Inference and Checking of Object Ownership

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

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

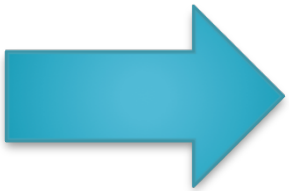
Ownership Types

- Enforce heap encapsulation
- Aid in program correctness and understanding
- Owner-as-Modifier (OaM)
  - An object can be modified only by its owner and by its peers
  - Universe types [Dietl & Müller JOT'05]
- Owner-as-Dominator (OaD)
  - An object can be accessed only through its owner
  - Ownership types [Clark et al. OOPSLA'98]

Annotation Burden is High

```
1   class Link {
2     Link next;  X data;
3     Link(X inData) {
4       next = null;
5       data = inData;
6     }
7   }
8   class XStack {
9     Link top;
10    void push(X data) {
11      Link newTop;
12      newTop = new Link(data);
13      newTop.next = top;
14      top = newTop;
15    }
16    X pop() {
17      Link oldTop = top;
18      top = oldTop.next;
19      return oldTop.data;
20    }
21    boolean isEmpty() {
22      return top == null; }
23    public static void
    main(String[] args) {
24      XStack s;
25      s = new XStack();
26      X x = new X();
27      s.push(x);
28      x = s.pop();
29    }
30  }
```

13 annotations are used in this small program!

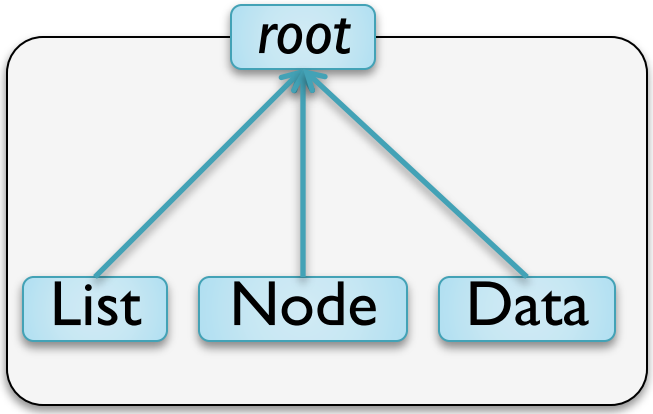


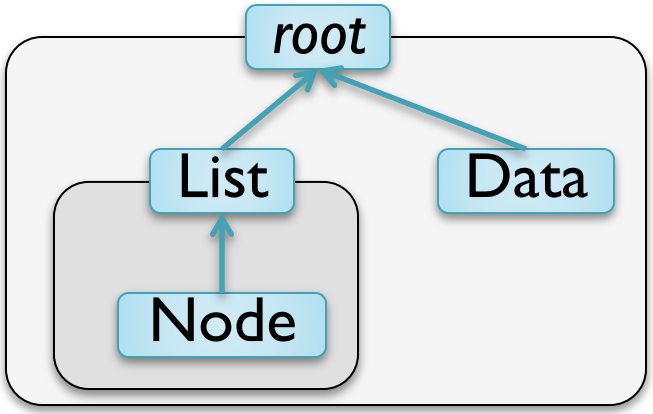
```
1   class Link {
2     <rep|p> Link next;  <p|p> X data;
3     Link(<p|p> X inData) {
4       next = null;
5       data = inData;
6     }
7   }
8   class XStack {
9     <rep|p> Link top;
10    void push(<p|p> X data) {
11      <rep|p> Link newTop;
12      newTop = new <rep|p> Link(data);
13      newTop.next = top;
14      top = newTop;
15    }
16    <p|p> X pop() {
17      <rep|p> Link oldTop = top;
18      top = oldTop.next;
19      return oldTop.data;
20    }
21    boolean isEmpty() {
22      return top == null; }
23    public static void main(String[]
    args) {
24      <rep|rep> XStack s;
25      s = new <rep|rep> XStack();
26      <rep|rep> X x = new <rep|rep> X();
27      s.push(x);
28      x = s.pop();
29    }
30  }
```

Ownership Type Inference

- Transforms un-annotated or partially-annotated programs into fully annotated ones
- Reveals how ownership concepts are expressed in existing programs

Many Valid Typings!





- A good typing should give rise to a deep ownership tree

- Goal: Infer the "best" typing
  - The typing that gives rise to the deepest tree

