

Parametric Shape Analysis via 3-Valued Logic

Chenguang Sun
sun47@purdue.edu

Previously on “Points-to” Analysis

- “Our method computes the points-to relationships between **stack** locations” (Page 242)
- “In the case of stack-based aliases a name exists for each stack location of interest.” (Page 243)
- “There are no natural names for each location (in heap)” (Page 243)
- “We use a single location called *heap* in our abstract stack for the points-to analysis.” (Page 254)

Previously on “Points-to” Analysis

- “The stack and heap problems can and should be separated.” (Page 254)

Sample Program

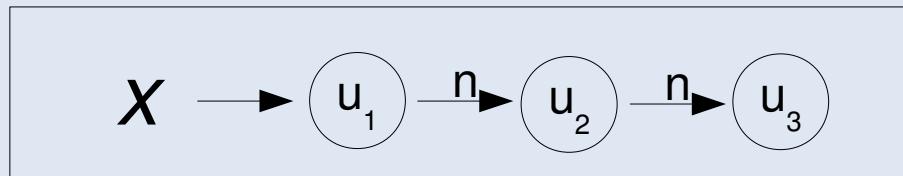
```
/* insert.c */
#include "list.h"
void insert(List x, int d) {
    List y, t, e;
    assert(acyclic_list(x) && x != NULL);
    y = x;
    while (y->n != NULL && ...) {
        y = y->n;
    }
    t = malloc();
    t->data = d;
    e = y->n;
    t->n = e;
    y->n = t;
}
```

(a)

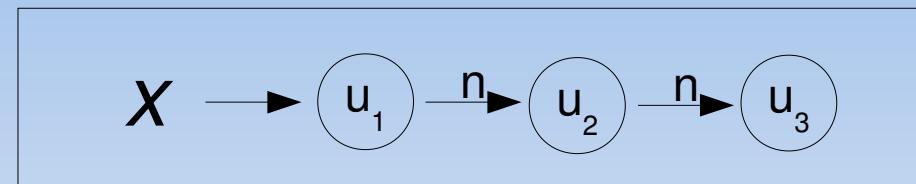
(b)

Representing Store via Graph

- A “store” is the memory state that arise at a given point in the program.
- In the graph
 - x : Variable
 - n : Field n of a node
 - u_i : Node

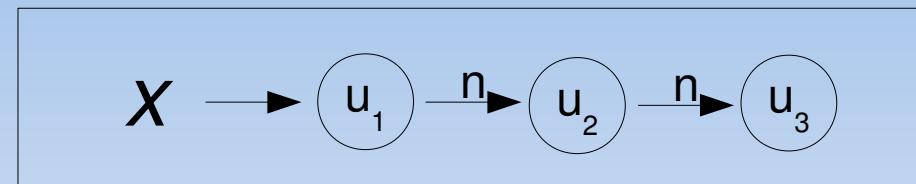


Representing Concrete Stores via First Order Logic

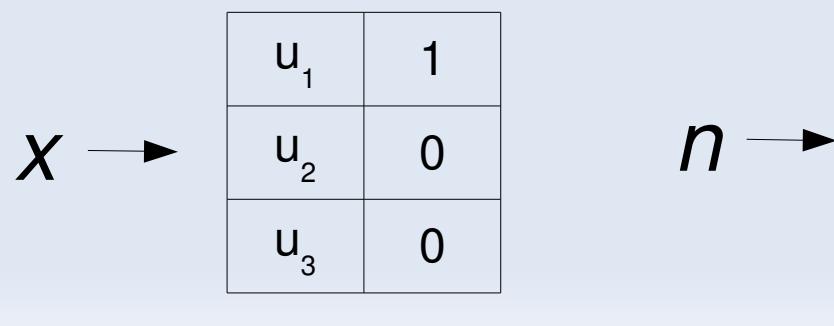


- Predicates
 - “pointed-to-by-variable” (Unary)
 - Pointers from stack into the heap
 - Example: x, y, t, e
 - “pointer-component-points-to” (Binary)
 - Pointer-valued fields of data structures
 - Example: n

Representing Concrete Stores via First Order Logic



- Logical structure $S = \langle U^S, \iota^S \rangle$
 - U^S : Universe of individuals
 - In this example, individuals are nodes
 - Example: u_1, u_2, u_3
 - ι^S : arity-k Predicates $\rightarrow (\text{Universe}^k \rightarrow \{0, 1\})$
 - Example:



