Pointer Analysis

CSE 501
Spring 15
Course Outline

• Static analysis
  – Dataflow and abstract interpretation
  – Applications

• Beyond general-purpose languages

• Program Verification

• Dynamic analysis

• New compilers
Today

• Intro to pointer analysis
  – What’s the big deal?
• Different aspects of the problem
• Two solutions
  – Andersen-style
  – Steensgaard-style
Pointer Analysis
What’s the problem?

```c
int * p = malloc(...)
int * q = ...
...
...
p = q;
p = &q2;
*p = q;
foo(p)
```
Uses

• Alias analysis:
  – For every pair of pointers in the program, determine if they can ever point to the same memory location

• Compiler optimization
  – \*p = a + b;
    x = a + b;
  – a + b is not redundant if \*p aliases a or b
  – Same for constant propagation, dead code elimination, etc
Uses

• Program parallelization
  – Converting sequential code into parallel implementations automatically

• Shape analysis
  – Find properties of data structures in the heap

• Detecting memory problems
  – Leaks, *NULL, security holes
Why is it hard?

• Complexity: huge in both space and time
  – How many pointers are there in a program?
  – Analyze every program point
  – Need to consider all paths to each program point

• Whole / part of the program?
  – Issues with external libraries

• The problem is undecidable
  [Landi 92, Ramalingam 94]
Designing a pointer analysis

- Must vs may
- Model programs and heap
- Model aggregates
- Analysis sensitivities
Representing points-to information

• Variable pairs that refer to the same memory location
  – <*a, b>, <*c, b>, <*a, *c>
  – *a and b alias, same with *c and b

• Points-to pairs:
  – <a → b>, <c → b>
  – a points to b, and c points to b (hence *a and *c are alias)

• Alias sets:
  – {*a, b, *c}
  – They all point to the same memory location

• Convert from one to another?
  – What are the tradeoffs?
Modeling the heap

• Lump everything into one

• By allocation site
  – Each call to new / malloc is a node
  – Doesn’t differentiate between multiple objects allocated by the same site

• Specialized data structures
  – Map of “memory address” to object
Modeling Aggregates

• Arrays
  – Each element is treated as individual location
  – Entire array as a single location
  – First / last element distinct from others

• Classes / Structures
  – Each field is treated as individual location
  – Lump all fields together
Sensitivity

• Flow sensitive
  \[
  x = y \\
  z = x \\
  z = x
  \]

• Path sensitive
  \[
  \begin{align*}
  \text{if (c)} & \quad x = z \\
  \text{else} & \quad x = y \\
  \text{if (c)} & \quad x = y \\
  \text{else} & \quad x = z
  \end{align*}
  \]

• 1-Context sensitive
  \[
  x = \text{foo}(y) \\
  z = \text{foo}(q) \\
  \text{foo (x) \{ return } x; \}
  \]

• Field sensitive
  \[
  \begin{align*}
  o.f &= x \\
  o.f &= y \\
  o.f &= x \\
  o.g &= y
  \end{align*}
  \]
## Pointer-induced Aliasing: A Problem Classification [Landi and Ryder, POPL 90]

<table>
<thead>
<tr>
<th>Alias Mechanism</th>
<th>Intraprocedural May Alias</th>
<th>Intraprocedural Must Alias</th>
<th>Interprocedural May Alias</th>
<th>Interprocedural Must Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference Formals, No Pointers, No Structures</td>
<td></td>
<td></td>
<td>Polynomial[1, 5]</td>
<td>Polynomial[1, 5]</td>
</tr>
<tr>
<td>Single level pointers, No Reference Formals, No Structures</td>
<td>Polynomial</td>
<td>Polynomial</td>
<td>Polynomial</td>
<td>Polynomial</td>
</tr>
<tr>
<td>Single level pointers, Reference Formals, No Pointer Reference Formals, No Structures</td>
<td></td>
<td></td>
<td>Polynomial</td>
<td>Polynomial</td>
</tr>
<tr>
<td>Multiple level pointers, No Reference Formals, No Structures</td>
<td>(N \mathcal{P})-hard</td>
<td>Complement is (N \mathcal{P})-hard</td>
<td>(N \mathcal{P})-hard</td>
<td>Complement is (N \mathcal{P})-hard</td>
</tr>
<tr>
<td>Single level pointers, Pointer Reference Formals, No Structures</td>
<td></td>
<td></td>
<td>(N \mathcal{P})-hard</td>
<td>Complement is (N \mathcal{P})-hard</td>
</tr>
<tr>
<td>Single level pointers, Structures, No Reference Formals</td>
<td>(N \mathcal{P})-hard[14]</td>
<td>Complement is (N \mathcal{P})-hard</td>
<td>(N \mathcal{P})-hard[14]</td>
<td>Complement is (N \mathcal{P})-hard</td>
</tr>
</tbody>
</table>

