $\to$
       $\mathcal{P}$(Views of Arguments $\to$ Effect $\times$ View of Result)
       $(\mathbf{r}_1, \ldots, \mathbf{r}_j \xrightarrow{m/m'} \mathbf{r}_0)$
     - Effect: pair of numbers m, m' representing potential consumed and released by the method.

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:
     - $A^{\text{get}}(C^{\mathbf{r}}, a)$: Class $\times$ View $\times$ Field $\to$ View (get-view)
     - $A^{\text{set}}(C^{\mathbf{r}}, a)$: Class $\times$ View $\times$ Field $\to$ View (set-view)
  5. RAJA types for methods:
     - $M(C^{\mathbf{r}}, m)$: Class $\times$ View $\times$ Method $\to$
       $\mathcal{P}$(Views of Arguments $\to$ Effect $\times$ View of Result)
       $(\mathbf{r}_1, \ldots, \mathbf{r}_j \xrightarrow{m/m'} \mathbf{r}_0)$
     - Effect: pair of numbers m, m' representing potential consumed and released by the method.

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:
     - $\mathsf{M}(C^{\mathbf{r}}, m)$: Class $\times$ View $\times$ Method $\to$
       $\mathcal{P}$(Views of Arguments $\to$ Effect $\times$ View of Result)
       $(\mathbf{r}_1, \ldots, \mathbf{r}_j \xrightarrow{m/m'} \mathbf{r}_0)$
     - Effect: pair of numbers m, m' representing potential consumed and released by the method.

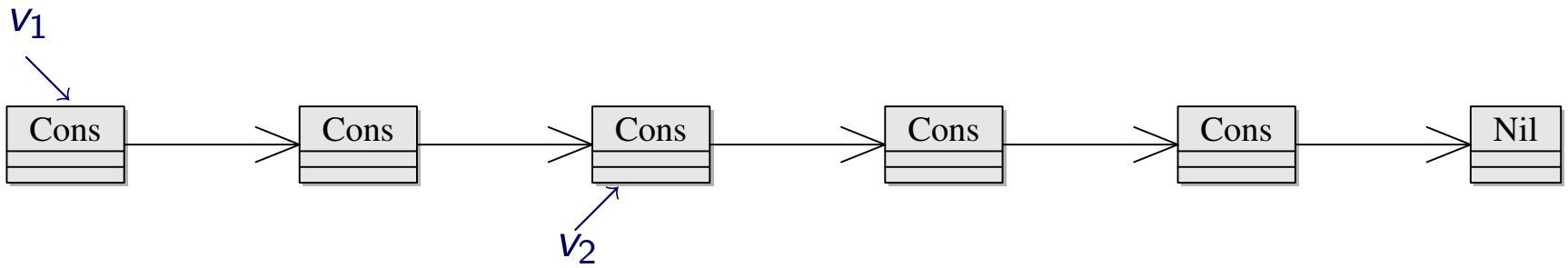
Sketch of RAJA System

- 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:
     - $\mathsf{M}(C^{\mathbf{r}}, m)$: Class $\times$ View $\times$ Method $\to$ $\mathcal{P}$(Views of Arguments $\to$ Effect $\times$ View of Result)

       $(\mathbf{r}_1, \ldots, \mathbf{r}_j \xrightarrow{m/m'} \mathbf{r}_0)$
     - Effect: pair of numbers m, m' representing potential consumed and released by the method.

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:

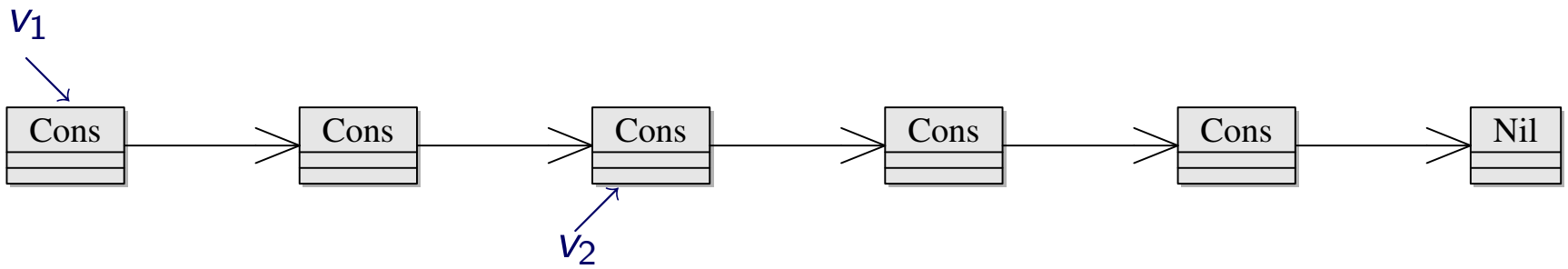


$$\mathrm{pot}_\sigma(v_1 : \mathsf{r}) = \begin{cases} 5 & \mathsf{r} = \mathsf{rich} \\ 0 & \mathsf{r} = \mathsf{poor} \end{cases} \qquad \mathrm{pot}_\sigma(v_2 : \mathsf{r}) = \begin{cases} 3 & \mathsf{r} = \mathsf{rich} \\ 0 & \mathsf{r} = \mathsf{poor} \end{cases}$$

$$\mathrm{pot}_\sigma(\eta : \Gamma) = \underbrace{\mathrm{pot}_\sigma(v_1 : \mathsf{rich})}_{5} + \underbrace{\mathrm{pot}_\sigma(v_2 : \mathsf{poor})}_{0} = 5$$

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:

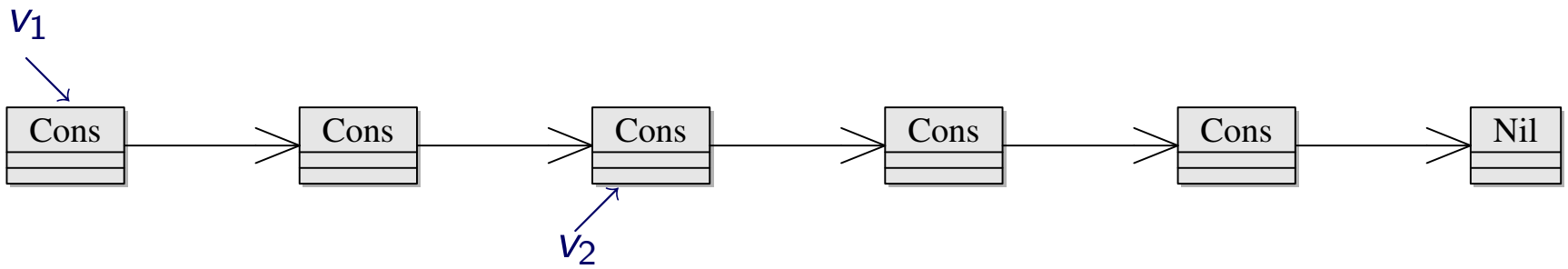


$$\mathsf{pot}_\sigma(v_1 : \mathsf{r}) = \begin{cases} 5 & \mathsf{r} = \mathbf{rich} \\ 0 & \mathsf{r} = \mathbf{poor} \end{cases} \qquad \mathsf{pot}_\sigma(v_2 : \mathsf{r}) = \begin{cases} 3 & \mathsf{r} = \mathbf{rich} \\ 0 & \mathsf{r} = \mathbf{poor} \end{cases}$$

$$\mathsf{pot}_\sigma(\eta : \Gamma) = \underbrace{\mathsf{pot}_\sigma(v_1 : \mathbf{rich})}_{5} + \underbrace{\mathsf{pot}_\sigma(v_2 : \mathbf{poor})}_{0} = 5$$

