Pointer Analysis:
Haven't We Solved This Problem Yet?

Michael Hind
IBM Watson Research Center
Pointer Analysis Pubs by Year

83 Publications in 14 years!  48 in the last 4 years!
Why should I care?

- needed for any "mod/ref" analysis
  - slicing, dep graph, constant prop, code motion, ...
  - call graph construction
    - needed for any whole program analysis

```c
p->data =
    = q->data;

x = 0;
*p=17;
    = x + ...

for (. . .) {
    ...
    p->data = 0;
}

(*p)(a, b, c);
q->foo();
```
OK, I need a pointer analysis, which one should I use?

- It depends ...
- Do you want
  - high precision?
  - high efficiency?
  - not a simple question
- Sit back and relax for the next 45 mins
Talk Roadmap

- **Ptr Analysis Dimensions**
- **Metrics**
- **Survey of Issues**
- **Conclusions**

**Feature:**

input from several ptr analysis experts
Pointer Analysis

**Goal:** statically determine what can be accessed by a pointer

**Bad news:** problem is undecidable

**Good news:** many approximation algorithms exist!
**Pointer Analysis**

*Goal:* statically determine what can be accessed by a pointer

*Bad news:* problem is undecidable

*Bad news:* many approximation algorithms exist!

*Worst case complexities:*

  - linear
  - ... doubly exponential

  *Is "Big-O" the same as "Big Ben"?*
Pointer Analysis Dimensions

- Flow sensitivity
- Context sensitivity
- Heap modeling
- Aggregate modeling
- Alias representation
- Whole program
Pointer Analysis Dimensions

- Heap modeling
  - allocation site
  - connection analysis
  - shape analysis
Pointer Analysis Dimensions

- Heap modeling
- Aggregate modeling

arrays

or

structs/objects

or
**Pointer Analysis Dimensions**

- Heap modeling
- Aggregate modeling

- Alias representation
  - points-to relations vs explicit alias representations

```
points-to          explicit alias rep
<a, b>             <*a,b>, <**a, c>
<b, c>             <*b, c>, <**a, *b>
```

Precision/efficiency tradeoffs exist [HBCC99,RLSZA01], but have not been studied!
Pointer Analysis Dimensions

- Heap modeling
- Aggregate modeling
- Alias representation

- Requires whole program?
(Pointer Analysis Dimensions

- Heap modeling
- Aggregate modeling
- Alias representation
- Requires whole program?

- Flow-sensitivity)
Pointer Analysis Dimensions

- Heap modeling
- Aggregate modeling
- Alias representation
- Requires whole program?

- Flow-sensitivity
  - considers control flow during the analysis
Pointer Analysis Dimensions

- Heap modeling
- Aggregate modeling
- Alias representation
- Requires whole program?

Flow-sensitivity

- considers control flow during the analysis
- Flow-sensitive
  - one solution/program point
  - more precise, less efficient (time and space)
Pointer Analysis Dimensions

- Heap modeling
- Aggregate modeling
- Alias representation
- Requires whole program?

- **Flow-sensitivity**
  - considers control flow during the analysis
  - Flow-sensitive
    - one solution/program point
    - more precise, less efficient (time and space)
  - Flow-insensitive
    - one solution/whole program or function
    - less precise, more efficient
    - equality-based (almost linear)
    - subset-based (polynomial)
Example

1: p = malloc();
2: q = malloc();
3: fp = &p;
4: fp = &q;
5: p = malloc();
6: ... = *p;

Points-to Relations at 6

Flow-sensitive analysis

```
fp
   p → heap5
     q → heap2
```
Example

1: p = malloc();
2: q = malloc();
3: fp = &p;
4: fp = &q;
5: p = malloc();
6: ... = *p;

Points-to Relations (at 6)

Subset-based flow-insensitive

\[
\begin{aligned}
\text{fp} & \rightarrow p & \rightarrow \text{heap1} \\
\text{fp} & \rightarrow q & \rightarrow \text{heap2} \\
p & \rightarrow \text{heap5} \\
q & \rightarrow \text{heap2}
\end{aligned}
\]
Example

1: p = malloc();
2: q = malloc();
3: fp = &p;
4: fp = &q;
5: p = malloc();
6: ... = *p;

