Relm & RelmInfer: Checking and Inference of Reference Immutability and Method Purity

Wei Huang¹, Ana Milanova¹, Werner Dietl², Michael D. Ernst²

¹Rensselaer Polytechnic Institute
²University of Washington
Reference Immutability

```javascript
md.setHours(3); // ✓
rd.setHours(2); // ✗
```
class Class{
    private Object [] signers;
    public Object [] getSigners() {
        return signers;
    }
}

Object [] signers = getSigners();
signers[0] = maliciousClass;

A real security flaw in Java 1.1
class Class{
    private Object [] signers;
    public Object readonly [] getSigners() {
        return signers;
    }
}

...

Object readonly [] signers = getSigners();
signers[0] = maliciousClass;
Contributions

- Relm: A context-sensitive type system for reference immutability
- RelmInfer: An inference algorithm for Relm
- Method purity – an application of Relm
- Implementation and evaluation
Motivation for Relm and RelmInfer

• Concrete need for method purity
  ◦ Available tools unstable and/or imprecise

• Javari [Tschantz & Ernst OOPSLA’05] and Javarifier [Quinonez et al. ECOOP’08] separate immutability of a container from its elements
  ◦ Unsuitable for purity inference

• Javarifier can be slow
Overview

Relm Typing Rules → Set-based Solver → Set-based Solution → Extract Concrete Typing → Maximal Typing → Preference Ranking over Qualifiers → Type Checking
Immutability Qualifiers

- mutable
  - A mutable reference can be used to mutate the referent

- readonly
  - A readonly reference cannot be used to mutate the referent

```java
readonly C x = ...;
x.f = z;       // not allowed
x.setField(z); // not allowed
```
Context-insensitive Typing

class DateCell {
    mutable Date date;
    mutable Date getDate(mutable DateCell this) {
        return this.date;
    }
    void setHours(mutable DateCell this) {
        Date md = this.getDate();
        md.hours = 1;
    }
    int getHours(mutable DateCell this) {
        Date rd = this.getDate();
        int hour = rd.hours;
        return hour;
    }
}
Immutability Qualifiers

- polyread
  - The mutability of a polyread reference depends on the context

```java
class C {
    polyread D f;
    ...
}
...
mutable C c1 = ...;
c1.f.g = 0;    // allowed
readonly C c2 = ...;
c2.f.g = 0;    // not allowed
```
class DateCell {
    polyread Date date;
    polyread Date getDate(polyread DateCell this) {
        return this.date;
    }
    void setHours(mutabale DateCell this) {
        mutable Date md = this.getDate();
        md.hour = 1;
    }
    int getHours(readonly DateCell this) {
        readonly Date rd = this.getDate();
        int hour = rd.hour;
        return hour;
    }
}
Viewpoint Adaptation

- Encodes context sensitivity
  - Adapts a type $q$ from the viewpoint of another type $q'$
  - Viewpoint adaptation operation:

\[
\begin{align*}
O_1 & \xrightarrow{q'} O_2 \\
O_2 & \xrightarrow{q} O_3
\end{align*}
\]

- $q' \triangleright q$

- Viewpoint adaptation operation:

\[
\begin{align*}
_\_ \triangleright \text{mutable} & = \text{mutable} \\
_\_ \triangleright \text{readonly} & = \text{readonly} \\
q \triangleright \text{polyread} & = q
\end{align*}
\]
Generalizes Viewpoint Adaptation

• Traditional viewpoint adaptation [Dietl & Müller JOT’05]
  ◦ Always adapts from the viewpoint of receiver
    • $x$ in field access $x.f$
    • $y$ in method call $y.m(z)$