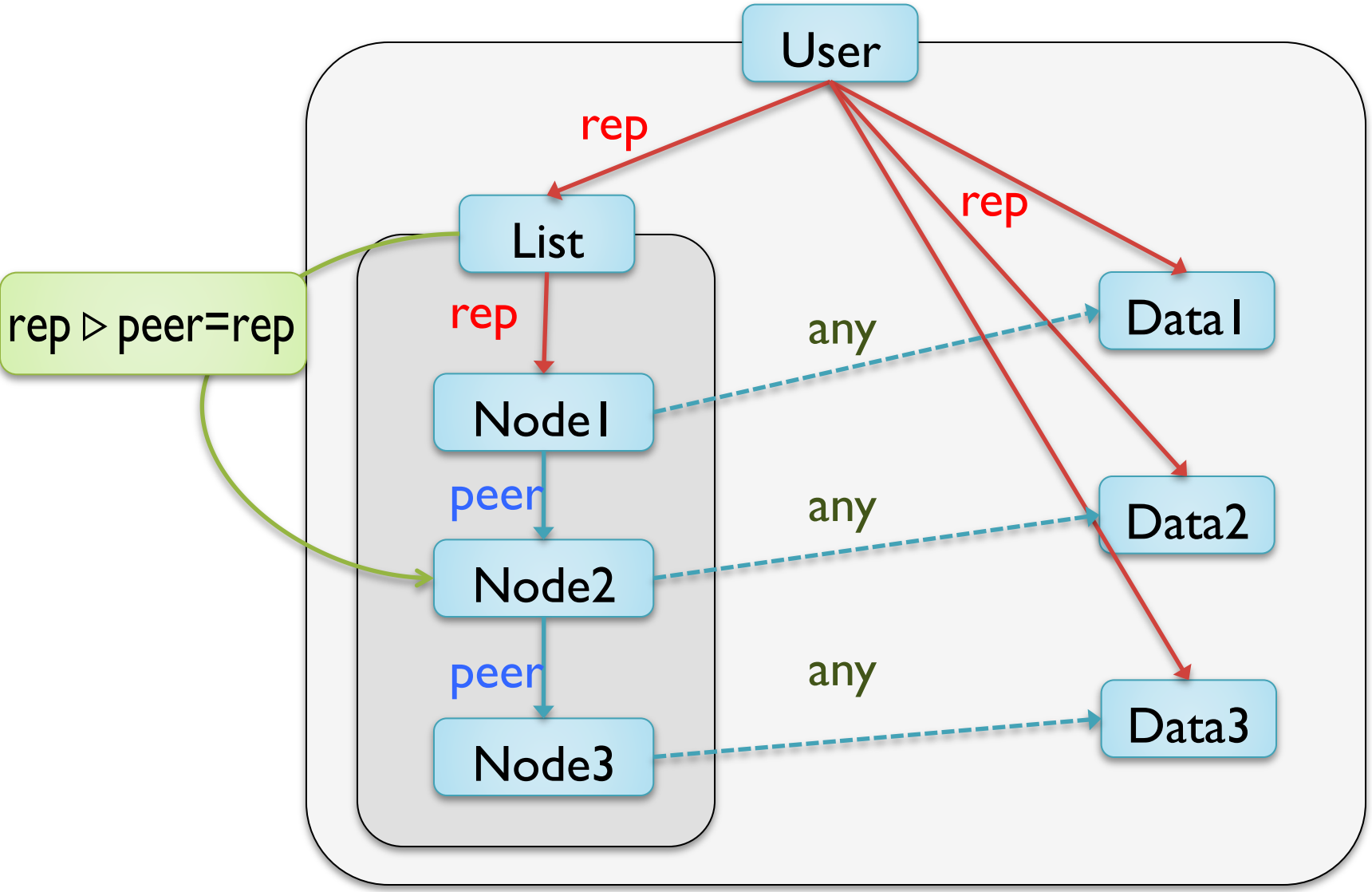
Contributions

- Unified typing rules
  - Universe Types (UT)
  - Ownership Types (OT)
- Unified inference approach
- Notion of "best" typing
- Implementation and evaluation
  - Results for UT and OT
  - Comparison of UT and OT

Universe Types

- Owner-as-Modifier encapsulation (OaM)
- Type qualifiers:
  - **rep**: owned by **this**
  - **peer**: has same owner as **this**
  - **any**: arbitrary ownership

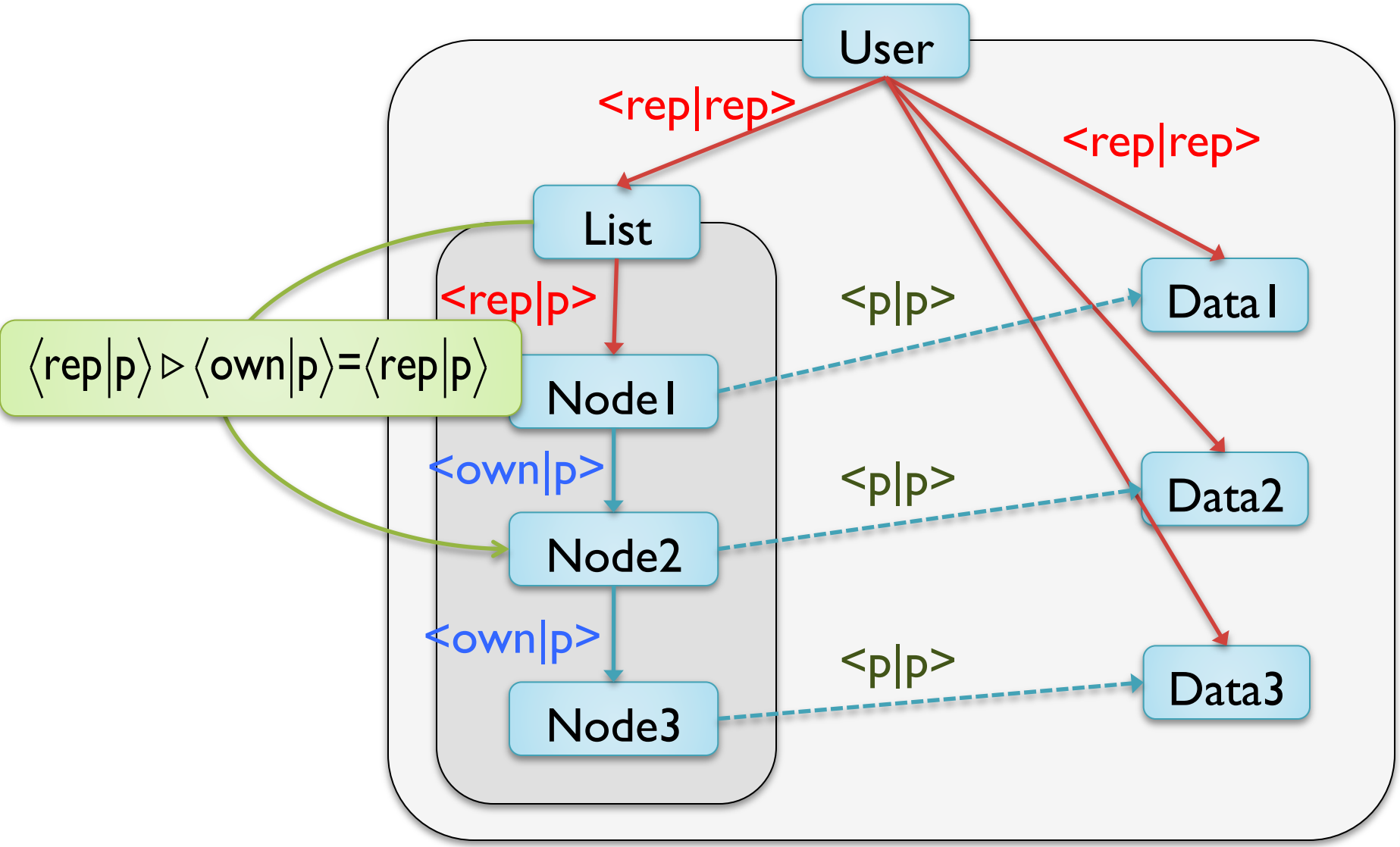
UT Viewpoint Adaptation Example



Classical Ownership Types

- Owner-as-Dominator encapsulation (OaD)
- Type qualifier $\langle q_0 | q_1 \rangle$
  - q_0 is the owner of the object
  - q_1 is the ownership parameter
  - **rep**: owned by **this**
  - **own**: has same owner as **this**
  - **p**: owned by the ownership parameter

OT Viewpoint Adaptation Example



Outline

- Unified typing rules
- Unified inference approach
- Notion of "best" typing
- Implementation and evaluation

