Implementation of Resource Aware JAva (RAJA)

Dulma Rodriguez

Department of Computer Science
Ludwig-Maximilians-University Munich

MOBIUS Annual Meeting 2008
June 17, 2008

Introduction

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

- Implementation in Ocaml
  - Result: Typechecker and interpreter for RAJA programs.
  - Challenge: Algorithmic presentation of the typing rules.
  - Goal: Testing and improving the system.

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

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

- $\approx$ Featherweight Java + imperative field update.

Copy example

In Java

```
abstract class List {
    abstract List copy();
}

class Nil extends List {
    List copy() {
        return this;
    }
}
class Cons extends List {
    int elem;
    List next;

    List copy() {
        Cons res = new Cons;
        res.elem = this.elem;
        res.next = this.next.copy();
        return res;
    }
}
```

In FJEU

```
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 Cons res = new Cons in
        let int elem = this.elem in
        let Cons res1 = res.elem ⟵ elem in
        let List next = this.next in
        let List rnext = next.copy() in
        let List res2 = res1.next ⟵ rnext in
        return res2;
    }
}
```

Copy example

In Java

```
abstract class List {
    abstract List copy();
}

class Nil extends List {
    List copy() {
        return this;
    }
}
class Cons extends List {
    int elem;
    List next;

    List copy() {
        Cons res = new Cons;
        res.elem = this.elem;
        res.next = this.next.copy();
        return res;
    }
}
```

In FJEU

```
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 Cons res = new Cons in
        let int elem = this.elem in
        let Cons res1 = res.elem ← elem in
        let List next = this.next in
        let List rnext = next.copy() in
        let List res2 = res1.next ← rnext in
        return res2;
    }
}
```

Sketch of RAJA System

- RAJA program P = *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:
   - $\Diamond(C^r)$: Class × View → $\mathbb{Q}^+$
4. (Get- and set-) views for attributes:
   - $\mathsf{A}^{\mathsf{get}}(C^r, a)$: Class × View × Field → View (get-view)
   - $\mathsf{A}^{\mathsf{set}}(C^r, a)$: Class × View × Field → View (set-view)
5. RAJA types for methods:
   - $\mathsf{M}(C^r, m)$: Class × View × Method → $\mathcal{P}$(Views of Arguments → Effect × View of Result)

     $(r_1, \ldots, r_j \xrightarrow{m/m'} r_0)$
   - 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 r we have an annotated version C^r.
  3. Potential:
     - $\Diamond(C^r)$: Class $\times$ View $\rightarrow \mathbb{Q}^+$
  4. (Get- and set-) views for attributes:
     - $A^{get}(C^r, a)$: Class $\times$ View $\times$ Field $\rightarrow$ View (get-view)
     - $A^{set}(C^r, a)$: Class $\times$ View $\times$ Field $\rightarrow$ View (set-view)
  5. RAJA types for methods:
     - $M(C^r, m)$: Class $\times$ View $\times$ Method $\rightarrow$ $\mathcal{P}$(Views of Arguments $\rightarrow$ Effect $\times$ View of Result)

       $(r_1, \ldots, r_j \xrightarrow{m/m'} r_0)$
     - 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 r we have an annotated version C^r.
3. Potential:
   - $\Diamond(C^r)$: Class $\times$ View $\rightarrow \mathbb{Q}^+$
4. (Get- and set-) views for attributes:
   - $\mathsf{A}^{\mathsf{get}}(C^r, a)$: Class $\times$ View $\times$ Field $\rightarrow$ View (get-view)
   - $\mathsf{A}^{\mathsf{set}}(C^r, a)$: Class $\times$ View $\times$ Field $\rightarrow$ View (set-view)
5. RAJA types for methods:
   - $\mathsf{M}(C^r, m)$: Class $\times$ View $\times$ Method $\rightarrow$

     $\mathcal{P}$(Views of Arguments $\rightarrow$ Effect $\times$ View of Result)

     $(r_1, \ldots, r_j \xrightarrow{m/m'} r_0)$
   - 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 r we have an annotated version C^r.
  3. Potential:
     - $\Diamond(C^r)$: Class × View → $\mathbb{Q}^+$
  4. (Get- and set-) views for attributes:
     - $\mathsf{A}^{\mathsf{get}}(C^r, a)$: Class × View × Field → View (get-view)
     - $\mathsf{A}^{\mathsf{set}}(C^r, a)$: Class × View × Field → View (set-view)
  5. RAJA types for methods:
     - $\mathsf{M}(C^r, m)$: Class × View × Method → $\mathcal{P}$(Views of Arguments → Effect × View of Result)
       $(r_1, \ldots, r_j \xrightarrow{m/m'} r_0)$
     - Effect: pair of numbers m, m' representing potential consumed and released by the method.