	u_1	u_2	u_3
u_1	0	1	0
u_2	0	0	1
u_3	0	0	0

Name	Logical Structure					Graphical Representation																																																		
S_0^\natural	unary preds. binary preds. <table border="1"> <tr> <td>indiv.</td> <td>x</td> <td>y</td> <td>t</td> <td>e</td> <td>n</td> </tr> </table>					indiv.	x	y	t	e	n																																													
indiv.	x	y	t	e	n																																																			
S_1^\natural	unary preds. binary preds. <table border="1"> <tr> <td>indiv.</td> <td>x</td> <td>y</td> <td>t</td> <td>e</td> <td>n</td> <td>u_1</td> </tr> <tr> <td>u_1</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>u_1</td> <td>0</td> </tr> </table>					indiv.	x	y	t	e	n	u_1	u_1	1	0	0	0	u_1	0	$x \Rightarrow u_1$																																				
indiv.	x	y	t	e	n	u_1																																																		
u_1	1	0	0	0	u_1	0																																																		
S_2^\natural	unary preds. binary preds. <table border="1"> <tr> <td>indiv.</td> <td>x</td> <td>y</td> <td>t</td> <td>e</td> <td>n</td> <td>u_1</td> <td>u_2</td> </tr> <tr> <td>u_1</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>u_1</td> <td>0</td> <td>1</td> </tr> <tr> <td>u_2</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>u_2</td> <td>0</td> <td>0</td> </tr> </table>					indiv.	x	y	t	e	n	u_1	u_2	u_1	1	0	0	0	u_1	0	1	u_2	0	0	0	0	u_2	0	0	$x \Rightarrow u_1 \xrightarrow{n} u_2$																										
indiv.	x	y	t	e	n	u_1	u_2																																																	
u_1	1	0	0	0	u_1	0	1																																																	
u_2	0	0	0	0	u_2	0	0																																																	
S_3^\natural	unary preds. binary preds. <table border="1"> <tr> <td>indiv.</td> <td>x</td> <td>y</td> <td>t</td> <td>e</td> <td>n</td> <td>u_1</td> <td>u_2</td> <td>u_3</td> </tr> <tr> <td>u_1</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>u_1</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>u_2</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>u_2</td> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>u_3</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>u_3</td> <td>0</td> <td>0</td> <td>0</td> </tr> </table>					indiv.	x	y	t	e	n	u_1	u_2	u_3	u_1	1	0	0	0	u_1	0	1	0	u_2	0	0	0	0	u_2	0	0	1	u_3	0	0	0	0	u_3	0	0	0	$x \Rightarrow u_1 \xrightarrow{n} u_2 \xrightarrow{n} u_3$														
indiv.	x	y	t	e	n	u_1	u_2	u_3																																																
u_1	1	0	0	0	u_1	0	1	0																																																
u_2	0	0	0	0	u_2	0	0	1																																																
u_3	0	0	0	0	u_3	0	0	0																																																
S_4^\natural	unary preds. binary preds. <table border="1"> <tr> <td>indiv.</td> <td>x</td> <td>y</td> <td>t</td> <td>e</td> <td>n</td> <td>u_1</td> <td>u_2</td> <td>u_3</td> <td>u_4</td> </tr> <tr> <td>u_1</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>u_1</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>u_2</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>u_2</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>u_3</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>u_3</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>u_4</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>u_4</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </table>					indiv.	x	y	t	e	n	u_1	u_2	u_3	u_4	u_1	1	0	0	0	u_1	0	1	0	0	u_2	0	0	0	0	u_2	0	0	1	0	u_3	0	0	0	0	u_3	0	0	0	1	u_4	0	0	0	0	u_4	0	0	0	0	$x \Rightarrow u_1 \xrightarrow{n} u_2 \xrightarrow{n} u_3 \xrightarrow{n} u_4$
indiv.	x	y	t	e	n	u_1	u_2	u_3	u_4																																															
u_1	1	0	0	0	u_1	0	1	0	0																																															
u_2	0	0	0	0	u_2	0	0	1	0																																															
u_3	0	0	0	0	u_3	0	0	0	1																																															
u_4	0	0	0	0	u_4	0	0	0	0																																															

There are infinite structures. We need a way to abstract.

Canonical Abstraction

- We consider unary predicates only. Since
 - $x(u_2) = x(u_3) = x(u_4)$
 - $y(u_2) = y(u_3) = y(u_4)$
 - $t(u_2) = t(u_3) = t(u_4)$
 - $e(u_2) = e(u_3) = e(u_4)$
- u_2, u_3, u_4 can be abstracted as one summary node u_{234}

unary preds.

indiv.	x	y	t	e
u_1	1	1	0	0
u_2	0	0	0	0
u_3	0	0	0	0
u_4	0	0	0	0

binary preds.

n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0

$x \rightarrow u_1 \xrightarrow{n} u_2 \xrightarrow{n} u_3 \xrightarrow{n} u_4$

\uparrow

S_a^{\natural}

Canonical Abstraction

indiv.	<i>x</i>	<i>y</i>	<i>t</i>	<i>e</i>
u_1	1	1	0	0
u_2	0	0	0	0
u_3	0	0	0	0
u_4	0	0	0	0

Merge $u_2 u_3 u_4$

indiv.	<i>x</i>	<i>y</i>	<i>t</i>	<i>e</i>
u_1	1	1	0	0
u_{234}	0	0	0	0

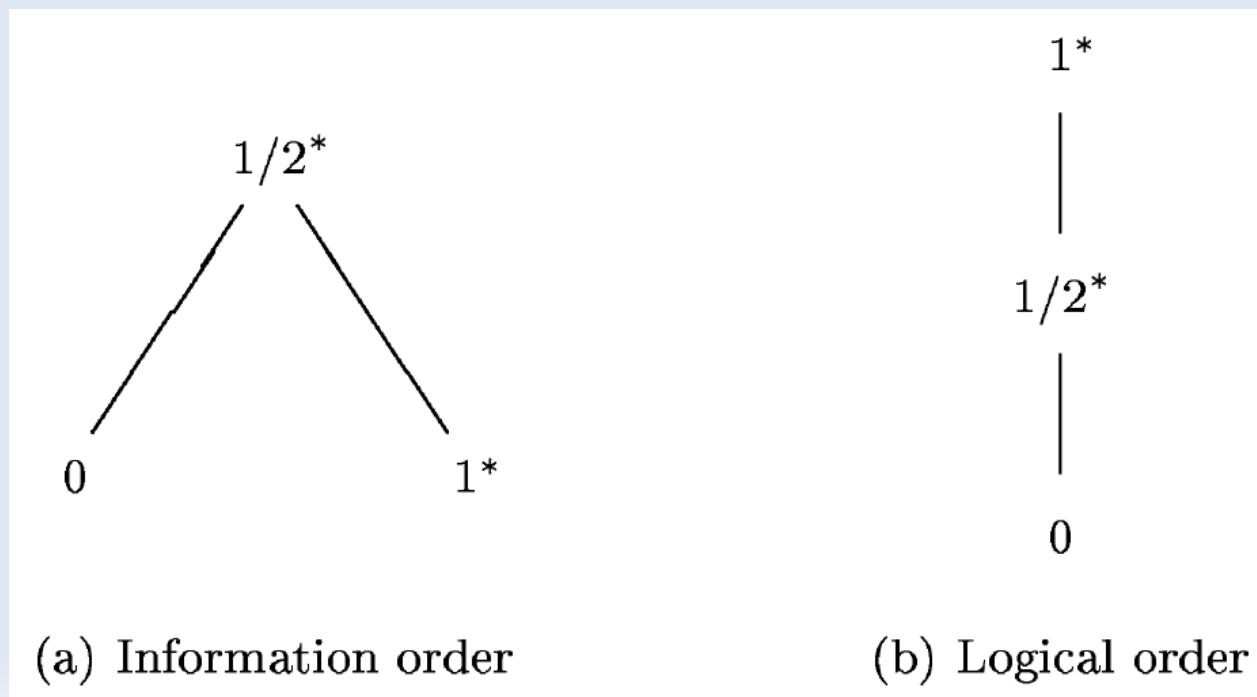
<i>n</i>	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0

Merge $u_2 u_3 u_4$

?