Typing Rule (TWRITE): x.f = y

UT: (TWRITE)

$$\frac{\Gamma(x) = q_x \quad \Gamma(y) = q_y \quad typeof(f) = q_f \quad q_y <: q_x \triangleright q_f \quad q_x \neq \text{any} \quad q_x \triangleright q_f \neq \text{lost}}{\Gamma \vdash x.f = y}$$

OT: (TWRITE)

$$\frac{\Gamma(x) = q_x \quad \Gamma(y) = q_y \quad typeof(f) = q_f \quad q_y <: q_x \triangleright q_f}{\Gamma \vdash x.f = y}$$

UT Adaptations:

rep ▷ peer=rep

peer ▷ peer=peer

…

OT Adaptations:

$\langle rep|p \rangle \triangleright \langle own|p \rangle = \langle rep|p \rangle$

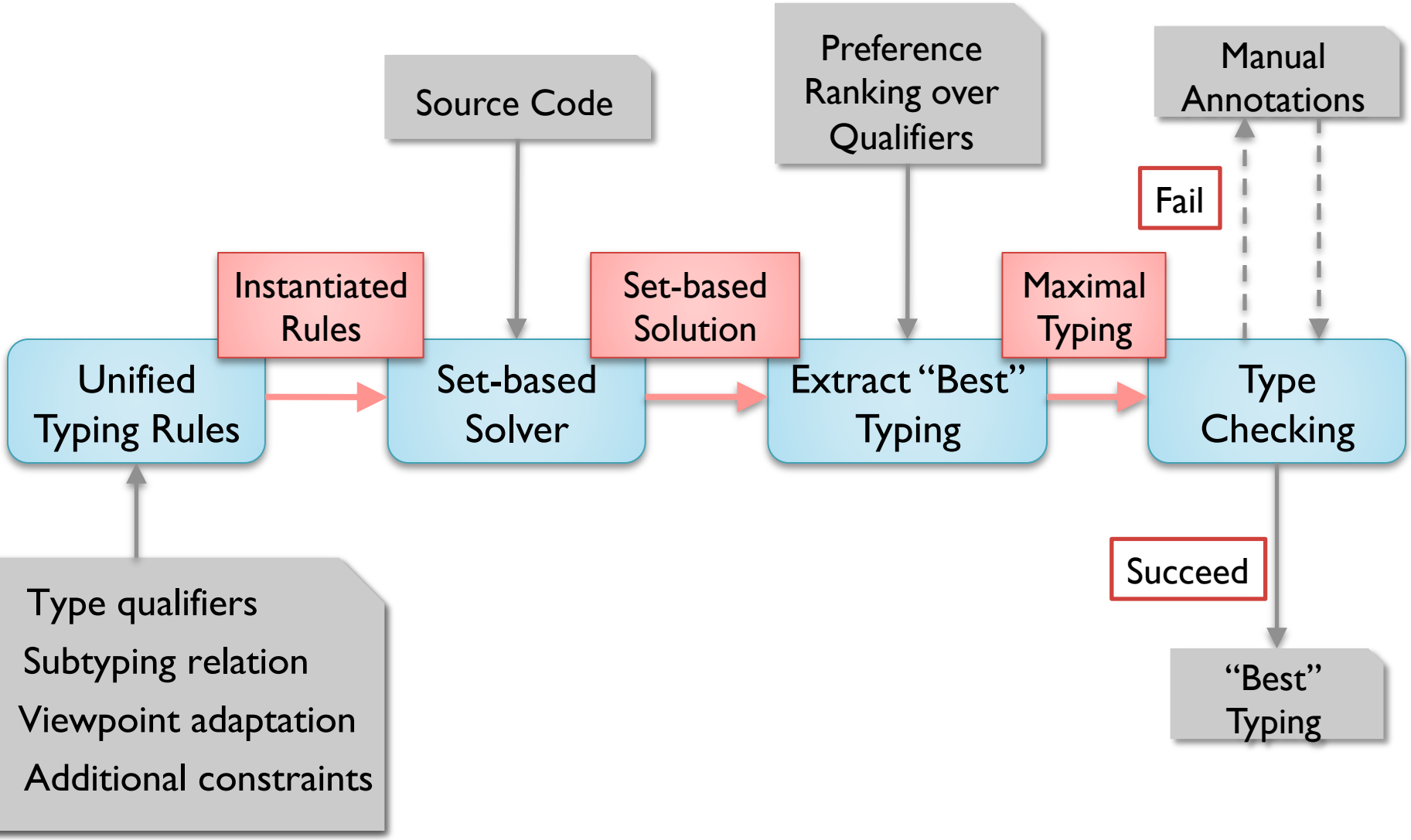
$\langle own|p \rangle \triangleright \langle own|p \rangle = \langle own|p \rangle$

…

Unified: (TWRITE)

$$\frac{\Gamma(x) = q_x \quad \Gamma(y) = q_y \quad typeof(f) = q_f \quad q_y <: q_x \triangleright q_f \quad \beta_{(\text{TWRITE})}}{\Gamma \vdash x.f = y}$$

Architecture



Outline

- Unified typing rules
- Unified inference approach
- Notion of "best" typing
- Implementation and evaluation

Set-based Solver

- Set Mapping: *S*: variable → {possible qualifiers}
  - e.g. *S*(x) = {any, rep, peer}
- Iterates over statements s
  - Applies the transfer function f_s
  - f_s removes infeasible qualifiers for each variable in statement *s* according to the instantiated rules
- Until
  - Reaches the fixpoint, or
  - Assigns the empty set to a variable

Example

```
1  class XStack {
2    {any, rep, peer} Link top;
3    void push( {any, rep, peer} X d) {
4    {any, rep, peer} Link newTop;
5    newTop = new {any, rep, peer} Link();
6    newTop.init(d);
7    ...
8   }
9  }
10 class Link {
11   ...
12   void init( {any, rep, peer} X inData) {
13     ...
14   }
15 }
```

First Iteration

```
1  class XStack {
2    {any, rep, peer} Link top;
3    void push( {any, rep, peer} X d) {
4    {any, rep, peer} Link newTop;
5    newTop = new {any, rep, peer} Link();
6    newTop.init(d);
7    ...
8   }
9  }
10 class Link {
11   ...
12   void init( {any, rep, peer} X inData) {
13     ...
14   }
15 }
```

First Iteration

```
1  class XStack {
2    {any, rep, peer} Link top;
3    void push( {any, rep, peer} X d) {
4    {any, rep, peer} Link newTop;
5    newTop = new {any, rep, peer} Link();
6    newTop.init(d);
7    ...
8   }
9  }
10 class Link {
11   ...
12   void init( {any, rep, peer} X inData) {
13     ...
14   }
15 }
```

