Demand-Driven Context-Sensitive Alias Analysis for Java

Dacong (Tony) Yan
Guoqing (Harry) Xu
Atanas Rountev
Ohio State University

PRESTO: Program Analyses and Software Tools Research Group, Ohio State University
Alias Analysis

- Many static analysis tools need highly precise and efficiently-computed alias information

- Desirable properties
  - Demand-driven: query “could x and y alias?”
  - Client-driven: client-defined time budget for a query

- Typical solution: use points-to analysis
  - Compute the objects that x may point to; same for y; are there common objects?

- Goal: answer alias queries precisely and efficiently, without computing the complete points-to sets
Redundancy in Points-to Analysis
Redundancy in Points-to Analysis

```c
m(a) {
    b = a;
    x = a.f;
    y = b.f;
}
```
Redundancy in Points-to Analysis

```c
m(a) {
    b = a;
    x = a.f;
    y = b.f;
}
```

Is `x` and `y` aliased?
Redundancy in Points-to Analysis

```c
m(a) {
    b = a;
    x = a.f;
    y = b.f;
}
```

Diagram:
- `x` points-to `O`
- `y` points-to `O`
- `x` alias `y`
Redundancy in Points-to Analysis

```c
m(a) {
    b = a;
    x = a.f;
    y = b.f;
}
```

```
x ←----- a
```
```
b ------ f y
```
Redundancy in Points-to Analysis

```c
m(a) {
    b = a;
    x = a.f;
    y = b.f;
}
```

Diagram:
```
x ← f a ← --- alias? b → f y
```
Redundancy in Points-to Analysis

\[ m(a) \{ \]
\[ \quad b = a; \]
\[ \quad x = a.f; \]
\[ \quad y = b.f; \]
\[ \} \]
Redundancy in Points-to Analysis

\( m(a) \) {
    b = a;
    x = a.f;
    y = b.f;
}

\[ x \leftarrow f \quad a \]
\[ \text{alias} \]
\[ b \rightarrow f \rightarrow y \]
Redundancy in Points-to Analysis

```c
m(a) {
    b = a;
    x = a.f;
    y = b.f;
}
```

![Diagram showing aliasing relationships](image)
Our Approach

• Alias analysis
  – Demand-driven and client-driven
  – Field-sensitive and calling-context-sensitive
  – Does not require complete points-to set computation
  – Better performance through method summaries

• Symbolic Points-to Graph
  – A specialized program representation that enables the demand-driven alias analysis
  – Facilitates computation and use of method summaries
Program Representation

• Intraprocedural Symbolic Points-To Graph
  – Introduce a *symbolic object node* \( s \) for each
    • formal parameter
    • field read \( a.fld \)
  – a call site that returns a reference-typed value
  – Compute intraprocedural points-to relationships

```java
m(a) {
    c = new ...;  // o_1
    a.f = c;
    return c.g;
}
```
Interprocedural Symbolic Points-To Graph

• Connect the intraprocedural graphs using entry and exit edges
Interprocedural Symbolic Points-To Graph

- Connect the intraprocedural graphs using entry and exit edges

```java
m(a) {
    c = new …; // o₁
    a.f = c;
    return c.g;
}
```
Interprocedural Symbolic Points-To Graph

• Connect the intraprocedural graphs using entry and exit edges

```java
m(a) {
    c = new ...; // o₁
    a.f = c;
    return c.g;
}
```

Diagram:

```
a → s_a → f → o₁ → c
    \^        \^  /  \\
    g         \g /   \\
    ret      /    S_g
```

6
Interprocedural Symbolic Points-To Graph

- Connect the intraprocedural graphs using entry and exit edges

```java
m(a) {
    c = new ...; // o₁
    a.f = c;
    return c.g;
}

d = new ...; // o₂
b = m(d); // call m
```
Interprocedural Symbolic Points-To Graph

• Connect the intraprocedural graphs using entry and exit edges

```java
m(a) {
    c = new …; // o1
    a.f = c;
    return c.g;
}

d = new …; // o2
b = m(d); // call m
```
Interprocedural Symbolic Points-To Graph

• Connect the intraprocedural graphs using entry and exit edges

```java
m(a) {
    c = new ...; // o₁
    a.f = c;
    return c.g;
}

d = new ...; // o₂
b = m(d); // call m
```
Alias Analysis Formulation

- Field-sensitivity and calling-context-sensitivity: matched **points-to edges** and **entry/exit edges**
- Traverses the object nodes (including symbolic ones) and the edges between them
Alias Analysis Formulation

• Field-sensitivity and calling-context-sensitivity: matched points-to edges and entry/exit edges

• Traverses the object nodes (including symbolic ones) and the edges between them
Alias Analysis Formulation

- Field-sensitivity and calling-context-sensitivity: matched points-to edges and entry/exit edges
- Traverses the object nodes (including symbolic ones) and the edges between them

\[
\begin{align*}
\text{a} & \rightarrow \text{o}_1 \\
\text{b} & \rightarrow \text{o}_2 \\
\text{entry}_1 & \rightarrow \text{s}_1 \\
\text{exit}_1 & \rightarrow \text{s}_1 \\
\text{f} & \\
\text{c} & \rightarrow \text{s}_3 \\
\text{d} & \rightarrow \text{s}_2 \\
\text{f} & \\
\text{a} & \text{ alias? c}
\end{align*}
\]
Alias Analysis Formulation

