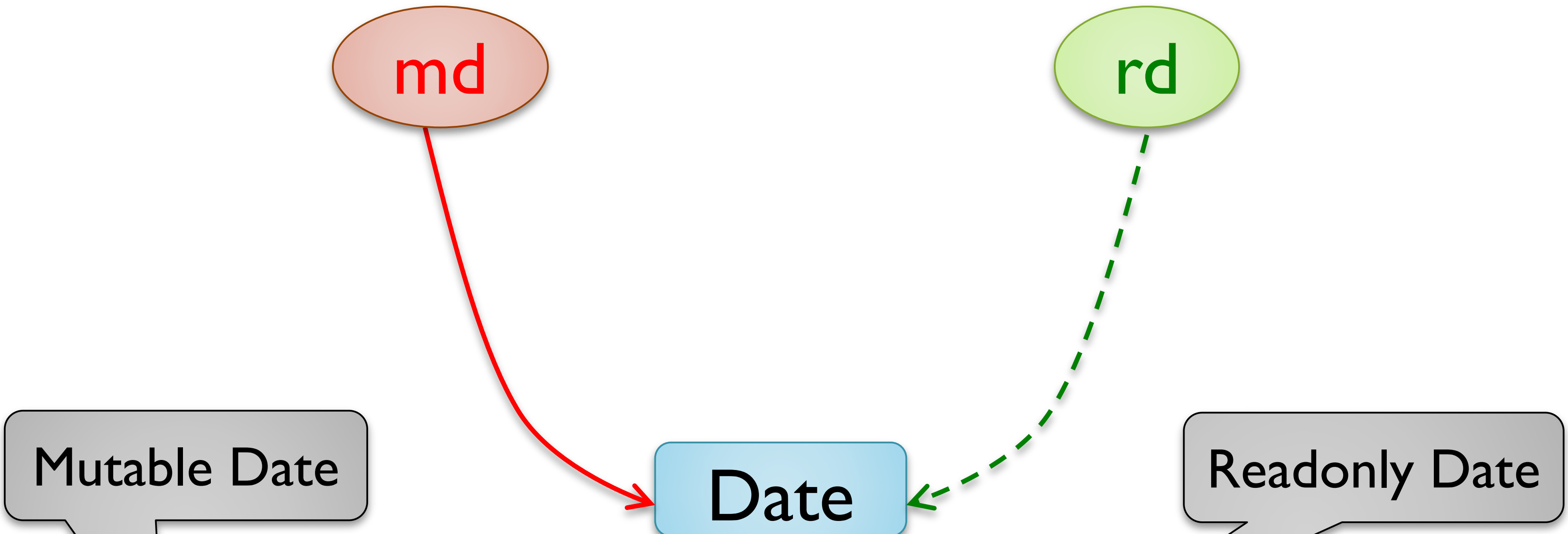
ReIm & ReImInfer: Checking and Inference of Reference Immutability and Method Purity

Wei Huang[1], Ana Milanova[1],
Werner Dietl[2], Michael D. Ernst[2]

[1]Rensselaer Polytechnic Institute
[2]University of Washington

Reference Immutability




```
md.setHours(3);
```

✓

```
rd.setHours(2);
```

✗

Motivating Example

```
class Class{
  private Object [] signers;
  public Object [] getSigners() {
    return signers;
  }
}

...

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

A real security flaw in Java 1.1

Reference Immutability Solution

```
class Class{
  private Object [] signers;
  public Object readonly[] getSigners() {
    return signers;
  }
}

...