Points-to Relations (at 6)

Equality-based flow-insensitive

fp

heap1

heap2

p

heap5

q
Example

1: p = malloc();
2: q = malloc();
3: fp = &p;
4: fp = &q;
5: p = malloc();
6: ... = *p;

Points-to Relations (at 6)

Equality-based flow-insensitive

```
fp -> p
  ^
  |
  v
q
```

```
heap1
heap2
heap5
```
Example

1: p = malloc();
2: q = malloc();
3: fp = &p;
4: fp = &q;
5: p = malloc();
6: ... = *p;

Aliases of *p at 6:

Flow-sensitive: heap5
FI subset: heap5 heap1
FI equality: heap5, heap1, heap2
Pointer Analysis Dimensions

- Heap modeling
- Aggregate modeling
- Alias representation
- Requires whole program?
- Flow sensitivity
- Context sensitivity

Is calling context considered when processing a method?

```plaintext
main() {
    1: f();
    2: p = malloc();
    3: g();
}

f() {
    4: p = malloc();
    5: g();
}

g() {
    ...}

p ← heap4
p ← heap2
p ← heap4
heap1
```
Talk Roadmap

- Ptr Analysis Dimensions
- Metrics
- Survey of Issues
- Conclusions
Metrics

Direct method: avg num objects at ptr deref

- Most popular
- Advantages
  - easy to understand
- Disadvantages
  - no inherent meaning
  - dependence on heap/recursive local model
  - client analyses
Metrics

- Direct method

- Pct of worst-case
  - not popular
  - incorporates language semantics
Metrics

- Direct method
- Worst-case

- Client impact
  - Adv: can see impact on client
  - Dis: only reports on one client
Metrics

- Direct method
- Worst-case
- Client impact

- Dynamic metric
  - direct method
  - client impact
  - Adv: gives lower bound
  - Dis: limited to one run, is lower bound tight?
Metrics

- Direct method
- Worst-case
- Client impact
- Dynamic metric

Recommendation: use combinations [DMM97]
Reproducible Results

- Given dimensions, many experiments are possible
- Often not performed, less often repeated
- Will it be published?
- Can be difficult because of
  - different intermediate representations
  - benchmark suites
  - benchmark versions
- Sharing infrastructure, benchmarks is crucial
- Isn't this at the heart of being a "science"?
Precision/Scalability

- Equality-based can analyze 1 MLOC
  - getting more precise [LH99, D00]
- Subset-based more precise, but haven't scaled well
  - but, getting more efficient!
    [FFSA98, SFA00, RC00, FRD00, RF01, HT01]
- Convergence may provide the answer, but ... is subset-based precision sufficient for all clients?
- More precise/expensive ptr analysis can make clients more efficient [SH97, HP00]
Efficiency (Time)

[HP00]

**Pointer Analysis Only**

**Ptr + All Client Analyses**

![Bar chart showing efficiency (time)](chart.png)

- **Address Taken**: 1.0
- **FI Subset**: 0.9
- **FI Equality**: 29.6
- **Flow Sensitive**: 79.49

- **AT**: 0.01
- **Equality**: 0.81
- **Subset**: 0.83
- **FS**: 0.40

**Graph legend**:
- **Address Taken**
- **FI Subset**
- **FI Equality**
- **Flow Sensitive**

**Bar chart**:
- **Clients**
- **Ptr Analysis**
Efficiency (Memory)

Pointer Analysis Only

<table>
<thead>
<tr>
<th></th>
<th>Address Taken</th>
<th>FI Subset</th>
<th>FI Equality</th>
<th>Flow Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg increase over Address Taken</td>
<td>1</td>
<td>8.52</td>
<td>1.15</td>
<td>12.19</td>
</tr>
</tbody>
</table>

Ptr + All Client Analyses

<table>
<thead>
<tr>
<th></th>
<th>Clients</th>
<th>Ptr Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>0.98</td>
<td>0.19</td>
</tr>
<tr>
<td>Equality</td>
<td>0.8</td>
<td>0.05</td>
</tr>
<tr>
<td>Subset</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td>FS</td>
<td>0.69</td>
<td></td>
</tr>
</tbody>
</table>
``It is easy to make a pointer analysis that is very fast and scales to large programs. But are the results worth anything? While more people have done work in the area, we still need a better understanding of what pointer analysis one should use.''

