Points-to Analysis in Almost Linear Time

Bjarne Steensgaard

Martin Velez
Class Presentation
ECS289C @ UC Davis
Winter 2015
Motivation

• Applications
  • Optimizing compilers
  • Program Understanding
  • Code browsing tools
  • Escape analysis
• Then state-of-the-art
  • Interprocedural analysis was slow but desired.
    • Whole-program optimization.
Flow-insensitive

• Order of statements is ignored
  • No control flow
  • Cannot do “strong updates” (overwrites)
    • \( x = \&a \)
      • \( a \) is added to \( \text{pts}(x) \)
Points-to Graph

- $|V| = n$
- $|E| = n^2$

```
Vars  x_1  ...  x_n

x_1   0     0
...
...

x_n   1     0

n^2 entries
```
**Smaller Points-to Graph**

- $|V| = n$
- $|E| = n$

$n$ entries

<table>
<thead>
<tr>
<th>Variable</th>
<th>Points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_j$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_i$</td>
<td>$x_k$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_n$</td>
<td>$x_l$</td>
</tr>
</tbody>
</table>
Example 1

1. \( p = \&x \)
2. \( r = \&p \)
3. \( q = \&y \)
4. \( s = \&q \)
5. \( r = s \)

Andersen's (More precise)

Join \( p \) and \( q \)

Steengaard's (Faster)

Join \( x \) and \( y \)
Example 2

1. $a = \&x$
2. $b = \&y$
3. if $p$ then
   4. $y = \&z$
4. else
   5. $y = \&x$
6. $c = \&y$
Example 3

```
1  q = &x
2  q = &y
3  p = q
4  q = &z
```

After Line 3

After Line 4

Simple example where Andersen’s algorithm does no more work.
Syntax

Syntax of relevant statements in C-like language

\[ S ::= x = y \]
\[ x = \&y \]
\[ x = \ast y \]
\[ x = \text{op}(y_1 \ldots y_n) \]
\[ x = \text{allocate}(y) \]
\[ *x = y \]
\[ x = \text{fun}(f_1 \ldots f_n) \rightarrow (r_1 \ldots r_m) \ S^* \]
\[ x_1 \ldots x_m = p(y_1 \ldots y_n) \]
Types

Non-standard

\[ \alpha ::= \tau \times \lambda \]
\[ \tau ::= \bot \mid \text{ref}(\alpha) \quad \text{locs or pointers to locs} \]
\[ \lambda ::= \bot \mid \text{lam}(\alpha_1 \ldots \alpha_n)(\alpha_{n+1} \ldots \alpha_{n+m}) \quad \text{functions or pointers to functions} \]

Note on type equality:
\[ t_i = t_j \iff \text{bottom or } i = j \]
Typing Rule Example

\[ A \vdash x : \text{ref}(\alpha_1) \]
\[ A \vdash y : \text{ref}(\alpha_2) \]
\[ \alpha_2 \triangleleft \alpha_1 \]
\[ \frac{}{A \vdash \text{welltyped}(x = y)} \]

\[ t_1 \triangleleft t_2 \iff (t_1 = \bot) \lor (t_1 = t_2) \]
\[ \times t_2 \triangleleft (t_3 \times t_4) \iff (t_1 \triangleleft t_3) \land (t_2 \triangleleft \]
All Typing Rules

\[
\begin{align*}
A \vdash x : \text{ref}(\alpha_1) \\
A \vdash y : \text{ref}(\alpha_2) \\
\alpha_2 \leq \alpha_1 \\
\hline
A \vdash \text{welltyped}(x = y)
\end{align*}
\]

\[
\begin{align*}
A \vdash x : \text{ref}(\tau \times _) \\
A \vdash y : \tau \\
\hline
A \vdash \text{welltyped}(x = \&y)
\end{align*}
\]

\[
\begin{align*}
A \vdash x : \text{ref}(\alpha_1) \\
A \vdash y : \text{ref}(\text{ref}(\alpha_2) \times _) \\
\alpha_2 \leq \alpha_1 \\
\hline
A \vdash \text{welltyped}(x = \*y)
\end{align*}
\]

\[
\begin{align*}
A \vdash x : \text{ref}(\alpha) \\
A \vdash y_i : \text{ref}(\alpha_i) \\
\forall i \in [1 \ldots n] : \alpha_i \leq \alpha \\
\hline
A \vdash \text{welltyped}(x = \text{op}(y_1 \ldots y_n))
\end{align*}
\]

\[
\begin{align*}
A \vdash x_i : \text{ref}(\alpha_{n+i}) \\
A \vdash p : \text{ref}(\_ \times \text{lam}(\alpha_1 \ldots \alpha_n)(\alpha_{n+1} \ldots \alpha_{n+m})) \\
A \vdash f_i : \text{ref}(\alpha_i) \\
A \vdash r_j : \text{ref}(\alpha_{n+j}) \\
\forall s \in S^* : A \vdash \text{welltyped}(s) \\
\forall i \in [1 \ldots n] : \alpha_i \leq \alpha_i' \\
\forall j \in [1 \ldots m] : \alpha_{n+j} \leq \alpha_{n+j}' \\
\hline
A \vdash \text{welltyped}(x_1 \ldots x_m = p(y_1 \ldots y_n))
\end{align*}
\]
Example 2 (revisit)

```plaintext
a = &x
b = &y
if p then
  y = &z;
else
  y = &x
fi
c = &y
```

```
a: \tau_1 = \text{ref}(\tau_4 \times \bot)
b: \tau_2 = \text{ref}(\tau_5 \times \bot)
c: \tau_3 = \text{ref}(\tau_5 \times \bot)
x: \tau_4 = \text{ref}(\bot \times \bot)
y: \tau_5 = \text{ref}(\tau_4 \times \bot)
z: \tau_4
p: \tau_6 = \text{ref}(\bot \times \bot)
```
Algorithm

• Find all pointer assignments in program

• Form points-to graph nodes for pointers to variables and functions, and variables.

• For each statement (arbitrary order)
  • construct points-to edges
  • Merge nodes (and edges) where indicated by unification constraints
Bottom Line

• Small programs - Andersen

• Large programs - Steensgaard