• Relm adapts from different viewpoints
  ◦ receiver at field access
    • $x$ in $x.f$
  ◦ left-hand-side of call assignment at method call
    • $x$ in $x = y.m(z)$
class DateCell {
    polyread Date date;
    polyread Date getDate(polyread DateCell this) {
        return this.date;
    }
    void setHours(mutable DateCell this) {
        mutable Date md = this.getDate();
        md.hour = 1;
    }
    int getHours(readonly DateCell this) {
        readonly Date rd = this.getDate();
        int hour = rd.hour;
        return hour;
    }
}
## Subtyping Hierarchy

- `mutable <: polyread <: readonly`

<table>
<thead>
<tr>
<th>mutable</th>
<th>Object mo;</th>
</tr>
</thead>
<tbody>
<tr>
<td>polyread</td>
<td>Object po;</td>
</tr>
<tr>
<td>readonly</td>
<td>Object ro;</td>
</tr>
</tbody>
</table>

- `ro = po; ✓  po = ro; ✗`
- `ro = mo; ✓  mo = ro; ✗`
- `po = mo; ✓  mo = po; ✗`
Typing Rules

(TREAD)
\[\Gamma(x) = q_x \quad \Gamma(y) = q_y \quad \text{typeof}(f) = q_f\]
\[q_y \triangleright q_f \preceq q_x\]
\[\Gamma \vdash x = y.f\]

(TWRITE)
\[\Gamma(x) = q_x \quad \Gamma(y) = q_y \quad \text{typeof}(f) = q_f\]
\[q_x = \text{mutable} \quad q_y \prec q_x \triangleright q_f\]
\[\Gamma \vdash x.f = y\]

(TCALL)
\[\Gamma(x) = q_x \quad \Gamma(y) = q_y \quad \Gamma(z) = q_z \quad \text{typeof}(m) = q_{\text{this}}, q_p \rightarrow q_{\text{ret}}\]
\[q_y \prec q_x \triangleright q_{\text{this}} \quad q_z \prec q_x \triangleright q_p \quad q_x \prec q_x \triangleright q_{\text{ret}}\]
\[\Gamma \vdash x = y.m(z)\]
Outline

- Relm type system
- Inference algorithm for Relm
- Method purity inference
- Implementation and evaluation
Set-based Solver

- Set Mapping $S$: variable $\rightarrow$ {readonly, polyread, mutable}
- Iterates over statements $s$
  - $f_s$ removes infeasible qualifiers for each variable in $s$ according to the typing rule
- Until
  - Reaches a fixpoint
Inference Example

class DateCell {
    readonly,polyread,mutable Date date;
    readonly,polyread,mutable Date getDate(
        DateCell this) {
        return this.date;
    }
    void setHours(
        DateCell this) {
        Date md = this.getDate();
        md.hour = 2;
    }
}
Inference Example

class DateCell {
    {readonly,polyread,mutable} Date date;
    {readonly,polyread,mutable} Date getDate(
        {readonly,polyread,mutable} DateCell this)
        {readonly,polyread,mutable} DateCell this)
    {
        return this.date;
    }

    void setHours(
        {readonly,polyread,mutable} DateCell this) {
        {readonly,polyread,mutable} Date md = this.getDate();
        md.hour = 2;
    }
}
}
class DateCell {
    Date date;

    Date getDate()
    {
        DateCell this;
        return this.date;
    }

    void setHours()
    {
        Date md = this.getDate();
        md.hour = 2;
    }
}
Inference Example

class DateCell {
    {readonly,polyread,mutable} Date date;
    {readonly,polyread,mutable} Date getDate(
        {readonly,polyread,mutable} DateCell this) {
        return this.date;
    }
    void setHours(
        {readonly,polyread,mutable} DateCell this) {
        {readonly,polyread,mutable} Date md = this.getDate();
        md.hour = 2;
    }
}
class DateCell {
    {readonly,polyread,mutable} Date date;
    {readonly,polyread,mutable} Date getDate(
        {readonly,polyread,mutable} DateCell this)
    {
        return this.date;
    }
    void setHours(
        {readonly,polyread,mutable} DateCell this) {
        {readonly,polyread,mutable} Date md = this.getDate();
        md.hour = 2;
    }
}
Maximal Typing

class DateCell {
    {readonly,polyread,mutable} Date date;
    {readonly,polyread,mutable} Date getDate(
        {readonly,polyread,mutable} DateCell this) {
        return this.date;
    }
    void setHours(
        {readonly,polyread,mutable} DateCell this) {
        {readonly,polyread,mutable} Date md = this.getDate();
        md.hour = 2;
    }
}