Kleene's Three-Valued Logic

- One more logical literal $\frac{1}{2}$
 - 0 and 1 are definite values;
 - $\frac{1}{2}$ means “unknown” which is a indefinite value.



Kleene's Three-Valued Logic

- $l_1 \sqsubseteq l_2$ denotes that l_1 has more definite information than l_2 ;
- \sqcup denotes least-upper-bound with respect to \sqsubseteq
 - $\sqcup\{0, 1\} = \frac{1}{2}$

Kleene's Three-Valued Logic

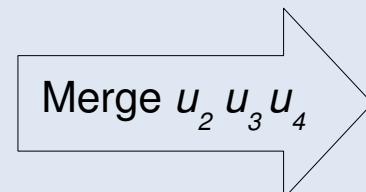
\wedge	0	1	1/2
0	0	0	0
1	0	1	1/2
1/2	0	1/2	1/2

\vee	0	1	1/2
0	0	1	1/2
1	1	1	1
1/2	1/2	1	1/2

\neg	
0	1
1	0
1/2	1/2

Canonical Abstraction

n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0



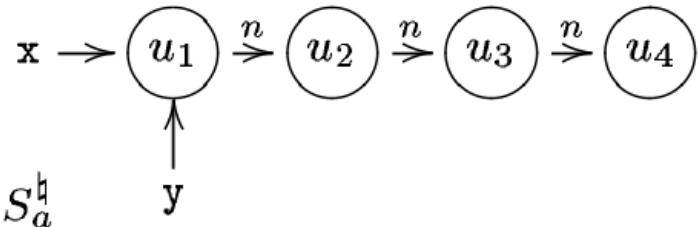
n	u_1	u_{234}
u_1	0	1/2
u_{234}	0	1/2

Canonical Abstraction

- An additional unary predicate, called *sm* (standing for “summary”) is added to capture whether a node is abstract.
 - $sm(\text{concrete node}) = 0$
 - $sm(\text{abstract node}) = \frac{1}{2}$
- *sm* is not an abstraction predicate
- $\llbracket v_1 = v_2 \rrbracket_3^S(Z) = \begin{cases} 0 & Z(v_1) \neq Z(v_2) \\ 1 & Z(v_1) = Z(v_2) \text{ and } \iota^S(sm)(Z(v_1)) = 0 \\ 1/2 & \text{otherwise} \end{cases}$

unary preds.

indiv.	x	y	t	e
u_1	1	1	0	0
u_2	0	0	0	0
u_3	0	0	0	0
u_4	0	0	0	0



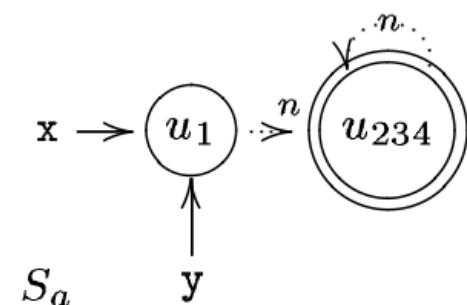
binary preds.

n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0

abstracts
to

unary preds.

indiv.	x	y	t	e	sm
u_1	1	1	0	0	0
u_{234}	0	0	0	0	1/2



binary preds.

n	u_1	u_{234}
u_1	0	1/2
u_{234}	0	1/2

The Meaning of Program Statements

- Predicate-update formula
 - For every statement st , the new values of every predicate p are defined via a predicate-update formula (φ_p^{st}).

st	φ_p^{st}
<code>x = NULL</code>	$\varphi_x^{st}(v) \stackrel{\text{def}}{=} \mathbf{0}$
<code>x = t</code>	$\varphi_x^{st}(v) \stackrel{\text{def}}{=} t(v)$
<code>x = t->sel</code>	$\varphi_x^{st}(v) \stackrel{\text{def}}{=} \exists v_1 : t(v_1) \wedge sel(v_1, v)$
<code>x->sel = NULL</code>	$\varphi_{sel}^{st}(v_1, v_2) \stackrel{\text{def}}{=} sel(v_1, v_2) \wedge \neg x(v_1)$
<code>x->sel = t</code> (assuming that <code>x->sel == NULL</code>)	$\varphi_{sel}^{st}(v_1, v_2) \stackrel{\text{def}}{=} sel(v_1, v_2) \vee (x(v_1) \wedge t(v_2))$
<code>x = malloc()</code>	$\varphi_x^{st}(v) \stackrel{\text{def}}{=} isNew(v)$ $\varphi_z^{st}(v) \stackrel{\text{def}}{=} z(v) \wedge \neg isNew(v), \text{ for each } z \in (PVar - \{x\})$ $\varphi_{sel}^{st}(v_1, v_2) \stackrel{\text{def}}{=} sel(v_1, v_2) \wedge \neg isNew(v_1) \wedge \neg isNew(v_2) \text{ for each } sel \in PSel$

The Meaning of Program Statements

- Structure transformer

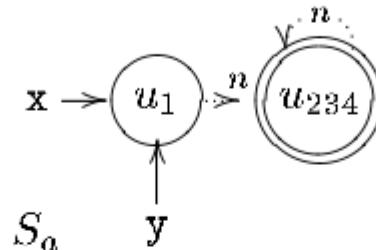
$$\llbracket st \rrbracket(S) = \langle U^S, \lambda p. \lambda u_1, \dots, u_k. \llbracket \varphi_p^{st} \rrbracket_2^S([v_1 \mapsto u_1, \dots, v_k \mapsto u_k]) \rangle$$