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:

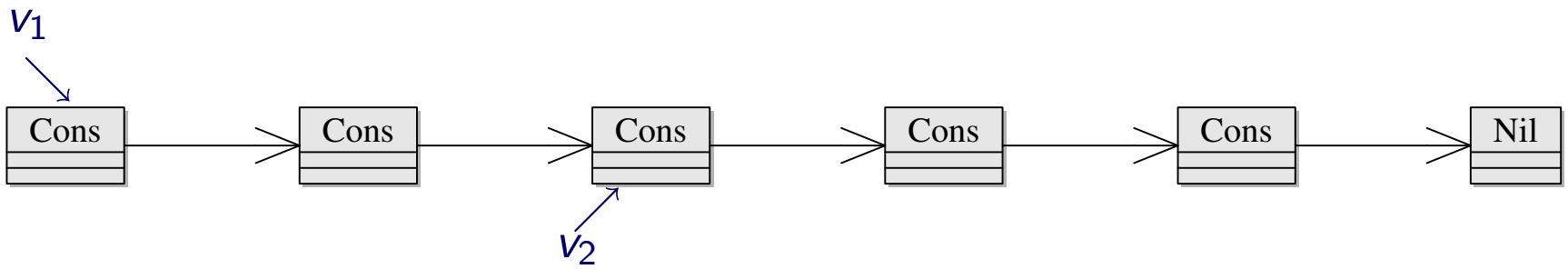


$$\mathsf{pot}_\sigma(v_1 : \mathsf{r}) = \begin{cases} 5 & \mathsf{r} = \mathbf{rich} \\ 0 & \mathsf{r} = \mathbf{poor} \end{cases} \qquad \mathsf{pot}_\sigma(v_2 : \mathsf{r}) = \begin{cases} 3 & \mathsf{r} = \mathbf{rich} \\ 0 & \mathsf{r} = \mathbf{poor} \end{cases}$$

$$\mathsf{pot}_\sigma(\eta : \Gamma) = \underbrace{\mathsf{pot}_\sigma(v_1 : \mathbf{rich})}_{5} + \underbrace{\mathsf{pot}_\sigma(v_2 : \mathbf{poor})}_{0} = 5$$

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:



$$\mathsf{pot}_\sigma(v_1 : \mathsf{r}) = \begin{cases} 5 & \mathsf{r} = \mathbf{rich} \\ 0 & \mathsf{r} = \mathbf{poor} \end{cases} \qquad \mathsf{pot}_\sigma(v_2 : \mathsf{r}) = \begin{cases} 3 & \mathsf{r} = \mathbf{rich} \\ 0 & \mathsf{r} = \mathbf{poor} \end{cases}$$

$$\mathsf{pot}_\sigma(\eta : \Gamma) = \underbrace{\mathsf{pot}_\sigma(v_1 : \mathbf{rich})}_{5} + \underbrace{\mathsf{pot}_\sigma(v_2 : \mathbf{poor})}_{0} = 5$$

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 $\Gamma \vdash^{n}_{n'} e : C^{\mathsf{r}}$ so that:
  - if $\Gamma \vdash^{n}_{n'} e : C^{\mathsf{r}}$ and $\eta, \sigma \vdash e \rightsquigarrow v, \tau$ then

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

Some RAJA typing rules

$$\frac{}{\emptyset \vdash^{\frac{1+\Diamond(C^r)}{0}} \mathsf{new}\ C : C^{\mathsf{r}}}\ (\Diamond New)$$

Some RAJA typing rules

$$\frac{}{\emptyset \vdash^{\frac{1 + \Diamond(C^r)}{0}} \mathsf{new}\ C : C^{\mathbf{r}}}\ (\Diamond New)$$

$$\frac{}{x : C^{\mathbf{r}} \vdash^{\frac{0}{1 + \Diamond(C^r)}} \mathsf{free}\,(x) : E^{\mathbf{s}}}\ (\Diamond Free)$$

Some RAJA typing rules

$$\frac{}{\emptyset \vdash^{\frac{1 + \Diamond(C^r)}{0}} \text{new } C : C^{\mathbf{r}}} \ (\Diamond New)$$

$$\frac{}{x : C^{\mathbf{r}} \vdash^{\frac{0}{1 + \Diamond(C^r)}} \text{free}(x) : E^{\mathbf{s}}} \ (\Diamond Free)$$

$$\frac{(E_1^{\mathbf{q_1}}, \ldots, E_j^{\mathbf{q_j}} \xrightarrow{n/n'} E_0^{\mathbf{q_0}}) \in M(C^{\mathbf{r}}, m)}{x : C^{\mathbf{r}}, y_1 : E_1^{\mathbf{q_1}}, \ldots, y_j : E_j^{\mathbf{q_j}} \vdash^{\frac{n}{n'}} x.m(y_1, \ldots, y_j) : E_0^{\mathbf{q_0}}} \ (\Diamond Invocation)$$

Sharing relation

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

  for all $D <: C : \lozenge(D^{\mathsf{r}}) \geq \lozenge(D^{\mathsf{s}_1}) + \ldots + \lozenge(D^{\mathsf{s}_i})$, etc.

Sharing relation

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

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

- $\curlyvee\left(\mathsf{List}^{\mathsf{rich}} \,\middle|\, \mathsf{List}^{\mathsf{rich}}, \mathsf{List}^{\mathsf{poor}}\right)$ ✓

Sharing relation

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

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

- $\curlyvee\left(\mathsf{List}^{\mathsf{rich}} \,\middle|\, \mathsf{List}^{\mathsf{rich}}, \mathsf{List}^{\mathsf{poor}}\right)$ ✓ $\quad\quad$ $\curlyvee\left(\mathsf{List}^{\mathsf{rich}} \,\middle|\, \mathsf{List}^{\mathsf{rich}}, \mathsf{List}^{\mathsf{rich}}\right)$ ✗

Sharing relation

- If this were allowed ...

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

Sharing relation

- 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!

Sharing relation

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

$$\frac{\Gamma, y_1 : D^{q_1}, y_2 : D^{q_2} \vdash^{n}_{n'} e : C^{r} \qquad \curlyvee(D^{s} \,|\, D^{q_1}, D^{q_2})}{\Gamma, x : D^{s} \vdash^{n}_{n'} e[x/y_1, x/y_2] : C^{r}}\ (\Diamond Share)$$

Sharing relation

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

$$\frac{\Gamma, y_1 : D^{q_1}, y_2 : D^{q_2} \vdash^{n}_{n'} e : C^{r} \qquad \curlyvee(D^{s} \,|\, D^{q_1}, D^{q_2})}{\Gamma, x : D^{s} \vdash^{n}_{n'} e[x/y_1, x/y_2] : C^{r}}\ (\Diamond Share)$$

- $(\Diamond Share)$ is a problematic rule for the implementation because
  - It is non-syntax-directed.
  - The intermediate views q_1, q_2 need to be found out.

Sharing relation

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

$$\frac{\Gamma, y_1 : D^{q_1}, y_2 : D^{q_2} \vdash^{n}_{n'} e : C^{r} \qquad \curlyvee(D^{s} \,|\, D^{q_1}, D^{q_2})}{\Gamma, x : D^{s} \vdash^{n}_{n'} e[x/y_1, x/y_2] : C^{r}}\ (\Diamond Share)$$

- ($\Diamond Share$) is a problematic rule for the implementation because
  - It is non-syntax-directed.
  - The intermediate views q_1, q_2 need to be found out.