Copy example in RAJA

```
views rich, poor
class List implements rich, poor {
    rich:pot = 0;
    poor:pot = 0;
    rich:List<poor>,0 copy(0) { return null;}
}
class Nil extends List implements rich, poor {
    rich:pot = 0;
    poor:pot = 0;
    rich:List<poor>,0 copy(0) { return this; }
}
class Cons extends List implements rich, poor {
    rich:pot = 1;
    poor:pot = 0;
    rich:int elem;
    poor:int elem;
    rich:List<rich,rich> next;
    poor:List<poor,poor> next;

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

$\Diamond(\cdot)$	rich	poor
List	0	0
Nil	0	0
Cons	1	0

	Cons^{rich}	Cons^{poor}
$A^{get}(\cdot, \text{next})$	rich	poor
$A^{set}(\cdot, \text{next})$	rich	poor

	List^{rich}	Nil^{rich}	Cons^{rich}
$M(\cdot, \text{copy})$	$() \xrightarrow{0/0} \text{poor}$		

Meaning of *rich* and *poor*:

$$\text{potential}(l : \text{List}^{rich}) = \text{length}(l)$$
$$\text{potential}(l : \text{List}^{poor}) = 0$$

we pay the copy of $l : \text{List}^{rich}$ from its potential but we *cannot* copy $l : \text{List}^{poor}$

How is potential defined?

- Object *cons* : $List^{rich}$ in the heap pointing to location 1
- [100, 200, 300, 400, 500]
- length(*cons*) $= 5$

```
{ 1 -> Cons [next: location 2; elem:100; ]  }
{ 2 -> Cons [next: location 3; elem:200; ]  }
{ 3 -> Cons [next: location 4; elem:300; ]  }
{ 4 -> Cons [next: location 5; elem:400; ]  }
{ 5 -> Cons [next: location 6; elem:500; ]  }
{ 6 -> Nil []  }
```

Figure: Heap configuration

How is potential defined?

```
1 -> Cons [next: location 2; elem:100; ]  }
2 -> Cons [next: location 3; elem:200; ]  }
3 -> Cons [next: location 4; elem:300; ]  }
4 -> Cons [next: location 5; elem:400; ]  }
5 -> Cons [next: location 6; elem:500; ]  }
{ 6 -> Nil []  }
```

$$\phi_\sigma((cons : rich).\epsilon) = \Diamond(\mathsf{Cons}^{rich}) = 1$$
$$\phi_\sigma((cons : rich).\mathtt{next}) = \Diamond(\mathsf{Cons}^{A^{get}(\mathsf{Cons}^{rich}, \mathtt{next})}) = 1 \quad \ldots$$
$$\phi_\sigma((cons : rich).\mathtt{next.next.next.next}) = 1$$
$$\phi_\sigma((cons : rich).\mathtt{next.next.next.next.next}) = \Diamond(\mathsf{Nil}^{rich}) = 0$$

Therefore: $\Phi_\sigma(cons : rich) = \sum_{\vec{p}} \phi_\sigma((cons : rich).\vec{p}) = 5$

but $\Phi_\sigma(cons : poor) = 0$

How is potential defined?

-

```
1 -> Cons [next: location 2; elem:100; ]  }
2 -> Cons [next: location 3; elem:200; ]  }
3 -> Cons [next: location 4; elem:300; ]  }
4 -> Cons [next: location 5; elem:400; ]  }
5 -> Cons [next: location 6; elem:500; ]  }
{ 6 -> Nil []  }
```

-

$\phi_\sigma((cons : rich).\epsilon) = \lozenge(\mathsf{Cons}^{rich}) = 1$

$\phi_\sigma((cons : rich).\mathtt{next}) = \lozenge(\mathsf{Cons}^{A^{get}(\mathsf{Cons}^{rich}, \mathsf{next})}) = 1 \quad \ldots$

$\phi_\sigma((cons : rich).\mathtt{next.next.next.next}) = 1$

$\phi_\sigma((cons : rich).\mathtt{next.next.next.next.next}) = \lozenge(\mathsf{Nil}^{rich}) = 0$

Therefore: $\Phi_\sigma(cons : rich) = \sum_{\vec{p}} \phi_\sigma((cons : rich).\vec{p}) = 5$

but $\Phi_\sigma(cons : poor) = 0$

How is potential defined?