Each Statement st Is A Transformer of S

- When st is not $malloc()$
 - U^S unchanged
 - $\iota^S(p) = \varphi_p^{st}$
- When st is $malloc()$
 - $U^S = U^S \cup \{u_{new}\}$
 - $\iota^S(p) = \varphi_p^{st}$

unary preds. **binary preds.**

indiv.	x	y	t	e	sm	n	u_1	u_{234}
u_1	1	1	0	0	0	u_1	0	$1/2$
u_{234}	0	0	0	0	$1/2$	u_{234}	0	$1/2$



$$y = y \rightarrow n$$

$$x'(v) = x(v)$$

$$y'(v) = \exists v_1 : y(v_1) \wedge n(v_1, v)$$

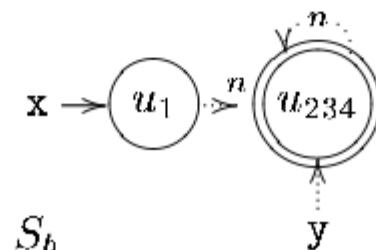
$$t'(v) = t(v)$$

$$e'(v) = t(v)$$

$$n'(v_1, v_2) = n(v_1, v_2)$$

unary preds. **binary preds.**

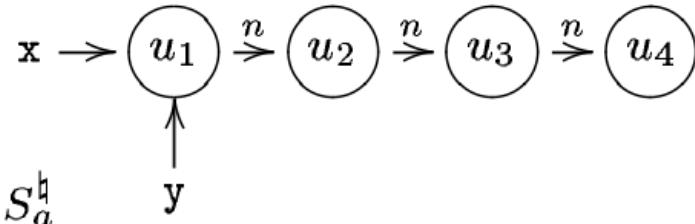
indiv.	x	y	t	e	sm	n	u_1	u_{234}
u_1	1	0	0	0	0	u_1	0	$1/2$
u_{234}	0	$1/2$	0	0	$1/2$	u_{234}	0	$1/2$



Is S_a acyclic?

unary preds.

indiv.	x	y	t	e
u_1	1	1	0	0
u_2	0	0	0	0
u_3	0	0	0	0
u_4	0	0	0	0



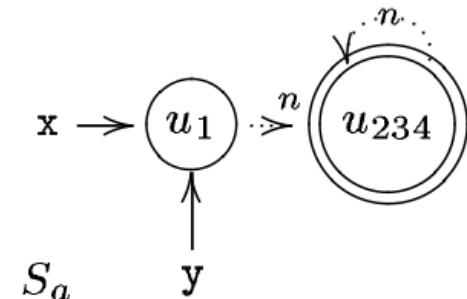
binary preds.

n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0

abstracts
to

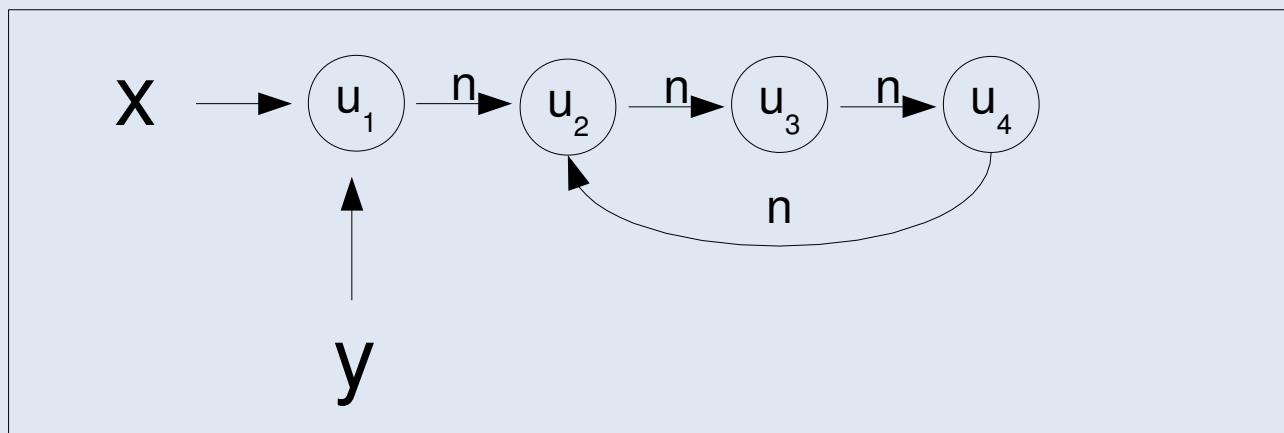
unary preds.

indiv.	x	y	t	e	sm
u_1	1	1	0	0	0
u_{234}	0	0	0	0	1/2



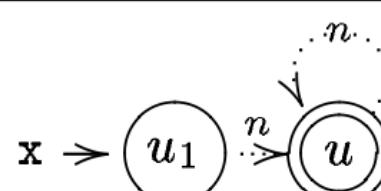
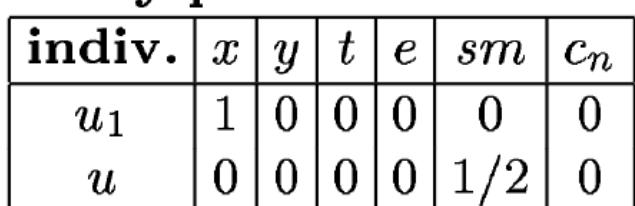
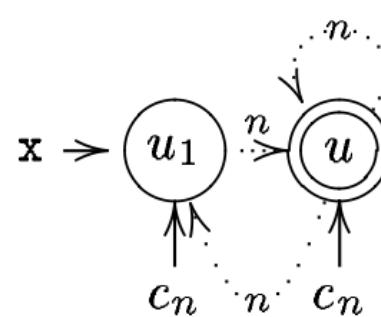
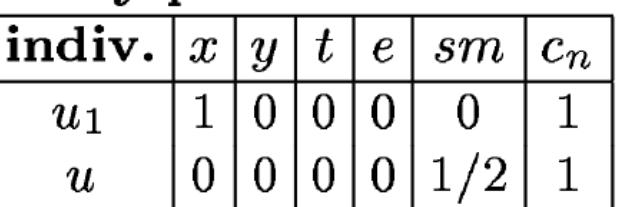
binary preds.

n	u_1	u_{234}
u_1	0	1/2
u_{234}	0	1/2



Instrumentation Predicates

- Solution
 - Add another predicate c_n . $c_n(u)$ is 1 when there is a path along n fields from u to u itself, otherwise 0.
 - Use c_n as an additional abstraction predicate.