Final Result: A Set-based Solution

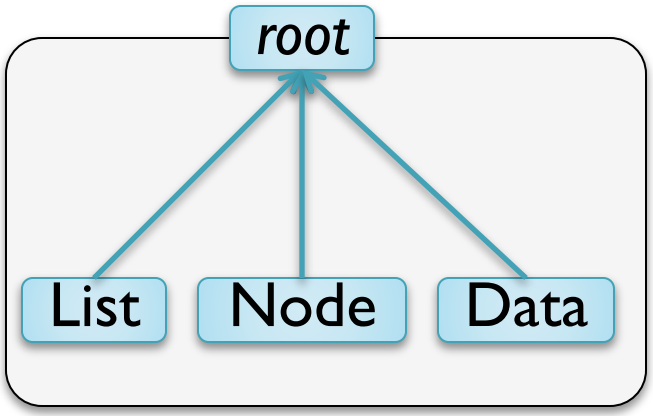
```
1  class XStack {
2    {any, rep, peer} Link top;
3    void push( {any, rep, peer} X d) {
4    {any, rep, peer} Link newTop;
5    newTop = new {any, rep, peer} Link();
6    newTop.init(d);
7    ...
8   }
9  }
10 class Link {
11   ...
12   void init( {any, rep, peer} X inData) {
13     ...
14   }
15 }
```

Outline

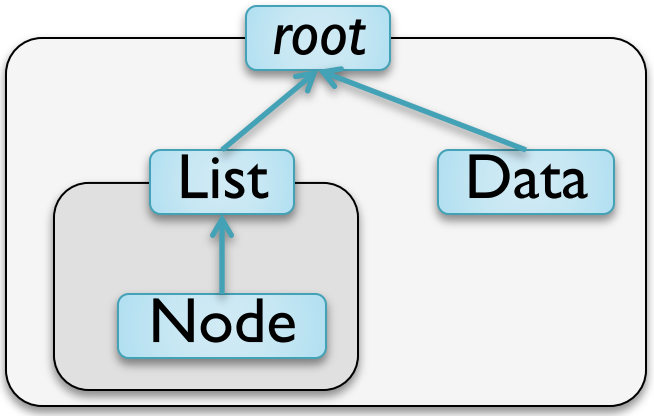
- Unified typing rules
- Unified inference approach
- Notion of "best" typing
- Implementation and evaluation

Set-based Solution

- Many valid typings can be extracted from the solution
- Which one is the "best"?
  - Deeper ownership tree has better encapsulation



Flatter tree



Deeper tree

Notion of "Best" Typing

- Objective functions rank valid typings

$$o_{UT}(T) = (\left|T^{-1}(\text{any})\right|, \left|T^{-1}(\text{rep})\right|, \left|T^{-1}(\text{peer})\right|)$$

ranks UT typings; a proxy for deep UT tree

$$o_{OT}(T) = (\left|T^{-1}(\langle \text{rep} | \_ \rangle)\right|, \left|T^{-1}(\langle \text{own} | \_ \rangle)\right|, \left|T^{-1}(\langle \text{p} | \_ \rangle)\right|)$$

ranks OT typings; a proxy for deep OT tree

- "Best" typing maximizes objective function

Maximal Typing

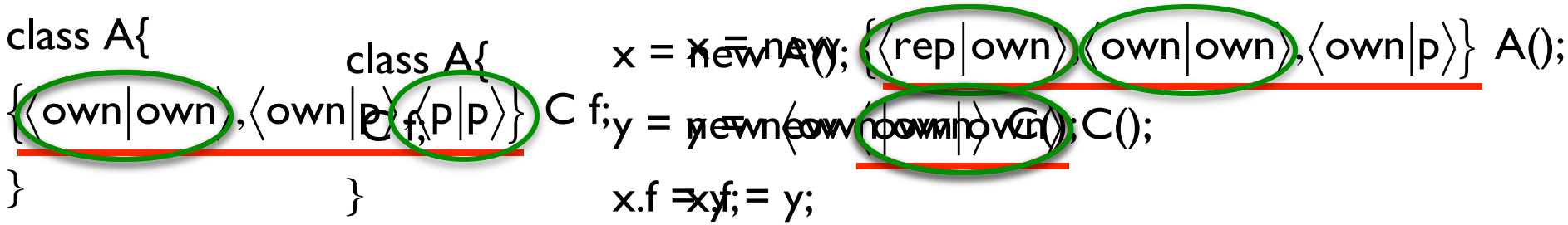
- Maximal typing assigns to each variable x the maximally preferred qualifier from *S*(x)
  - Preference ranking over qualifiers
    - UT: any > rep > peer
    - OT: $\langle \text{rep} | \text{rep} \rangle > \langle \text{rep} | \text{own} \rangle > \langle \text{rep} | \text{p} \rangle > \langle \text{own} | \text{own} \rangle > \langle \text{own} | \text{p} \rangle > \langle \text{p} | \text{p} \rangle$
- If the maximal typing type-checks, then it is the "best" typing
  - UT: the maximal typing always type-checks
  - OT: it does not always type-check

UT: Maximal Typing Always Type Checks

```
1  class XStack {
2    {any, rep, peer} Link top;
3    void push( {any, rep, peer} X d) {
4    {any, rep, peer} Link newTop;
5    newTop = new {any, rep, peer} Link();
6    newTop.init(d);
7    ...
8   }
9  }
10 class Link {
11   ...
12   void init( {any, rep, peer} X inData) {
13     ...
14   }
15 }
```

OT: Maximal Typing Does Not Always Type Check

- Conflict: picking the maximal qualifiers doesn't type-check
- Tool prompts user for manual annotations



$$x \quad . \quad f \quad = \quad y$$

$$\langle rep|own\rangle \triangleright \langle own|own\rangle = \langle rep|rep\rangle \neq \langle own|own\rangle$$

Outline

- Unified typing rules
- Unified inference approach
- Notion of "best" typing
- Implementation and evaluation