Table 1: Alias problem decomposition and classification
A Pointer Language

• (Assume x and y are pointers)
• \( y = \&x \)
  – y points to x
• \( y = x \)
  – If x points to z then y points to z
• \( *y = x \)
  – If y points to z and z is a pointer, and if x points to w then z now points to w
• \( y = *x \)
  – If x points to z and z is a pointer, and if z points to w then y not points to w
A Pointer Language

- points-to(x): set of variables that pointer variable x may point to

- Example: points-to(x) = \{y, z\}
  - x can point to either y or z
Andersen’s-style Pointer Analysis
• Flow, context insensitive, inclusion-based algorithm

<table>
<thead>
<tr>
<th>Statement</th>
<th>Constraint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y = &amp; x )</td>
<td>( y \supseteq {x} )</td>
<td>( x \in \text{points-to}(y) )</td>
</tr>
<tr>
<td>( y = x )</td>
<td>( y \supseteq x )</td>
<td>( \text{points-to}(y) \supseteq \text{points-to}(x) )</td>
</tr>
<tr>
<td>( y = *x )</td>
<td>( y \supseteq *x )</td>
<td>( \forall v \in \text{points-to}(x). \text{points-to}(y) \supseteq \text{points-to}(x) )</td>
</tr>
<tr>
<td>( *y = x )</td>
<td>( *y \supseteq x )</td>
<td>( \forall v \in \text{points-to}(y). \text{points-to}(v) \supseteq \text{points-to}(x) )</td>
</tr>
</tbody>
</table>
An Example

\[
p = \&a; \\
q = p; \\
p = \&b; \\
r = p;
\]

\[
p \supseteq \{a\} \\
q \supseteq p \\
p \supseteq \{b\} \\
r \supseteq p
\]

Solving the equations:

<table>
<thead>
<tr>
<th>Points-to</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>{a, b}</td>
</tr>
<tr>
<td>q</td>
<td>{a, b}</td>
</tr>
<tr>
<td>r</td>
<td>{a, b}</td>
</tr>
<tr>
<td>a</td>
<td>{}</td>
</tr>
<tr>
<td>b</td>
<td>{}</td>
</tr>
</tbody>
</table>

Example from Prof. Stephen Chong
Another Example

\[
p = \&a;
p \supseteq \{a\}
\]
\[
q = \&b;
q \supseteq p
\]
\[
*p = q;
*p \supseteq q
\]
\[
r = \&c;
r \supseteq \{c\}
\]
\[
s = p;
s \supseteq p
\]
\[
t = *p;
t \supseteq *p
\]
\[
*s = r;
*s \supseteq r
\]

<table>
<thead>
<tr>
<th>Points-to</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>{a}</td>
</tr>
<tr>
<td>q</td>
<td>{b}</td>
</tr>
<tr>
<td>r</td>
<td>{c}</td>
</tr>
<tr>
<td>s</td>
<td>{a}</td>
</tr>
<tr>
<td>t</td>
<td>{b, c}</td>
</tr>
<tr>
<td>a</td>
<td>{b, c}</td>
</tr>
<tr>
<td>b</td>
<td>{}</td>
</tr>
<tr>
<td>c</td>
<td>{}</td>
</tr>
</tbody>
</table>
Precision

\( p = \&a; \quad p \rightarrow a \)

\( q = \&b; \quad p \rightarrow a \quad q \rightarrow b \)

\( *p = q; \quad p \rightarrow a \quad q \rightarrow b \)

\( r = \&c; \quad p \rightarrow a \quad q \rightarrow b \quad r \rightarrow c \)

\( s = p; \quad p \rightarrow a \quad r \rightarrow c \)

\( t = *p; \quad s \rightarrow a \quad r \rightarrow c \)

\( *s = r; \quad p \rightarrow a \quad q \rightarrow b \quad t \rightarrow a \)

<table>
<thead>
<tr>
<th>Points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
</tr>
<tr>
<td>q</td>
</tr>
<tr>
<td>r</td>
</tr>
<tr>
<td>s</td>
</tr>
<tr>
<td>t</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
</tbody>
</table>
Precision

\[ p = \&a; \]
\[ q = \&b; \]
\[ *p = q; \]
\[ r = \&c; \]
\[ s = p; \]
\[ t = *p; \]
\[ *s = r; \]