Maximal Typing: Pick the maximal qualifier from each set

Ranking: readonly > polyread > mutable

Maximal Typing always provably type checks!
Outline

• Relm type system
• Inference algorithm for Relm
• Method purity inference
• Implementation and evaluation
Purity

- A method is *pure* if it does not mutate any object that exists in *prestates* [Sălcianu & Rinard VMCAI’05]

- If a method does not access static states
  - The *prestates* are from parameters
  - If any of the parameters is *mutable*, the method is *impure*; otherwise, it is *pure*
Purity Example

class List {
    Node head;
    int len;
    void add(mutable Node this, mutable Node n) {
        n.next = this.head;
        this.head = n;
        this.len++;
    }
    int size(readonly Node this) { return this.len;
    }
}
Outline

- Relm type system
- Inference algorithm for Relm
- Method purity inference
- Implementation and evaluation
Implementation

• Built on top of the Checker Framework
  [Papi et al. ISSTA’08, Dietl et al. ICSE’11]
• Extends the framework to specify:
  ◦ Ranking over qualifiers
  ◦ Viewpoint adaptation operation
• Publicly available at
  ◦ http://code.google.com/p/type-inference/
Reference Immutability Evaluation

- 13 benchmarks, comprising 766K LOC in total
  - 4 whole Java programs and 9 Java libraries
- Comparison with Javarifier [Quinonez et al. ECOOP’08]
  - Equally precise results
    - Differences are due to different semantics of Javari and Relm
  - Better scalability
Reference Immutability Results

- readonly
- polyread
- mutable

Average
Performance Comparison

![Graph showing performance comparison between RelmInfer and Javarifier]

- Jolden: 6kLOC - 4.0s
- tinySQL: 32kLOC - 11.3s
- htmlparser: 62kLOC - 11.4s
- ejc: 110kLOC - 43.3s
- xalan: 348kLOC - 57.4s
- RelmInfer
- Javarifier

Inference Time (in seconds)
Purity Evaluation

- Comparison with JPPA [Sălcianu & Rinard VMCAI’05] and JPure [Pearce CC’11]
  - Equal or better precision
    - Differences are due to different definitions of purity
  - Works with both whole programs and libraries
  - More robust!
Purity Inference Results

- JOlden: 60% Pure, 40% Impure
- tinySQL: 60% Pure, 40% Impure
- htmlparser: 50% Pure, 50% Impure
- ejc: 50% Pure, 50% Impure
- xalan: 50% Pure, 50% Impure
- javad: 50% Pure, 50% Impure
- SPECjbb: 50% Pure, 50% Impure
- commons-pool: 50% Pure, 50% Impure
- jdbm: 50% Pure, 50% Impure
- jdbf: 50% Pure, 50% Impure
- jtds: 50% Pure, 50% Impure
- java.lang: 40% Pure, 60% Impure
- java.util: 30% Pure, 70% Impure
- Average: 40% Pure, 60% Impure
Related Work

- **Javari** [Tschantz & Ernst OOPSLA’05] and Javarifier [Quinonez et al. ECOOP’08]
  - Javari allows excluding fields from state
  - Handles generics and arrays differently
- **JPPA** [Sălcianu & Rinard VMCAI’05]
  - Relies on pointer and escape analysis
  - Works on whole program
- **JPure** [Pearce CC’11]
  - Modular purity system for Java
  - Exploits *freshness* and *locality*
Conclusions