Name	Logical Structure						Graphical Representation
$S_{acyclic}$	unary preds.			binary preds.			
	indiv.			x	y	t	
	u_1	1	0	0	0	0	
S_{cyclic}	unary preds.			binary preds.			
	indiv.			x	y	t	
	u_1	1	0	0	0	0	

Predicate-Update Formula for C_n

st	$\varphi_{c_n}^{st}(v)$
$x = \text{NULL}$	$c_n(v)$
$x = t$	$c_n(v)$
$x = t \rightarrow n$	$c_n(v)$
$x \rightarrow n = \text{NULL}$	$c_n(v) \wedge \neg(\exists v' : x(v') \wedge c_n(v') \wedge r_{x,n}(v))$
$x \rightarrow n = t$ (assuming that $x \rightarrow n == \text{NULL}$)	$c_n(v) \vee \exists v' : x(v') \wedge r_{t,n}(v') \wedge r_{t,n}(v)$
$x = \text{malloc}()$	$c_n(v) \wedge \neg new(v)$

Other Instrumentation Predicates

Pred.	Intended Meaning	Purpose	Ref.
$is(v)$	Do two or more fields of heap elements point to v ?	lists and trees	[Chase et al. 1990], [Sagiv et al. 1998]
$r_{x,n}(v)$	Is v (transitively) reachable from pointer variable x along n fields?	separating disjoint data structures	[Sagiv et al. 1998]
$r_n(v)$	Is v reachable from some pointer variable along n fields (i.e., is v a non-garbage element)?	compile-time garbage collection	
$c_n(v)$	Is v on a directed cycle of n fields?	reference counting	[Jones and Muchnick 1981]
$c_{f.b}(v)$	Does a field- f deref. from v , followed by a field- b deref., yield v ?	doubly-linked lists	[Hendren et al. 1992], [Plevyak et al. 1993]
$c_{b.f}(v)$	Does a field- b deref. from v , followed by a field- f deref., yield v ?	doubly-linked lists	[Hendren et al. 1992], [Plevyak et al. 1993]

Other Instrumentation Predicates

$$\varphi_{is}(v) \stackrel{\text{def}}{=} \exists v_1, v_2 : n(v_1, v) \wedge n(v_2, v) \wedge v_1 \neq v_2$$

$$\varphi_{r_{x,n}}(v) \stackrel{\text{def}}{=} x(v) \vee \exists v_1 : x(v_1) \wedge n^+(v_1, v)$$

$$\varphi_{r_n}(v) \stackrel{\text{def}}{=} \bigvee_{\mathbf{x} \in PVar} (x(v) \vee \exists v_1 : x(v_1) \wedge n^+(v_1, v))$$

$$\varphi_{c_n}(v) \stackrel{\text{def}}{=} n^+(v, v)$$

$$\varphi_{c_{f.b}}(v) \stackrel{\text{def}}{=} \forall v_1 : f(v, v_1) \Rightarrow b(v_1, v)$$

$$\varphi_{c_{b.f}}(v) \stackrel{\text{def}}{=} \forall v_1 : b(v, v_1) \Rightarrow f(v_1, v)$$

Predicate-Update Formula for $r_{z,n}$

st	cond.	$\varphi_{r_z,n}^{st}(v)$
$x = \text{NULL}$	$z \equiv x$ $z \not\equiv x$	0 $r_{z,n}(v)$
$x = t$	$z \equiv x$ $z \not\equiv x$	$r_{t,n}(v)$ $r_{z,n}(v)$
$x = t \rightarrow n$	$z \equiv x$ $z \not\equiv x$	$r_{t,n}(v) \wedge (c_n(v) \vee \neg t(v))$ $r_{z,n}(v)$
$x \rightarrow n = \text{NULL}$	$z \equiv x$ $z \not\equiv x$	$x(v)$ $\begin{cases} \varphi_{r_z,n}[n \mapsto \varphi_n^{st}] & \text{if } c_n(v) \wedge r_{x,n}(v) \\ r_{z,n}(v) \wedge \neg(\exists v' : r_{z,n}(v') \wedge x(v') \wedge r_{x,n}(v) \wedge \neg x(v)) & \text{otherwise} \end{cases}$
$x \rightarrow n = t$ (assuming that $x \rightarrow n == \text{NULL}$)		$r_{z,n}(v) \vee (\exists v' : r_{z,n}(v') \wedge x(v') \wedge r_{t,n}(v))$
$x = \text{malloc()}$	$z \equiv x$ $z \not\equiv x$	$new(v)$ $r_{z,n}(v) \wedge \neg new(v)$

Predicate-Update Formula for Instrumentation Predicates

Definition 4.6 A predicate-update formula φ_p^{st} **maintains a correct instrumentation for predicate** $p \in \mathcal{I}$ if, for all $S^\natural \in 2\text{-CSTRUCT}[\mathcal{P}, F]$ and for all Z ,

$$\llbracket \varphi_p^{st} \rrbracket_3^{S^\natural}(Z) = \llbracket \varphi_p \rrbracket_3^{\llbracket st \rrbracket(S^\natural)}(Z). \quad (23)$$

Instrumentation Predicates

- Speed and Accuracy
 - More instrumentation-predicates;
 - More information (more accurate);
 - More abstraction nodes (slower to process);