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

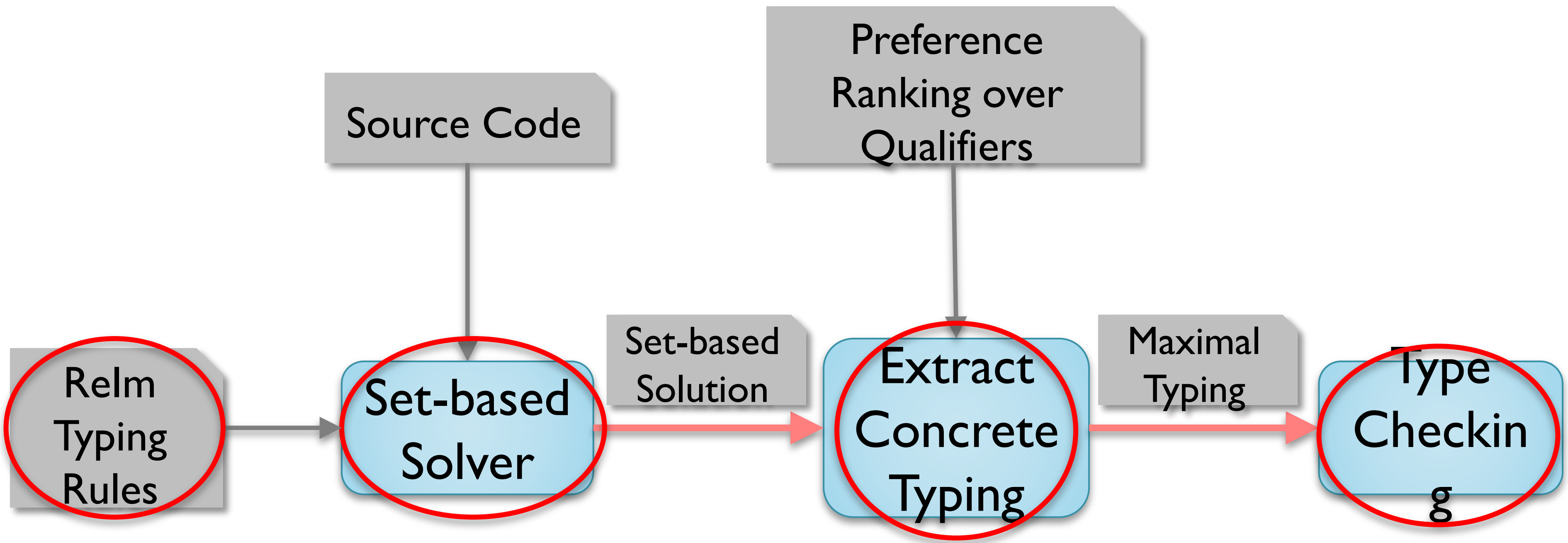
Contributions

- 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 ReIm and ReImInfer

- 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



Source Code
Preference Ranking over Qualifiers
Relm Typing Rules
Set-based Solver
Set-based Solution
Extract Concrete Typing
Maximal Typing
Type Checkin g

Immutability Qualifiers

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

```
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) {
       mutable  Date md = this.getDate();
               md.hours = 1;
  }
  int getHours(mutable DateCell this) {
       readonly Date rd = this.getDate();
               int hour = rd.hours;
               return hour;
  }
}
```

It could have been readonly

Immutability Qualifiers

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

```
class C {
  polyread D f;
   ...
}
...
mutable C c1 = ...;
c1.f.g = 0;         // allowed
readonly C c2 = ...;
c2.f.g = 0;         // not allowed
```

Relm Typing

```
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;
  }
}
```

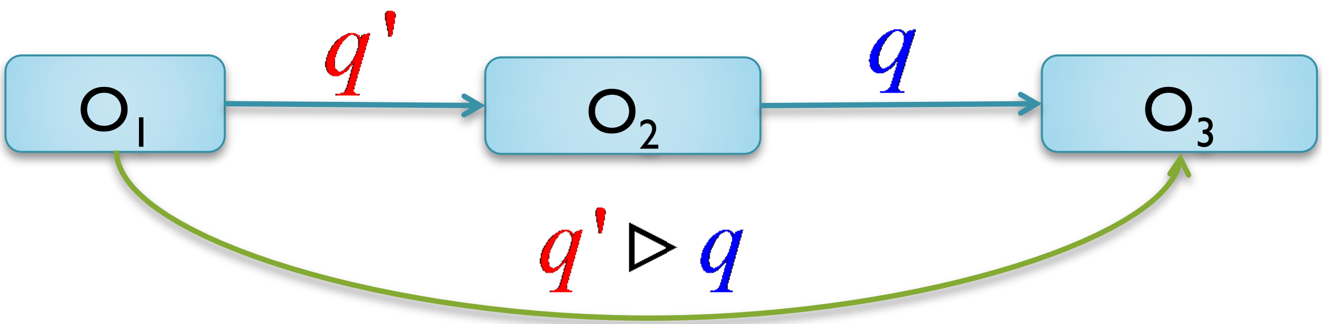
Instantiated to **mutable**

It is **readonly**

Instantiated to **readonly**

Viewpoint Adaptation

- Encodes context sensitivity
  - Adapts a type q from the viewpoint of another type q'