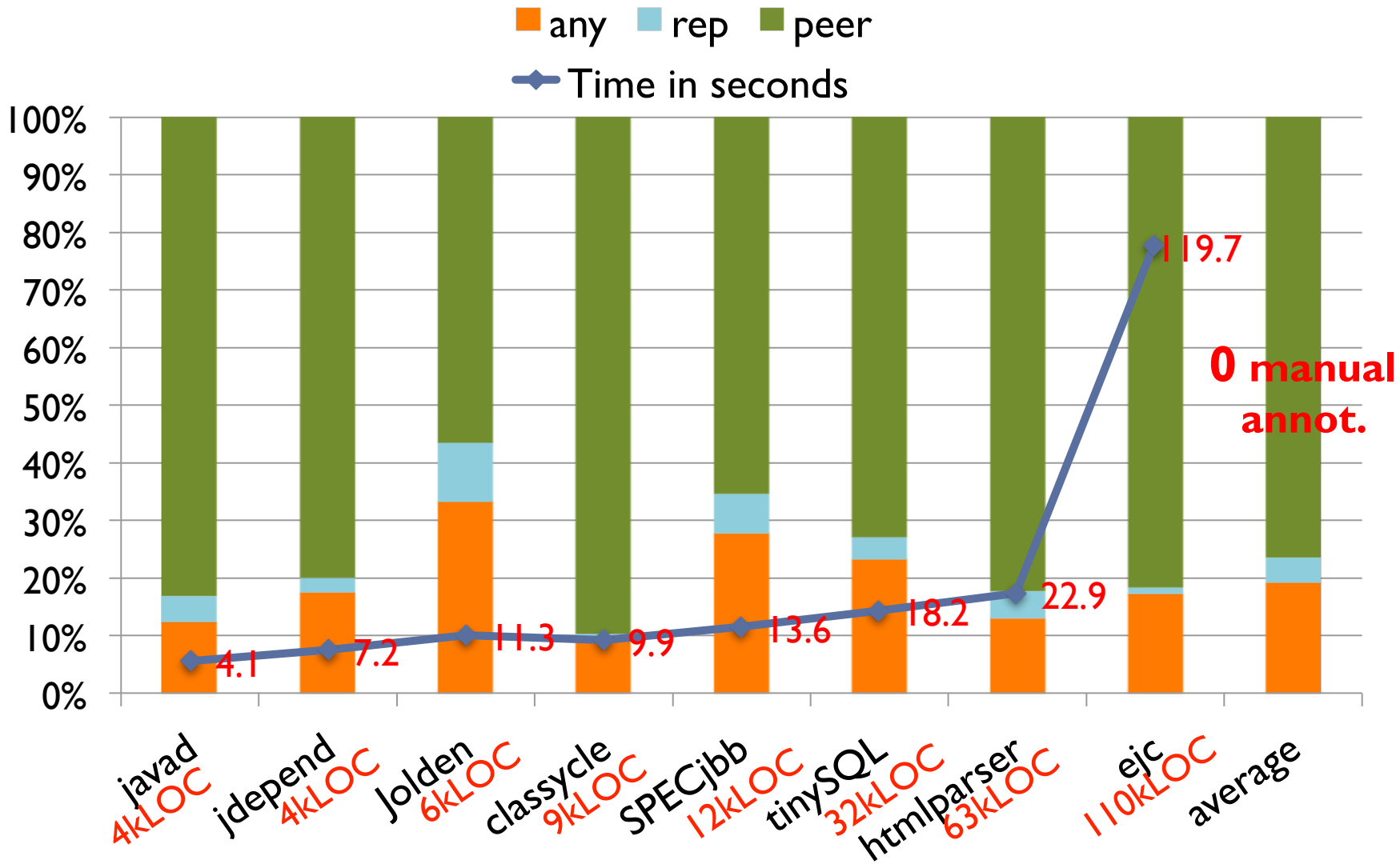
Implementation

- Built on top of the Checker Framework (CF) [Papi et al. ISSTA'08, Dietl et al. ICSE'11]
- Extends the CF to specify:
  - Preference ranking over qualifiers
  - Viewpoint adaptation function
  - Additional constraints
- Publicly available at
  - http://www.cs.rpi.edu/~huangw5/cf-inference

