Parametric Shape Analysis via 3-valued Logic

Authors: Mooly Sagiv, Thomas Reps
& Reinhard Wilhelm

Talk: Mike Dodds

Approach

Represent structures by logical structures

- Concretely - predicates represent connections between nodes
- Abstractly - use 3-valued logic to summarise properties

Construct sets of 3-valued structures by shape analysis

- Define a semantics for abstract execution over 3-value structures
- Construct fixed-points of abstract execution

2-valued structures

Represent stores by *logical structures*: $S = \langle U^S, \iota^S \rangle$

- U^S is a universe of *individuals*
- ι^S associates predicates with values

In a 2-value structure ι^S maps each arity-k predicate and tuple $(u_1, \ldots, u_k)$ to 0 or 1.

Also require a variable interpretation $Z \colon \{v_1, v_2, \ldots\} \to U^S$

2-valued logic

Write formulas φ with the following operators:

- first-order conjunction, disjunction, universal quantification.
- Equality assertions
- Transitive closure, $(TC\ \ v_1, v_2 \colon \varphi)(v_3, v_4)$

Given a variable interpretation $Z \colon \{v_1, v_2, \ldots\} \to U^S$ we denote the 2-valued meaning of a formula φ by:

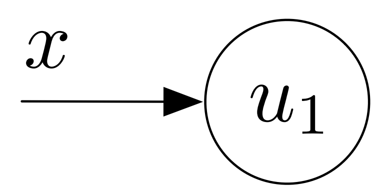
$$[\![\varphi]\!]_2^S(Z)$$

2-valued representation

Define *core predicates* recording the structure of a data structure by logical values

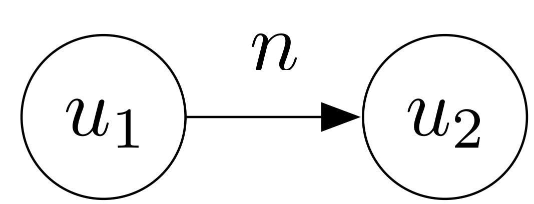
Unary predicates hold for a variable if the variable points to the argument value:

$$x(u_1)$$



Edges are recorded by binary predicates:

$$n(u_1, u_2)$$



2-valued list

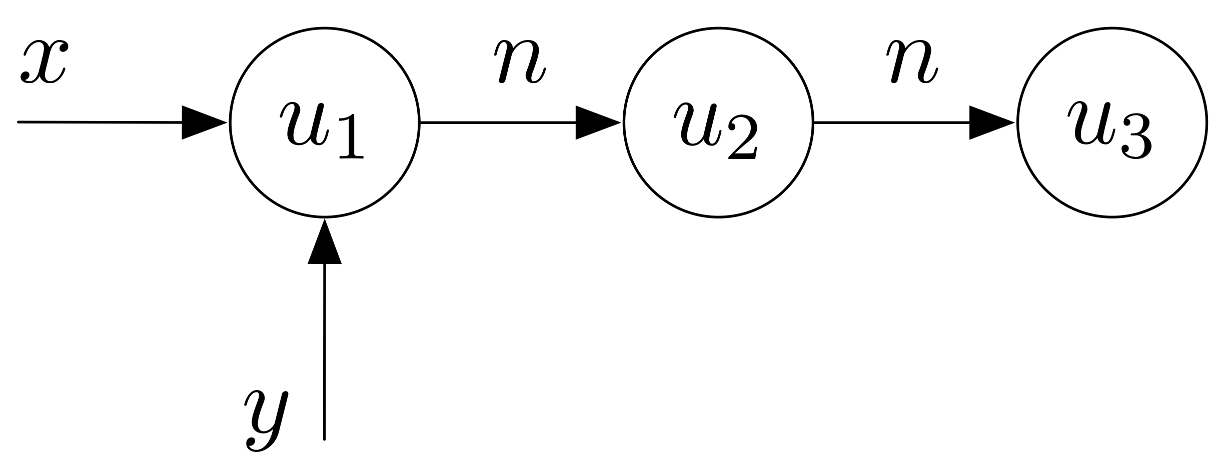
Unary predicates x and y:

	x	y
u_1	1	1
u_2	0	0
u_3	0	0

Binary predicate n:

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

These logical values represent the following structure:



Compatibility constraints

In order to represent pointer structures, logical formulas must obey compatibility constraints.

- Every individual has exactly one *n*-labelled out-edge

$$\forall v_1, v_2 \colon (\exists v_3 \colon n(v_3, v_1) \wedge n(v_3, v_2)) \Rightarrow v_1 = v_2$$

- Every variable points to at most one individual

$$\text{for each } x \in PVar, \forall v_1, v_2 \colon x(v_1) \wedge x(v_2) \Rightarrow v_1 = v_2$$

We have to enforce these constraints explicitly during analysis by *coercion*

Operational semantics

Define the operational semantics of state updates by logical formulas on variables.