  - Viewpoint adaptation operation:

$$
\boxed{
\begin{array}{lll}
\_ \rhd \text{mutable} & = & \text{mutable} \\
\_ \rhd \text{readonly} & = & \text{readonly} \\
q \rhd \text{polyread} & = & q
\end{array}
}
$$

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)`

Viewpoint Adaptation Example

```
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;
  }
}
```

polyread $\triangleright$ polyread = polyread

mutable $\triangleright$ polyread = mutable

readonly $\triangleright$ polyread = readonly

Subtyping Hierarchy

- mutable <: polyread <: readonly

```
mutable  Object mo;
polyread Object po;
readonly Object ro;

ro = po; ✓  po = ro; ✗
ro = mo; ✓     mo = ro; ✗
po = mo; ✓     mo = po; ✗
```

Typing Rules

(TREAD)

$$\frac{\Gamma(\mathsf{x}) = q_{\mathsf{x}} \quad \Gamma(\mathsf{y}) = q_{\mathsf{y}} \quad \mathit{typeof}(\mathsf{f}) = q_{\mathsf{f}} \qquad q_{\mathsf{y}} \rhd q_{\mathsf{f}} <: q_{\mathsf{x}}}{\Gamma \vdash \mathsf{x} = \mathsf{y.f}}$$

(TWRITE)

$$\frac{\Gamma(\mathsf{x}) = q_{\mathsf{x}} \quad \Gamma(\mathsf{y}) = q_{\mathsf{y}} \quad \mathit{typeof}(\mathsf{f}) = q_{\mathsf{f}} \qquad q_{\mathsf{x}} = \mathsf{mutable} \quad q_{\mathsf{y}} <: q_{\mathsf{x}} \rhd q_{\mathsf{f}}}{\Gamma \vdash \mathsf{x.f} = \mathsf{y}}$$

(TCALL)

$$\frac{\Gamma(\mathsf{x}) = q_{\mathsf{x}} \quad \Gamma(\mathsf{y}) = q_{\mathsf{y}} \quad \Gamma(\mathsf{z}) = q_{\mathsf{z}} \quad \mathit{typeof}(\mathsf{m}) = q_{\mathsf{this}}, q_{\mathsf{p}} \to q_{\mathsf{ret}} \qquad q_{\mathsf{y}} <: q_{\mathsf{x}} \rhd q_{\mathsf{this}} \quad q_{\mathsf{z}} <: q_{\mathsf{x}} \rhd q_{\mathsf{p}} \quad q_{\mathsf{x}} <: q_{\mathsf{x}} \rhd q_{\mathsf{ret}}}{\Gamma \vdash \mathsf{x} = \mathsf{y.m(z)}}$$

Outline

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

Set-based Solver

- Set Mapping S:
  - variable → {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(
          {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;
   }
}
```

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;
   }
}
```

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;
   }
}
```

mutable ▷ readonly = readonly

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;
   }
}
```

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;
   }
}
```

Maximal Typing

Ranking:
readonly > polyread > mutable