Amer Diwan
Precision/Scalability

Bill Landi:

- relaxing safety
- Flow and context-sensitive analysis
  - days to minutes
  - false positives/negatives are a problem, maybe?
- users: false positives $\Rightarrow$ poorly written code

Susan Horwitz:

- determine part of program (code region, ptr variable, etc.) that needs high accuracy
- find special cases where analysis works well, even if it is not general.
Satisfying the Client

- Precision/efficiency required depends on client
- Barbara Ryder:
  - should look for classes of clients with similar needs
- Manuel Fahndrich:
  - two such clients
    - optimizations
      - current analyses may be sufficient
    - error detection & program understanding tools
      - lower bound on precision
- Manuvir Das:
  - error detection => Killer App for pointer analysis
Does Flow-Sensitivity Matter?

- Flow-sensitive analysis does not provide significant precision improvement over subset-based flow-insensitive [HP00]
  - Assuming:
    - no CS, malloc site, pts-to, whole program, aggregates summarized

- Need more studies, clients
Direct Precision

[HPO0]

Mod

Stmt Mod

Avg Num Objects/ptr dereference

30
25
20
15
10
5
0

Address Taken
FI Subset
FI Equality
Flow Sensitive

[HP00]
Live Variables and Dead Assignments

### Live Variables

![Bar chart for Live Variables](chart1.png)

- **Address Taken**: 34.24
- **FI Subset**: 20.13
- **FI Equality**: 18.36
- **Flow Sensitive**: 18.3

### Dead Assignments

![Bar chart for Dead Assignments](chart2.png)

- **Address Taken**: 1.91
- **FI Subset**: 1.91
- **FI Equality**: 1.96
- **Flow Sensitive**: 1.96
Reaching Defs and Flow Dependences

Reaching Defs

Flow Dependences

<table>
<thead>
<tr>
<th>Address Taken</th>
<th>FI Subset</th>
<th>FI Equality</th>
<th>Flow Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>36.39</td>
<td>23/23</td>
<td>22.04</td>
<td>12/23</td>
</tr>
<tr>
<td>20.21</td>
<td>5/23</td>
<td>20.16</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address Taken</th>
<th>FI Subset</th>
<th>FI Equality</th>
<th>Flow Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>52.51</td>
<td>21/23</td>
<td>44.24</td>
<td>43.84</td>
</tr>
<tr>
<td>43.84</td>
<td>9/23</td>
<td>43.84</td>
<td></td>
</tr>
</tbody>
</table>
Constant Propagation and Unexecutable Stmts

Constants

Unexecutable Stmts

<table>
<thead>
<tr>
<th></th>
<th>Avg Num of Constant Exprs/Program</th>
<th>Avg Num of Unexecutable Stmts/Program</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Address Taken</td>
<td>FI Subset</td>
</tr>
<tr>
<td>3/22</td>
<td>7.8</td>
<td>10.6</td>
</tr>
<tr>
<td>1/22</td>
<td>10.6</td>
<td>10.7</td>
</tr>
<tr>
<td>0/22</td>
<td>10.7</td>
<td>10.7</td>
</tr>
</tbody>
</table>
Does Context-Sensitivity Matter?

- Exponential worst-case => improving efficiency [EGH94, WL95]
- Does it improve precision?
  - flow-sensitive analysis
    - probably not [Ruf95]
  - subset-based FI
    - little [FFA00]
  - extended version of equality-based
    - little [DLFR01]
  - for equality-based FI
    - yes [FFA00]
- Assumptions
  - alloc site, pts-to, aggregates summarized, whole program
  - direct metric: [Ruf95, FFA00]
  - alias frequency: [DLFR01]
Context-Sensitivity

- Erik Ruf:
  - Fixed CS strategy may not be appropriate for client
    - Ex, traditional CS approach can yield bad code
  - Eagerly building clones inside a stand-alone ptr analysis is undesirable (potentially exponential)
  - Even highly parameterized standalone analyses pay costs for unneeded contexts
  - Ptr analysis should be integrated with client
Heap Modeling