• A type system for reference immutability
• An efficient type inference algorithm
• Method purity inference
• Evaluation on 766 kLOC
• Publicly available at
  ◦ http://code.google.com/p/type-inference/
Conclusions

- A type system for reference immutability
- An efficient type inference algorithm
- Method purity inference
- Evaluation on 766 kLOC
- Publicly available at
  - http://code.google.com/p/type-inference/
## Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOlden</td>
<td>6,223</td>
<td>Benchmark suite of 10 small programs</td>
</tr>
<tr>
<td>javad</td>
<td>4,207</td>
<td>Java class file disassembler</td>
</tr>
<tr>
<td>SPECjbb</td>
<td>12,076</td>
<td>A SPEC's benchmark</td>
</tr>
<tr>
<td>ejc</td>
<td>110,822</td>
<td>Java compiler of the Eclipse IDE</td>
</tr>
<tr>
<td>commons-pool</td>
<td>4,755</td>
<td>A generic object-pooling library</td>
</tr>
<tr>
<td>jdbm</td>
<td>11,610</td>
<td>A lightweight transactional persistence engine</td>
</tr>
<tr>
<td>jdbf</td>
<td>15,961</td>
<td>An object-relational mapping system</td>
</tr>
<tr>
<td>tinySQL</td>
<td>31,980</td>
<td>Database engine</td>
</tr>
<tr>
<td>jtds</td>
<td>38,064</td>
<td>A JDBC driver for Microsoft SQL Server and Sybase</td>
</tr>
<tr>
<td>java.lang</td>
<td>43,282</td>
<td>A package from JDK 1.6</td>
</tr>
<tr>
<td>java.util</td>
<td>59,960</td>
<td>A package from JDK 1.6</td>
</tr>
<tr>
<td>htmlparser</td>
<td>62,627</td>
<td>HTML parser</td>
</tr>
<tr>
<td>xalan</td>
<td>348,229</td>
<td>A library for transforming XML documents to HTML</td>
</tr>
</tbody>
</table>
# Precision Evaluation on JOlden

<table>
<thead>
<tr>
<th>Program</th>
<th>#Meth</th>
<th>JPPA</th>
<th>JPure</th>
<th>RelmInfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>BH</td>
<td>69</td>
<td>20 (29%)</td>
<td>N/A</td>
<td>33 (48%)</td>
</tr>
<tr>
<td>BiSort</td>
<td>13</td>
<td>4 (31%)</td>
<td>3 (23%)</td>
<td>5 (38%)</td>
</tr>
<tr>
<td>Em3d</td>
<td>19</td>
<td>4 (21%)</td>
<td>1 (5%)</td>
<td>8 (42%)</td>
</tr>
<tr>
<td>Health</td>
<td>26</td>
<td>6 (23%)</td>
<td>2 (8%)</td>
<td>11 (42%)</td>
</tr>
<tr>
<td>MST</td>
<td>33</td>
<td>15 (45%)</td>
<td>12 (36%)</td>
<td>16 (48%)</td>
</tr>
<tr>
<td>Perimeter</td>
<td>42</td>
<td>27 (64%)</td>
<td>31 (74%)</td>
<td>38 (90%)</td>
</tr>
<tr>
<td>Power</td>
<td>29</td>
<td>4 (14%)</td>
<td>2 (7%)</td>
<td>10 (34%)</td>
</tr>
<tr>
<td>TSP</td>
<td>14</td>
<td>4 (29%)</td>
<td>0 (0%)</td>
<td>1 (7%)</td>
</tr>
<tr>
<td>TreeAdd</td>
<td>10</td>
<td>1 (10%)</td>
<td>1 (10%)</td>
<td>6 (60%)</td>
</tr>
<tr>
<td>Voronoi</td>
<td>71</td>
<td>40 (56%)</td>
<td>30 (42%)</td>
<td>47 (66%)</td>
</tr>
</tbody>
</table>
Precision Comparison