```
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

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;
   }
}
```

impure

pure

Outline

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

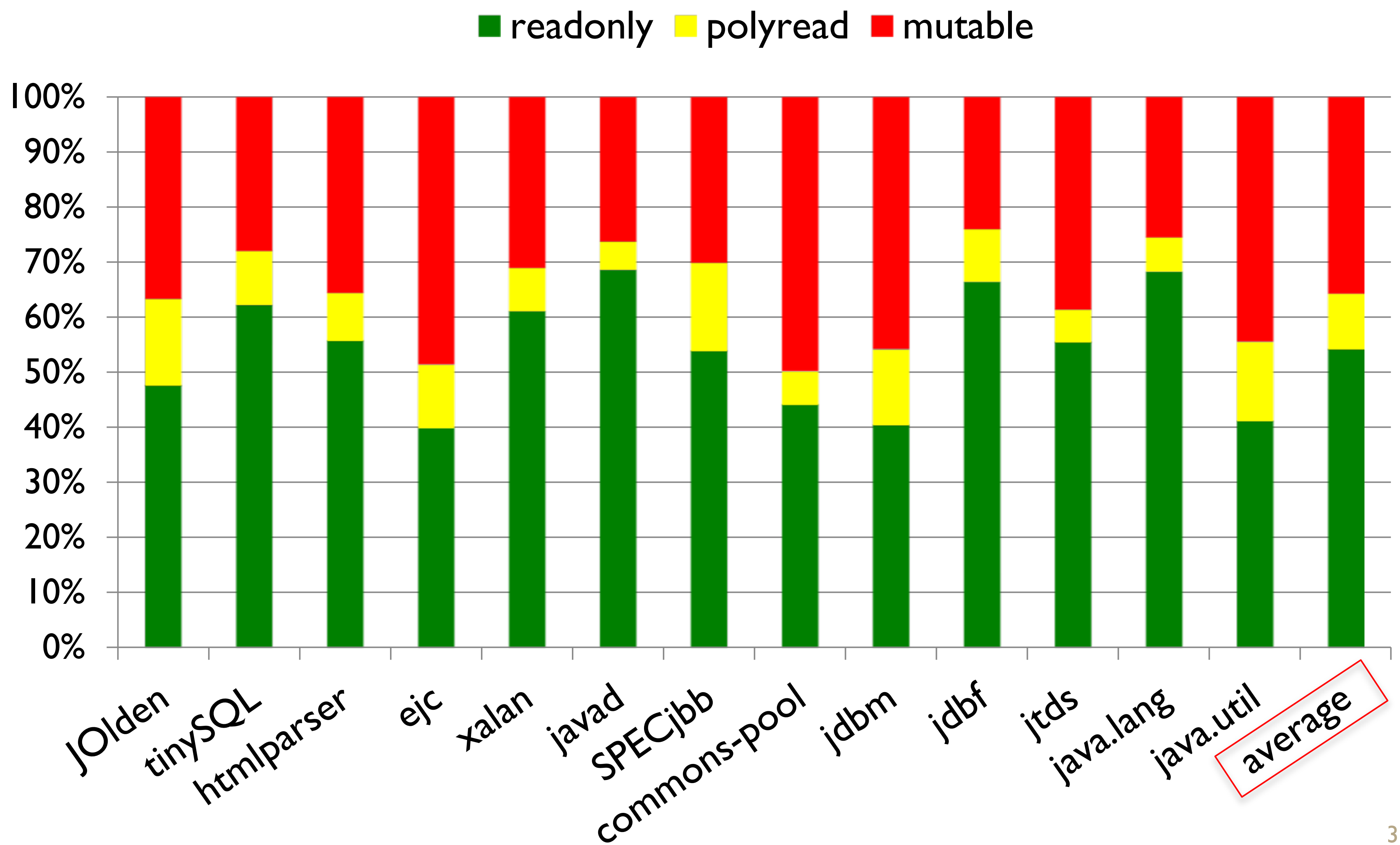
Implementation

- 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/

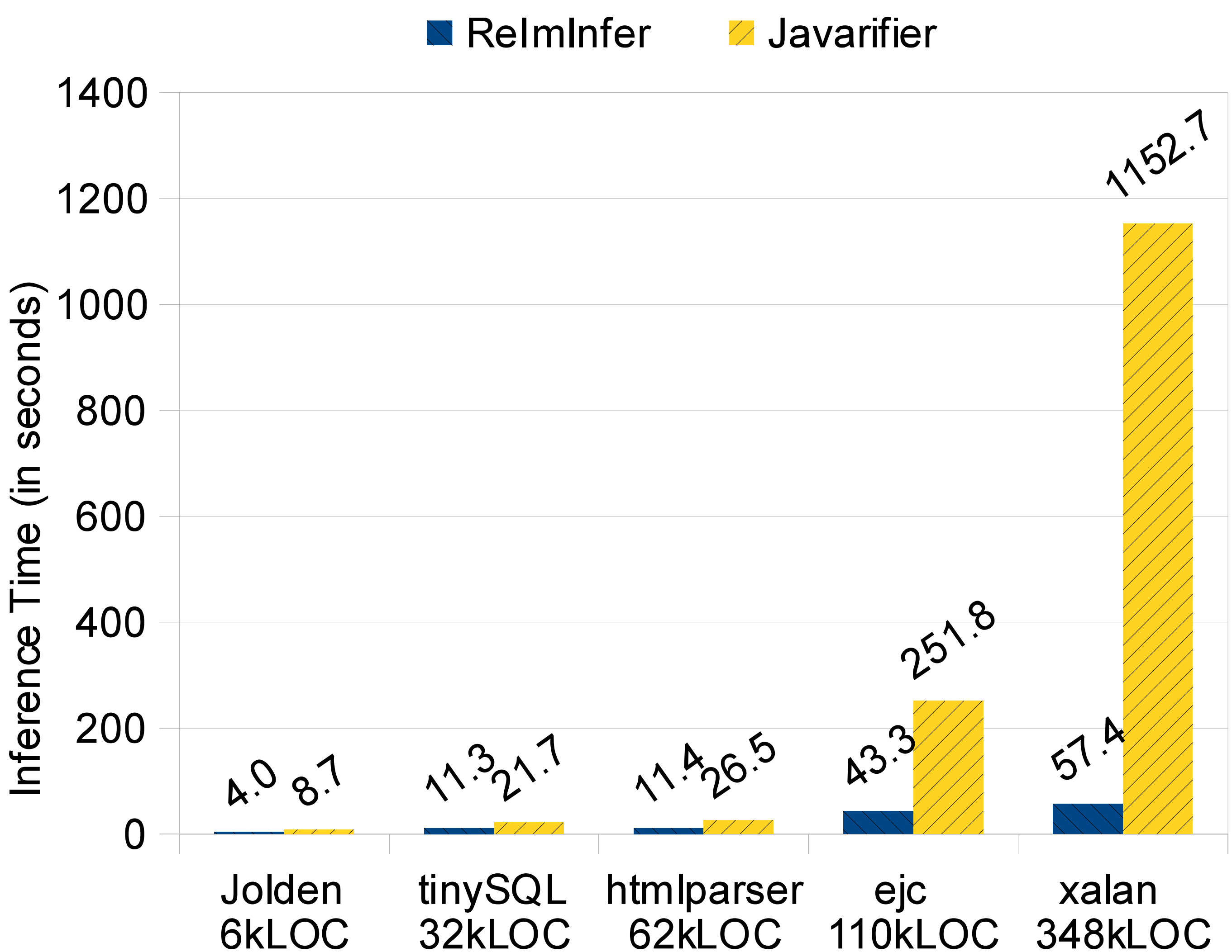
Reference Immutability Evaluation

- 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 ReIm