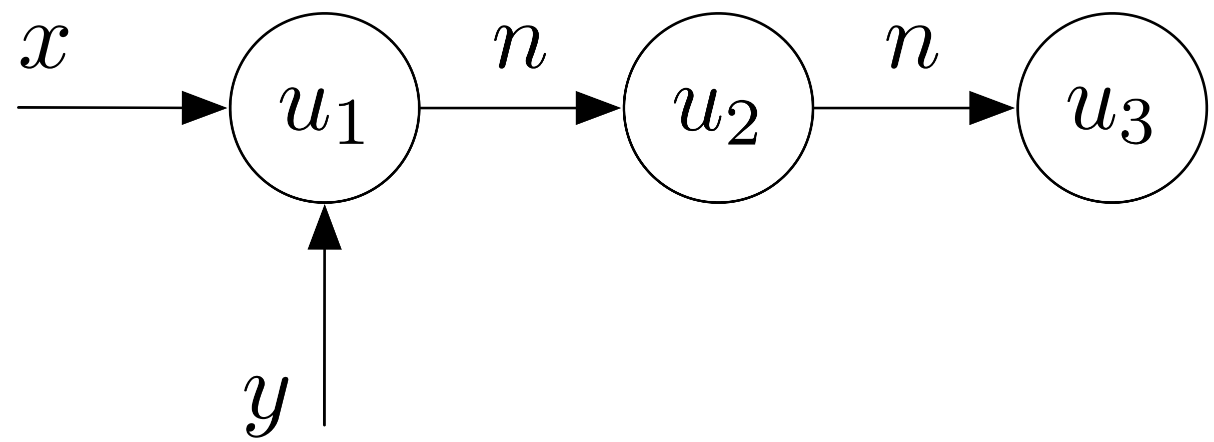
For a statement `y := y->n`

...we have update: $y'(v) \quad = \quad \exists v_1 .\, y(v_1) \wedge n(v_1, v)$

Other predicates are unchanged, as they are unaffected by the rewrite.

Handle memory allocation by adding a new individual to the universe, then applying an update as above.

Updating the List



	x	y
u_1	1	1
u_2	0	0
u_3	0	0

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

Statement: `y := y->n`

Updating the List

	x	y
u_1	1	1
u_2	0	0
u_3	0	0

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

Statement: `y := y->n`

Updates:

$$
\begin{aligned}
x'(v) &= x(v) \\
y'(v) &= \exists v_1 .\, y(v_1) \wedge n(v_1, v) \\
n'(v_1, v_2) &= n(v_1, v_2)
\end{aligned}
$$

Updatin

only y
is updated according
to the semantics

	x	y
u_1	1	1
u_2	0	0
u_3	0	0

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

Statement: `y := y->n`

Updates:

$$\begin{aligned} x'(v) &= x(v) \\ y'(v) &= \exists v_1 .\, y(v_1) \wedge n(v_1, v) \\ n'(v_1, v_2) &= n(v_1, v_2) \end{aligned}$$

Updatin

only y
is updated according
to the semantics

	x	y
u_1	1	0
u_2	0	1
u_3	0	0

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

Statement: `y := y->n`

Updates:

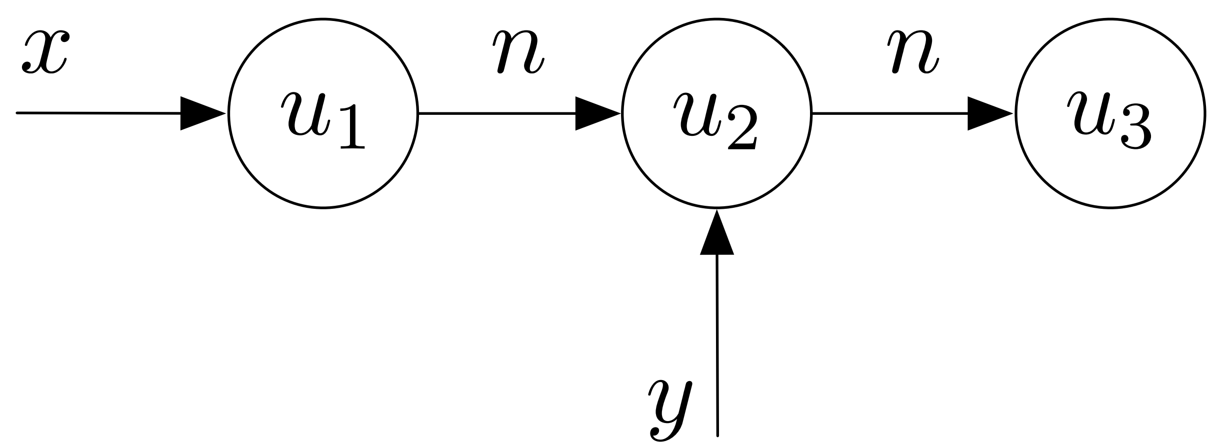
$$\begin{aligned} x'(v) &= x(v) \\ y'(v) &= \exists v_1 .\, y(v_1) \land n(v_1, v) \\ n'(v_1, v_2) &= n(v_1, v_2) \end{aligned}$$

Updating the List

	x	y
u_1	1	0
u_2	0	1
u_3	0	0

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

Statement: `y := y->n`



3-valued structures

We call 1 and 0 *definite values,* and $1/2$ the *indefinite value*.

In a *3-value* structure ι^S maps each arity-k predicate and tuple $(u_1, \ldots, u_k)$ to 0, 1, or $1/2$

3-valued logic

Operators in 3-valued logic have definitions as if the indefinite value could be *either* 0 or 1

$$\begin{aligned} 1 \wedge {}^1\!/_2 &= {}^1\!/_2 \\ 0 \vee {}^1\!/_2 &= {}^1\!/_2 \\ &\ldots \end{aligned}$$

Given a variable interpretation Z we denote the 3-valued meaning of a formula φ by:

$$[\![\varphi]\!]_3^S(Z)$$

Abstraction

Use 3-valued structures to represent classes of 2-valued structures

Associate definite values with elements that are guaranteed to be present in the structure.

The indefinite value $1/2$ represents things that *may* be present.