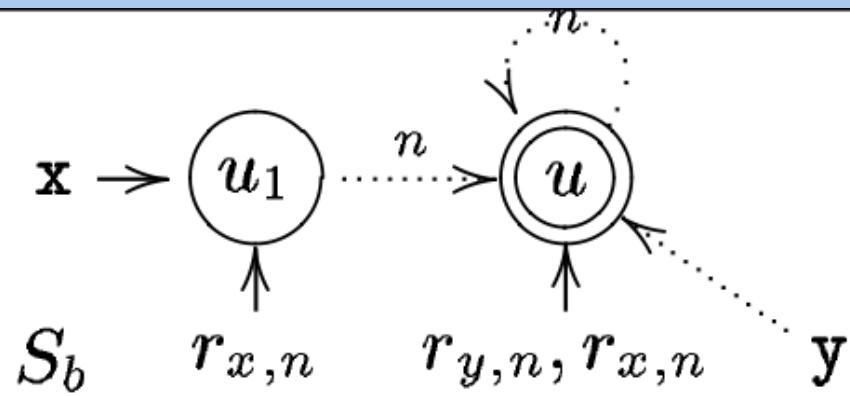
Improve Abstract Semantics

input structure	$x, y \rightarrow u_1$	$\xrightarrow{n} u$
	S_a	$r_{x,n}, r_{y,n}$
update formulae	$\varphi_y^{st_0}(v)$ $\exists v_1 : y(v_1) \wedge n(v_1, v)$	$\varphi_{r_{n,y}}^{st_0}(v)$ $r_{y,n}(v) \wedge (c_n(v) \vee \neg y(v))$
output structure	$x \rightarrow u_1$	$\xrightarrow{n} u$
	S_b	$r_{x,n}$
		$r_{y,n}, r_{x,n}$
		y

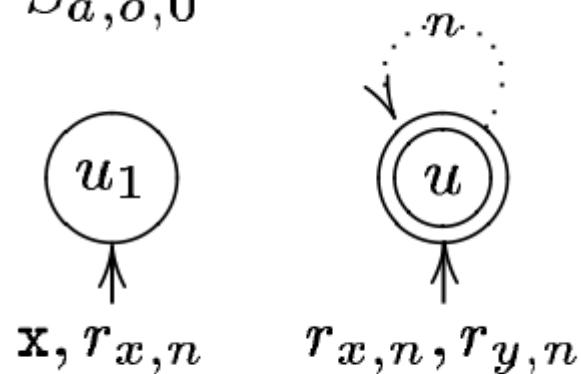
- New value of y becomes indefinite. $st_0: y = y \rightarrow n$

Impossible Structures That Could Be Represented By S_b

output
structure



$S_{a,o,0}$

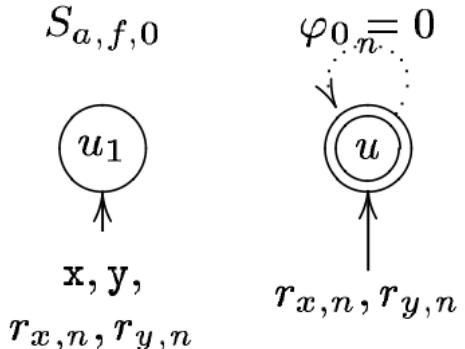
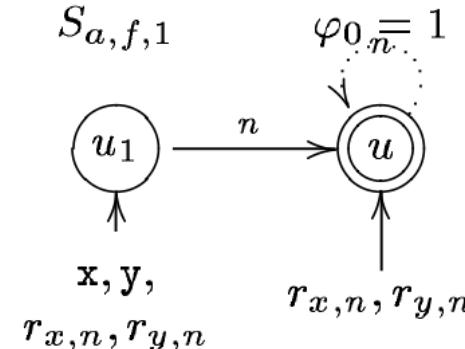
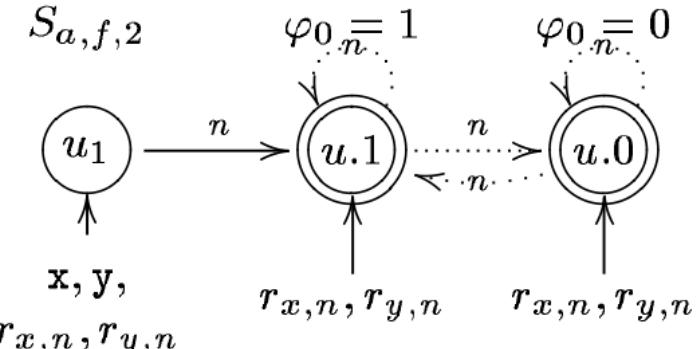
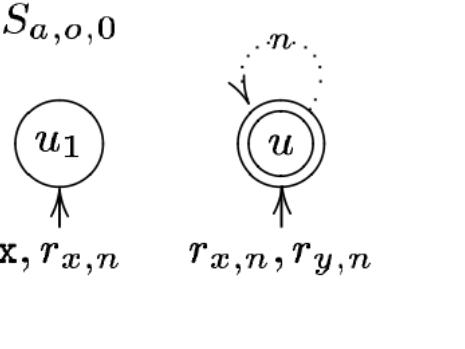
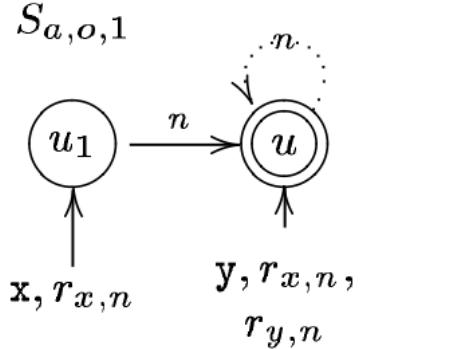
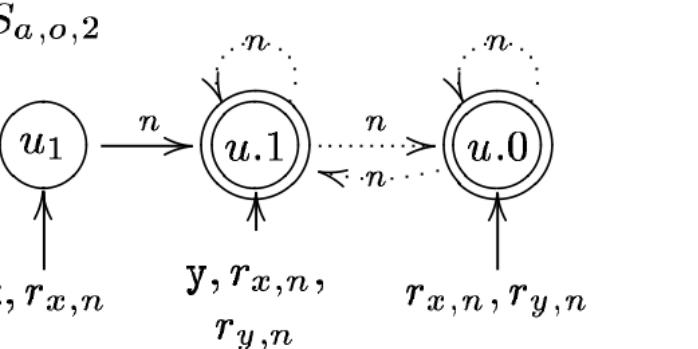