  - Better scalability

Reference Immutability Results



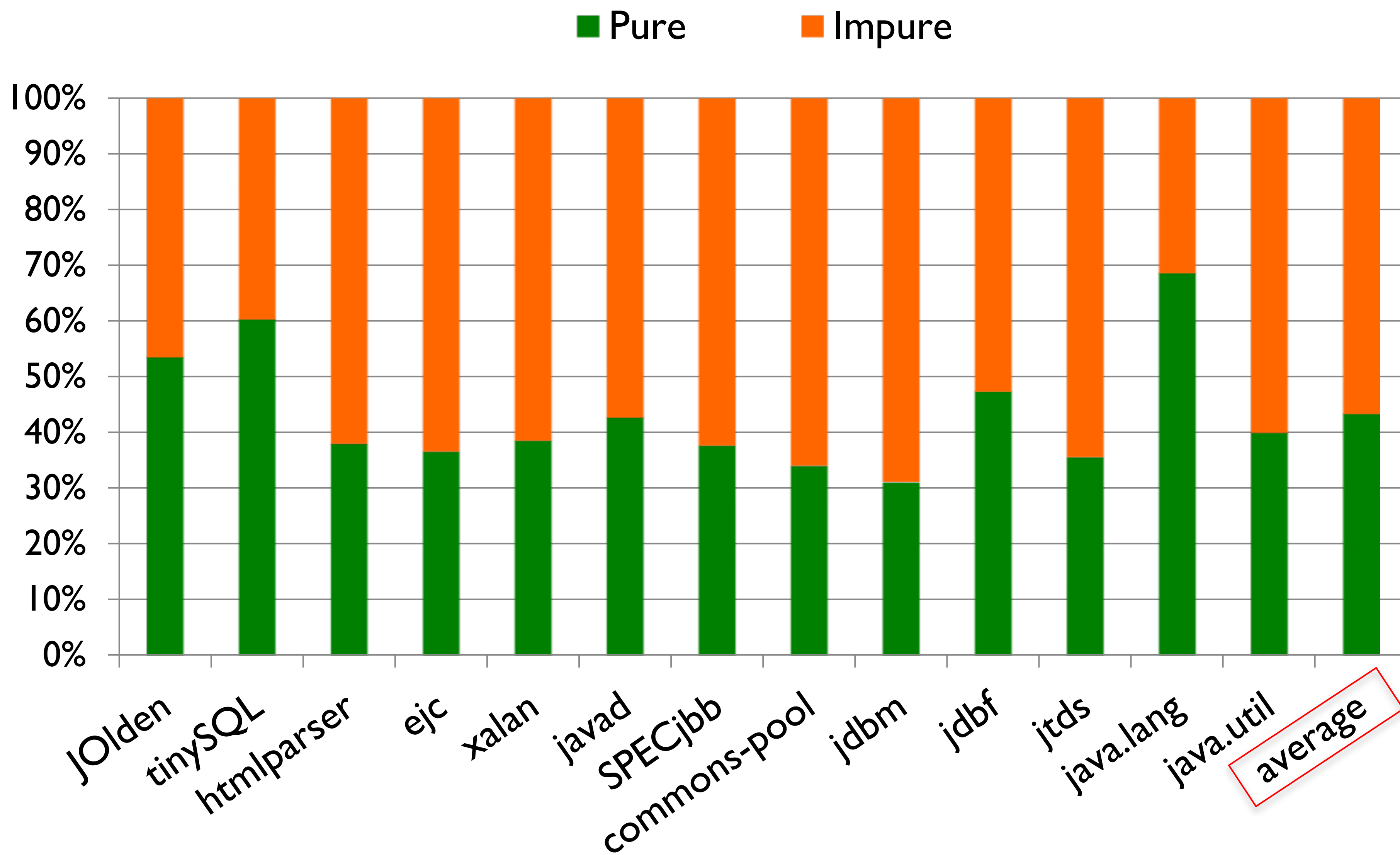
Performance Comparison



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



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

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

Precision Evaluation on JOlden

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

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

q_{get} = polyread