-

```
1 -> Cons [next: location 2; elem:100; ]  }
2 -> Cons [next: location 3; elem:200; ]  }
3 -> Cons [next: location 4; elem:300; ]  }
4 -> Cons [next: location 5; elem:400; ]  }
5 -> Cons [next: location 6; elem:500; ]  }
{ 6 -> Nil []  }
```

-

$$\phi_\sigma((cons : rich).\epsilon) = \Diamond(\mathsf{Cons}^{rich}) = 1$$

$$\phi_\sigma((cons : rich).\mathtt{next}) = \Diamond(\mathsf{Cons}^{A^{get}(\mathsf{Cons}^{rich},\,\mathsf{next})}) = 1 \quad \ldots$$

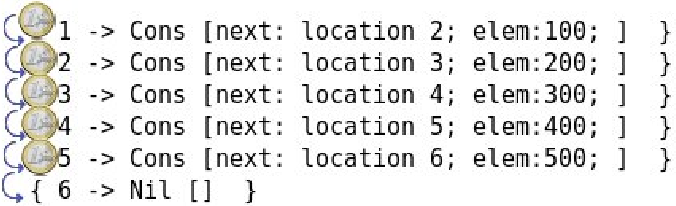
$$\phi_\sigma((cons : rich).\mathtt{next.next.next.next}) = 1$$

$$\phi_\sigma((cons : rich).\mathtt{next.next.next.next.next}) = \Diamond(\mathsf{Nil}^{rich}) = 0$$

Therefore: $\Phi_\sigma(cons : rich) = \sum_{\vec{p}} \phi_\sigma((cons : rich).\vec{p}) = 5$

but $\Phi_\sigma(cons : poor) = 0$

How is potential defined?

- 

- $$\phi_\sigma((cons : rich).\epsilon) = \Diamond(\mathsf{Cons}^{rich}) = 1$$
  $$\phi_\sigma((cons : rich).\mathtt{next}) = \Diamond(\mathsf{Cons}^{A^{get}(\mathsf{Cons}^{rich}, \mathsf{next})}) = 1 \quad \ldots$$
  $$\phi_\sigma((cons : rich).\mathtt{next.next.next.next}) = 1$$
  $$\phi_\sigma((cons : rich).\mathtt{next.next.next.next.next}) = \Diamond(\mathsf{Nil}^{rich}) = 0$$

Therefore: $\Phi_\sigma(cons : rich) = \sum_{\vec{p}} \phi_\sigma((cons : rich).\vec{p}) = 5$

but $\Phi_\sigma(cons : poor) = 0$

How is potential defined?

