Alias Analysis

Last time
– Reuse optimization

Today
– Alias analysis (pointer analysis)

Next time
– More alias analysis (pointer analysis)

Aliasing

What is aliasing?
– When two expressions denote the same mutable memory location
– e.g., \( p = \text{new Object;} \)
  \( q = p; \) \( \Rightarrow *p \) and \( *q \) alias

How do aliases arise?
– Pointers
– Call by reference (parameters can alias each other or non-locals)
– Array indexing
– C \texttt{union}, Pascal variant records, Fortran \texttt{EQUIVALENCE} and \texttt{COMMON} blocks
Aliasing Examples

Pointers (e.g., in C)

int *p, i;
p = &i;

* p and i alias

Parameter passing by reference (e.g., in Pascal)

procedure procl(var a:integer; var b:integer);
...
procl(x,x);
procl(x,glob);

a and b alias in body of procl
b and glob alias in body of procl

Array indexing (e.g., in C)

int i,j, a[128];
i = j;

a[i] and a[j] alias

What Can Alias?

Stack storage and globals

void fun(int p1) {
    int i, j, temp;
    ...
}

do i, j, or temp alias?

Heap allocated objects

n = new Node;
n->data = x;
n->next = new Node;
...

do n and n->next alias?
What Can Alias? (cont)

Arrays
for (i=1; i<=n; i++) {
    b[c[i]] = a[i];
}

Can c[i_1] and c[i_2] alias?

Fortran

Java

<table>
<thead>
<tr>
<th>7</th>
<th>1</th>
<th>4</th>
<th>2</th>
<th>3</th>
<th>1</th>
<th>9</th>
<th>0</th>
</tr>
</thead>
</table>

Alias Analysis

Goal: Statically identify aliases
- Can memory reference m and n access the same state at program point p?
- What program state can memory reference m access?

Why is alias analysis important?
- Many analyses need to know what storage is read and written
e.g., available expressions (CSE)
    *p = a + b;
    y = a + b;
    If *p aliases a or b, the second expression is not redundant (CSE fails)

- e.g., Reaching definitions (constant propagation)
    d_1: x = 3;
    d_2: *p = 4;
    d_3: y = x;
    If *p aliases x, d_3 reaches this point; otherwise, both d_1 and d_2 reach

Otherwise we must be very conservative
How hard is this problem?

Undecidable
- Landi 1992
- Ramalingan 1994

All solutions are conservative approximations

Is this problem solved?
- Why haven’t we solved this problem? [Hind 2001]
- Wednesday and next week we will look at some open issues

Alias/Pointer Analysis Survey

Today
- Address Taken
- Steensgaard (unification)

Tomorrow
- Anderson (inclusion)
- Emami

Next Week
- Burk
- Choi
Trivial Alias Analyses

Easiest approach
– Assume that nothing must alias
– Assume that everything may alias everything else
– Yuck!

Address taken: A slightly better approach (for C)
– Assume that nothing must alias
– Assume that all pointer dereferences may alias each other
– Assume that variables whose addresses are taken (and globals) may alias all pointer dereferences

\texttt{e.g.,}
\begin{verbatim}
  p = &a;
  \ldots
  a = 3; b = 4;
  *q = 5;
\end{verbatim}

*\texttt{q} and \texttt{a} may alias, so \texttt{a} may be 3 or 5, but \texttt{*q} does not alias \texttt{b}, so \texttt{b} is 4

Enhance with type information?

Properties of Alias Analysis

Scope: Intraprocedural (per procedure) or Interprocedural (whole program)

Representation
– Alias pairs?
– Points-to sets?
– Others. . .?

Flow sensitivity: Sensitive versus insensitive?

Context sensitivity: Sensitive versus insensitive?

Definiteness: May versus must?

Heap Modeling?

Aggregate Modeling?
## Representations of Aliasing

### Equivalence sets
- All memory references in the same set are aliases
- e.g., \{\*a, b\}, \{\*b, c, **a\}

### Alias pairs
- Pairs that refer to the same memory
  - e.g., (\*a, b), (\*b, c), (**a, c)
  - Completely general

* Points-to pairs [Emami94]
  - Pairs where the first member points to the second
    - e.g., (a -> b), (b -> c)
  - Possibly more compact than alias pairs

---

```c
int **a, *b, c, *d, e;
1: a = &b;
2: b = &c;
```

## Flow Sensitivity of Alias Analysis

### Flow-sensitive alias analysis
- Compute aliasing information at each program point
  - e.g.,
    ```c
    p = &x;
    ...  
    p = &y;  
    ```
    *p and x alias here
    *p and y alias here

### Flow-insensitive alias analysis
- Compute aliasing information for entire procedure
  - e.g.,
    ```c
    p = &x;
    ...  
    p = &y;  
    ```
    *p may alias x or y in this procedure
Definiteness of Alias Information

May (possible) alias information
- Indicates what might be true
  \[ \text{e.g., } \quad \text{if (c) } p = &i; \]
  \[ *p \text{ and } i \text{ may alias } \]

Must (definite) alias information
- Indicates what is definitely true
  \[ \text{e.g., } \quad p = &i; \]
  \[ *p \text{ and } i \text{ must alias } \]

Often need both
- e.g., Consider liveness analysis
  \[ s: *p = *q + 4; \]
  \[ \text{Recall: } \text{in}[s] = \text{use}[s] \cup (\text{out}[s] - \text{def}[s]) \]
  \[ (1) *p \text{ must alias } v \Rightarrow \text{def}[s] = \text{kill}[s] = \{v\} \]
  \[ (2) *q \text{ may alias } v \Rightarrow \text{use}[s] = \text{gen}[s] = \{v\} \]
  \[ \text{Suppose out}[s] = \{v\} \]

FIAlias [Landi & Ryder] equivalent to Steensgaard

Overview
- Put all interesting memory references in separate equivalence sets
- Merge equivalence sets based on pointer assignments
- Merge equivalence sets based on type 2 alias effects, (e.g., merging \(*a \text{ with } d \text{ will cause merge of equiv sets with } b \text{ and } d, \text{ and those with } e \text{ and } c) \]

Characterization of Steensgaard
- Whole program
- Flow-insensitive
- Context-insensitive
- May analysis
- Alias representation: equivalence sets
- Heap modeling?
- Aggregate modeling?
Next Time

**Reading**
- [Emami95]

**Lecture**
- Alias Analysis II
  - Andersen
  - Emami