CS 162 Programming languages

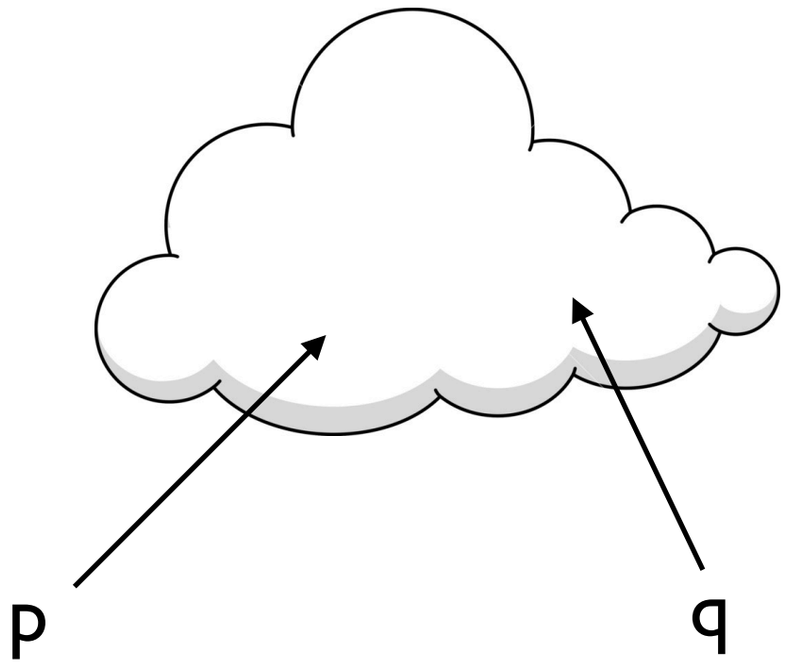
Lecture 16: Program Analysis in Datalog

Yu Feng

Winter 2020

Pointer analysis

```
int * p = malloc(…);
int * q = …
…
…
p = q;
p = &q2;
*p = q;
foo(p);
```



Compute the locations that each variable **may** point to

Applications of pointer analysis

- Information flow analysis:
    - For every variables in the program, determine if they can point to a sensitive location (credit card#, SSN, emails, etc)
- Compiler optimizations
    - Common subexpression elimination
    - *p = a + b;

      x = a + b;
    - a + b is not redundant if *p aliases with a or b
    - Same for constant propagation, dead code elimination, register allocation, etc

Pointers in C

- Assuming x and y are pointers. E.g., x = malloc(T)
- Address taken: y = &x
  - y points to x
- Assign: y = x
  - If x points to z then y **now** points to z
- Store: *y = x
  - If y points to z and z is a pointer, and if x points to w then z **now** points to w
- Load: y = *x
  - If x points to z and z is a pointer, and if z points to w then y **now** points to w

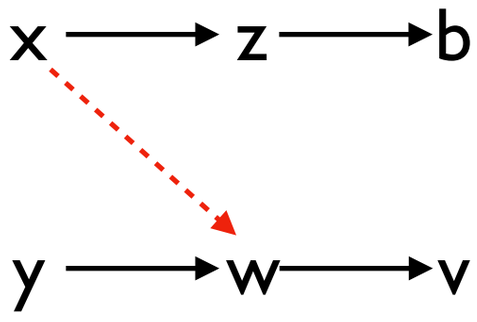
Andersen pointer analysis

- View pointer assignments as **subset** constraints
- Use constraints to propagate points-to information
- Worst case complexity: $O(n^3)$, where n = program size

