Alias Analysis Motivation

```
a = 1;
b = 2;

c = a + b;
```

```
a = 1;
b = 2;

c = 3;
```

Alias Analysis Motivation

```
a = 1;
b = 2;
*x = 4;
c = a + b;
```

```
a = 1;
b = 2;
*x = 4;
c = ???;
```

Alias Analysis Motivation

```
a = 1;
b = 2;
*x = 4;
c = a + b;
```

```
a = 1;
b = 2;
*x = 4;
c = ???;
```

If `x` must equal `&a`, `c = 6`.
If `x` must equal `&b`, `c = 5`.
If not (`x` may equal `&a` or `x` may equal `&b`), `c = 3`.

Sources of Aliases

1. Call by reference

```
int f(var x, var y) {
  x = 1;
  y = 2;
  return x + y;
}
```

Sources of Aliases

1. Call by reference

```
int f(var x, var y) {
  x = 1;
  y = 2;
  return x + y;
}
int z = 1;
z = f(z, z);
```

Sources of Aliases

2. Address-of operator

```
int a;
int* x = &a;
```

3. Dynamic memory allocation

```
int* x = malloc(sizeof(int));
```

4. Array expressions

```
a[x] = 1;
a[y] = 2;
c = a[x]; // is c = 1 or 2?
```

Address Taken

```
a = 1;
b = 2;
*x = 4;
c = a + b;
```

IF `&a`, `&b` never occur in the program,
THEN `x` can never equal `&a` or `&b`.
(except for evil pointer arithmetic)

Local variables:
in Java, address cannot be taken
in C, quite rare to take their address

Type-Based Analysis

```
double* x;
int a = 1;
int b = 2;
*x = 4;
c = a + b;
```

IF the language enforces declared types,
THEN `x` can never equal `&a` or `&b`.

Alias Pairs vs. Points-To Sets

Alias pairs

(α, β) if the expressions α and β must/may point to the same location

Points-to sets

$o \in pt(p)$ if p must/may point to o

Flow-sensitive vs. Flow-insensitive

Flow-sensitive

- Compute separate analysis information at each program point.
- Consider order in which statements execute.
- Allow strong updates.

Flow-insensitive

- Compute single result for whole program (which holds at every program point).
- Ignore statement execution order.
- Only weak updates allowed.

Subset-based vs. Equality-based

Assignment statement: $x = y$

Subset-based aka Propagation-based aka Andersen

$$pt(y) \subseteq pt(x)$$

x points to everything that y points to.
$O(n^3)$ in theory.

Equality-based aka Unification-based aka Steensgaard

$$pt(y) = pt(x)$$

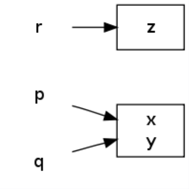
x and y point to the same set of objects.
$O(n\alpha(n))$ but less precise.

Equality-based points-to analysis

Let $S = \{s_1, s_2, s_3, \ldots\}$ be the set of all possible targets of pointers. We partition it into disjoint subsets, and model each pointer to point to one of the disjoint subsets.

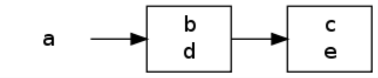
Example

p = &x;
q = &y;
p = q;
r = &z;



Example

a = &b;
b = &c;
a = &d;
d = &e;



Field Sensitivity

	Field Sensitive	Field Insensitive	Field Based
Abstraction	o.f	o.*	*.f
Efficiency	slowest	medium	fastest single iteration
Precision	most precise	very imprecise for OO-style code	medium
Soundness			only if field types enforced
Language	Java, C	C	Java