```
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*

(TSWRITE)

$$\frac{\mathit{methodof}(\mathsf{sf} = \mathsf{x}) = \mathsf{m} \quad \mathit{staticttypeof}(\mathsf{m}) = q_{\mathsf{m}} \quad q_{\mathsf{m}} = \mathsf{mutable}}{\Gamma \vdash \mathsf{sf} = \mathsf{x}}$$

(TSREAD)

$$\frac{\mathit{methodof}(\mathsf{x} = \mathsf{sf}) = \mathsf{m} \quad \mathit{statictypeof}(\mathsf{m}) = q_{\mathsf{m}} \qquad \Gamma(\mathsf{x}) = q_{\mathsf{x}} \quad q_{\mathsf{m}} <: q_{\mathsf{x}}}{\Gamma \vdash \mathsf{x} = \mathsf{sf}}$$

Example

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

- Extended typing rule (TSREAD) enforce

$$\boxed{q_m <: q_x}$$

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 $S(\mathrm{m})$ = {readonly, polyread, mutable} and solved by the set-based solver
- The purity of m is decided by:

$$
pure(\mathrm{m}) = \begin{cases} \text{false} & \text{if } q_{\text{this}} = \text{mutable or} \\ & \quad q_p = \text{mutable or} \\ & \quad q_{\mathrm{m}} = \text{mutable} \\ \text{true} & \text{otherwise} \end{cases}
$$

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 ReIm

Summary

- ReImInfer produces equally precise results

- ReImInfer 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 ReImInfer

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 ReImInfer

Summary

- ReImInfer shows good precision compared to JPPA and JPure
- ReImInfer scales well to large programs
- ReImInfer works with both whole programs and libraries
- ReImInfer is robust!

Viewpoint Adaptation Example

```
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;
  }
}
```

polyread $\triangleright$ polyread = polyread

mutable $\triangleright$ polyread = mutable

readonly $\triangleright$ polyread = readonly