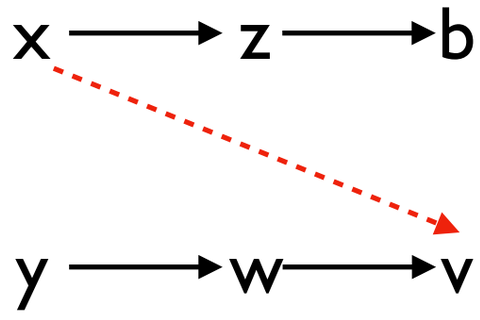
Constraint type	Assignment	Constraint	Meaning
Base	a = &b	$a \supseteq \{b\}$	$loc(b) \in pts(a)$
Simple	a = b	$a \supseteq b$	$pts(a) \supseteq pts(b)$
Complex	a = *b	$a \supseteq *b$	$\forall v \in pts(b).\ pts(a) \supseteq pts(v)$
Complex	*a = b	$*a \supseteq b$	$\forall v \in pts(a).\ pts(v) \supseteq pts(b)$

L. Andersen. Program Analysis and Specialization for the C Programming Language. PhD thesis, University of Copenhagen, 1994

Andersen pointer analysis by example



x = y



x = *y

Fixed-point computation: worklist algorithm of complexity O(n^3)

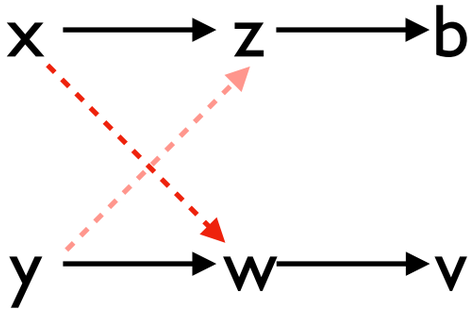
Steensgaard pointer analysis

- View pointer assignments as **equality** constraints
- Use constraints to propagate points-to information
- Almost linear time

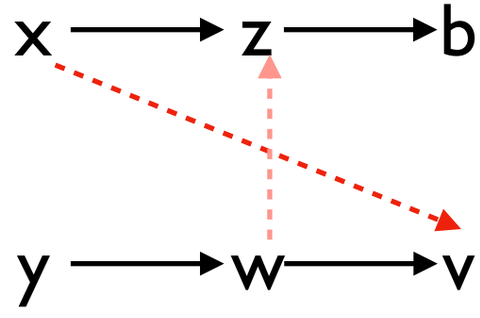
Constraint type	Assignment	Constraint	Meaning
Base	a = &b	a ⊇ {b}	loc(b) ∈ pts(a)
Simple	a = b	a = b	pts(a) = pts(b)
Complex	a = *b	a = *b	∀v∈pts(b). pts(a) = pts(v)
Complex	*a = b	*a = b	∀v∈pts(a). pts(v) = pts(b)