Embedding

We define an information order $\sqsubseteq$ on logical values so $l \sqsubseteq l'$ if $l = l'$ or $l' = 1/2$

For two structures S, S' and a function $f : U^S \to U^{S'}$ we say *f embeds S in S'* if:

$$\iota^S(p)(u_1, \ldots, u_k) \sqsubseteq \iota^{S'}(p)(f(u_1), \ldots, f(u_k))$$

...for all predicates p *and* $u_i \in U^S$

Embedding theorem

Let S, S' be two structures and $f : U^S \to U^{S'}$ and an embedding function such that $S \sqsubseteq^f S'$

Then for any formula φ and complete assignment Z :

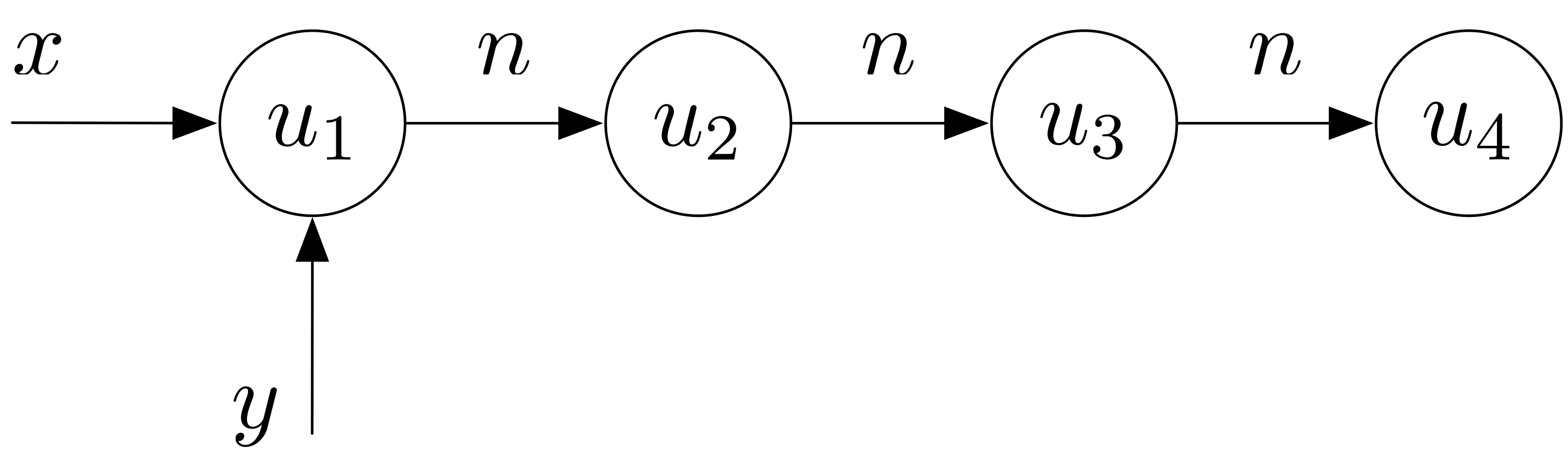
$$[\![\varphi]\!]_3^S(Z) \sqsubseteq [\![\varphi]\!]_3^{S'}(Z)$$

That is, we can use a three-value structure to summarise any structure embedded in it, for any formula.

List abstraction

	x	y
u_1	1	1
u_2	0	0
u_3	0	0
u_4	0	0

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



List abstraction

	x	y
u_1	1	1
u_2	0	0
u_3	0	0
u_4	0	0

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

	x	y	sm
u_1	1	1	0
u_{234}	0	0	$1/2$

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

List abstraction

	x	y
u_1	1	1
u_2	0	0
u_3	0	0
u_4	0	0

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
[illegible]	0	0	0	0

node u_{234} summarises nodes u_2, u_3, u_4

	x	y	sm
u_1	1	1	0
u_{234}	0	0	$1/2$

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

List abstraction

	x	y
u_1	1	1
u_2	0	0
u_3	0	0
u_4	0	0

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
[illegible]	[illegible]	[illegible]	0	0

predicate *sm*
records that a node has been summarised

....

