Abstractions from Tests

Mayur Naik (Georgia Institute of Technology)
Hongseok Yang (University of Oxford)
Ghila Castelnuovo (Tel-Aviv University)
Mooly Sagiv (Tel-Aviv University)
Motivation

• Great success stories in automatic program verification based on static analysis techniques (SDV, Astree, etc).

• Yet balancing precision and performance of a static analysis is still an art.

• We want to do this balancing automatically.
Typical static analysis

program P
query q

static analysis

proved
don’t know
Our approach

program $P$
query $q$

parameterised static analysis

proved

don’t know
Our approach

Program $P$ query $q$

Dynamic analysis

Parameter inference

Parameterised static analysis

Disproved

Info

Parameter

Proved

Don’t know

Wednesday, 1 February 2012
Hypothesis

- If a query is simple, we can find why the query holds simply by looking at a few execution traces.
Parameter inference based on separability and minimality

instrumented states $s, s'$

parameter inference

GOOD | BAD

$s, s'$

parameter $\eta$

$\eta_0 \quad \eta \quad \eta_1$
Parameter inference based on separability and minimality

instrumented states $s, s'$ → parameter inference → parameter $\eta$

GOOD | BAD
--- | ---
$s, s'$

Can separate?

$\eta_0 \quad \eta \quad \eta_1$
Parameter inference based on separability and minimality

instrumented states $s, s'$

parameter inference

Can separate?

GOOD | BAD

$\eta_0$ | $\eta$ | $\eta_1$
Parameter inference based on separability and minimality

instrumented states $s, s'$ → parameter inference → parameter $\eta$

GOOD | BAD

Can separate?

$\eta_0 \quad \eta \quad \eta_1$
Parameter inference based on separability and minimality

- Computes a separability condition.
Parameter inference based on separability and minimality

- Computes a separability condition.
- Among separable \( \eta_i \)'s, choose a minimal \( \eta \) according to an order (which approximately reflects precision).
Thread-escape query

• Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
   x0 = new h0;
   x1 = new h1; x1.f1 = x0;
   x2 = new h2; x2.f2 = x1;
   x3 = new h3; x3.f3 = x2;
   x0.start();
   pc: x2.id = i; //local(x2)?
   x3.start();
}
```
Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; // local(x2)?
    x3.start();
}
```
Thread-escape query

• Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape query

• Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape query

• Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape query

• Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; // local(x2)?
    x3.start();
}
```
Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; // local(x2)?
    x3.start();
}
```
Thread-escape query

• Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```

By now, people should understand what thread-escape query means, and how this program works.

People might wonder why one should care about thread-escape queries. Answers for them help race detectors or other verifiers for concurrent programs.

Difficulty. Shape analysis. Difficult to scale. Also, note that the allocation-site abstraction doesn’t work for this example.
Thread-escape analysis

• Summarise all heap objects with only two abstract nodes E and L.

• $\varepsilon(E)$ consists of all the thread-escaping objects and possibly more.

• $\varepsilon(L)$ contains only thread-local objects.
Parameterisation

\[ \text{Param} = \text{AllocSite} \rightarrow \{ L, E \} \]

- For each allocation site, it decides whether L or E is used to summarise allocated objects.
- Changes the transfer function of “\( x = \text{new } h_i \)”.
- Objects summarised by L can move to E, but not vice versa.
Thread-escape analysis

- Parameter $\eta = [{h0,h1} \mapsto E, {h2,h3} \mapsto L]$

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape analysis

- Parameter $\eta = \{h0,h1\} \mapsto E, \{h2,h3\} \mapsto L$

```java
for (i = 0; i < n; i++) {
    x0 = new h0/E;
    x1 = new h1/E; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; // local(x2)?
    x3.start();
}
```
Thread-escape analysis

- Parameter $\eta = [\{h_0, h_1\} \mapsto E, \{h_2, h_3\} \mapsto L]$

```java
for (i = 0; i < n; i++) {
    x0 = new h0/E;
    x1 = new h1/E; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape analysis

• Parameter $\eta = [{h0,h1}\mapsto E, {h2,h3}\mapsto L]$

```java
for (i = 0; i < n; i++) {
    x0 = new h0/E;
    x1 = new h1/E; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape analysis

- Parameter $\eta = \{h0, h1\} \mapsto E, \{h2, h3\} \mapsto L$

```java
for (i = 0; i < n; i++) {
    x0 = new h0/E;
    x1 = new h1/E; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape analysis

- Parameter $\eta = \{h0,h1\} \mapsto E, \{h2,h3\} \mapsto L$

```java
for (i = 0; i < n; i++) {
    x0 = new h0/E;
    x1 = new h1/E; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape analysis

- Parameter $\eta = [{h0,h1} \mapsto E, {h2,h3} \mapsto L]$

```java
for (i = 0; i < n; i++) {
    x0 = new h0/E;
    x1 = new h1/E; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; // local(x2)?
    x3.start();
}
```
Thread-escape analysis

- Parameter $\eta = \{h_0, h_1\} \mapsto E, \{h_2, h_3\} \mapsto L$

for (i = 0; i < n; i++) {
    x0 = new h0/E;
    x1 = new h1/E; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: $x2.id = i$; //local(x2)?
    x3.start();
}
Thread-escape analysis

- Parameter \( \eta = \{h_0, h_1\} \rightarrow E, \{h_2, h_3\} \rightarrow L \)

```java
for (i = 0; i < n; i++) {
    x0 = new h0/E;
    x1 = new h1/E; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Thread-escape analysis

- Parameter $\eta = \{h_0, h_1\} \mapsto E, \{h_2, h_3\} \mapsto L$

```java
for (i = 0; i < n; i++) {
    x0 = new h0/E;
    x1 = new h1/E; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Difficulties in choosing a good parameter

• Using more L makes the analysis more expensive.
Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.
Difficulties in choosing a good parameter

• Using more L makes the analysis more expensive.
• More L doesn’t always mean more precision.
Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.
- More L doesn’t always mean more precision.

```java
for (i = 0; i < n; i++) {
    x0 = new h0;
    x1 = new h1; x1.f1 = x0;
    x2 = new h2; x2.f2 = x1;
    x3 = new h3; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.
- More L doesn’t always mean more precision.