The Focus Operation

input structure	<p>$S_a \quad r_{x,n}, r_{y,n}$</p>					
focus formulae	$\{\varphi_0(v)\}$, where $\varphi_0(v) \stackrel{\text{def}}{=} \exists v_1 : y(v_1) \wedge n(v_1, v)$					
focused structures	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center; padding: 10px;"> $S_{a,f,0}$ <p>$x, y,$ $r_{x,n}, r_{y,n}$</p> </td><td style="text-align: center; padding: 10px;"> $\varphi_0 \equiv 0$ <p>$r_{x,n}, r_{y,n}$</p> </td><td style="text-align: center; padding: 10px;"> $S_{a,f,1}$ <p>$x, y,$ $r_{x,n}, r_{y,n}$</p> </td></tr> </table>			$S_{a,f,0}$ <p>$x, y,$ $r_{x,n}, r_{y,n}$</p>	$\varphi_0 \equiv 0$ <p>$r_{x,n}, r_{y,n}$</p>	$S_{a,f,1}$ <p>$x, y,$ $r_{x,n}, r_{y,n}$</p>
$S_{a,f,0}$ <p>$x, y,$ $r_{x,n}, r_{y,n}$</p>	$\varphi_0 \equiv 0$ <p>$r_{x,n}, r_{y,n}$</p>	$S_{a,f,1}$ <p>$x, y,$ $r_{x,n}, r_{y,n}$</p>				

- φ_0 is the predicate update formula for y
- Partition the set of structures represented by S_a to three subset of structures represented by $S_{a,f,0}$, $S_{a,f,1}$, and $S_{a,f,2}$ respectively, where φ_0 evaluates to definite values.

Structure Transformation

focused structures	$S_{a,f,0}$  $\varphi_{0,n} = 0$	$S_{a,f,1}$  $\varphi_{0,n} = 1$	$S_{a,f,2}$  $\varphi_{0,n} = 1$ $\varphi_{0,n} = 0$
update formulae		$\varphi_y^{st_0}(v)$ $\exists v_1 : y(v_1) \wedge n(v_1, v)$	$\varphi_{r_n,y}^{st_0}(v)$ $r_{y,n}(v) \wedge (c_n(v) \vee \neg y(v))$
output structures	$S_{a,o,0}$  $x, r_{x,n}$ $r_{x,n}, r_{y,n}$	$S_{a,o,1}$  $x, r_{x,n}$ $y, r_{x,n}, r_{y,n}$	$S_{a,o,2}$  $x, r_{x,n}$ $y, r_{x,n}, r_{y,n}$ $r_{x,n}, r_{y,n}$

- $st_0: y = y \rightarrow n$

Compatibility Constraints

- Constraints from the semantics of the programming language (*C language*)

$$(\exists v : sm(v)) \triangleright \mathbf{0}$$

for each $\mathbf{x} \in PVar, x(v_1) \wedge x(v_2) \triangleright v_1 = v_2$

$$(\exists v_3 : n(v_3, v_1) \wedge n(v_3, v_2)) \triangleright v_1 = v_2$$

Compatibility Constraints

- Constraints from the definitions of the instrumentation predicates

$$(\exists v_1, v_2 : n(v_1, v) \wedge n(v_2, v) \wedge v_1 \neq v_2) \triangleright is(v)$$

$$\neg(\exists v_1, v_2 : n(v_1, v) \wedge n(v_2, v) \wedge v_1 \neq v_2) \triangleright \neg is(v)$$

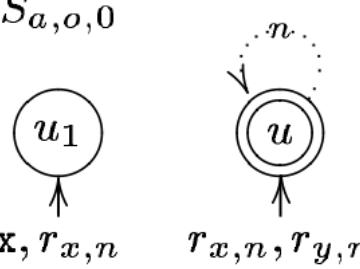
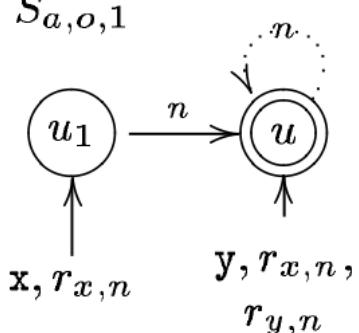
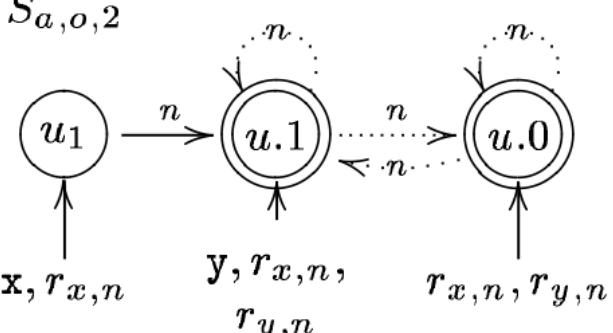
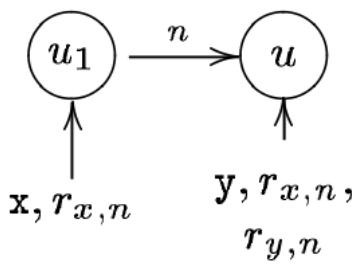
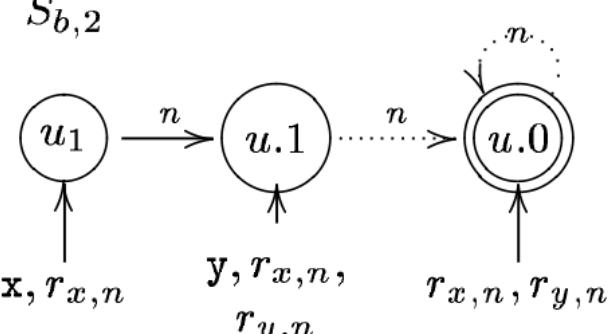
$$n^+(v, v) \triangleright c_n(v)$$

$$\neg n^+(v, v) \triangleright \neg c_n(v)$$

$$\text{for each } x \in PVar : x(v) \vee \exists v_1 : x(v_1) \wedge n^+(v_1, v) \triangleright r_{x,n}(v)$$

$$\text{for each } x \in PVar : \neg(x(v) \vee \exists v_1 : x(v_1) \wedge n^+(v_1, v)) \triangleright \neg r_{x,n}(v)$$