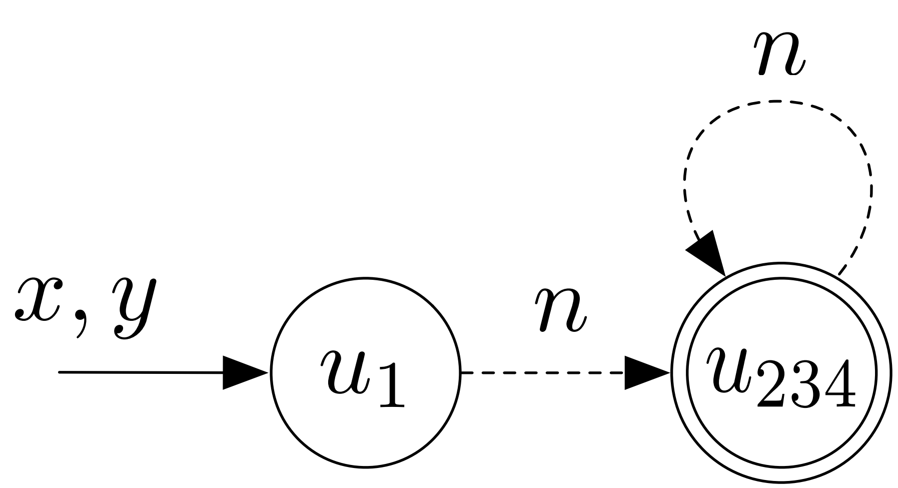
	x	y	sm
u_1	1	1	0
u_{234}	0	0	$1/2$

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

List abstraction

	x	y	sm
u_1	1	1	0
u_{234}	0	0	$1/2$

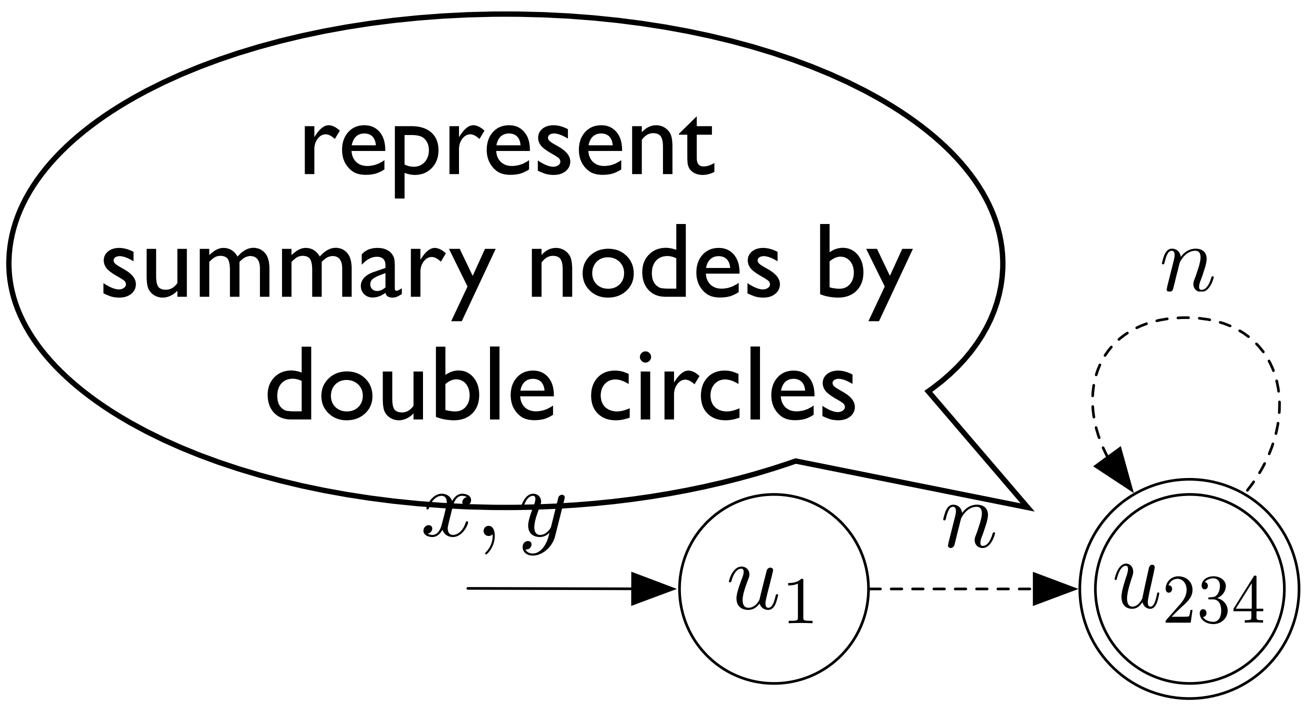
n	u_1	u_{234}
u_1	0	$1/2$
u_{234}	0	$1/2$



List abstraction

	x	y	sm
u_1	1	1	0
u_{234}	0	0	$1/2$

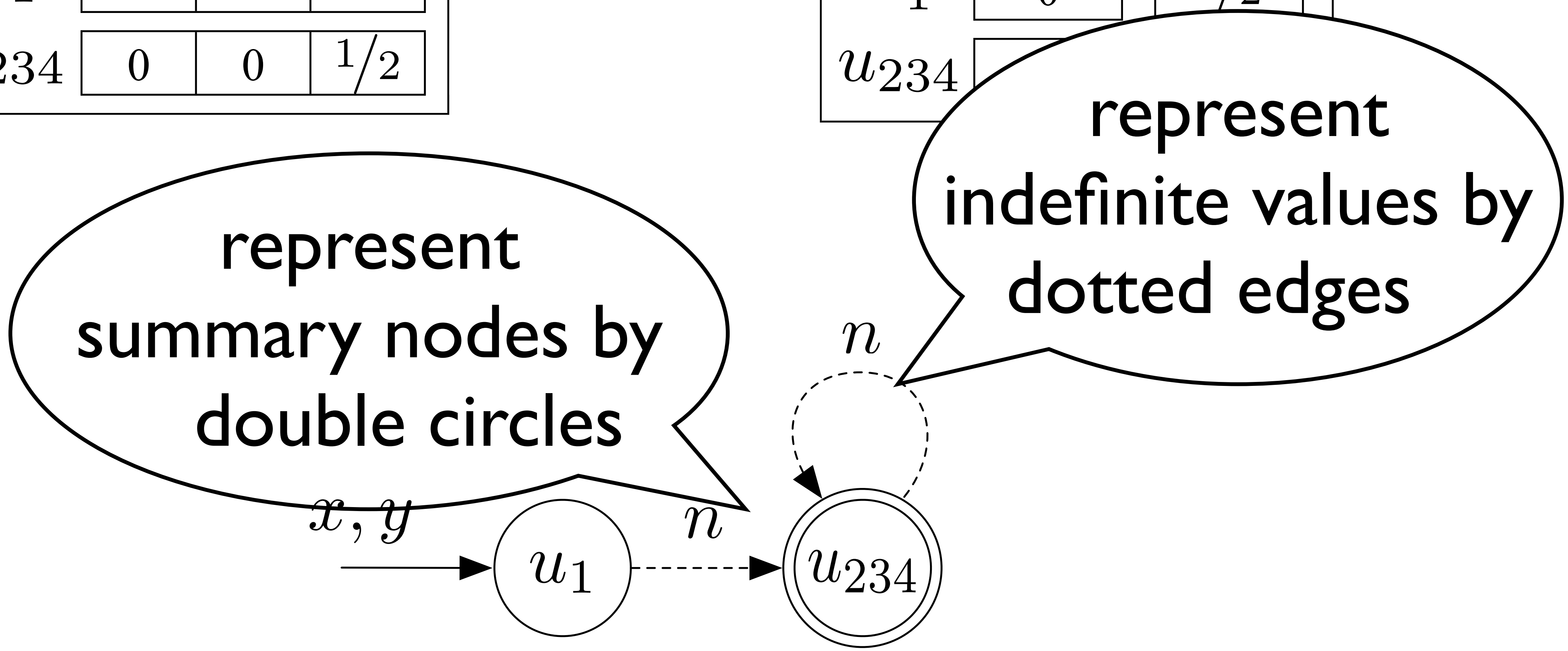
n	u_1	u_{234}
u_1	0	$1/2$
u_{234}	0	$1/2$