-

```
1 -> Cons [next: location 2; elem:100; ]  }
2 -> Cons [next: location 3; elem:200; ]  }
3 -> Cons [next: location 4; elem:300; ]  }
4 -> Cons [next: location 5; elem:400; ]  }
5 -> Cons [next: location 6; elem:500; ]  }
{ 6 -> Nil []  }
```

-

$$\phi_\sigma((cons : rich).\epsilon) = \Diamond(\mathsf{Cons}^{rich}) = 1$$

$$\phi_\sigma((cons : rich).\texttt{next}) = \Diamond(\mathsf{Cons}^{A^{get}(\mathsf{Cons}^{rich},\, \mathsf{next})}) = 1 \quad \ldots$$

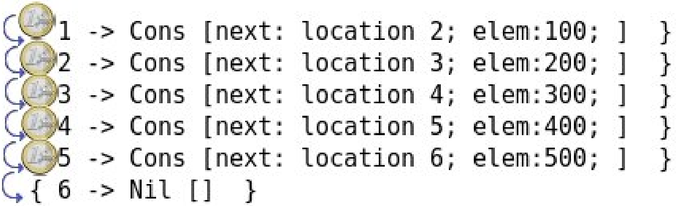
$$\phi_\sigma((cons : rich).\texttt{next.next.next.next}) = 1$$

$$\phi_\sigma((cons : rich).\texttt{next.next.next.next.next}) = \Diamond(\mathsf{Nil}^{rich}) = 0$$

Therefore: $\Phi_\sigma(cons : rich) = \sum_{\vec{p}} \phi_\sigma((cons : rich).\vec{p}) = 5$

but $\Phi_\sigma(cons : poor) = 0$

How is potential defined?

- 

- $$\phi_\sigma((cons : rich).\epsilon) = \Diamond(\mathrm{Cons}^{rich}) = 1$$
$$\phi_\sigma((cons : rich).\mathtt{next}) = \Diamond(\mathrm{Cons}^{A^{get}(\mathrm{Cons}^{rich}, \mathrm{next})}) = 1 \quad \ldots$$
$$\phi_\sigma((cons : rich).\mathtt{next.next.next.next}) = 1$$
$$\phi_\sigma((cons : rich).\mathtt{next.next.next.next.next}) = \Diamond(\mathrm{Nil}^{rich}) = 0$$

Therefore: $\Phi_\sigma(cons : rich) = \sum_{\vec{p}} \phi_\sigma((cons : rich).\vec{p}) = 5$

but $\Phi_\sigma(cons : poor) = 0$

Typing and semantics

- Semantics: $\eta, \sigma \vdash^{n}_{n'}\circ\ e \rightsquigarrow v, \tau$ means:
  - expression e evaluates successfully to value v beginning with stack η and heap σ and ending with heap τ.
  - n: unused heap units before evaluating e.
  - n': unused heap units after evaluating e.
- Typing judgement $\Gamma \vdash^{n}_{n'} e : C^{r}$.

Theorem

If $\Gamma \vdash^{n}_{n'} e : C^{r}$ *and* $\eta, \sigma \vdash\circ\ e \rightsquigarrow v, \tau$ *then*

$$\eta, \sigma \vdash^{n + \Phi_{\sigma}(\eta : \Gamma) + \Phi_{\sigma}(\eta : \Delta)}_{n' + \Phi_{\tau}(v : r) + \Phi_{\tau}(\eta : \Delta)}\circ\ e \rightsquigarrow v, \tau$$

Typing and semantics

- Semantics: $\eta, \sigma \vdash^{n}_{n'}\circ\; e \rightsquigarrow v, \tau$ means:
  - expression e evaluates successfully to value v beginning with stack η and heap σ and ending with heap τ.
  - n : unused heap units before evaluating e.
  - n': unused heap units after evaluating e.
- Typing judgement $\Gamma \vdash^{n}_{n'} e : C^r$.

Theorem

If $\Gamma \vdash^{n}_{n'} e : C^r$ *and* $\eta, \sigma \vdash^{\circ} e \rightsquigarrow v, \tau$ *then*

$$\eta, \sigma \vdash^{n + \Phi_\sigma(\eta : \Gamma) + \Phi_\sigma(\eta : \Delta)}_{n' + \Phi_\tau(v : r) + \Phi_\tau(\eta : \Delta)}\circ\; e \rightsquigarrow v, \tau$$