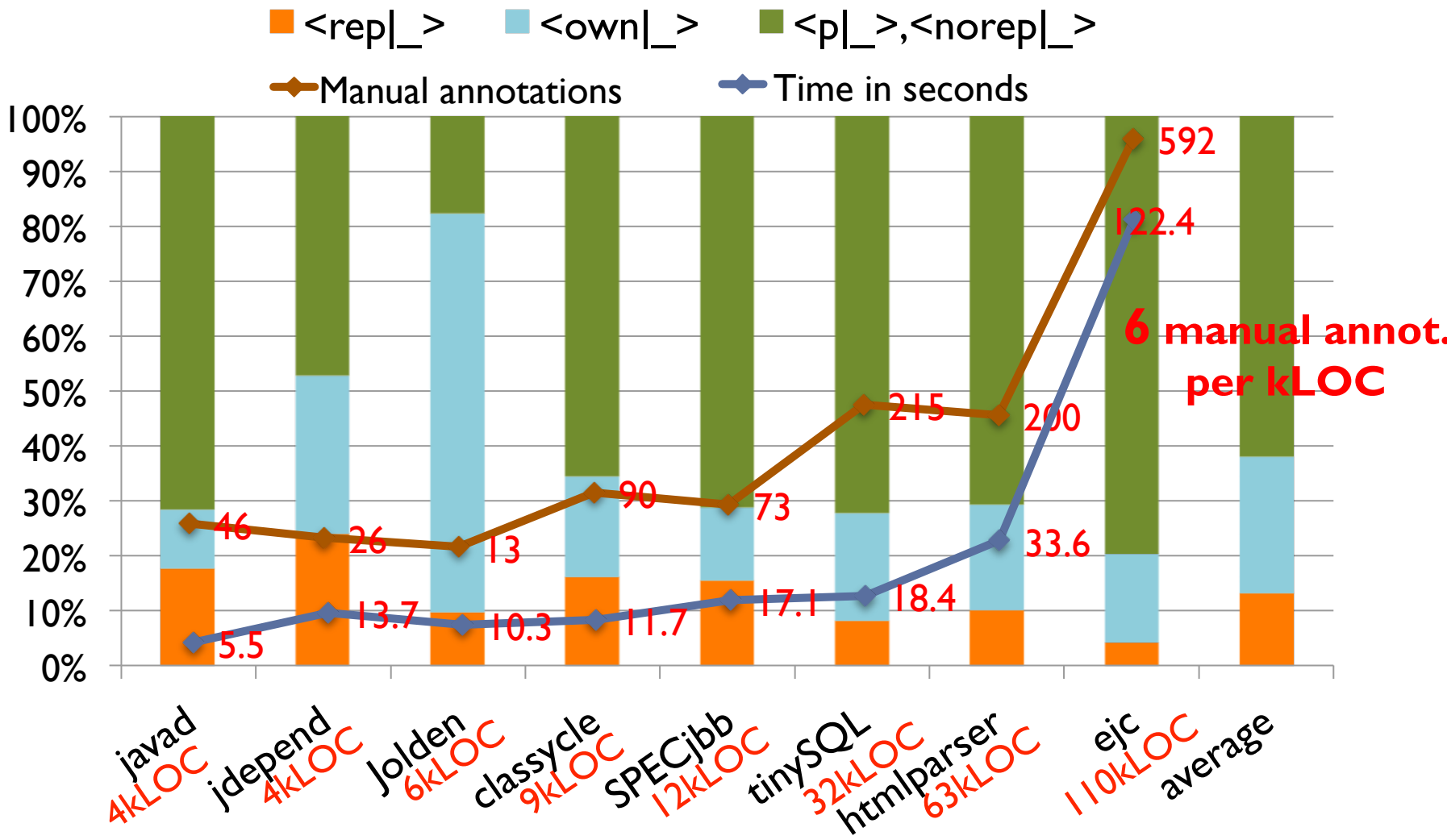
Benchmarks

Benchmark	#Line	Description
javad	4,207	Java class file disassembler
jdepend	4,351	Java package dependency analyzer
JOlden	6,223	Benchmark suite of 10 small programs
classycle	8,972	Java class and package dependency analyzer
SPECjbb	12,076	SPEC's benchmark for evaluating server side Java
tinySQL	31,980	Database engine
htmlparser	62,627	HTML parser
ejc	110,822	Java compiler of the Eclipse IDE

UT Result


any
rep
peer
Time in seconds
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
4.1
7.2
11.3
9.9
13.6
18.2
22.9
119.7
0 manual annot.
javad 4kLOC
jdepend 4kLOC
Jolden 6kLOC
classycle 9kLOC
SPECjbb 12kLOC
tinySQL 32kLOC
htmlparser 63kLOC
ejc 110kLOC
average

OT Result

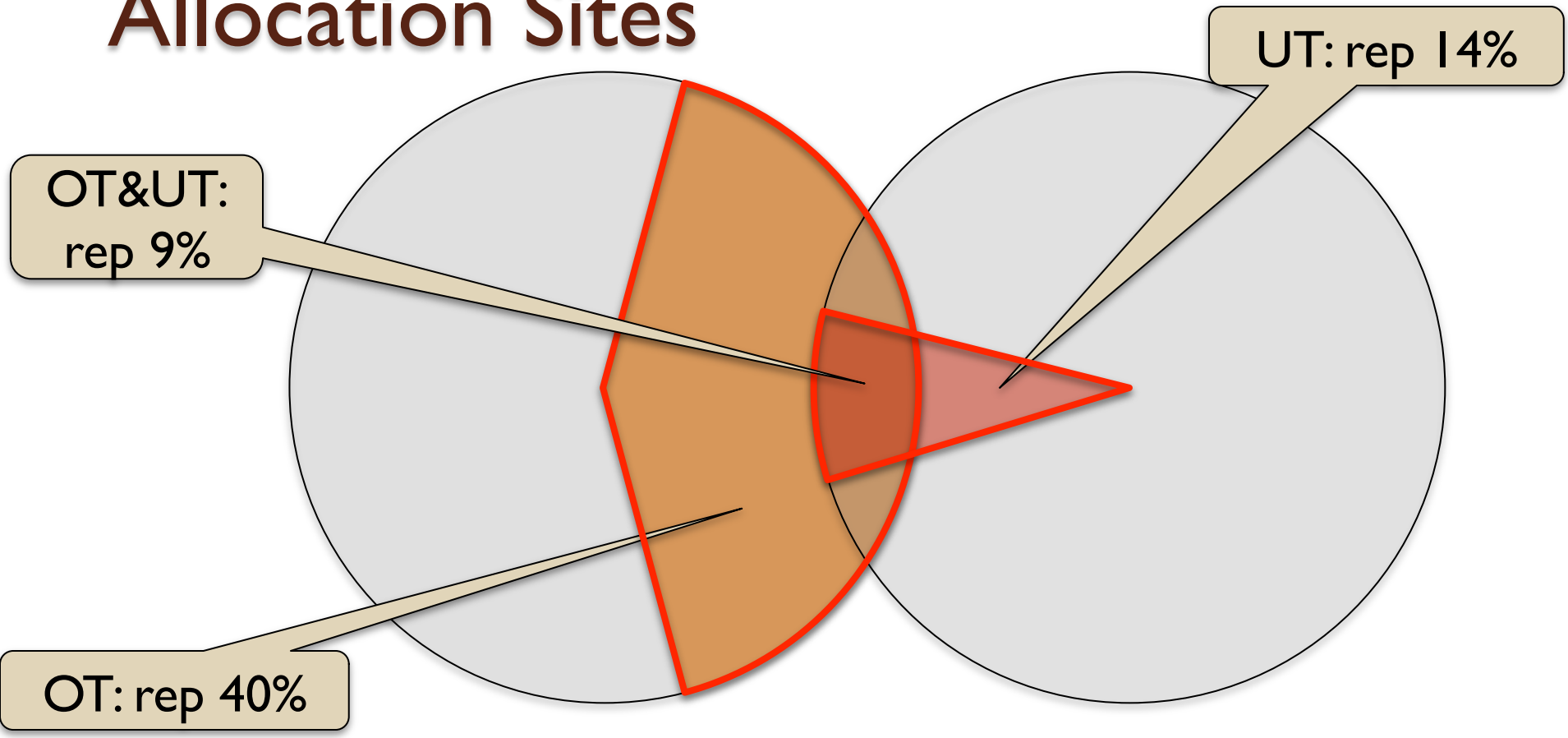

<rep|_>
<own|_>
<p|_>,<norep|_>
Manual annotations
Time in seconds
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
46
26
13
90
73
215
200
592
5.5
13.7
10.3
11.7
17.1
18.4
33.6
122.4
6 manual annot. per kLOC
javad 4kLOC
jdepend 4kLOC
JoIden 6kLOC
classycle 9kLOC
SPECjbb 12kLOC
tinySQL 32kLOC
htmlparser 63kLOC
ejc 110kLOC
average

Owner-as-Modifier vs Owner-as-Dominator

- Goal: compare UT (OaM) to OT (OaD)
- In certain cases, UT gives rise to a deeper tree than OT
- In other cases, OT gives rise to a deeper tree

- Does UT or OT has deeper trees?
- Do UT and OT give rise to different trees?

Allocation Sites



OT has deeper trees

OT and UT give rise to different ownership trees

Summary of Results

- Many objects are owned (encapsulated)
  - UT: 14% of allocation sites are rep (upper bound!)
  - OT: 40% of allocation sites are rep (close to upper bound!)