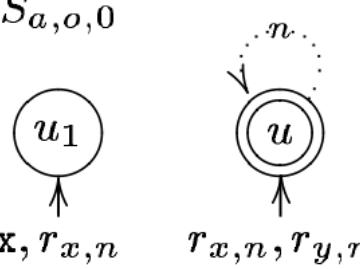
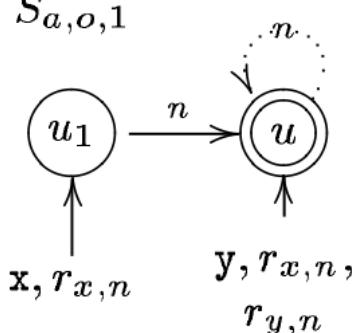
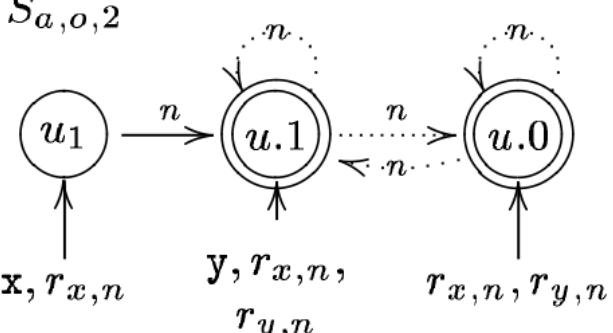
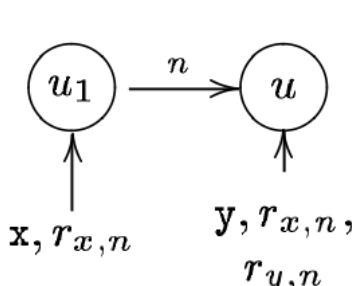
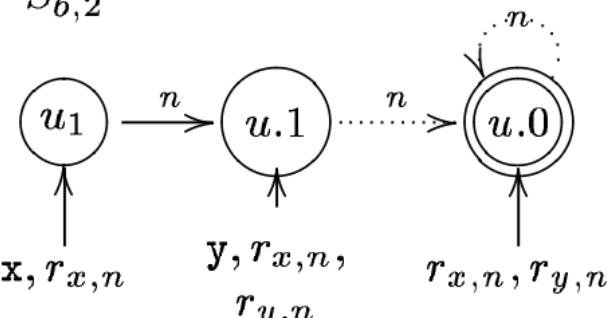
The Coerce Operation

output structures	$S_{a,o,0}$ 	$S_{a,o,1}$ 	$S_{a,o,2}$ 
coerced structures		$S_{b,1}$ 	$S_{b,2}$ 

- $S_{a,o,0}$ violates the constraint (irreparable):

for each $x \in PVar : \neg(x(v) \vee \exists v_1 : x(v_1) \wedge n^+(v_1, v)) \triangleright \neg r_{x,n}(v)$

The Coerce Operation

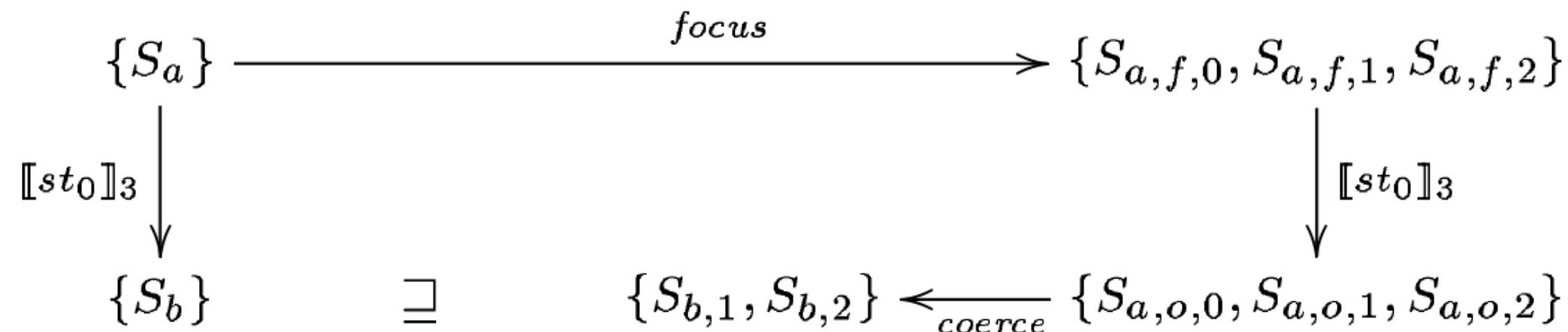
output structures	$S_{a,o,0}$ 	$S_{a,o,1}$ 	$S_{a,o,2}$ 
coerced structures		$S_{b,1}$ 	$S_{b,2}$ 

- $S_{a,o,1}$ and $S_{a,o,0}$ violate the constraints (fixable):

for each $x \in PVar, x(v_1) \wedge x(v_2) \triangleright v_1 = v_2$
 $(\exists v_3 : n(v_3, v_1) \wedge n(v_3, v_2)) \triangleright v_1 = v_2$

Semantic Reduction

- The *Focus* and *Coerce* convert a set of three-valued structures into a more precise set of structures that describe the same set of stores.



The Shape-Analysis Algorithm

- The shape-analysis algorithm itself is an iterative procedure that computes a set of structures, $\text{StructSet}[v]$, for each vertex v of control-flow graph G , as a least fixed point of the following system of equations.

$$\text{StructSet}[v] = \begin{cases} \bigcup_{w \rightarrow v \in G} \{t_embed_c[\![st(w)]\!](S) \mid S \in \text{StructSet}[w]\} & \text{if } v \neq \text{start} \\ \{\langle \emptyset, \lambda p. \lambda u_1. \dots, u_k. 1/2 \rangle\} & \text{if } v = \text{start} \end{cases}$$

Convergence of The Shape-Analysis Algorithm

- The number of predicates is fixed.
- With canonical abstraction, the number of individuals is bounded.
 - $|U^S| \leq 2^{|A|}$ (*A is the set of abstraction predicates*)
- The number of possible structures is bounded.

To Beat A Dead Horse Again

- Why we need instrument predicates?
 - To collect the information we are interested in.
- Why we need *Focus* operations?
 - To maintain the precision of these information by making sure that the formulas that define the meaning of *st* evaluate to definite values.
- Why we need *Coerce* operations?
 - To minimize the set of possible structures by removing impossible structures.

Thanks!