List abstraction

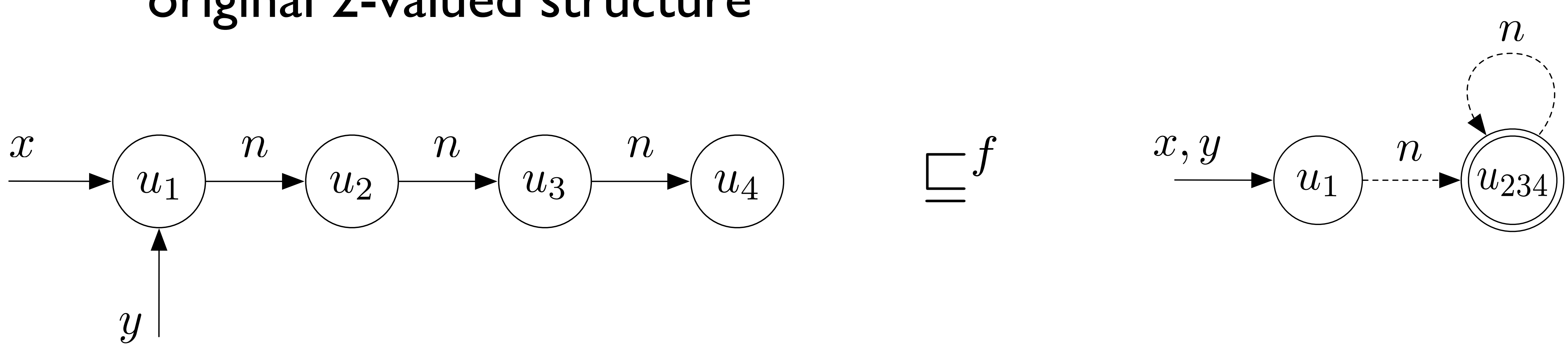
	x	y	sm
u_1	1	1	0
u_{234}	0	0	$1/2$

n	u_1	u_{234}
u_1	0	$1/2$
u_{234}	[illegible]	[illegible]



Embedding and abstraction

The resulting 3-valued structure should embed the original 2-valued structure



Note that we could have more indefinite values than we need, eg by making u_1 indefinite.

Call a minimally-indefinite embedding a *tight* embedding

Analysis algorithm

Construct a control-flow graph G for the program.

Assign a set of 3-valued structures $StructSet[v]$ to every vertex v of the graph.

$StructSet[v]$ is defined as the least fixed-point of the following system of equations

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

Shape analysis algorithm

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

$st(w)$ is the update formula for the transition $w \to v$

$embed[\![S, st(w)]\!]$ takes a structure S, applies update $st(w)$ and constructs a set of 3-value structures summarising the resulting structures

$\langle \emptyset, \lambda p.\, \lambda u_1, \ldots, u_k, {}^1\!/_2 \rangle$ is the empty structure, where all predicates have indefinite values

Termination

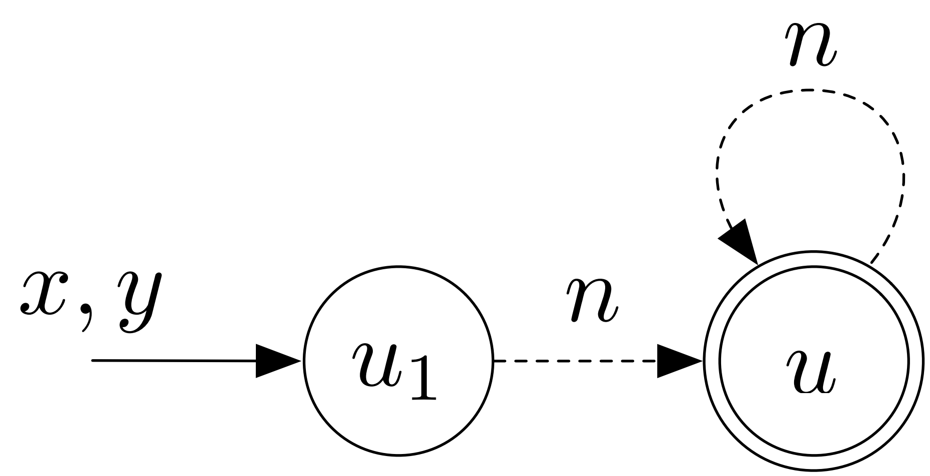
Termination is ensured by defining a finite class of *bounded structures* for a set of predicate symbols.

A structure $S = \langle U^S, \iota^S \rangle$ is bounded if for every pair of elements $u_1, u_2 \in U^S$ where $u_1 \neq u_2$ there exists a unary predicate p such that:

- $\iota^S(p)(u_1) \neq 1/2$ and $\iota^S(p)(u_2) \neq 1/2$
- $\iota^S(p)(u_1) \neq \iota^S(p)(u_2)$

The set of bounded structures is finite, and the embedding of a structure into a bounded structure is unique.