- UT requires no manual annotations
  - Programs can be refactored to have better OaM structure
- OT requires manual annotations
  - Annotations are hard to understand

Related Work

- Tip et al. (TOPLAS 2011)
  - Similar algorithm: starts with all possible answers and iteratively removes infeasible elements
  - We also use qualifier preference ranking
- Dietl et al. (ECOOP 2011)
  - Tunable Inference for Generic Universe Types
  - Encodes type constraints and solved by Max-SAT solver
- Sergey & Clark (ESOP 2012)
  - Gradual Ownership Types
  - Requires both static and dynamic analyses
  - Analyzes 8,200 lines of code in total

Conclusions

- An inference framework for ownership-like type systems
- Definition of "best" typing
- Evaluation on 241 kLOC in total
- Publicly available at
  - http://www.cs.rpi.edu/~huangw5/cf-inference

Conclusions

- An inference framework for ownership-like type systems
- Definition of "best" typing
- Evaluation on 241 kLOC in total
- Publicly available at
  - http://www.cs.rpi.edu/~huangw5/cf-inference

Typing Rule (TCALL): x = y.m(z)

UT: (TCALL)

$$\frac{typeof(\mathrm{m}) = q_p \rightarrow q_{ret} \quad \Gamma(\mathrm{x}) = q_x \quad \Gamma(\mathrm{y}) = q_y \quad typeof(\mathrm{z}) = q_z \quad q_z <: q_y \triangleright q_p \quad q_y \triangleright q_{ret} <: q_x \quad q_y \triangleright q_p \neq \mathrm{lost} \;\; impure(\mathrm{m}) \Rightarrow q_y \neq \mathrm{any}}{\Gamma \mapsto \mathrm{x} = \mathrm{y.m(z)}}$$

OT: (TWRITE)

$$\frac{typeof(\mathrm{m}) = q_p \rightarrow q_{ret} \quad \Gamma(\mathrm{x}) = q_x \quad \Gamma(\mathrm{y}) = q_y \quad typeof(\mathrm{z}) = q_z \quad q_z <: q_y \triangleright q_p \quad q_y \triangleright q_{ret} <: q_x \quad q_p \neq \langle \mathrm{rep} | \_ \rangle}{\Gamma \mapsto \mathrm{x} = \mathrm{y.m(z)}}$$

Unified: (TWRITE)

$$\frac{typeof(\mathrm{m}) = q_p \rightarrow q_{ret} \quad \Gamma(\mathrm{x}) = q_x \quad \Gamma(\mathrm{y}) = q_y \quad typeof(\mathrm{z}) = q_z \quad q_z <: q_y \triangleright q_p \quad q_y \triangleright q_{ret} <: q_x \quad \beta_{(\mathrm{TCALL})}}{\Gamma \mapsto \mathrm{x} = \mathrm{y.m(z)}}$$

Delelte

UT Result

Benchmark	TotalVar	any	rep	peer	#Manual	Time(s)
JOlden	685	227	71	387	0	11.3
tinySQL	2711	630	104	1977	0	18.2
htmlparser	3269	426	153	2690	0	22.9
ejc	10957	1897	122	8938	0	119.7
javad	249	31	11	207	0	4.1
SPECjbb	1066	295	74	697	0	13.6
jdepend	542	95	14	433	0	7.2
classycle	946	87	11	848	0	9.9

- Running times range from 4 sec. to 120 sec.
- Zero manual annotations are required

Delelte

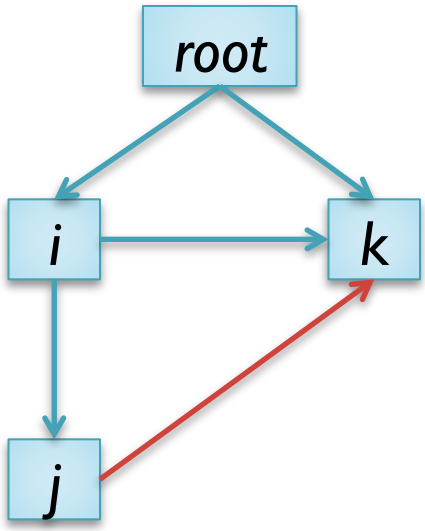
OT Result

Benchmark	TotalVar					#Manual	Time(s)
JOlden	685	$\#\langle rep\mid_{-}\rangle$ 67	$\#\langle own\mid_{-}\rangle$ 497	$\#\langle p\mid_{-}\rangle$ 24	$\#\langle norep\mid_{-}\rangle$ 97	13(2/KLOC)	10.3
tinySQL	2711	224	530	5	1952	215(7/KLOC)	18.4
htmlparser	3269	330	629	36	2274	200(3/KLOC)	33.6
ejc	10957	467	1768	50	8672	592(5/KLOC)	122.4
javad	249	44	27	74	104	46(10/KLOC)	5.5
SPECjbb	1066	166	141	71	688	73(6/KLOC)	17.1
jdepend	542	130	156	128	128	26(6/KLOC)	13.7
classycle	946	153	173	28	592	90(10/KLOC)	11.7

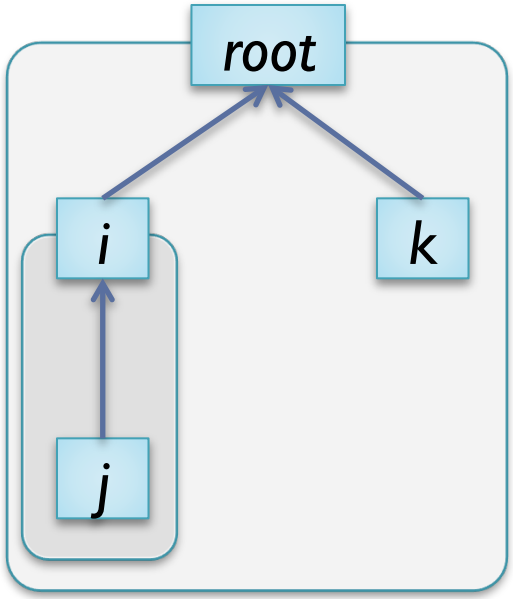
- Running times range from 4 sec. to 120 sec.
- 6/KLOC manual annotations on average

OaM vs OaD

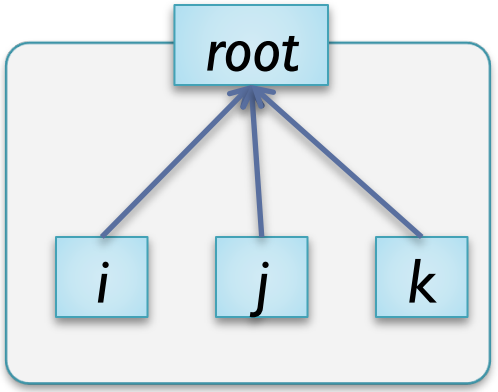
- OT gives rise to a deeper tree
- Object *j* modifies object *k* of an enclosing context