Sharing relation

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

$$\frac{\Gamma, y_1 : D^{q_1}, y_2 : D^{q_2} \vdash^{n}_{n'} e : C^{r} \qquad \curlyvee(D^{s} \,|\, D^{q_1}, D^{q_2})}{\Gamma, x : D^{s} \vdash^{n}_{n'} e[x/y_1, x/y_2] : C^{r}}\ (\Diamond Share)$$

- $(\Diamond Share)$ is a problematic rule for the implementation because
  - It is non-syntax-directed.
  - The intermediate views q_1, q_2 need to be found out.

Gaining potential from this

- Method body rule (*too weak*)

$$\frac{\curlyvee(C^{\mathsf{r}} \mid C^{\mathsf{q}}, C^{\mathsf{s}}) \qquad \texttt{this} : C^{\mathsf{r}}, x_1 : E_1^{\mathsf{r}_1}, \ldots, x_j : E_j^{\mathsf{r}_j} \vdash^{n}_{n'} \mathsf{M}_{\mathsf{body}}(C, m) : E_0^{\mathsf{r}_0}}{\vdash m : E_1^{\mathsf{r}_1}, \ldots, E_j^{\mathsf{r}_j} \xrightarrow{n/n'} E_0^{\mathsf{r}_0} \in \mathsf{M}(C^{\mathsf{r}}, m) \ \mathsf{ok}} \ (\Diamond MBody)$$

- Example:

```
class Cons extends List {
    ...
    rich: List<poor>,0 copy(0) {
        Cons<poor> res = new Cons;
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\curlyvee\left(\mathsf{Cons}^{\mathsf{rich}} \,\middle|\, \mathsf{Cons}^{1}, \mathsf{Cons}^{\mathsf{rich} \dot{-} 1}\right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \dot{-} \mathbf{n}$ and $\mathbf{n}$.

Gaining potential from this

- Method body rule (*too weak*)

$$\frac{\curlyvee(C^{\mathsf{r}} \mid C^{\mathsf{q}}, C^{\mathsf{s}}) \qquad \mathtt{this}: C^{\mathsf{r}}, x_1 : E_1^{\mathsf{r}_1}, \ldots, x_j : E_j^{\mathsf{r}_j} \vdash^{n}_{n'} \mathsf{M}_{\mathsf{body}}(C, m) : E_0^{\mathsf{r}_0}}{\vdash m : E_1^{\mathsf{r}_1}, \ldots, E_j^{\mathsf{r}_j} \xrightarrow{n/n'} E_0^{\mathsf{r}_0} \in \mathsf{M}(C^{\mathsf{r}}, m) \ \mathsf{ok}} \ (\Diamond MBody)$$

- Example:

```
class Cons extends List {
                ...
    rich: List<poor>,0 copy(0) {
        Cons<poor> res = new Cons;
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\curlyvee\left(\mathsf{Cons}^{\mathsf{rich}} \,\middle|\, \mathsf{Cons}^{1}, \mathsf{Cons}^{\mathsf{rich} \dot{-} 1}\right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \dot{-} \mathbf{n}$ and $\mathbf{n}$.

Gaining potential from this

- Method body rule (*too weak*)

$$\frac{\text{this}: C^{\mathsf{r}}, x_1 : E_1^{\mathsf{r}_1}, \ldots, x_j : E_j^{\mathsf{r}_j} \vdash^{n}_{n'} \mathsf{M}_{\text{body}}(C, m) : E_0^{\mathsf{r}_0}}{\vdash m : E_1^{\mathsf{r}_1}, \ldots, E_j^{\mathsf{r}_j} \xrightarrow{n/n'} E_0^{\mathsf{r}_0} \in \mathsf{M}(C^{\mathsf{r}}, m) \ \text{ok}} \ (\Diamond MBody)$$

$$\curlyvee(C^{\mathsf{r}} \,|\, C^{\mathsf{q}}, C^{\mathsf{s}})$$

- Example:

```
class Cons extends List {
                ...
    rich: List<poor>,0 copy(0) {  /* this : Cons^rich |-0/0 M_body(Cons, copy) : List^poor */
        Cons<poor> res = new Cons;
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\curlyvee\left(\text{Cons}^{\text{rich}} \,\middle|\, \text{Cons}^{1}, \text{Cons}^{\text{rich} \dot{-} 1}\right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \dot{-} \mathbf{n}$ and $\mathbf{n}$.

Gaining potential from this

- Method body rule (*too weak*)

$$\frac{\text{this}: C^{\mathsf{r}}, x_1 : E_1^{\mathsf{r}_1}, \ldots, x_j : E_j^{\mathsf{r}_j} \vdash^{n}_{n'} \mathsf{M}_{\text{body}}(C, m) : E_0^{\mathsf{r}_0}}{\vdash m : E_1^{\mathsf{r}_1}, \ldots, E_j^{\mathsf{r}_j} \xrightarrow{n/n'} E_0^{\mathsf{r}_0} \in \mathsf{M}(C^{\mathsf{r}}, m) \ \text{ok}} \ (\Diamond MBody)$$

- Example:

```
class Cons extends List {
                ...
    rich: List<poor>,0 copy(0) {  /* this : Cons^rich |-0/0 M_body(Cons, copy) : List^poor */
        Cons<poor> res = new Cons;     /* potential available: 0     */
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\curlyvee\left(\text{Cons}^{\text{rich}} \,\middle|\, \text{Cons}^{1}, \text{Cons}^{\text{rich} \dot{-} 1}\right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \dot{-} \mathbf{n}$ and $\mathbf{n}$.

Gaining potential from this

- Method body rule (*too weak*)

$$\frac{\forall(C^{r} \mid C^{q}, C^{s})}{\texttt{this}: C^{r}, x_1 : E_1^{r_1}, \ldots, x_j : E_j^{r_j} \vdash^{n}_{n'} \mathsf{M}_{\mathsf{body}}(C, m) : E_0^{r_0}} \; (\Diamond MBody)$$
$$\vdash m : E_1^{r_1}, \ldots, E_j^{r_j} \xrightarrow{n/n'} E_0^{r_0} \in \mathsf{M}(C^{r}, m) \;\; \mathsf{ok}$$

- Example:

```
class Cons extends List {
                ...
    rich: List<poor>,0 copy(0) {  /* this : Cons^rich |-0/0 M_body(Cons, copy) : List^poor */
        Cons<poor> res = new Cons;     /* potential available: 0  ✘ */
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\forall\left(\mathsf{Cons}^{\mathsf{rich}} \,\middle|\, \mathsf{Cons}^{1}, \mathsf{Cons}^{\mathsf{rich} \dot{-} 1}\right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \dot{-} \mathbf{n}$ and $\mathbf{n}$.

Gaining potential from this

- Method body rule

$$\frac{\texttt{this}: C^{\mathbf{q}}, x_1 : E_1^{\mathbf{r}_1}, \ldots, x_j : E_j^{\mathbf{r}_j} \vdash \frac{n + \Diamond(C^s)}{n'} \; \mathsf{M}_{\mathsf{body}}(C, m) : E_0^{\mathbf{r}_0}}{\vdash m : E_1^{\mathbf{r}_1}, \ldots, E_j^{\mathbf{r}_j} \xrightarrow{n/n'} E_0^{\mathbf{r}_0} \in \mathsf{M}(C^{\mathbf{r}}, m) \;\; \mathsf{ok}} \; (\Diamond MBody)$$

- Example:

```
class Cons extends List {
                ...
    rich: List<poor>,0 copy(0) {
        Cons<poor> res = new Cons;
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\curlyvee\left(\mathsf{Cons}^{\mathbf{rich}} \,\middle|\, \mathsf{Cons}^{\mathbf{1}}, \mathsf{Cons}^{\mathbf{rich} \dot{-} \mathbf{1}} \right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \dot{-} \mathbf{n}$ and $\mathbf{n}$.