Naively updating structures

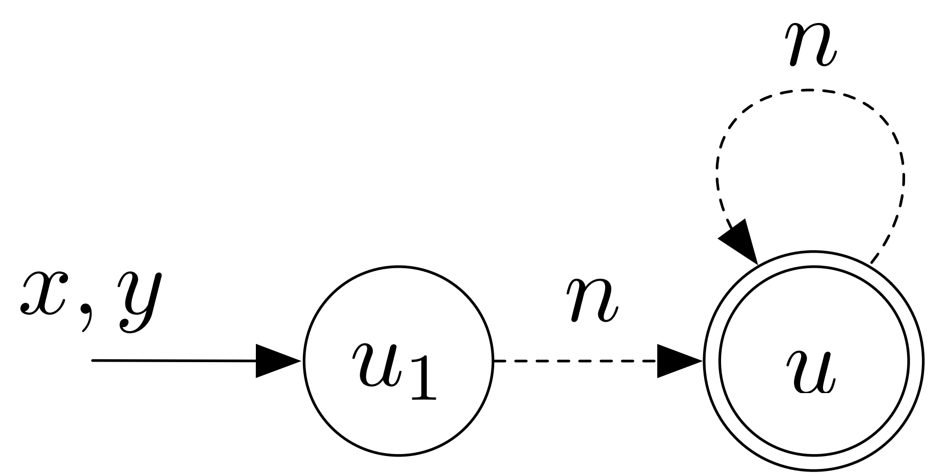


Statement: `x := x->n`

Apply the same update as in a 2-value structure:

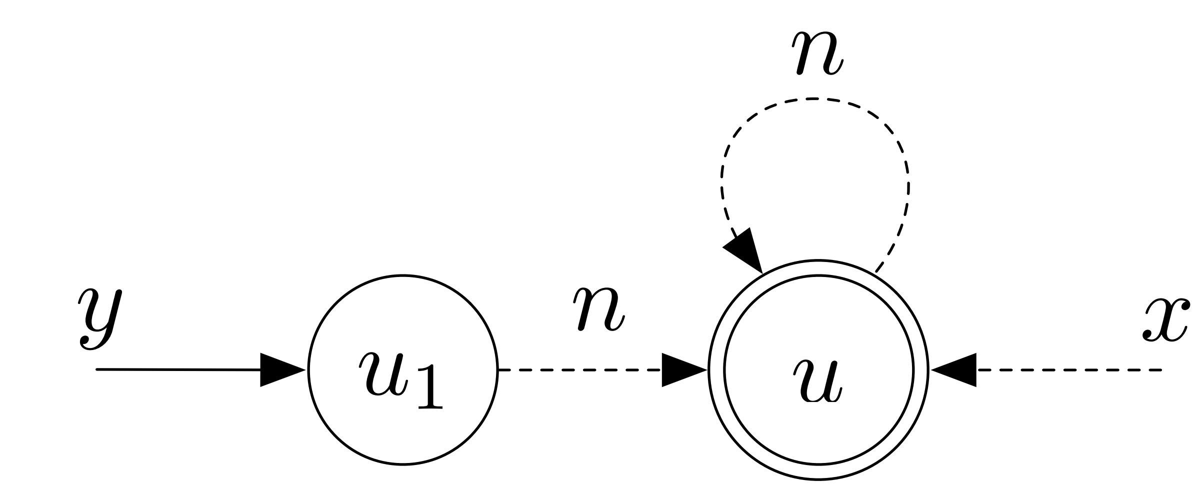
$$x'(v) \quad = \quad \exists v_1.\, x(v_1) \wedge n(v_1, v)$$