Typing and semantics: new and free

$$\frac{}{\emptyset \vdash^{n}_{n - (\Diamond(C^r) + \mathrm{Size}\,(\mathrm{C}))} \ \mathsf{new}\ C : C^r}\ (\Diamond New)$$

$$\frac{\tau = \sigma[v \mapsto (C, a_1 : 0, \ldots, a_k : 0)]}{\eta, \sigma \vdash^{n + \mathrm{Size}\,(\mathrm{C})}_{n} \circ \ \mathsf{new}\ C \rightsquigarrow v, \tau}\ (\vdash New)$$

$$\frac{}{x : C^r \vdash^{n}_{n + \Diamond(C^r) + \mathrm{Size}\,(\mathrm{C})} \ \mathsf{free}\,(x) : E^r}\ (\Diamond Free)$$

$$\frac{\eta_x = v \qquad \sigma_v = (C, a_1 : v_1, \ldots, a_k : v_k)}{\eta, \sigma \vdash^{n}_{n + \mathrm{Size}\,(\mathrm{C})} \circ \ \mathit{free}(x) \rightsquigarrow 0, (\sigma[v \mapsto \mathit{invalid}])}\ (\vdash Free)$$

RAJA typing rules

$$\frac{\mathsf{A}^{\text{get}}(C^{r}, a) = s \qquad C.a = D}{x : C^{r} \vdash^{n}_{n} x.a : D^{s}} \;(\Diamond Access)$$

$$\frac{\mathsf{A}^{\text{set}}(C^{r}, a) = s \qquad C.a = D}{x : C^{r}, y : D^{s} \vdash^{n}_{n} x.a \leftarrow y : C^{r}} \;(\Diamond Update)$$

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

$$\frac{x \in \Gamma \qquad \Gamma \vdash^{n}_{n'} e_1 : C^{r} \qquad \Gamma \vdash^{n}_{n''} e_2 : C^{r}}{\Gamma \vdash^{n}_{\max(n', n'')} \text{if } x \text{ instanceof } E \text{ then } e_1 \text{ else } e_2 : C^{r}} \;(\Diamond Conditional)$$

Sharing relation

- $\curlyvee(r \,|\, s_1, \ldots, s_i)$: coinductively defined relation between view r and the multiset of views $s_1, \ldots, s_i$:

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

$\Diamond(\cdot)$	rich	poor	rest	first	poorest
List	0	0	0	0	0
Nil	0	0	0	0	0
Cons	1	0	0	1	0

	Cons^{rich}	Cons^{poor}	Cons^{rest}	Cons^{first}	$\mathrm{Cons}^{poorest}$
$A^{get}(\cdot\,, \texttt{next})$	rich	poor	rich	poorest	poorest
$A^{set}(\cdot\,, \texttt{next})$	rich	poor	rich	rich	rich

- $\curlyvee(rich \,|\, rich, poorest)$
- $\curlyvee(rich \,|\, rest, first)$
- NOT $\curlyvee(rich \,|\, rich, rich)$

Sharing relation

- $\curlyvee(r \mid s_1, \ldots, s_i)$: coinductively defined relation between view r and the multiset of views $s_1, \ldots, s_i$:

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

$\Diamond(\cdot)$	rich	poor	rest	first	poorest
List	0	0	0	0	0
Nil	0	0	0	0	0
Cons	1	0	0	1	0

	Cons^{rich}	Cons^{poor}	Cons^{rest}	Cons^{first}	$\text{Cons}^{poorest}$
$A^{get}(\cdot, \texttt{next})$	rich	poor	rich	poorest	poorest
$A^{set}(\cdot, \texttt{next})$	rich	poor	rich	rich	rich

- $\curlyvee(rich \mid rich, poorest)$
- $\curlyvee(rich \mid rest, first)$
- NOT $\curlyvee(rich \mid rich, rich)$

Sharing relation: examples

- *cons* : $List^{rich}$