Gaining potential from this

- Method body rule

$$\frac{\textbf{this}: C^{\mathbf{q}}, x_1 : E_1^{\mathbf{r}_1}, \ldots, x_j : E_j^{\mathbf{r}_j} \vdash \underset{n'}{\overset{n + \Diamond(C^s)}{}} \; \mathsf{M}_{\mathsf{body}}(C, m) : E_0^{\mathbf{r}_0} \quad \curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{q}}, C^{\mathbf{s}})}{\vdash m : E_1^{\mathbf{r}_1}, \ldots, E_j^{\mathbf{r}_j} \xrightarrow{n/n'} E_0^{\mathbf{r}_0} \in \mathsf{M}(C^{\mathbf{r}}, m) \;\; \mathsf{ok}} \; (\Diamond MBody)$$

- Example:

```
class Cons extends List {
                ...
    rich: List<poor>,0 copy(0) {
        Cons<poor> res = new Cons;
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\curlyvee\left(\mathsf{Cons}^{\mathsf{rich}} \,\middle|\, \mathsf{Cons}^{1}, \mathsf{Cons}^{\mathsf{rich} \dot{-} 1} \right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \dot{-} \mathbf{n}$ and $\mathbf{n}$.

Gaining potential from this

- Method body rule

$$\frac{\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{q}}, C^{\mathbf{s}}) \qquad \mathtt{this}: C^{\mathbf{q}}, x_1 : E_1^{\mathbf{r}_1}, \ldots, x_j : E_j^{\mathbf{r}_j} \vdash \frac{n + \Diamond(C^s)}{n'}\ \mathsf{M}_{\mathsf{body}}(C, m) : E_0^{\mathbf{r}_0}}{\vdash m : E_1^{\mathbf{r}_1}, \ldots, E_j^{\mathbf{r}_j} \xrightarrow{n/n'} E_0^{\mathbf{r}_0} \in \mathsf{M}(C^{\mathbf{r}}, m)\ \ \mathsf{ok}}\ (\Diamond MBody)$$

- Example:

```
class Cons extends List {
                ...
    rich: List<poor>,0 copy(0) {
        Cons<poor> res = new Cons;
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\curlyvee\left(\mathsf{Cons}^{\mathsf{rich}} \,\middle|\, \mathsf{Cons}^{1}, \mathsf{Cons}^{\mathsf{rich} \dot{-} 1}\right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \dot{-} \mathbf{n}$ and $\mathbf{n}$.

Gaining potential from this

- Method body rule

$$\frac{\curlyvee(C^{\mathbf{r}} \mid C^{\mathbf{q}}, C^{\mathbf{s}}) \qquad \mathtt{this} : C^{\mathbf{q}}, x_1 : E_1^{\mathbf{r}_1}, \ldots, x_j : E_j^{\mathbf{r}_j} \vdash^{\frac{n + \Diamond(C^s)}{n'}} \mathsf{M}_{\mathsf{body}}(C, m) : E_0^{\mathbf{r}_0}}{\vdash m : E_1^{\mathbf{r}_1}, \ldots, E_j^{\mathbf{r}_j} \xrightarrow{n/n'} E_0^{\mathbf{r}_0} \in \mathsf{M}(C^{\mathbf{r}}, m) \ \mathsf{ok}} \ (\Diamond MBody)$$

- Example:

```
class Cons extends List {
                ...
    rich: List<poor>,0 copy(0) {  /* this : Cons^{rich ∸ 1} |-(0 + ◇(Cons^1))/0  ...*/
        Cons<poor> res = new Cons;
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\curlyvee\left(\mathsf{Cons}^{\mathbf{rich}} \,\middle|\, \mathsf{Cons}^{1}, \mathsf{Cons}^{\mathbf{rich} \dot{-} 1}\right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \dot{-} \mathbf{n}$ and $\mathbf{n}$.

Gaining potential from this

- Method body rule

$$\frac{\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{q}}, C^{\mathbf{s}}) \qquad \mathtt{this} : C^{\mathbf{q}}, x_1 : E_1^{\mathbf{r}_1}, \ldots, x_j : E_j^{\mathbf{r}_j} \vdash \frac{n + \Diamond(C^s)}{n'} \; \mathsf{M}_{\mathsf{body}}(C, m) : E_0^{\mathbf{r}_0}}{\vdash m : E_1^{\mathbf{r}_1}, \ldots, E_j^{\mathbf{r}_j} \xrightarrow{n/n'} E_0^{\mathbf{r}_0} \in \mathsf{M}(C^{\mathbf{r}}, m) \;\; \mathsf{ok}} \; (\Diamond MBody)$$

- Example:

```
class Cons extends List {
                ...
    rich: List<poor>,0 copy(0) {  /* this : Cons^{rich ∸ 1} | (0 + ◇(Cons^1)) / 0  ...*/
        Cons<poor> res = new Cons;     /* potential available: 1     */
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\curlyvee\left(\mathsf{Cons}^{\mathbf{rich}} \,\middle|\, \mathsf{Cons}^{1}, \mathsf{Cons}^{\mathbf{rich} \dot{-} 1}\right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \dot{-} \mathbf{n}$ and $\mathbf{n}$.

Gaining potential from this

- Method body rule

$$\frac{\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{q}}, C^{\mathbf{s}}) \qquad \mathtt{this} : C^{\mathbf{q}}, x_1 : E_1^{\mathbf{r}_1}, \ldots, x_j : E_j^{\mathbf{r}_j} \vdash \frac{n + \Diamond(C^s)}{n'} \; \mathsf{M}_{\mathsf{body}}(C, m) : E_0^{\mathbf{r}_0}}{\vdash m : E_1^{\mathbf{r}_1}, \ldots, E_j^{\mathbf{r}_j} \xrightarrow{n/n'} E_0^{\mathbf{r}_0} \in \mathsf{M}(C^{\mathbf{r}}, m) \;\; \mathsf{ok}} \; (\Diamond MBody)$$

- Example:

```
class Cons extends List {
                ...
    rich: List<poor>,0 copy(0) {  /* this : Cons^{rich ∸ 1} | (0 + ◇(Cons^1)) / 0  ...*/
        Cons<poor> res = new Cons;     /* potential available: 1    */
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\curlyvee\left(\mathsf{Cons}^{\mathbf{rich}} \,\middle|\, \mathsf{Cons}^{\mathbf{1}}, \mathsf{Cons}^{\mathbf{rich} \,\dot{-}\, \mathbf{1}}\right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \,\dot{-}\, \mathbf{n}$ and $\mathbf{n}$.

Gaining potential from this

- Method body rule

$$\frac{\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{q}}, C^{\mathbf{s}}) \qquad \mathbf{this} : C^{\mathbf{q}}, x_1 : E_1^{\mathbf{r}_1}, \ldots, x_j : E_j^{\mathbf{r}_j} \vdash \frac{n + \Diamond(C^s)}{n'} \; \mathsf{M}_{\mathsf{body}}(C, m) : E_0^{\mathbf{r}_0}}{\vdash m : E_1^{\mathbf{r}_1}, \ldots, E_j^{\mathbf{r}_j} \xrightarrow{n/n'} E_0^{\mathbf{r}_0} \in \mathsf{M}(C^{\mathbf{r}}, m) \;\; \mathsf{ok}} \; (\Diamond MBody)$$

- Example:

```
class Cons extends List {
                ...
    rich: List<poor>,0 copy(0) {  /* this : Cons^{rich ∸ 1} | (0 + ◇(Cons^1)) / 0  ...*/
        Cons<poor> res = new Cons;     /* potential available: 1    */
        res.elem = elem;
        res.next = this.next.copy();
        return res;
    }
}
```

- where $\curlyvee\left(\mathbf{Cons}^{\mathbf{rich}} \,\middle|\, \mathbf{Cons}^{\mathbf{1}}, \mathbf{Cons}^{\mathbf{rich} \dot{-} \mathbf{1}}\right)$
- Intermediate views $\mathbf{q}, \mathbf{s}$ are automatically inferred using algorithmic views $\mathbf{r} \dot{-} \mathbf{n}$ and $\mathbf{n}$.

Algorithmic views

Let $\mathbf{r}, \mathbf{s} \in \mathscr{V}, n \in \mathbb{N}$.

- $\mathbf{r} \dot{-} \mathbf{n}$ $\Rightarrow \mathbf{r} \dot{-} \mathbf{n}$ is defined like $\mathbf{r}$ but with
  $\Rightarrow \Diamond(C^{\mathbf{r} \dot{-} \mathbf{n}}) = \Diamond(C^{\mathbf{r}}) \dot{-} n$.

- $\mathbf{n}$ $\Rightarrow \Diamond(C^{\mathbf{n}}) = n$.

- $\mathbf{r} + \mathbf{s}$ $\Rightarrow C^{\mathbf{r}+\mathbf{s}}$ is a formal sum operation of $C^{\mathbf{r}}$ and $C^{\mathbf{s}}$,
  $\Rightarrow \Diamond(C^{\mathbf{r}+\mathbf{s}}) = \Diamond(C^{\mathbf{r}}) + \Diamond(C^{\mathbf{s}})$

- $\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{n}}, C^{\mathbf{r}-\mathbf{n}})$ $\curlyvee(C^{r} \,|\, C^{s_1}, C^{s_2}) \iff C^{r} <: C^{s_1+s_2}$ $C^{r+0} = C^{r}$

Algorithmic views

Let $\mathbf{r}, \mathbf{s} \in \mathscr{V}, n \in \mathbb{N}$.

- $\mathbf{r} \dot{-} \mathbf{n}$ $\Rightarrow \mathbf{r} \dot{-} \mathbf{n}$ is defined like $\mathbf{r}$ but with
  $\Rightarrow \Diamond(C^{\mathbf{r} \dot{-} \mathbf{n}}) = \Diamond(C^{\mathbf{r}}) \dot{-} n$.

- $\mathbf{n}$ $\Rightarrow \Diamond(C^{\mathbf{n}}) = n$.

- $\mathbf{r} + \mathbf{s}$ $\Rightarrow C^{\mathbf{r}+\mathbf{s}}$ is a formal sum operation of $C^{\mathbf{r}}$ and $C^{\mathbf{s}}$,
  $\Rightarrow \Diamond(C^{\mathbf{r}+\mathbf{s}}) = \Diamond(C^{\mathbf{r}}) + \Diamond(C^{\mathbf{s}})$

- $\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{n}}, C^{\mathbf{r}-\mathbf{n}})$ $\curlyvee(C^{r} \,|\, C^{s_1}, C^{s_2}) \iff C^{r} <: C^{s_1+s_2}$ $C^{\mathbf{r}+\mathbf{0}} = C^{\mathbf{r}}$

Algorithmic views

Let $\mathbf{r}, \mathbf{s} \in \mathscr{V}, n \in \mathbb{N}$.

- $\mathbf{r} \dot{-} \mathbf{n}$ $\Rightarrow \mathbf{r} \dot{-} \mathbf{n}$ is defined like $\mathbf{r}$ but with
  $\Rightarrow \Diamond(C^{\mathbf{r} \dot{-} \mathbf{n}}) = \Diamond(C^{\mathbf{r}}) \dot{-} n.$

- $\mathbf{n}$ $\Rightarrow \Diamond(C^{\mathbf{n}}) = n.$

- $\mathbf{r} + \mathbf{s}$ $\Rightarrow C^{\mathbf{r}+\mathbf{s}}$ is a formal sum operation of $C^{\mathbf{r}}$ and $C^{\mathbf{s}}$,
  $\Rightarrow \Diamond(C^{\mathbf{r}+\mathbf{s}}) = \Diamond(C^{\mathbf{r}}) + \Diamond(C^{\mathbf{s}})$

- $\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{n}}, C^{\mathbf{r}-\mathbf{n}})$ $\curlyvee(C^{r} \,|\, C^{s_1}, C^{s_2}) \iff C^{r} <: C^{s_1+s_2}$ $C^{r+0} = C^{r}$

Algorithmic views

Let $\mathbf{r}, \mathbf{s} \in \mathscr{V}, n \in \mathbb{N}$.

- $\mathbf{r} \dot{-} \mathbf{n}$ $\Rightarrow \mathbf{r} \dot{-} \mathbf{n}$ is defined like $\mathbf{r}$ but with
  $\Rightarrow \Diamond(C^{\mathbf{r} \dot{-} \mathbf{n}}) = \Diamond(C^{\mathbf{r}}) \dot{-} n.$

- $\mathbf{n}$ $\Rightarrow \Diamond(C^{\mathbf{n}}) = n.$

- $\mathbf{r} + \mathbf{s}$ $\Rightarrow C^{\mathbf{r}+\mathbf{s}}$ is a formal sum operation of $C^{\mathbf{r}}$ and $C^{\mathbf{s}}$,
  $\Rightarrow \Diamond(C^{\mathbf{r}+\mathbf{s}}) = \Diamond(C^{\mathbf{r}}) + \Diamond(C^{\mathbf{s}})$

- $\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{n}}, C^{\mathbf{r}-\mathbf{n}})$ $\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{s}_1}, C^{\mathbf{s}_2}) \iff C^{\mathbf{r}} <: C^{\mathbf{s}_1+\mathbf{s}_2}$ $C^{\mathbf{r}+0} = C^{\mathbf{r}}$

Algorithmic views

Let $\mathbf{r}, \mathbf{s} \in \mathscr{V}, n \in \mathbb{N}$.

- $\mathbf{r} \dot{-} \mathbf{n}$ $\Rightarrow \mathbf{r} \dot{-} \mathbf{n}$ is defined like $\mathbf{r}$ but with
  $\Rightarrow \Diamond(C^{\mathbf{r} \dot{-} \mathbf{n}}) = \Diamond(C^{\mathbf{r}}) \dot{-} n.$

- $\mathbf{n}$ $\Rightarrow \Diamond(C^{\mathbf{n}}) = n.$

- $\mathbf{r} + \mathbf{s}$ $\Rightarrow C^{\mathbf{r}+\mathbf{s}}$ is a formal sum operation of $C^{\mathbf{r}}$ and $C^{\mathbf{s}}$,
  $\Rightarrow \Diamond(C^{\mathbf{r}+\mathbf{s}}) = \Diamond(C^{\mathbf{r}}) + \Diamond(C^{\mathbf{s}})$

- $\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{n}}, C^{\mathbf{r}-\mathbf{n}})$ $\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{s}_1}, C^{\mathbf{s}_2}) \iff C^{\mathbf{r}} <: C^{\mathbf{s}_1+\mathbf{s}_2}$ $C^{\mathbf{r}+0} = C^{\mathbf{r}}$

Algorithmic views

Let $\mathbf{r}, \mathbf{s} \in \mathscr{V}, n \in \mathbb{N}$.

- $\mathbf{r} \dot{-} \mathbf{n}$ $\Rightarrow \mathbf{r} \dot{-} \mathbf{n}$ is defined like $\mathbf{r}$ but with
  $\Rightarrow \Diamond(C^{\mathbf{r} \dot{-} \mathbf{n}}) = \Diamond(C^{\mathbf{r}}) \dot{-} n.$

- $\mathbf{n}$ $\Rightarrow \Diamond(C^{\mathbf{n}}) = n.$

- $\mathbf{r} + \mathbf{s}$ $\Rightarrow C^{\mathbf{r}+\mathbf{s}}$ is a formal sum operation of $C^{\mathbf{r}}$ and $C^{\mathbf{s}}$,
  $\Rightarrow \Diamond(C^{\mathbf{r}+\mathbf{s}}) = \Diamond(C^{\mathbf{r}}) + \Diamond(C^{\mathbf{s}})$

- $\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{n}}, C^{\mathbf{r}-\mathbf{n}})$ $\curlyvee(C^{\mathbf{r}} \,|\, C^{\mathbf{s}_1}, C^{\mathbf{s}_2}) \iff C^{\mathbf{r}} <: C^{\mathbf{s}_1+\mathbf{s}_2}$ $C^{\mathbf{r}+\mathbf{0}} = C^{\mathbf{r}}$

Idea of the algorithm