Naively updating structures

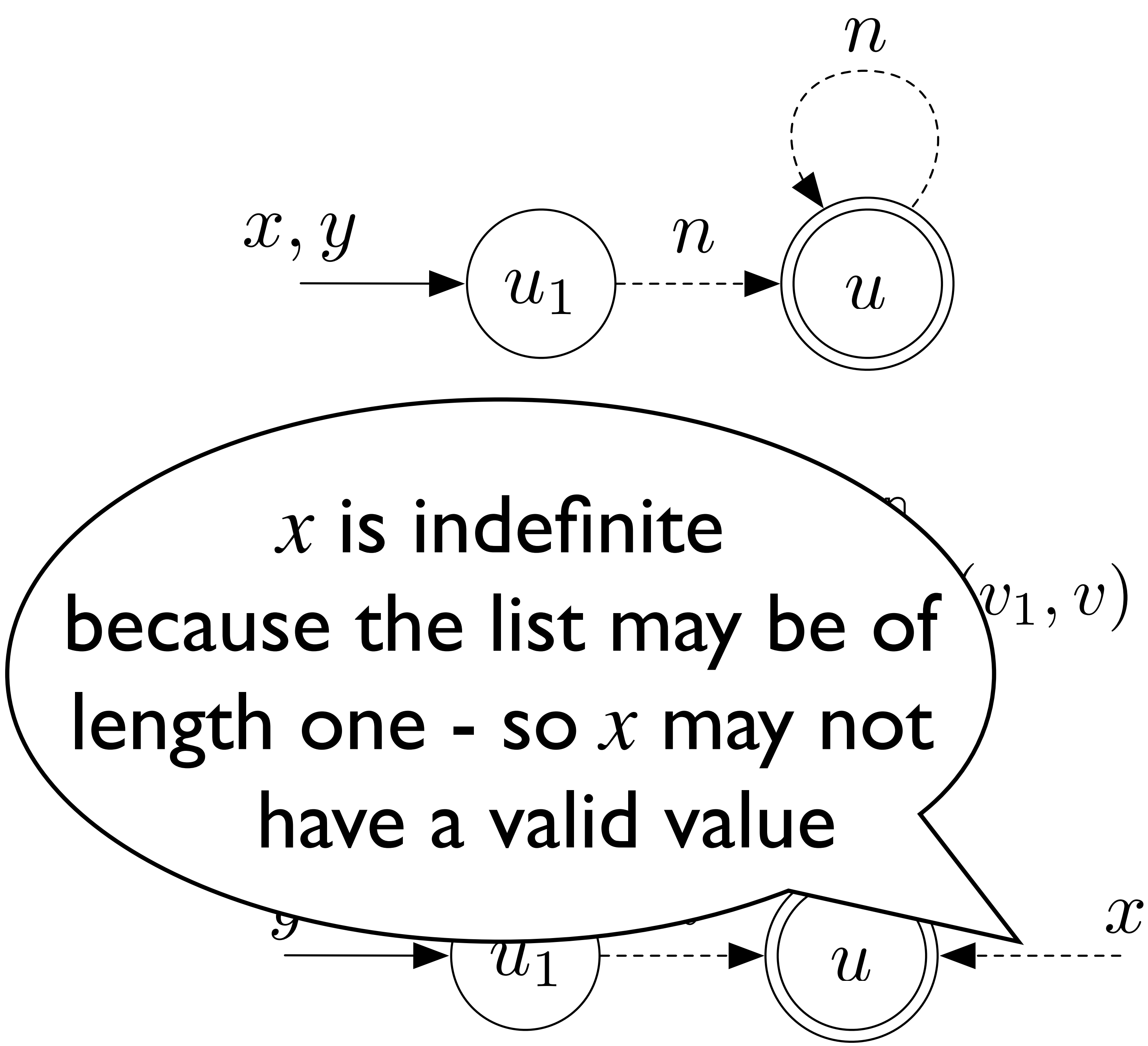


Statement: `x := x->n`

$$x'(v) \quad = \quad \exists v_1 .\, x(v_1) \wedge n(v_1, v)$$



Naively updating structures



Improving precision

Three methods of improving precision:

- Instrumentation predicates - attach more information in the structure
- Focussing - split cases to ensure more precise updating
- Coercion - make structures more precise by eliminating indefinite values and inconsistent structures

Instrumentation predicates

Core predicates do not capture important properties

- Sharing, patterns of edges
- Reachability, cyclicity, etc.

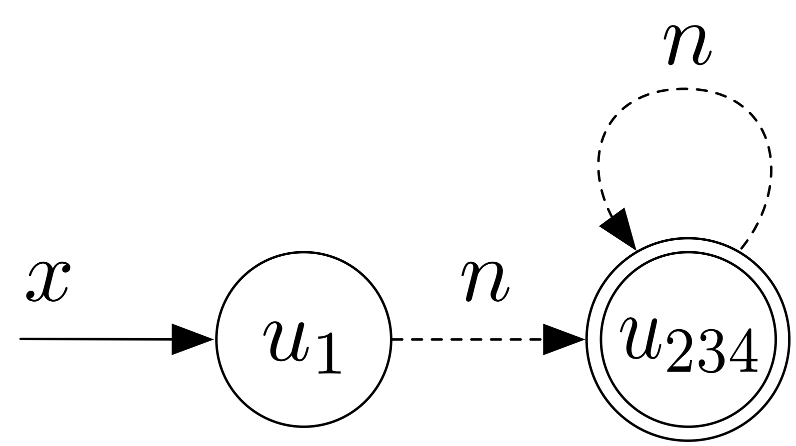
Shape analysis counters this with *instrumentation* predicates

- separate cases using predicates
- explicitly record properties

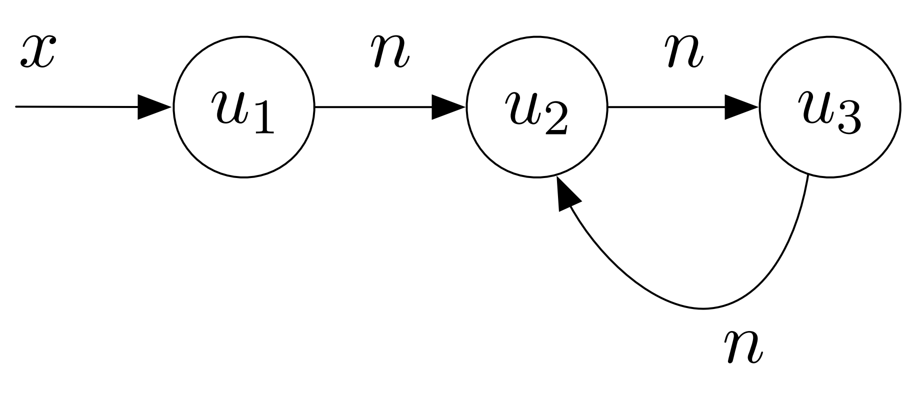
Sharing

	x	y	sm
u_1	1	1	0
u_{234}	0	0	$1/2$

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



This three-value structure also summarises lists with cycles, such as:



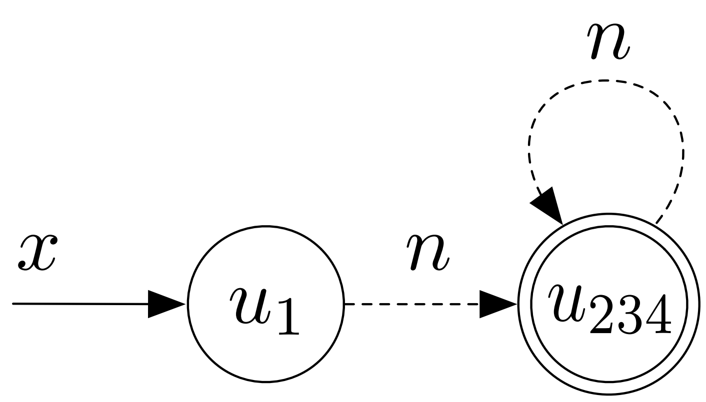
Add a sharing predicate

Predicate $is(u)$ holds if the node u is shared by two or more fields of heap elements