```
1 -> Cons [next: location 2; elem:100; ]
2 -> Cons [next: location 3; elem:200; ]
3 -> Cons [next: location 4; elem:300; ]
4 -> Cons [next: location 5; elem:400; ]
5 -> Cons [next: location 6; elem:500; ]
{ 6 -> Nil []  }
```

- *cons* : $List^{poorest}$

```
{ 1 -> Cons [next: location 2; elem:100; ]
{ 2 -> Cons [next: location 3; elem:200; ]
{ 3 -> Cons [next: location 4; elem:300; ]
{ 4 -> Cons [next: location 5; elem:400; ]
{ 5 -> Cons [next: location 6; elem:500; ]
{ 6 -> Nil []  }
```

- $\curlyvee(rich \,|\, rich, poorest)$

Sharing relation: examples

- $cons : List^{first}$

```
 1 -> Cons [next: location 2; elem:100; ]
{ 2 -> Cons [next: location 3; elem:200; ]
{ 3 -> Cons [next: location 4; elem:300; ]
{ 4 -> Cons [next: location 5; elem:400; ]
{ 5 -> Cons [next: location 6; elem:500; ]
{ 6 -> Nil []  }
```

- $cons : List^{rest}$

```
{ 1 -> Cons [next: location 2; elem:100; ]
 2 -> Cons [next: location 3; elem:200; ]
 3 -> Cons [next: location 4; elem:300; ]
 4 -> Cons [next: location 5; elem:400; ]
 5 -> Cons [next: location 6; elem:500; ]
{ 6 -> Nil []  }
```

- $\curlyvee(rich \mid rest, first)$

Gaining potential from this

$$\frac{\curlyvee(r \,|\, q, s) \qquad \mathsf{M}(C^r, m) \ni E_1^{r_1}, \ldots, E_j^{r_j} \xrightarrow{n/n'} E_0^{r_0}}{\texttt{this}: C^q, x_1 : E_1^{r_1}, \ldots, x_j : E_j^{r_j} \; \vdash^{n + \Diamond(C^s)}_{n'} \; \mathsf{M}_{\mathsf{body}}(C, m) : E_0^{r_0}}$$

- Implementation without inference:

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

- Typechecker checks sharing: $\curlyvee(\mathit{rich} \,|\, \mathit{rest}, \mathit{first})$

Gaining potential from this

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

- Implementation without inference:

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

- Typechecker checks sharing: $\curlyvee(\mathit{rich} \mid \mathit{rest}, \mathit{first})$

Share rule

- Handling of aliasing: sharing rule

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

Implementation of sharing

- Problems with sharing:
  1. $(\Diamond Share)$ not syntax directed.
  2. Sharing needs to be guessed or calculated.
- Implementation without inference: annotation of every variable occurrence.

$$\frac{}{\Gamma, x : C^{(r|q_1,\ldots,q_i)}, y : D^{(s|q'_1,\ldots,q'_j)} \vdash^{n}_{n} [x \text{ as } q].a \leftarrow [y \text{ as } q'] : C^q} \; (\Diamond Update)$$

- every rule updates the context:

$$\Gamma, x : C^{(r|q_1,\ldots,q_i,q)}, y : D^{(s|q'_1,\ldots,q'_j,q')}$$

- after typechecking, sharing is checked:

$$\text{for every } x : C^{(r|q_1,\ldots,q_n)} \in \Gamma : \text{ check } \curlyvee(r\,|q_1,\ldots,q_n)$$

Implementation of sharing

- Problems with sharing:
  1. $(\Diamond \mathit{Share})$ not syntax directed.
  2. Sharing needs to be guessed or calculated.
- Implementation without inference: annotation of every variable occurrence.

$$\frac{}{\Gamma, x : C^{(r \mid q_1,\ldots,q_i)}, y : D^{(s \mid q'_1,\ldots,q'_j)} \vdash^{n}_{n} [x \text{ as } q].a \leftarrow [y \text{ as } q'] : C^{q}} \; (\Diamond \mathit{Update})$$

- every rule updates the context:

$$\Gamma, x : C^{(r \mid q_1,\ldots,q_i,q)}, y : D^{(s \mid q'_1,\ldots,q'_j,q')}$$

- after typechecking, sharing is checked:

$$\text{for every } x : C^{(r \mid q_1,\ldots,q_n)} \in \Gamma : \ \text{check } \curlyvee(r \mid q_1, \ldots, q_n)$$

Implementation and Examples

- More implementation features:
  - Typechecker for RAJA programs: prediction of free-list size.
  - Interpreter for FJEU programs with built-in calculation of free-list size.
  - Integers as primitives.
  - Arithmetic expressions.
  - Booleans as built-in classes Bool, True and False.

- Examples
  1. Copy of lists: correct prediction of linear space consumption.
  2. Handling with circular data: doubly linked lists.
  3. More computations: merge sort.

- Demo...

Implementation and Examples

- More implementation features:
  - Typechecker for RAJA programs: prediction of free-list size.
  - Interpreter for FJEU programs with built-in calculation of free-list size.
  - Integers as primitives.
  - Arithmetic expressions.
  - Booleans as built-in classes Bool, True and False.
- Examples
  1. Copy of lists: correct prediction of linear space consumption.
  2. Handling with circular data: doubly linked lists.
  3. More computations: merge sort.
- Demo...

Conclusions and further work

- Conclusions
  - Our type-based analysis encompasses:
    1. Objects
    2. Inheritance
    3. Downcast
    4. Imperative update
    5. Aliasing
    6. Circular data
  - Implementation allows:
    - Experimenting with the system
    - Testing the system with bigger programs
- Further work
  - More examples: Iterators on lists, etc.
  - Improving the type system with new features: polymorphism, etc.
  - Type inference