- Subtyping and sharing are integrated in the rules.
- Variable occurrences are annotated with a view (x^{r}).
- The views from the different occurrences are summed up.
- Judgment: $\Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leqq C^{\gamma}$
- where Δ is an FJEU context and Ψ a map from variables to *algorithmic* views.
- $\Rightarrow \Delta^{\Psi}$ is a map from variables to algorithmic RAJA types.

$$\mathsf{typecheck}(\Delta, e^{\circ}, C^{\gamma}) = \begin{cases} \\ \mathsf{fail} \end{cases}$$

Idea of the algorithm

- Subtyping and sharing are integrated in the rules.
- Variable occurrences are annotated with a view (x^{r}).
- The views from the different occurrences are summed up.
- Judgment: $\Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leftleftarrows C^{\gamma}$
- where Δ is an FJEU context and Ψ a map from variables to *algorithmic* views.
- $\Rightarrow \Delta^{\Psi}$ is a map from variables to algorithmic RAJA types.

$$\mathsf{typecheck}(\Delta, e^{\circ}, C^{\gamma}) = \begin{cases} \text{Success} & \text{if } \\ \text{Fail} & \text{otherwise} \end{cases}$$

Idea of the algorithm

- Subtyping and sharing are integrated in the rules.
- Variable occurrences are annotated with a view (x^{r}).
- The views from the different occurrences are summed up.
- Judgment: $\Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leftleftarrows C^{\gamma}$
- where Δ is an FJEU context and Ψ a map from variables to *algorithmic* views.
- $\Rightarrow \Delta^{\Psi}$ is a map from variables to algorithmic RAJA types.

$$\mathsf{typecheck}(\Delta, e^{\circ}, C^{\gamma}) = \begin{cases} \\ \\ \end{cases}$$

Idea of the algorithm

- Subtyping and sharing are integrated in the rules.
- Variable occurrences are annotated with a view (x^{r}).
- The views from the different occurrences are summed up.
- Judgment: $\Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leftleftarrows C^{\gamma}$
- where Δ is an FJEU context and Ψ a map from variables to *algorithmic* views.
- $\Rightarrow \Delta^{\Psi}$ is a map from variables to algorithmic RAJA types.

$$\mathsf{typecheck}(\Delta, e^{\circ}, C^{\gamma}) = \begin{cases} \\ \\ \end{cases}$$

Idea of the algorithm

- Subtyping and sharing are integrated in the rules.
- Variable occurrences are annotated with a view (x^{r}).
- The views from the different occurrences are summed up.
- Judgment: $\Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leftleftarrows C^{\gamma}$
- where Δ is an FJEU context and Ψ a map from variables to *algorithmic* views.
- $\Rightarrow \Delta^{\Psi}$ is a map from variables to algorithmic RAJA types.

$$\mathsf{typecheck}(\Delta, e^{\circ}, C^{\gamma}) = \begin{cases} (\Psi, n, n') & \text{if } \Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leftleftarrows C^{\gamma} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

Idea of the algorithm

- Subtyping and sharing are integrated in the rules.
- Variable occurrences are annotated with a view (x^{r}).
- The views from the different occurrences are summed up.
- Judgment: $\Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leftleftarrows C^{\gamma}$
- where Δ is an FJEU context and Ψ a map from variables to *algorithmic* views.
- $\Rightarrow \Delta^{\Psi}$ is a map from variables to algorithmic RAJA types.

$$\mathsf{typecheck}(\Delta, e^{\circ}, C^{\gamma}) = \begin{cases} (\Psi, n, n') & \text{if } \Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leftleftarrows C^{\gamma} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

Idea of the algorithm

- Subtyping and sharing are integrated in the rules.
- Variable occurrences are annotated with a view (x^{r}).
- The views from the different occurrences are summed up.
- Judgment: $\Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leftleftarrows C^{\gamma}$
- where Δ is an FJEU context and Ψ a map from variables to *algorithmic* views.
- $\Rightarrow \Delta^{\Psi}$ is a map from variables to algorithmic RAJA types.

$$\mathsf{typecheck}(\Delta, e^{\circ}, C^{\gamma}) = \begin{cases} (\Psi, n, n') & \text{if } \Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leftleftarrows C^{\gamma} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

Idea of the algorithm

- Subtyping and sharing are integrated in the rules.
- Variable occurrences are annotated with a view (x^{r}).
- The views from the different occurrences are summed up.
- Judgment: $\Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leftleftarrows C^{\gamma}$
- where Δ is an FJEU context and Ψ a map from variables to *algorithmic* views.
- $\Rightarrow \Delta^{\Psi}$ is a map from variables to algorithmic RAJA types.

$$\mathsf{typecheck}(\Delta, e^{\circ}, C^{\gamma}) = \begin{cases} (\Psi, n, n') & \text{if } \Delta^{\Psi} \vdash^{n}_{n'} e^{\circ} \leftleftarrows C^{\gamma} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

Idea of the algorithm

- Subtyping and sharing are integrated in the rules.
- Variable occurrences are annotated with a view (x^{r}).
- The views from the different occurrences are summed up.
- Judgment: $\Delta^{\Psi} \vdash_{n'}^{n} e^{\circ} \leftleftarrows C^{\gamma}$
- where Δ is an FJEU context and Ψ a map from variables to *algorithmic* views.
- $\Rightarrow \Delta^{\Psi}$ is a map from variables to algorithmic RAJA types.

$$\mathsf{typecheck}(\Delta, e^{\circ}, C^{\gamma}) = \begin{cases} (\Psi, n, n') & \text{if } \Delta^{\Psi} \vdash_{n'}^{n} e^{\circ} \leftleftarrows C^{\gamma} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

Typechecking Algorithm on an Example