Bjarne Steensgaard, points-to analysis in almost linear time, POPL'96

Steensgaard pointer analysis by example



x = y



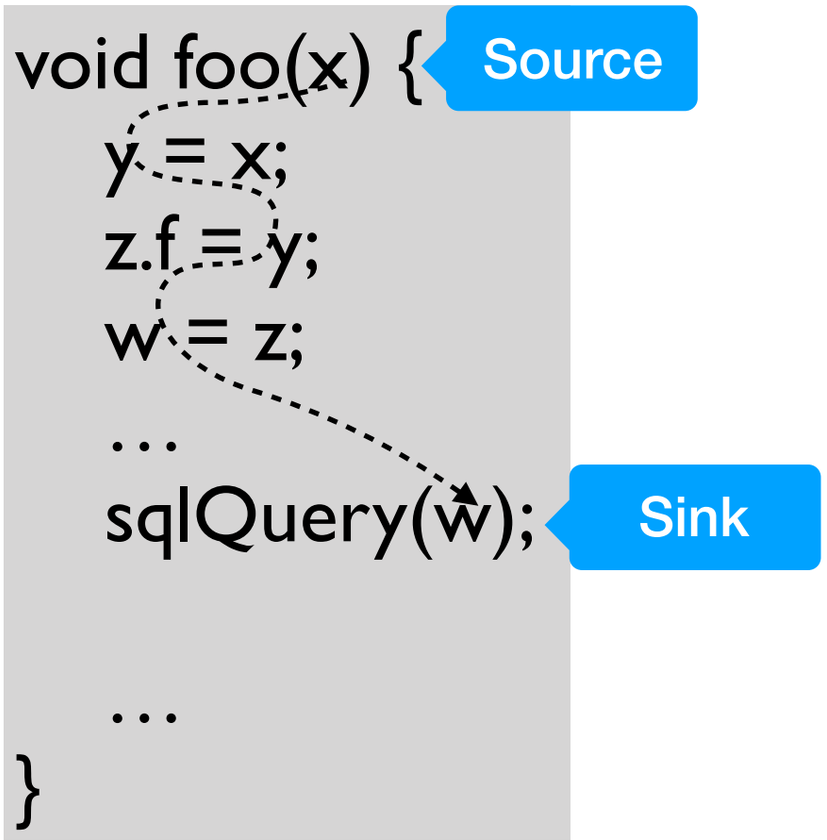
x = *y

Almost linear time!

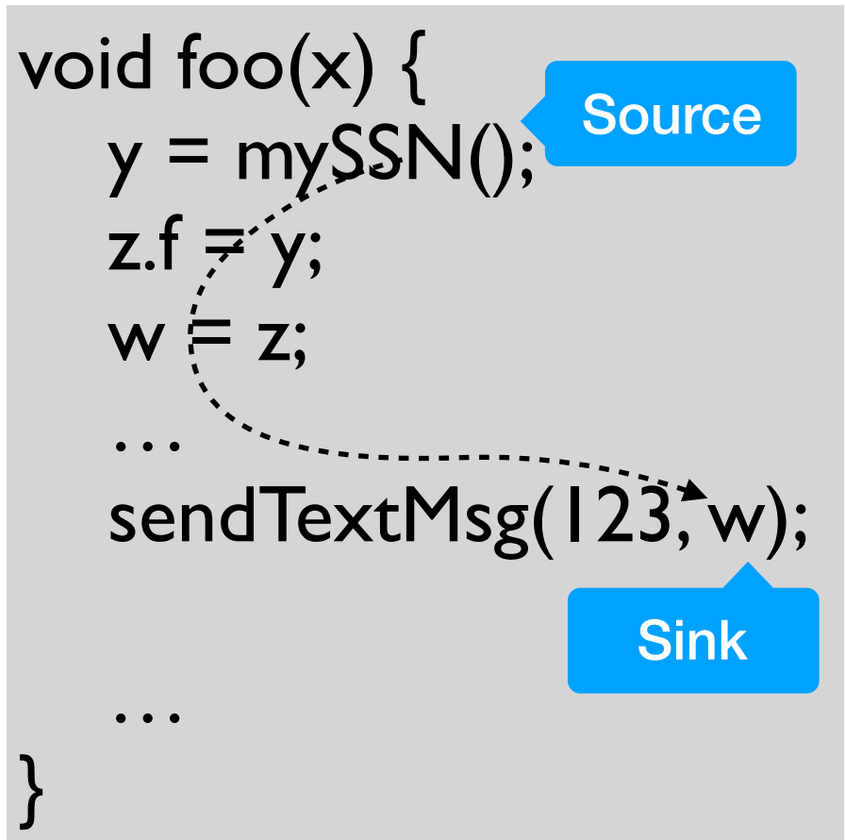
Pointer analysis in practice

- Complex data types (Object, structs)
- Function calls
- Loops and recursion
- Control-flow (if-else)
- Third-party libraries
- Class hierarchy

Information flow (taint) analysis



SQL injection



Malware analysis

Taint analysis in Datalog

- v = source(): v is tainted by a sensitive API called “source”
- v = w: if w is tainted, then v is also tainted.
- v.f = q: if q is tainted, then for all objects w pointed by v, w.f is also tainted
- sink(…, x): the value held by x is leaked via untrusted method “sink”
- flow(x, y): there is a taint flow from source x to sink y.