Object Graph



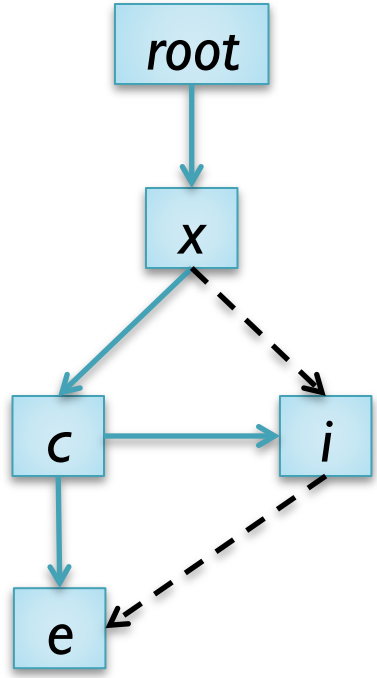
OT Tree



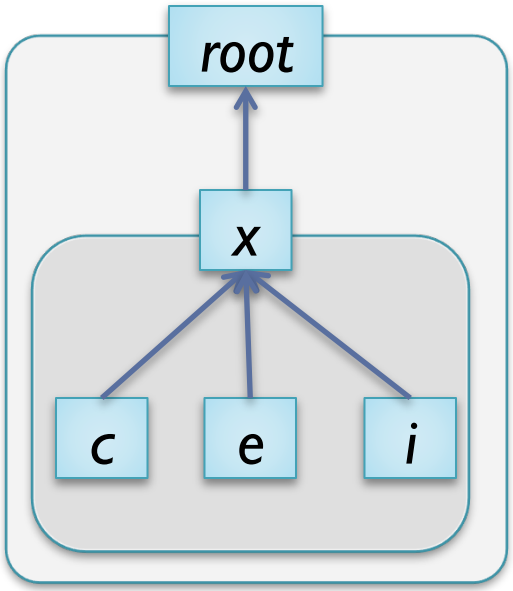
UT Tree

OaM vs OaD

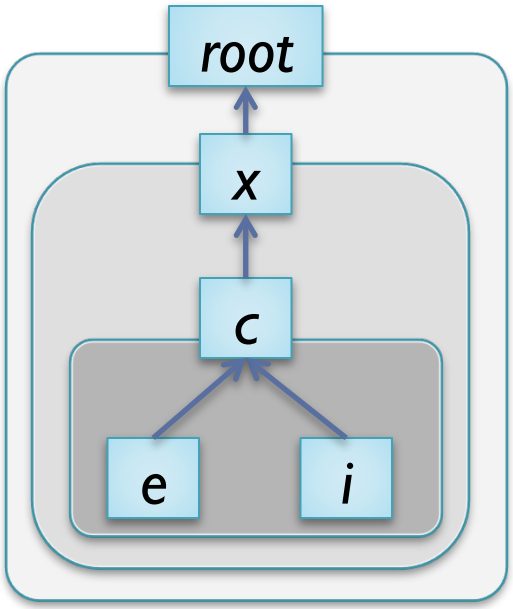
- UT gives rise to a deeper tree
- Access to object *e* from *i* is readonly



Object Graph

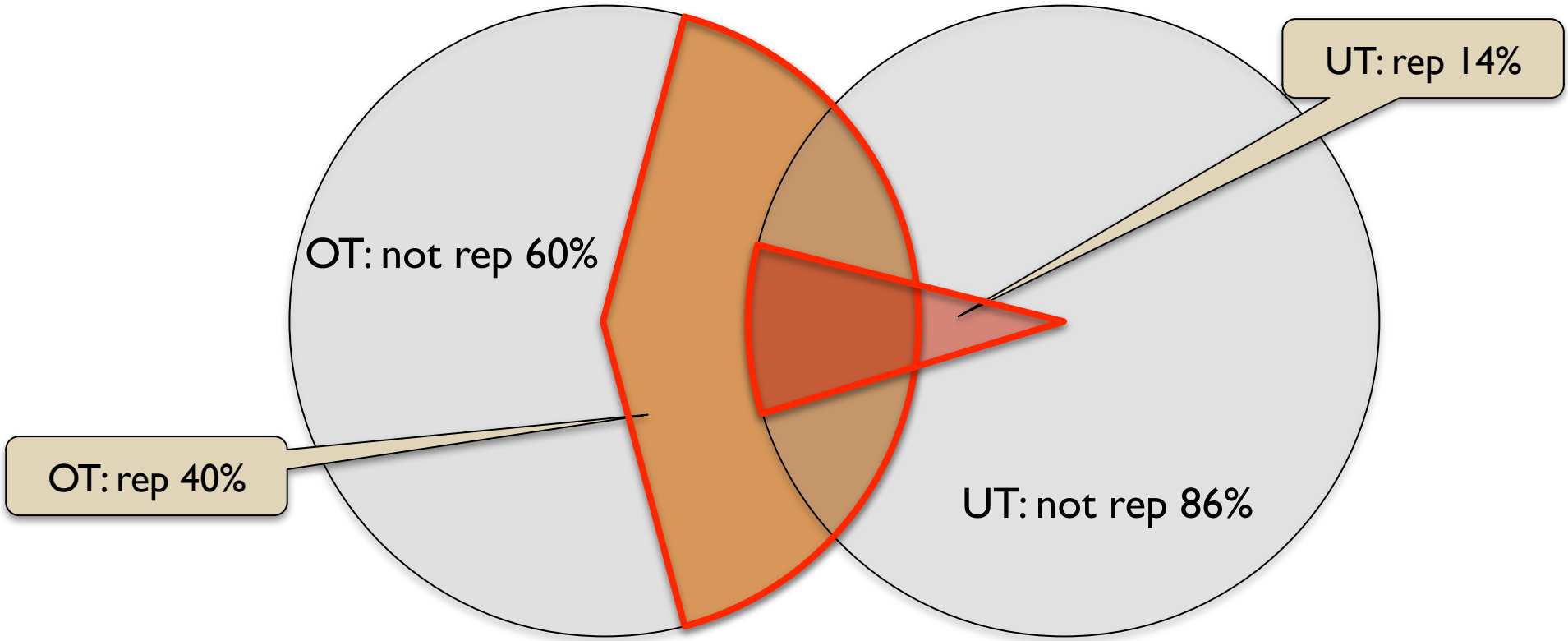


OT Tree



UT Tree

Allocation Sites in All Benchmarks



Modification of objects from enclosing context happens more often than readonly exposure