```java
for (i = 0; i < n; i++) {
    x0 = new h0/L;
    x1 = new h1/L;  x1.f1 = x0;
    x2 = new h2/L;  x2.f2 = x1;
    x3 = new h3/L;  x3.f3 = x2;
    x0.start();
    pc: x2.id = i;  //local(x2)?
    x3.start();
}
```
Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.
- More L doesn’t always mean more precision.

```java
for (i = 0; i < n; i++) {
    x0 = new h0/L;
    x1 = new h1/L; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Difficulties in choosing a good parameter

• Using more L makes the analysis more expensive.

• More L doesn’t always mean more precision.

```java
for (i = 0; i < n; i++) {
    x0 = new h0/L;
    x1 = new h1/L; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)?
    x3.start();
}
```
Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.
- More L doesn’t always mean more precision.

```java
for (i = 0; i < n; i++) {
    x0 = new h0/L;
    x1 = new h1/L; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    x3 = new h3/L; x3.f3 = x2;
    x0.start();
    pc: x2.id = i; //local(x2)!
    x3.start();
}
```
Separability question

<table>
<thead>
<tr>
<th>local(x2)</th>
<th>¬local(x2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>s, s'</td>
</tr>
</tbody>
</table>

- Does analysis(η) have an abstract element d separating \{s, s'\} from ¬local(x2)?

- We use a generic answer to this question during our parameter inference.
Separability from $\neg\text{local}(x_2)$

- This state satisfies $\text{local}(x_2)$. 
Separability from \( \lnot \text{local}(x_2) \)

- This state satisfies \( \text{local}(x_2) \).

- Separated from \( \lnot \text{local}(x_2) \) by analysis(\( \eta \)) iff
  \( (\eta \circ \text{allocSite} \circ \text{backReach})(x_2) = \{L\} \).
1. Testing gives states where \textit{local}(x2) holds.

\begin{itemize}
    \item Compute the alloc. sites \(H\) of objects backward-reachable from \(x2\).
    \item \(\eta(h) = \text{L, if } h \text{ is in } H; \eta(h) = \text{E, otherwise.}\)
    \item Return \(\eta\).
\end{itemize}
1. Testing gives states where local(x2) holds.

2. Compute the alloc. sites H of objects that can reach x2.

\[ H = \{h2, h3\} \]
Parameter inference

1. Testing gives states where local(x2) holds.
2. Compute the alloc. sites $H$ of objects that can reach x2.
3. $\eta(h) = L$, if $h$ is in $H$; $\eta(h) = E$, otherwise.

$H = \{h2, h3\}$

$\eta = [\{h0, h1\} \rightarrow E, \{h2, h3\} \rightarrow L]$
Parameter inference

1. Testing gives states where local(x2) holds.
2. Compute the alloc. sites H of objects that can reach x2.
3. \( \eta(h) = L \), if h is in H; \( \eta(h) = E \), otherwise.

\[ H = \{h2, h3\} \]
\[ \eta = [{h0, h1} \rightarrow E, \{h2, h3\} \rightarrow L] \]
1. Testing gives states where \( \text{local}(x_2) \) holds.

2. Compute the alloc. sites \( H \) of objects that can reach \( x_2 \).

3. \( \eta(h) = L \), if \( h \) is in \( H \); \( \eta(h) = E \), otherwise.

\[ H = \{h_2, h_3\} \]
\[ \eta = \{\{h_0, h_1\} \mapsto E, \{h_2, h_3\} \mapsto L\} \]
Does it work?
Setting of experiments

- 6 concurrent Java programs from Dacapo:
  - 161K - 491K bytecode (including analysed JDK).
  - Up to 5K allocation sites per program.
- 47K queries, but only 17K (37%) reached during testing.
- Considered only these reachable queries.
6 Java prog. (161K-491K) up to 5K sites
17K queries

dynamic analysis ➔ info ➔ parameter inference ➔ parameterised static analysis

disproved ➔ proved ➔ don’t know

info
6 Java prog. (161K-491K) up to 5K sites
17K queries

dynamic analysis

28% disproved

info

parameter inference

parameter

parameterised static analysis

52% proved

20% don’t know
6 Java prog. (161K-491K) up to 5K sites
17K queries

Dynamic analysis

Parameter inference

Parameterised static analysis

Per program: 38s - 86m

Info

28% disproved

Parameter

52% proved

20% don’t know

Per prog:
6s - 8m
6 Java progs. (161K-491K) up to 5K sites
17K queries

Dynamic analysis

Parameter inference

L-mapped sites:
avg 4.8, max 195

Parameterised static analysis

Per prog:
6s - 8m

Per program:
38s - 86m

28% disproved
52% proved
20% don’t know

Info per program:
38s - 86m
6 Java prog. (161K-491K) up to 5K sites
17K queries

All sites mapped to L

parameterised static analysis

proved
don’t know
6 Java prog. (161K-491K) up to 5K sites
17K queries

Out of memory for all programs

All sites mapped to L

parameterised static analysis

proved
don’t know