Using types to optimize programs

Amer Diwan

The Question

Are type declarations in statically typed languages useful for optimizing programs?

Disadvantage: Too imprecise. Or are they?
Advantages:
– Fast
– Do not require the whole program
Assumption: Object-oriented programming language
Outline

- Using types to do pointer analysis
  - this class
- Using pointer analysis to improve program performance
  - next class
  - evaluation to see how well they work

A Running Example

$T \ast t$
$S1 \ast s1$
$S2 \ast s2$

$T: \text{INT}$
$g: \text{INT}$

$t$, $s1$, and $s2$ are pointers to memory locations in the heap
Running Example (cont.)

Redundant load elimination eliminates lexically identical redundant heap references

Another example

- $t = \text{NEW}(s1)$
  $x^\wedge = \text{NEW}(s2)$
  $t.f();$
- What does $t.f()$ call?
Analysis I: T-TBAA

Use type compatibility only:

\[
T-TBAA(p, q) = \text{Subtypes}(\text{Type}(p)) \cap \text{Subtypes}(\text{Type}(q)) \neq 0
\]

Weaknesses of T-TBAA

- Ignores some type information
  - E.g., field names could also be used
  - TF-TBAA
- Ignores instructions in the program: only considers type declarations
  - TFM-TBAA
  - TM-TBAA
Analysis II: TF-TBAA

Use other properties of types, e.g.,
– Accesses to distinct fields cannot alias each other
– An array reference cannot alias a field reference
– Must consider by reference, ...

\[
\begin{array}{c}
(t^\wedge, s1^\wedge) \\
(t^\wedge, s2^\wedge) \\
(t^\wedge, t^\wedge) \\
(t^\wedge, t^\wedge) \\
(s1^\wedge, s2^\wedge)
\end{array}
\]

Analysis III: TFM-TBAA

• Incorporate flow insensitive analysis. \( t \) aliases \( s1 \) if
  – at some point, a reference to an object of type \( S1 \) may have been assigned to a location of type \( T \) (\( S1 \) is merged into \( T \))

\[\text{e.g., } t = \text{new } S1;\]

\[
\begin{array}{c}
(t, s1) \\
(t, s1) \\
(t, t) \\
(t, t) \\
(s1, s2)
\end{array}
\]
Qualitative analysis

• How do TBAA compare to Steensgaard in precision?
  - \( t_1: T; t_2: T; \)
    \( t_1 = \text{new } T; \)
    \( t_2 = \text{new } T; \)
  - According to Steensgaard \( t_1 \) and \( t_2 \) do not point to the same object
    According to TBAA they may point to the same object

Qualitative analysis (cont)

• \( s_1: S_1; s_2: S_2; t: T; \)
  \( s_1 = \text{new } S_1; s_2: \text{new } S_2; \)
  \( t = s_1; t = s_2; \)
• According to Steensgaard, \( t, s_1, \) and \( s_2 \) may all point to the same object
• According to TBAA, \( s_1 \) and \( s_2 \) may not point to the same object
• Bottom line: incomparable w.r.t. precision
When will TBAA work best?

• (When programs are written in a safe language)
• When programs use types carefully rather than having everything by a void* or ROOT
  – t: T; s1: T; s2: T versus
t: T; s1: S1; s2: S2
• How well does it work for real programs?

Now that I have a pointer analysis, what do I do with it?

• Eliminate redundant loads
• Replace method invocations by direct calls
  – Type hierarchy analysis
  – Intra and Interprocedural type propagation
  – ...with and without TBAA
Evaluation Environment

Some of the benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Lines</th>
<th>Dynamic heap loads (% total instrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>395</td>
<td>10</td>
</tr>
<tr>
<td>dformat</td>
<td>602</td>
<td>9</td>
</tr>
<tr>
<td>write-pickle</td>
<td>654</td>
<td>13</td>
</tr>
<tr>
<td>k-tree</td>
<td>726</td>
<td>10</td>
</tr>
<tr>
<td>slisp</td>
<td>1,645</td>
<td>27</td>
</tr>
<tr>
<td>m2tom3</td>
<td>10,574</td>
<td>8</td>
</tr>
<tr>
<td>m3cg</td>
<td>16,475</td>
<td>8</td>
</tr>
<tr>
<td>trestle</td>
<td>28,977</td>
<td></td>
</tr>
</tbody>
</table>
Static Evaluation

Measure alias pairs.
E.g., (p,q) \equiv p \text{ and } q \text{ are references in the program that } \textit{may} \text{ reference the same heap location.}

\Rightarrow \text{ Enables comparing analyses}

What it does \textbf{not} do:
- Allow us to compare analyses with different strengths
- Tell us how effective the analysis is w.r.t. clients
- Tell us how much better we could do

Static evaluation

Alias pairs within procedure as a percent of all possible pairs within procedure

<table>
<thead>
<tr>
<th></th>
<th>T-TBAA</th>
<th>TF-TBAA</th>
<th>TFM-TBAA</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>31</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>dformat</td>
<td>24</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>write-pickle</td>
<td>24</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>k-tree</td>
<td>29</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>slisp</td>
<td>45</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>m2tom3</td>
<td>41</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>m3cg</td>
<td>32</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>trestle</td>
<td>23</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

• These numbers look pretty bad by themselves!
• TF-TBAA better than T-TBAA
• TFM-TBAA doesn’t offer much
Dynamic Evaluation

Measure run-time impact of RLE and method resolution
⇒ Directly measures impact of an analysis on its clients

What it does not do:
– Give results for all inputs and optimizations
– Tell us how much better we could do

Run-time improvements with RLE

![Bar chart showing run-time improvements with RLE for different formats and configurations.](chart.png)
Overall run-time improvement

Limit Evaluation

Measure upper-bounds on performance:

– count heap references that are still redundant after redundant load elimination.
– count methods that are still unresolved but call the same procedure at run time

⇒ Reveals potential room for improvement

What it does not do:

– Give results for all inputs and optimizations
Limit Evaluation (intraprocedural):
Loads that are still redundant

Why still redundant?

"Rest" is the upper-bound on what a better alias analysis can do.
Summary of types for pointer analysis

- Despite large number of alias pairs, type-based alias analysis is nearly perfect for these benchmarks and optimizations

- More precise analysis is not necessarily better
  - The three evaluation techniques tell us different things and should all be used.

- Type-safety can be used to improve program performance!
Discussion

• Strengths
  – Simple and fast analysis that works for an important application
• Weaknesses
  – Doesn't work for unsafe languages
  – How well do TBA work for other uses of pointer analysis?

Next topic

• Closures
• Reading: Scott 3.3