Points-to

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>{a}</td>
</tr>
<tr>
<td>q</td>
<td>{b}</td>
</tr>
<tr>
<td>r</td>
<td>{c}</td>
</tr>
<tr>
<td>s</td>
<td>{a}</td>
</tr>
<tr>
<td>t</td>
<td>{b, c}</td>
</tr>
<tr>
<td>a</td>
<td>{b, c}</td>
</tr>
<tr>
<td>b</td>
<td>{}</td>
</tr>
<tr>
<td>c</td>
<td>{}</td>
</tr>
</tbody>
</table>
Precision

\[ p = \&a; \quad p \rightarrow a \]
\[ q = \&b; \quad p \rightarrow a \quad q \rightarrow b \]
\[ *p = q; \quad p \rightarrow a \quad q \rightarrow b \quad *p \rightarrow q \]
\[ r = \&c; \quad p \rightarrow a \quad r \rightarrow c \quad q \rightarrow b \]
\[ s = p; \quad s \rightarrow p \quad r \rightarrow c \]
\[ t = *p; \quad s \rightarrow p \quad r \rightarrow c \]
\[ *s = r; \quad *s \rightarrow r \quad t \rightarrow b \]

<table>
<thead>
<tr>
<th>Points-to</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>{a}</td>
</tr>
<tr>
<td>q</td>
<td>{b}</td>
</tr>
<tr>
<td>r</td>
<td>{c}</td>
</tr>
<tr>
<td>s</td>
<td>{a}</td>
</tr>
<tr>
<td>t</td>
<td>{b, c}</td>
</tr>
<tr>
<td>a</td>
<td>{b, c}</td>
</tr>
<tr>
<td>b</td>
<td>{}</td>
</tr>
<tr>
<td>c</td>
<td>{}</td>
</tr>
</tbody>
</table>
Andersen as Graph Closure

• One node for each memory location
  – i.e., elements in any points-to set
• Each node contains a points-to set

• Solve equations by computing transitive closure of graph, and add edges according to constraints
## Andersen as Graph Closure

<table>
<thead>
<tr>
<th>Statement</th>
<th>Constraint</th>
<th>Meaning</th>
<th>Graph Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = &amp; x$</td>
<td>$y \supseteq {x}$</td>
<td>$x \in \text{points-to}(y)$</td>
<td>Nothing</td>
</tr>
<tr>
<td>$y = x$</td>
<td>$y \supseteq x$</td>
<td>$\text{points-to}(y) \supseteq \text{points-to}(x)$</td>
<td>Add edge from $x$ to $y$</td>
</tr>
<tr>
<td>$y = *x$</td>
<td>$y \supseteq *x$</td>
<td>$\forall v \in \text{points-to}(x). \text{points-to}(y) \supseteq \text{points-to}(x)$</td>
<td>Nothing</td>
</tr>
<tr>
<td>$*y = x$</td>
<td>$*y \supseteq x$</td>
<td>$\forall v \in \text{points-to}(y). \text{points-to}(v) \supseteq \text{points-to}(x)$</td>
<td>Nothing</td>
</tr>
</tbody>
</table>
Same Example, as Graph

\[
p = \&a; \quad p \supseteq \{a\}
q = \&b; \quad q \supseteq p
*p = q; \quad *p \supseteq q
r = \&c; \quad r \supseteq \{c\}
s = p; \quad s \supseteq p
t = *p; \quad t \supseteq *p
*s = r; \quad *s \supseteq r
\]

\[
y = x \quad y \supseteq x \quad \text{points-to}(y) \supseteq \text{points-to}(x) \quad \text{Add edge from } x \text{ to } y
\]
Same Example, as Graph

\[
\begin{align*}
p &= &a; & p \supseteq \{a\} \\
q &= &b; & q \supseteq p \\
*p &= q; & *p \supseteq q \\
r &= &c; & r \supseteq \{c\} \\
s &= p; & s \supseteq p \\
t &= *p; & t \supseteq *p \\
*s &= r; & *s \supseteq r
\end{align*}
\]

\[
\begin{array}{c}
y = x \quad y \supseteq x \quad \text{points-to}(y) \supseteq \text{points-to}(x) \quad \text{Add edge from x to y}
\end{array}
\]
Worklist Algorithm