- Field-sensitivity and calling-context-sensitivity: matched points-to edges and entry/exit edges
- Traverses the object nodes (including symbolic ones) and the edges between them

**may represent the same concrete object?**

![Diagram with nodes and edges labeled a, o₁, o₂, s₁, s₂, s₃, c, f, d, entry₁, exit₁, b, o₁, s₃]
Alias Analysis Formulation

- Field-sensitivity and calling-context-sensitivity: matched points-to edges and entry/exit edges
- Traverses the object nodes (including symbolic ones) and the edges between them
Alias Analysis Formulation

• Field-sensitivity and calling-context-sensitivity: matched points-to edges and entry/exit edges
• Traverses the object nodes (including symbolic ones) and the edges between them

![Diagram showing traversal of object nodes and edges]
Alias Analysis Formulation

- Field-sensitivity and calling-context-sensitivity: matched points-to edges and entry/exit edges
- Traverses the object nodes (including symbolic ones) and the edges between them

\[a \rightarrow o_1 \quad o_1 \rightarrow o_2 \quad b \rightarrow o_2 \quad s_3 \leftarrow c \quad s_2 \leftarrow d\]

\[\text{entry}_1 \rightarrow s_1 \quad \text{exit}_1\]

\[f \rightarrow o_1 \quad o_1 \rightarrow o_2 \quad \text{entry}_1 \rightarrow s_1 \quad \text{exit}_1 \rightarrow s_2 \quad s_3\]
Alias Analysis Formulation

- Field-sensitivity and calling-context-sensitivity: matched points-to edges and entry/exit edges
- Traverses the object nodes (including symbolic ones) and the edges between them
Alias Analysis Formulation

- Field-sensitivity and calling-context-sensitivity: matched points-to edges and entry/exit edges
- Traverses the object nodes (including symbolic ones) and the edges between them

![Diagram](attachment:image.png)
Alias Analysis Formulation

- Field-sensitivity and calling-context-sensitivity: matched points-to edges and entry/exit edges
- Traverses the object nodes (including symbolic ones) and the edges between them

![Diagram of alias analysis formulation]

- Nodes and edges represent field-sensitivity and calling-context-sensitivity.
- Points-to edges match fields to objects.
- Entry/exit edges indicate entry and exit points of functions.
- The diagram illustrates the traversal of object nodes and edges in the alias analysis.
Using Method Summaries

• Several reachability strings for a selected method
  – Each string is a sequence of field points-to edges between boundary objects of the method
Using Method Summaries

• Several reachability strings for a selected method
  – Each string is a sequence of field points-to edges between boundary objects of the method

```java
m(a) {  
c = new ...; // o1  
a.f = c;  
return c.g;
}
```

Diagram:
```
a -> sa -> o1 -> c  
g: f  
ret -> sg
```
Using Method Summaries

• Several reachability strings for a selected method
  – Each string is a sequence of field points-to edges between boundary objects of the method

```
summary(m):
  m(a) {
    c = new ...; // o₁
    a.f = c;
    return c.g;
  }
```

```
  a → sa → f  \ o₁ → c
    ↑            ↑
  ret → sg
```

```
f, g
    ────→
      sa    sg
```
Using Method Summaries

• Several reachability strings for a selected method
  – Each string is a sequence of field points-to edges between boundary objects of the method

• Selecting methods for summarization
  – Compute a summary for a method only if it is invoked from many different call sites
Using Method Summaries

• Several reachability strings for a selected method
  – Each string is a sequence of field points-to edges
    between boundary objects of the method

• Selecting methods for summarization
  – Compute a summary for a method only if it is invoked
    from many different call sites
  – Summarization ratio for method \( m \)
    • the number of incoming call graph edges of \( m \),
      divided by the average number of incoming call
      graph edges for all methods
Experimental Evaluation

• 19 Java programs
  – Number of methods in the whole-program call graph ranging from 2344 to 8789

• **Experiment 1**: compare the precision with a field and context-sensitive, demand-driven, client-driven points-to analysis [PLDI’06]
  – Under the same time budget per alias query
  – Result 1: for 14 programs, the number of alias pairs is lower when using our analysis
  – Result 2: summarization leads to better precision

• **Experiment 2**: compare running times with and without method summaries
Running Time Reduction When Using Method Summaries

Running Time Reduction = \( \frac{RT_{no-summ} - RT_{summ}}{RT_{no-summ}} \)
Running Time Reduction When Using Method Summaries

Running Time Reduction = \( \frac{R_{\text{no-summ}} - R_{\text{summ}}}{R_{\text{no-summ}}} \)

24% average reduction with threshold T=2
Conclusions

• A demand-driven alias analysis
  – Answers alias queries directly, without computing the complete points-to sets
  – Selects methods for online summarization to reduce analysis running time
  – Outperforms a highly precise state-of-the-art points-to analysis
Thank you