• Compare with Javarifier [Quinonez et al. ECOOP’08]
• JOlden benchmark
  ◦ 34 differences from Javarifier, out of 758 identifiable references
• Other benchmarks
  ◦ Randomly select 4 classes from each benchmarks
  ◦ 2 differences from Javarifier, out of 868 identifiable references
Method Purity

- A method is *pure* if it does not mutate any object that exists in *prestates*

- Applications
  - Compiler optimization [Lhoták & Hendren CC’05]
  - Model checking [Tkachuk & Dwyer ESEC/FSE’03]
  - Atomicity [Flanagna et al. TOSE’05]
Prestates From Static Fields

- Static immutability type $q_m$ for each method $q_m$ can be:
  - mutable: $m$ mutates static states
  - readonly: $m$ never mutates static states
  - polyread: $m$ never mutates static states, but the static states it returns to its callers are mutated

```plaintext
polyread X static get() { return sf; }
...
polyread X x = get();  x.f = 0;
```
Extended Typing Rules

- Extends Relm typing rules to enforce *static immutability types*

\[(\text{TSWRITE})\]
\[
\begin{align*}
\text{methodof}(sf \equiv x) &= m \\
\text{statictypeof}(m) &= q_m \\
\Gamma \vdash sf = x
\end{align*}
\]

\[ q_m = \text{mutable} \]

\[(\text{TSREAD})\]
\[
\begin{align*}
\text{methodof}(x \equiv sf) &= m \\
\text{statictypeof}(m) &= q_m \\
\Gamma(x) &= q_x \\
\Gamma \vdash x = sf
\end{align*}
\]

\[ q_m \preceq q_x \]
Example

```c
void m() {
    x = sf; // a static field read
    y = x.f;
    z = id(y);
    z.g = 0;
    ...
}
```

- Extended typing rule (TSREAD) enforce

\[ q_m \triangleleft q_x \]

Because \( q_x \) is mutable, then \( q_m \) is mutable
Infer Purity

- \( q_m \) is inferred as immutability types
- Each method \( m \) is mapped to \( S(m) = \{\text{readonly, polyread, mutable}\} \) and solved by the set-based solver
- The purity of \( m \) is decided by:

\[
\text{pure}(m) = \begin{cases} 
\text{false} & \text{if } q_{\text{this}} = \text{mutable} \text{ or } q_p = \text{mutable} \text{ or } q_m = \text{mutable} \\
\text{true} & \text{otherwise}
\end{cases}
\]
Precision Comparison

- Compare with Javarifier [Quinonez et al. ECOOP’08]
- 36 differences from Javarifier, out of 1526 identifiable references
  - Due to different semantics of Javari and Relm
Summary

- Relmlnfer produces equally precise results
- Relmlnfer scales better than Javarifier
Precision Comparison with JPPA

• 59 differences from JPPA out of 326 user methods for JOlden benchmark
  ◦ 4 are due to differences in definitions/assumptions
  ◦ 51 are due to limitations/bugs in JPPA
  ◦ 4 are due to limitations in RelmInfer
Precision Comparison with JPure

- **60** differences from JPure out of **257** user methods for JOlden benchmark, excluding the BH program
  - **29** differences are caused by different definitions/assumptions
  - **29** differences are caused by limitations/bugs in JPure
  - **2** are caused by limitations in RelmInfer
Summary

• RelmlInfer shows good precision compared to JPPA and JPure

• RelmlInfer scales well to large programs

• RelmlInfer works with both whole programs and libraries

• RelmlInfer is robust!
class DateCell {
    polyread Date date;
    polyread Date getDate(polyread DateCell this) {
        return this.date;
    }
    void setHours(mutable DateCell this) {
        mutable Date md = this.getDate();
        md.hour = 1;
    }
    int getHours(readonly DateCell this) {
        readonly Date rd = this.getDate();
        int hour = rd.hour;
        return hour;
    }
}