- Shape analysis[SRW98, GH96,...] has high precision over alloc site naming
- Scalability of most precise analyses is in doubt
- Tom Reps:
  - plenty of interesting issues remain, such as
    - a better understanding of how to identify the important ingredients
    - efficiency
  - producing insights into other problems, such as system/memory configurations that can arise as a computation evolves
Aggregate Modeling

- **Structs/objects**
  - C/C++: absence of strong-typing makes struct field disambiguation nontrivial
    - many analyses didn't distinguish, exceptions [WL95, YHR99, ...]
  - Java's strong-type makes distinguishing fields easier
    - most Java analyses distinguish
  - Few empirical studies exist [YHR99, RLSZA01, LPH01, RMRO1]

- **Arrays**
  - Only [RR99] distinguish array elements, no empirical studies
  - Leverage dependence analysis work?
Aggregate Modeling

- Rakesh Ghiya:
  Need to improve the basis ptr analysis info (especially malloc-site identification in the presence of user-defined memory management, and handling of fields), as opposed to solely focusing on incremental improvements in the propagation techniques.
Demand-Driven/Incremental

- Ptr analysis efficiency is important
- Precision requirements depends on the client
- Why not a demand-driven analysis?
  - Solutions exists for subset-based FI
    [R94,R98,D00,HT01,FRD00,RF01,DLFR01]
  - Open problem for FS

- How about an incremental analysis?
  - Some work [YRL99, VR01]
Java and OO Languages

- Most ptr analysis work is for C
- Does this work transfer to Java?
  - Good news: conservative fallback is not as bad (type info)
  - Good news: can't point to stack variables
  - Bad news: everything is a heap pointer
  - Promising approaches
    - Simpler shape analysis [GH96]
    - Type-based analyses [DMM97, FKS00]
- Need to revalidate studies based on C
Thoughts on Java

- Bjarne Steensgaard:
  - Many ptr analysis that worked well for C perform poorly for Java
  - Ptr analysis designers will adapt to programming languages/styles and output (tools and other analyses)

- Laurie Hendren:
  - Ex. finding properties of complex OO programs like verifying the correctness of iterators in Java
Incomplete Programs

- Most ptr analyses require whole program

- Michael Burke:
  - Component programming/library are becoming more prevalent
  - Whole program analysis less useful
  - Need parameterized ptr analyses wrt how they are configured in a full application
  - Some work [RRL99,RR01] exists, but problem not solved

- Manual Fahndrich:
  - Interface declarations that describe sharing and non-sharing relationships between data structures (shape descriptions) could lead to more precise ptr info
Engineering Insights

- Efficiency (time and memory) of a pointer analysis is important.
- Careful engineering of a pointer analysis, particularly for FS, can dramatically improve its performance and scalability.
- Conference ptr analysis papers
  - background
  - algorithm
  - empirical comparison
  - related work
  - implementation details
- Last section rarely gets written !!!
- To impact production systems, we must describe engineering
Terminology

- "context-sensitive" = "poly-variant"
- "context-insensitive" = "mono-variant"
- flow-insensitive analyses
  - equality = unification = Steensgaard-style = term or equality constraints
  - subset = Andersen-style = inclusion constraints
- pointer analysis, points-to analysis, alias analysis
- formulation
  - data flow, constraint-based, abstract interpretation, non-standard type inference
So, Have We Solved This Problem?

- No!
- Better question: will we ever "solve" this problem?
- Maybe, maybe not
  - need to focus on classes of clients
    - optimizations vs program understanding
  - new algorithms are nice, but we need strong empirical studies

- Maybe language designers will solve it for us?
  - latest ANSI C allows programmer to severely limit possible aliases
  - Fortran 90, Ada 95 require programmer to declare ptr targets

- But we still need more help for abstractions, such as collections
Thanks!

- Matthew Arnold
- Michael Burke
- Jong-Deok Choi
- Manuvir Das
- Amer Diwan
- Manuel Fahndrich
- Stephen Fink
- David Grove
- Rakesh Ghiya

- Laurie Hendren
- Susan Horwitz
- Bill Landi
- G. Ramalingam
- Tom Reps
- Erik Ruf
- Barbara Ryder
- Mooly Sagiv
- Bjarne Steensgaard