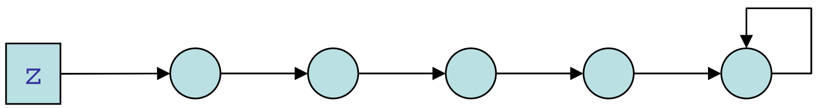
Shape Analysis

Static Analysis 2009

Michael I. Schwartzbach
Computer Science, University of Aarhus

Looking Into The Heap

- A non-trivial heap structure:



- Deciding *disjointness* of data structures:
  - `x` and `y` are not disjoint
  - `y` and `z` are disjoint

Shape Graphs

- Graphs that describe possible heaps:
  - nodes are pointer targets
  - edges are *possible* pointer references
- The lattice of shape graphs is:

  $$2^{Targets \times Targets}$$

  ordered under subset inclusion
- For every CFG node, v, we introduce a constraint variable [[v]] describing the heap *after* v

Shape Constraints

- For pointer operations:
  - id = `malloc`: $[[v]] = JOIN(v)\downarrow id \cup \{ (\&id, \texttt{malloc-}i) \}$
  - $id_1 = \&id_2$: $[[v]] = JOIN(v)\downarrow id_1 \cup \{ (\&id_1, \&id_2) \}$
  - $id_1 = id_2$: $[[v]] = assign(JOIN(v), id_1, id_2)$
  - $id_1 = *id_2$: $[[v]] = right(JOIN(v), id_1, id_2)$
  - $*id_1 = id_2$: $[[v]] = left(JOIN(v), id_1, id_2)$
  - id = `null`: $[[v]] = JOIN(v)\downarrow id$
- For all other CFG nodes:
  - $[[v]] = JOIN(v)$

Auxiliary Functions

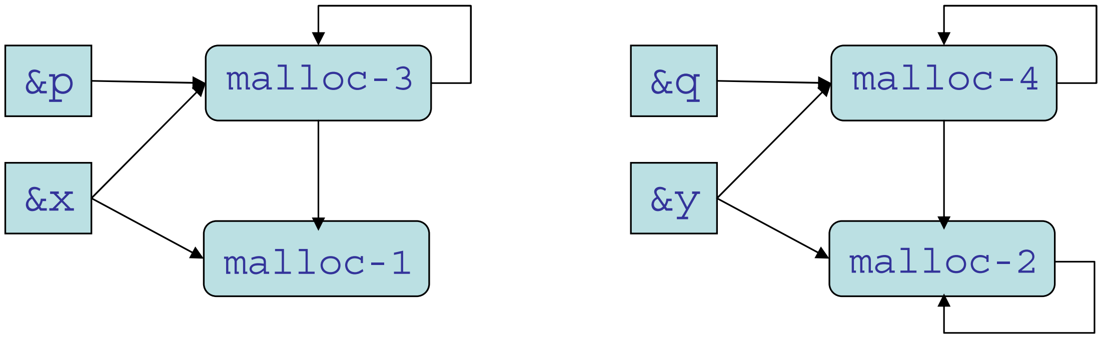
- $JOIN(v) = \bigcup_{w \in pred(v)} [[w]]$
- $\sigma \downarrow x = \{ (s,t) \in \sigma \mid s \neq \&x \}$
- $assign(\sigma,x,y) = \sigma \downarrow x \cup \bigcup_{(\&y,t) \in \sigma} \{ (\&x,t) \}$
- $right(\sigma,x,y) = \sigma \downarrow x \cup \bigcup_{(\&y,s),(s,t) \in \sigma} \{ (\&x,t) \}$
- $left(\sigma,x,y) = \begin{cases} \sigma & \{s \mid (\&x,s) \in \sigma\} = \emptyset \\ \bigcup_{(\&x,s) \in \sigma} \sigma \downarrow s & \{s \mid (\&x,s) \in \sigma\} \neq \emptyset \wedge \{t \mid (\&y,t) \in \sigma\} = \emptyset \\ \bigcup_{(\&x,s),(\&y,t) \in \sigma} \sigma \downarrow s \cup \{(s,t)\} & otherwise \end{cases}$

Example Program

```
var x,y,n,p,q;
x = malloc; y = malloc;
*x = null; *y = y;
n = input;
while (n>0) {
  p = malloc; q = malloc;
  *p = x; *q = y;
  x = p; y = q;
  n = n-1;
}
```

Result of Shape Analysis

- After the loop we have the shape graph:



- We conclude that `x` and `y` will always be disjoint

Points-To Maps From Shape Graphs

- A shape analysis is also a flow-sensitive points-to analysis, since:

  $$pt(\mathtt{p}) = \{ t \mid (\&\mathtt{p}, t) \in [[v]] \}$$

  is a points-to map for each program point v

- More expensive, but more precise:
  - Andersen: $pt(\mathtt{x}) = \{ \&\mathtt{y}, \&\mathtt{z} \}$
  - Shape analysis: $pt(\mathtt{x}) = \{ \&\mathtt{z} \}$

```
x = &y;
x = &z;
```

- This may even be iterated...