```
rich: List<poor>,0 copy(0) {
  let   res = new Cons in
  let   _ = res.elem ← this.elem in
  let   rnext = this.next.copy() in
  let   _ = res.next ← rnext in
  return res; }
```

Typechecking Algorithm on an Example

```
rich: List<poor>,0 copy(0) {
  let   res = new Cons in
  let   _ = [res as 0].elem ← [this as 0].elem in
  let   rnext = [this as rich-1].next.copy() in
  let   _ = [res as poor].next ← [rnext as poor] in
  return [res as poor]; }
```

view annotations are provided by the user

Typechecking Algorithm on an Example

```
rich: List<poor>,0 copy(0) {
  let List res = new Cons in
        ↖
          FJEU types annotations are automatically inferred
  let List _ = [res as 0].elem ← [this as 0].elem in
  let List rnext = [this as rich-1].next.copy() in
  let List _ = [res as poor].next ← [rnext as poor] in
  return [res as poor]; }
```

Typechecking Algorithm on an Example

$$\texttt{this} : \texttt{Cons}^{?} \vdash^{?}_{?} \mathsf{M}_{\mathsf{body}}(\mathit{Cons}, \mathit{copy}) \leftleftarrows \mathit{List}^{\mathbf{poor}}$$

```
rich: List<poor>,0 copy(0) {
  let List res = new Cons in
  let List _ = [res as 0].elem ← [this as 0].elem in
  let List rnext = [this as rich-1].next.copy() in
  let List _ = [res as poor].next ← [rnext as poor] in
  return [res as poor]; }
```

Typechecking Algorithm on an Example

$$\texttt{this} : \texttt{Cons}^{?} \vdash^{?}_{?} \mathsf{M}_{\mathsf{body}}(\mathit{Cons}, \mathit{copy}) \leftleftarrows \mathit{List}^{\mathbf{poor}}$$

```
rich: List<poor>,0 copy(0) {
  let List<poor+poor> res = new Cons in
  let List _ = [res as 0].elem ← [this as 0].elem in
  let List rnext = [this as rich-1].next.copy() in
  let List _ = [res as poor].next ← [rnext as poor] in
  return [res as poor]; }
```

Typechecking Algorithm on an Example

$$\texttt{this} : \texttt{Cons}^{?} \overset{?}{\vdash_{?}} \mathsf{M}_{\mathsf{body}}(\mathit{Cons}, \mathit{copy}) \leftleftarrows \mathit{List}^{\mathsf{poor}}$$

```
rich: List<poor>,0 copy(0) {
  let List<poor+poor> res = new Cons in
  let List _ = [res as 0].elem ← [this as 0].elem in
  let List<poor> rnext = [this as rich-1].next.copy() in
  let List _ = [res as poor].next ← [rnext as poor] in
  return [res as poor]; }
```

Typechecking Algorithm on an Example

$$\texttt{this} : \texttt{Cons}^{\texttt{rich}-1} \vdash \frac{1 = 0 + \Diamond(\mathit{Cons}^{1})}{0} \; \mathsf{M}_{\mathsf{body}}(\mathit{Cons}, \mathit{copy}) \Leftarrow \mathit{List}^{\mathsf{poor}}$$

