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

Parameterised

Proved
don’t know
Our approach

program P
query q
dynamic analysis
info
parameter inference
parameter
disproved
parameterised static analysis
proved
don’t know
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

Parameter $\eta$

GOOD

BAD

$s, s'$

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

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

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

Can separate? $\eta_0 \eta \eta_1$
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

instrumented states $s, s'$  \rightarrow \text{parameter inference}  \rightarrow \text{parameter } \eta

GOOD | BAD

Can separate?

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

- Computes a separability condition.

$\eta$
Parameter inference based on separability and minimality

instrumented states \( s, s' \) → parameter inference → parameter \( \eta \)

GOOD | BAD
---|---
\( s, s' \) | 

Can separate? YES \( \eta_0 \) NO \( \eta_1 \)

- 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();
}
```
Thread-escape analysis

- Summarise all heap objects with only two abstract nodes E and L.
- $\varphi(E)$ consists of all the thread-escaping objects and possibly more.
- $\varphi(L)$ contains only thread-local objects.
Parameterisation

Param = 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=new hi”.
• 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 $\eta = [{h_0, h_1} \mapsto E, {h_2, h_3} \mapsto L]$

```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 \( \eta = [\{h_0, h_1\} \mapsto E, \{h_2, h_3\} \mapsto L] \)

```c
for (i = 0; i < n; i++) {
    x0 = new h0/E;
    x1 = new h1/E; x1.f1 = x0;
    x2 = new h2/L; x2.f2 = x1;
    \textcolor{red}{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 \( \eta = [{h_0,h_1} \mapsto E, {h_2,h_3} \mapsto L] \)

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

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

```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();
    //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

- Does analysis($\eta$) have an abstract element $d$ separating $\{s, s'\}$ from $\neg$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 $\neg\text{local}(x_2)$

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

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

2. Compute the alloc. sites $H$ of objects that can reach $x_2$.

$$H = \{h_2, h_3\}$$
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.

\[ H = \{h2, h3\} \]
\[ \eta = \{\{h0, h1\} \mapsto E, \{h2, h3\} \mapsto L\} \]
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.

\[
H = \{h2, h3\}
\]
\[
\eta = [\{h0, h1\} \mapsto E, \{h2, h3\} \mapsto L]
\]
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.

\[ H = \{h2, h3\} \]
\[ \eta = [\{h0, h1\}\rightarrow E, \{h2, h3\}\rightarrow L] \]
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

Parameter inference

Parameterised static analysis

Disproved

Info

Proved

Don't know

Monday, 27 February 2012
6 Java prog. (161K-491K) up to 5K sites
17K queries

dynamic analysis → info → parameter inference

28% disproved

parameter → parameterised static analysis

52% proved
20% don’t know
6 Java prog. (161K-491K) up to 5K sites
17K queries

per prog: 6s - 8m

28% disproved

dynamic analysis

info

parameter inference

52% proved

per program: 38s - 86m

parameterised static analysis

20% don’t know

Monday, 27 February 2012
6 Java prog. (161K-491K) up to 5K sites
17K queries

per prog: 6s - 8m

dynamic analysis

28% disproved

info

parameter inference

L-mapped sites: 
avg 4.8, max 195

parameter

per program: 38s - 86m

parameterised static analysis

52% proved

20% don’t know

don’t know

Monday, 27 February 2012
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

All sites mapped to L

parameterised static analysis

proved
don't know

Out of memory for all programs

Monday, 27 February 2012