// Init graph and points-to sets using base constraints
W = { nodes with non-empty points-to sets }
while W is not empty {
    v = choose from W
    for each constraint v ⊇ x
        add edge x → v, and add x to W if edge is new
    for each a ∈ points-to(v) do {
        for each constraint p ⊇ *v
            add edge a → p, and add a to W if edge is new
        for each constraint *v ⊇ q
            add edge q → a, and add q to W if edge is new
    }
    for each edge v → q do {
        points-to(q) = points-to(q) U points-to(v),
        and add q to W if points-to(q) changed
    }
}
Worklist Algorithm

• Complexity is $O(n^3)$, where $n =$ number of nodes in graph

• In practice, improve by eliminating cycles
  – Detect strongly connected components in points-to graph and collapse to single node

• How to detect cycles?
  – Some reduction can be done statically, some on-the-fly as new edges added
Steensgaard-style Analysis

• Similar to Andersen, except that each node can only point to one other node in points-to graph
# Steensgaard-style Analysis

- Flow, context insensitive, unification-based algorithm

<table>
<thead>
<tr>
<th>Statement</th>
<th>Constraint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = &amp; x$</td>
<td>$y \supseteq {x}$</td>
<td>$x \in \text{points-to}(y)$</td>
</tr>
<tr>
<td>$y = x$</td>
<td>$y = x$</td>
<td>$\text{points-to}(y) = \text{points-to}(x)$</td>
</tr>
<tr>
<td>$y = \ast x$</td>
<td>$y = \ast x$</td>
<td>$\forall v \in \text{points-to}(x).$ $\text{points-to}(y) = \text{points-to}(x)$</td>
</tr>
<tr>
<td>$\ast y = x$</td>
<td>$\ast y = x$</td>
<td>$\forall v \in \text{points-to}(y).$ $\text{points-to}(v) = \text{points-to}(x)$</td>
</tr>
</tbody>
</table>
Steensgaard-style Analysis
• Flow, context insensitive, unification-based algorithm

<table>
<thead>
<tr>
<th>Statement</th>
<th>Constraint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = &amp;x$</td>
<td>$y \supseteq {x}$</td>
<td>$x \in \text{points-to}(y)$</td>
</tr>
<tr>
<td>$y = x$</td>
<td>$y = x$</td>
<td>$\text{points-to}(y) = \text{points-to}(x)$</td>
</tr>
<tr>
<td>$y = *x$</td>
<td>$y = *x$</td>
<td>$\forall v \in \text{points-to}(x). \text{points-to}(y) = \text{points-to}(x)$</td>
</tr>
<tr>
<td>$*y = x$</td>
<td>$*y = x$</td>
<td>$\forall v \in \text{points-to}(y). \text{points-to}(v) = \text{points-to}(x)$</td>
</tr>
</tbody>
</table>
Steensgaard-style Analysis

• Implications for using equality constraints
  – Each statement is processed exactly once
  – Only one iteration of the worklist algorithm
  – Union-find / disjoint set data structure
  – Worst case complexity: $O(n)$ (almost), where $n =$ number of nodes in graph

  – Less precise than Andersen’s
Example

\[
\begin{align*}
  x & \rightarrow z \rightarrow a \\
  y & \rightarrow w \rightarrow v \rightarrow b
\end{align*}
\]

\[
\text{Statement} = *y
\]

<table>
<thead>
<tr>
<th>Statement</th>
<th>Constraint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x = *y)</td>
<td>(x = *y)</td>
<td>(\forall v \in \text{points-to}(y). \text{points-to}(x) = \text{points-to}(y))</td>
</tr>
</tbody>
</table>
Example

\[
\begin{align*}
  x &= *y \\
  y &\rightarrow w \quad v \\
  v &\rightarrow b \\
  y &\rightarrow v \\
  x &\rightarrow z \\
  z &\rightarrow a
\end{align*}
\]

<table>
<thead>
<tr>
<th>Statement</th>
<th>Constraint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x = *y)</td>
<td>(x = *y)</td>
<td>(\forall v \in \text{points-to}(y). \quad \text{points-to}(x) = \text{points-to}(y))</td>
</tr>
</tbody>
</table>
Example

\[
x = *y
\]

<table>
<thead>
<tr>
<th>Statement</th>
<th>Constraint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = *y$</td>
<td>$x = *y$</td>
<td>$\forall v \in \text{points-to}(y)$. points-to(x) = points-to(y)</td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>Statement</th>
<th>Constraint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = *y$</td>
<td>$x = *y$</td>
<td>$\forall v \in \text{points-to}(y).$ points-to($x$) = points-to($y$)</td>
</tr>
</tbody>
</table>