Better Shape Analysis

- The shape graph is missing information:
  - `malloc-2` nodes always form a self-loop

- To conclude this, we need a more detailed lattice:

$$2^{Targets \times Targets} \times 2^{Targets} \times 2^{Targets}$$

  where for an element (X,Y,Z) we have:
  - X denotes the *possible* edges
  - Y denotes the targets that have been allocated
  - $Z \subseteq Y$ denotes the targets that have been *uniquely* allocated

Constraints

- Assume *JOIN*(v) = (X,Y,Z)
- The assignment *id* = `malloc` has the constraint:

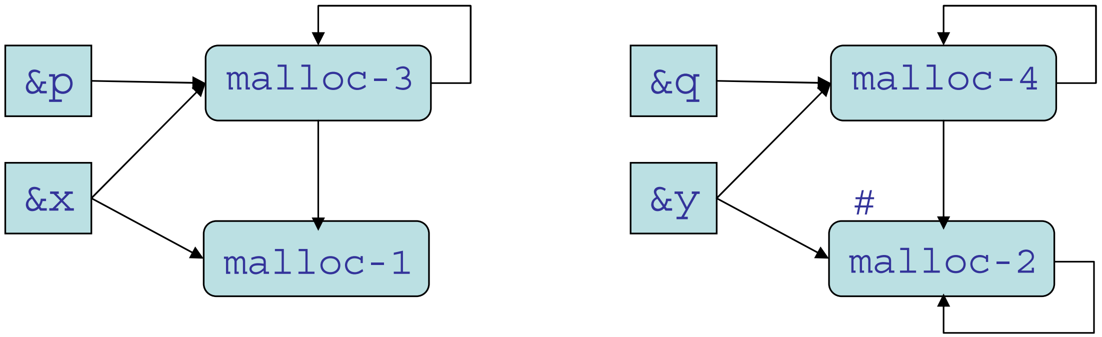
$$[[v]] = (X{\downarrow}id \cup \{(\&id,\texttt{malloc-}i)\},\; Y{\downarrow}id \cup \{\texttt{malloc-}i\},\; unique(Y,Z,\texttt{malloc-}i))$$

where we have the auxiliary function:

$$unique(Y,Z,t) = \begin{cases} Z \cup \{t\} & \text{if } t \notin Y \\ Z \setminus \{t\} & \text{otherwise} \end{cases}$$

Better Results

- After the loop we have the shape graph:



- Here, # means that the target is unique
- Thus, `malloc-2` nodes form a self-loop

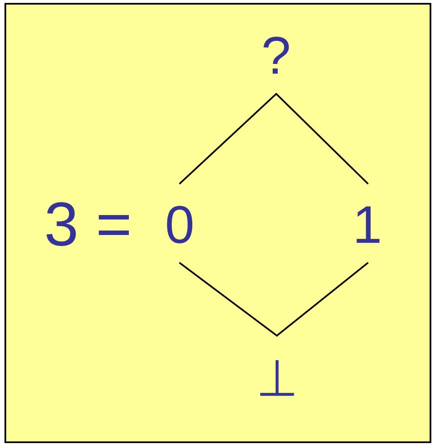
Parametric Shape Analysis (1/3)

- A less ad-hoc approach to analyzing heaps
- Characterize targets by a collection of unary *instrumentation predicates*:
  - does this node have two or more incoming pointers?
  - is this nodes reachable from the variable `x`?
  - is this node on a cycle?
- Shape graph nodes are polyvariant:
  - one copy for each 3-valued interpretation of predicates
  - nodes correspond to $3^{Targets} \times 3^{Targets} \times \ldots \times 3^{Targets}$

Parametric Shape Analysis (2/3)

- The shape graph itself is then:

$$3^{(3^{Targets} \times 3^{Targets} \times \ldots \times 3^{Targets})^2}$$



or amusingly:

$$3^{((3^{Targets})^k)^2}$$

if we have *k* instrumentation predicates.

Parametric Shape Analysis (3/3)

- Constraints must now update all information:
  - quite a heavy burden
  - a puzzle to get everything right
  - some automatic support is possible
- A powerful technique:
  - verify that insert operation on red-black search trees maintain the data structure invariant

Escape Analysis

- Perform the simple shape analysis
- Look at return expression
- Check reachability in the shape graph to arguments or variables defined in the function itself

- None of those

  $\Downarrow$

  no escaping stack cells

```
baz() {
  var x;
  return &x;
}

main() {
  var p;
  p=baz();
  *p=1;
  return *p;
}
```