Alias Analysis

Last time
– Midterm

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 union, Pascal variant records, Fortran EQUIVALENCE and COMMON blocks

Aliasing Examples

Pointers (e.g., in C)
\[ \text{int } *p, i; \]
\[ p = &i; \]
\( *p \) and \( i \) alias

Parameter passing by reference (e.g., in Pascal)
\[ \text{procedure procl(var } a:\text{integer; } \text{var } b:\text{integer);} \]
\[ \ldots \]
\[ \text{procl}(x,x); \]
\( a \) and \( b \) alias in body of \( \text{procl} \)
\[ \text{procl}(x,\text{glob}); \]
\( b \) and \( \text{glob} \) alias in body of \( \text{procl} \)

Array indexing (e.g., in C)
\[ \text{int } i,j, a[128]; \]
\[ i = j; \]
\( a[i] \) and \( a[j] \) alias

What Can Alias?

Stack storage and globals
\[ \text{void fun(int } p1) \{ \]
\[ \text{int } i, j, \text{temp}; \]
\[ \ldots \]
\[ \} \]
\( \text{do } i, j, \text{ or } \text{temp alias?} \)

Heap allocated objects
\[ n = \text{new Node}; \]
\[ n->\text{data} = x; \]
\[ n->\text{next} = \text{new Node}; \]
\[ \ldots \]
\( \text{do } n \text{ and } n->\text{next alias?} \)
What Can Alias? (cont)

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

Can c[i1] and c[i2] alias?

Fortran

Java

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]
- Next week we will look at some open issues

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)
d1: x = 3;
d2: *p = 4;
d3: y = x;
If *p aliases x, d2 reaches this point; otherwise, both d1 and d2 reach

Otherwise we must be very conservative

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
e.g.
p = &a;
    a = 3; b = 4;
    *q = 5;
    
    *q and a may alias, so a may be 3 or 5, but
    *q does not alias b, so 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

Flow Sensitivity of Alias Analysis

Flow-sensitive alias analysis
  - Compute aliasing information at each program point
    e.g.,
    \[\text{p = &x;}\]
    \[\ldots\]
    \[\text{p = &y;}\]
    \[\text{*p and x alias here}\]
    \[\text{*p and y alias here}\]

Flow-insensitive alias analysis
  - Compute aliasing information for entire procedure
    e.g.,
    \[\text{p = &x;}\]
    \[\ldots\]
    \[\text{p = &y;}\]
    \[\text{*p may alias x or y in this procedure}\]

Definiteness of Alias Information

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

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

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

Analogous flow functions

- $\cap$ is $\cup$
- $s$: $p = \& x$
  out\[s\] = $\{ (p \rightarrow x) \cup (\text{in}[s] \setminus \{p \rightarrow y\} \forall y) \}$
- $s$: $p = q$
  out\[s\] = $\{ (p \rightarrow t) \mid (q \rightarrow t) \in \text{in}[s] \} \cup (\text{in}[s] \setminus \{p \rightarrow y\} \forall y))$
- $s$: $p = * q$
  out\[s\] = $\{ (r \rightarrow t) \mid (p \rightarrow r) \in \text{in}[s] \} \cup (\text{in}[s] \setminus \{p \rightarrow x\} \forall x)$
- $s$: $* p = q$
  out\[s\] = $\{ (r \rightarrow t) \mid (p \rightarrow r) \in \text{in}[s] \} \cup (\text{in}[s] \setminus \{p \rightarrow x\} \forall x)$

Must Points-To Analysis

Analogous flow functions

- $\cap$ is $\cap$
- $s$: $p = \& x$
  out\[must\][s] = $\{ (p \rightarrow x) \}$
- $s$: $p = q$
  out\[must\][s] = $\{ (p \rightarrow t) \mid (q \rightarrow t) \in \text{in}[must][s] \} \cup (\text{in}[must][s] \setminus \{p \rightarrow y\} \forall y))$
- $s$: $p = * q$
  out\[must\][s] = $\{ (r \rightarrow t) \mid (p \rightarrow r) \in \text{in}[must][s] \} \cup (\text{in}[must][s] \setminus \{p \rightarrow x\} \forall x)$
- $s$: $* p = q$
  out\[must\][s] = $\{ (r \rightarrow t) \mid (p \rightarrow r) \in \text{in}[must][s] \} \cup (\text{in}[must][s] \setminus \{p \rightarrow x\} \forall x)$

Other Issues (Modeling the Heap)

Issue
- Each allocation creates a new piece of storage
  e.g., $p = \text{new T}$

Proposal?
- Generate (at compile-time) a new “variable” to stand for new storage
  - newvar: Creates a new variable

Flow function
- $s$: $p = \text{new T} \quad \text{newvar}$
  out\[s\] = $\{ (p \rightarrow \text{newvar}) \cup (\text{in}[s] \setminus \{p \rightarrow x\} \forall x) \}$

Problem
- Domain is unbounded!
- Iterative data-flow analysis may not converge

Modeling the Heap (cont)

Simple solution
- Create a summary “variable” (node) for each allocation statement
- Domain: $2^{(\text{Var} \cup \text{Stmt}) \times (\text{Var} \cup \text{Stmt})}$ rather than $2^{\text{Var} \times \text{Var}}$

- Monotonic flow function
  $s$: $p = \text{new T} \quad \text{newvar}$
  out\[s\] = $\{ (p \rightarrow \text{newvar}) \} \cup (\text{in}[s] \setminus \{p \rightarrow x\} \forall x)$
- Less precise (but finite)

Alternatives
- Summary node for entire heap
- Summary node for each type
- K-limited summary
  - Maintain distinct nodes up to k links removed from root variables
Using Alias Information

Example: reaching definitions
- Compute at each point in the program a set of \((s,v)\) pairs, indicating that statement \(s\) may define variable \(v\).

Flow functions
- \(s: \,*p = x;\)
  \[\text{out}_{\text{reach}}[s] = \{(s,z) | (p \to z) \in \text{in}_{\text{maypt}}[s]\} \cup \{(t,y) | (p \to y) \in \text{in}_{\text{mustpt}}[s]\}\]
- \(s: \, x = *p;\)
  \[\text{out}_{\text{reach}}[s] = \{(s,x)\} \cup \{(t,x) | (t \to x) \forall t\}\]
- \(\ldots\)

Function Calls

Question
- How do function calls affect our points-to sets?
  - \(\text{e.g., } p1 = \&x;\)
  - \(p2 = \&p1;\)
  - \(\ldots\)
  - \(\text{foo}();\)

Be conservative
- Assume that any reachable pointer may be changed
- Pointers can be “reached” via globals and parameters
  - May pass through objects in the heap
  - Can be changed to anything reachable or something else
- Can we prune aliases using types?

Problem
- Lose a lot of information

Concepts

What is aliasing and how does it arise?

Properties of alias analyses
- Definiteness: may or must
- Flow sensitivity: sensitive or insensitive
- Context sensitivity: sensitive or insensitive (interprocedural only)
- Representation: alias pairs, points-to sets

Function calls degrade alias information
- Context-sensitive interprocedural analysis

Next Time

Reading
- [Emami94]

Lecture
- Interprocedural analysis