```
rich: List<poor>,0 copy(0) {
  let List<poor+poor> res = new Cons in
  let List _ = [res as 0].elem ← [this as 0].elem in
  let List<poor> rnext = [this as rich-1].next.copy() in
  let List _ = [res as poor].next ← [rnext as poor] in
  return [res as poor]; }
```

Correctness of the Algorithm

Lemma (Soundness of algorithmic RAJA typing)

If $\Delta^{\Psi} \vdash^{n}_{n'} e \leftleftarrows C^{\gamma}$ *then* $\Delta^{\Psi} \vdash^{n}_{n'} |e| : C^{\gamma}$.

Lemma (Completeness of algorithmic RAJA typing)

If $\Gamma \vdash^{n}_{n'} e : C^{r}$ *then there is an annotated version* e° *of the expression* e *with* $|\Gamma|^{\Psi} \vdash^{u}_{u'} e^{\circ} \leftleftarrows C^{r}$ *for some* $u \leq n$ *and* $u - u' \leq n - n'$ *so that* $\Gamma <: |\Gamma|^{\Psi}$.

- More details in the article "Efficient Type-Checking for Amortised Heap-Space Analysis" in CSL'09.

Correctness of the Algorithm

Lemma (Soundness of algorithmic RAJA typing)

If $\Delta^{\Psi} \vdash^{n}_{n'} e \leftleftarrows C^{\gamma}$ *then* $\Delta^{\Psi} \vdash^{n}_{n'} |e| : C^{\gamma}$.

Lemma (Completeness of algorithmic RAJA typing)

If $\Gamma \vdash^{n}_{n'} e : C^{\mathbf{r}}$ *then there is an annotated version* e° *of the expression* e *with* $|\Gamma|^{\Psi} \vdash^{u}_{u'} e^{\circ} \leftleftarrows C^{\mathbf{r}}$ *for some* $u \leq n$ *and* $u - u' \leq n - n'$ *so that* $\Gamma <: |\Gamma|^{\Psi}$.

- More details in the article "Efficient Type-Checking for Amortised Heap-Space Analysis" in CSL'09.

Correctness of the Algorithm

Lemma (Soundness of algorithmic RAJA typing)

If $\Delta^{\Psi} \vdash^{n}_{n'} e \leftleftarrows C^{\gamma}$ *then* $\Delta^{\Psi} \vdash^{n}_{n'} |e| : C^{\gamma}$.

Lemma (Completeness of algorithmic RAJA typing)

If $\Gamma \vdash^{n}_{n'} e : C^{\mathbf{r}}$ *then there is an annotated version* e° *of the expression* e *with* $|\Gamma|^{\Psi} \vdash^{u}_{u'} e^{\circ} \leftleftarrows C^{\mathbf{r}}$ *for some* $u \leq n$ *and* $u - u' \leq n - n'$ *so that* $\Gamma <: |\Gamma|^{\Psi}$.

- More details in the article "Efficient Type-Checking for Amortised Heap-Space Analysis" in CSL'09.

Subtyping algorithm

- With algorithmic views a sharing task can be reduced into a subtyping task: $\curlyvee(C^{\mathsf{r}} \,|\, C^{\mathsf{s}_1}, C^{\mathsf{s}_2}) \iff C^{\mathsf{r}} <: C^{\mathsf{s}_1+\mathsf{s}_2}$.
- $\Rightarrow$ It is enough to have a subtyping algorithm.
- $C^{\mathsf{r}} <: D^{\mathsf{s}} \iff (C^{\mathsf{r}}, D^{\mathsf{s}})$ are elements of the greatest fixpoint of some operator.
- There is an efficient goal-directed algorithm for membership checking in greatest fixpoints, but only for a certain kind of operators (TPL, Pierce, 2002).
- The operator that describes subtyping is not of that kind.
- $\Rightarrow$ We extended the given algorithm for all kinds of operators. (FICS'09)

Subtyping algorithm

- With algorithmic views a sharing task can be reduced into a subtyping task: $\curlyvee(C^{\mathsf{r}} \,|\, C^{\mathsf{s}_1}, C^{\mathsf{s}_2}) \iff C^{\mathsf{r}} <: C^{\mathsf{s}_1+\mathsf{s}_2}$.
- $\Rightarrow$ It is enough to have a subtyping algorithm.
- $C^{\mathsf{r}} <: D^{\mathsf{s}} \iff (C^{\mathsf{r}}, D^{\mathsf{s}})$ are elements of the greatest fixpoint of some operator.
- There is an efficient goal-directed algorithm for membership checking in greatest fixpoints, but only for a certain kind of operators (TPL, Pierce, 2002).
- The operator that describes subtyping is not of that kind.
- $\Rightarrow$ We extended the given algorithm for all kinds of operators. (FICS'09)

Subtyping algorithm

- With algorithmic views a sharing task can be reduced into a subtyping task: $\curlyvee(C^{\mathsf{r}} \,|\, C^{\mathsf{s}_1}, C^{\mathsf{s}_2}) \iff C^{\mathsf{r}} <: C^{\mathsf{s}_1+\mathsf{s}_2}$.
- $\Rightarrow$ It is enough to have a subtyping algorithm.
- $C^{\mathsf{r}} <: D^{\mathsf{s}} \iff (C^{\mathsf{r}}, D^{\mathsf{s}})$ are elements of the greatest fixpoint of some operator.
- There is an efficient goal-directed algorithm for membership checking in greatest fixpoints, but only for a certain kind of operators (TPL, Pierce, 2002).
- The operator that describes subtyping is not of that kind.
- $\Rightarrow$ We extended the given algorithm for all kinds of operators. (FICS'09)

Subtyping algorithm

- With algorithmic views a sharing task can be reduced into a subtyping task: $\curlyvee(C^{\mathsf{r}} \,|\, C^{\mathsf{s}_1}, C^{\mathsf{s}_2}) \iff C^{\mathsf{r}} <: C^{\mathsf{s}_1+\mathsf{s}_2}$.
- $\Rightarrow$ It is enough to have a subtyping algorithm.
- $C^{\mathsf{r}} <: D^{\mathsf{s}} \iff (C^{\mathsf{r}}, D^{\mathsf{s}})$ are elements of the greatest fixpoint of some operator.
- There is an efficient goal-directed algorithm for membership checking in greatest fixpoints, but only for a certain kind of operators (TPL, Pierce, 2002).
- The operator that describes subtyping is not of that kind.
- $\Rightarrow$ We extended the given algorithm for all kinds of operators. (FICS'09)

Subtyping algorithm

- With algorithmic views a sharing task can be reduced into a subtyping task: $\curlyvee(C^{\mathsf{r}} \,|\, C^{\mathsf{s}_1}, C^{\mathsf{s}_2}) \iff C^{\mathsf{r}} <: C^{\mathsf{s}_1+\mathsf{s}_2}$.
- $\Rightarrow$ It is enough to have a subtyping algorithm.
- $C^{\mathsf{r}} <: D^{\mathsf{s}} \iff (C^{\mathsf{r}}, D^{\mathsf{s}})$ are elements of the greatest fixpoint of some operator.
- There is an efficient goal-directed algorithm for membership checking in greatest fixpoints, but only for a certain kind of operators (TPL, Pierce, 2002).
- The operator that describes subtyping is not of that kind.
- $\Rightarrow$ We extended the given algorithm for all kinds of operators. (FICS'09)

Subtyping algorithm

- With algorithmic views a sharing task can be reduced into a subtyping task: $\curlyvee(C^{\mathsf{r}} \,|\, C^{\mathsf{s}_1}, C^{\mathsf{s}_2}) \iff C^{\mathsf{r}} <: C^{\mathsf{s}_1+\mathsf{s}_2}$.
- $\Rightarrow$ It is enough to have a subtyping algorithm.
- $C^{\mathsf{r}} <: D^{\mathsf{s}} \iff (C^{\mathsf{r}}, D^{\mathsf{s}})$ are elements of the greatest fixpoint of some operator.
- There is an efficient goal-directed algorithm for membership checking in greatest fixpoints, but only for a certain kind of operators (TPL, Pierce, 2002).
- The operator that describes subtyping is not of that kind.
- $\Rightarrow$ We extended the given algorithm for all kinds of operators. (FICS'09)

Implementation Features

- Typechecker for RAJA programs: prediction of free-list size.
- Interpreter with built-in calculation of free-list size.
- Datatypes
  - Integers
  - Booleans
  - Strings
- I/O features
  - Opening and closing files
  - Reading from files
  - Printing to the standard output
- Analysis gives an upper bound on the free-list size as a function of the size of the read files:

$$|\mathsf{free-list}| \leq m + \sum_i (p_i * |\mathsf{file_i}|)$$

  - where p_i is the potential given to each line of the file $\mathsf{file_i}$
  - and m is a constant.

Implementation Features

- Typechecker for RAJA programs: prediction of free-list size.
- Interpreter with built-in calculation of free-list size.
- Datatypes
  - Integers
  - Booleans
  - Strings
- I/O features
  - Opening and closing files
  - Reading from files
  - Printing to the standard output
- Analysis gives an upper bound on the free-list size as a function of the size of the read files:

$$|\mathsf{free-list}| \leq m + \sum_i (p_i * |\mathsf{file_i}|)$$

  - where p_i is the potential given to each line of the file $\mathsf{file_i}$
  - and m is a constant.

Implementation Features

- Typechecker for RAJA programs: prediction of free-list size.
- Interpreter with built-in calculation of free-list size.
- Datatypes
  - Integers
  - Booleans
  - Strings
- I/O features
  - Opening and closing files
  - Reading from files
  - Printing to the standard output
- Analysis gives an upper bound on the free-list size as a function of the size of the read files:

$$|\mathsf{free-list}| \leq m + \sum_i (p_i * |\mathsf{file_i}|)$$

  - where p_i is the potential given to each line of the file $\mathsf{file_i}$
  - and m is a constant.

Implementation Features

- Typechecker for RAJA programs: prediction of free-list size.
- Interpreter with built-in calculation of free-list size.
- Datatypes
  - Integers
  - Booleans
  - Strings
- I/O features
  - Opening and closing files
  - Reading from files
  - Printing to the standard output
- Analysis gives an upper bound on the free-list size as a function of the size of the read files:

$$|\mathsf{free-list}| \leq m + \sum_i (p_i * |\mathsf{file_i}|)$$

  - where p_i is the potential given to each line of the file $\mathsf{file_i}$
  - and m is a constant.

Conclusions and further work

- Conclusions
  - Our type-based analysis encompasses:
    1. Objects
    2. Inheritance
    3. Downcast
    4. Imperative update
    5. Aliasing
    6. Circular data
  - The (improved) implementation allows:
    - Testing the system with bigger programs and bigger inputs
    - Improving the system
- Further work
  - Elimination of the annotations of the variable occurrences.
  - Type inference
  - More examples: Iterators on lists, etc.
  - Making the system more expressive (maybe with type states).

Conclusions and further work

- Conclusions
  - Our type-based analysis encompasses:
    1. Objects
    2. Inheritance
    3. Downcast
    4. Imperative update
    5. Aliasing
    6. Circular data
  - The (improved) implementation allows:
    - Testing the system with bigger programs and bigger inputs
    - Improving the system
- Further work
  - Elimination of the annotations of the variable occurrences.
  - Type inference
  - More examples: Iterators on lists, etc.
  - Making the system more expressive (maybe with type states).