Acyclic list:

	x	y	sm	is
u_1	1	0	0	0
u_{234}	0	0	$1/2$	0

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

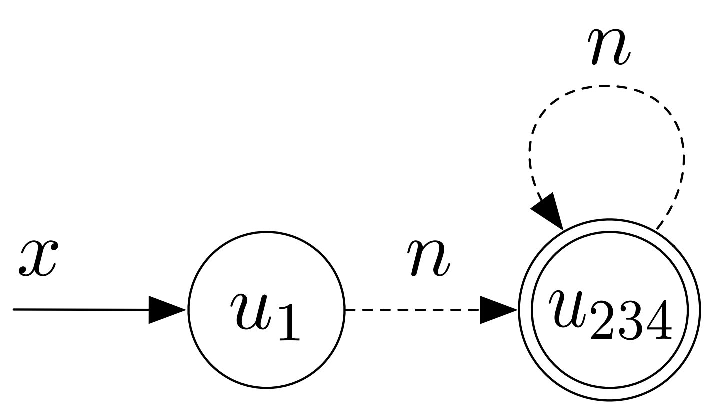


Add a sharing predicate

Acyclic list:

	x	y	sm	is
u_1	1	0	0	0
u_{234}	0	0	1/2	0

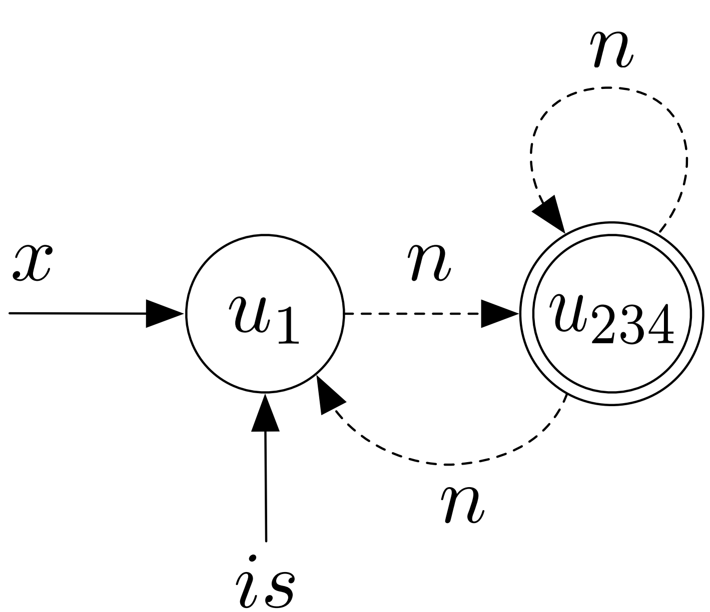
n	u_1	u_{234}
u_1	0	1/2
u_{234}	0	1/2



Cyclic list:

	x	y	sm	is
u_1	1	0	0	1
u_{234}	0	0	1/2	0

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



Updating the *is* predicates

Instrumentation predicates are updated in the same way as core predicates.

Statement: `x->n = y`

Update formula for $is(u)$:

$$
\begin{array}{rcl}
is'(v) & \stackrel{\text{def}}{=} & \left(\begin{array}{c} is(v) \land \exists v_1, v_2.\, v_1 \neq v_2 \\ \land\, n(v_1, v) \land n(v_2, v) \\ \land\, \neg x(v_1) \land \neg x(v_2) \end{array}\right) \\
& & \lor\, (y(v) \land \exists v_1.\, n(v_1, v) \land \neg x(v_1))
\end{array}
$$

Updating the *is* predicates

Instrumentati... ...tes are updated in the same way as core pre...

v is shared between two elements that aren't pointed to by x

Staten...

Update form...

$$is'(v) \stackrel{\mathbf{def}}{=} \begin{pmatrix} is(v) \wedge \exists v_1, v_2.\, v_1 \neq v_2 \\ \wedge\, n(v_1, v) \wedge n(v_2, v) \\ \wedge\, \neg x(v_1) \wedge \neg x(v_2) \end{pmatrix} \vee (y(v) \wedge \exists v_1.\, n(v_1, v) \wedge \neg x(v_1))$$

Updating the *is* predicates

Instrumentati… …tes are updated in the same way as core pre…

Staten…

Update form…

v is shared between two elements that aren't pointed to by x

$$is'(v) \stackrel{\textbf{def}}{=} \left(\begin{array}{l} is(v) \wedge \exists v_1, v_2 .\, v_1 \ldots \\ \quad \wedge\, n(v_1, v) \wedge n(v_2, \ldots \\ \quad \wedge\, \neg x(v_1) \wedge \neg x(v \ldots \end{array} \right) \\ \vee\, (y(v) \wedge \exists v_1 .\, n(v_1, v) \wedge \neg x(v_1))$$

v is now shared between x and y

Other predicates

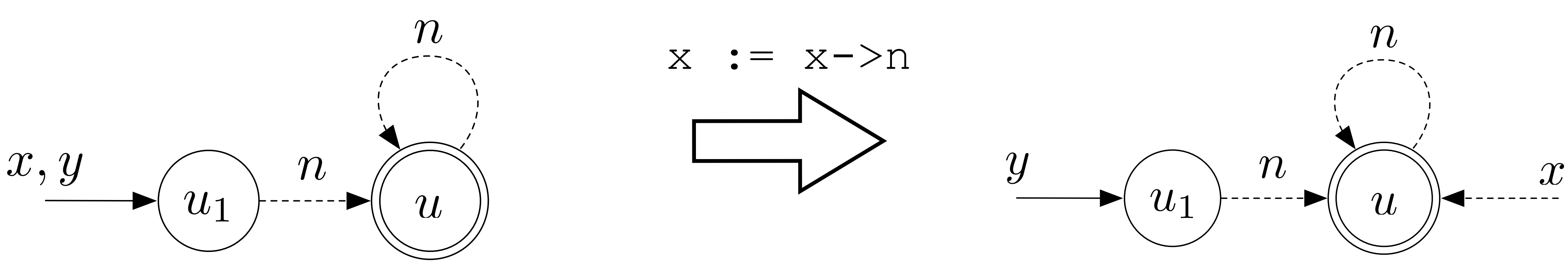
The is-shared predicate is a comparatively simple instrumentation predicate.

The analysis also uses:

- Edge-pattern predicates, e.g `*an n edge must be followed by a t edge*'
- Reachability predicate
- Cyclicity predicate

Focussing

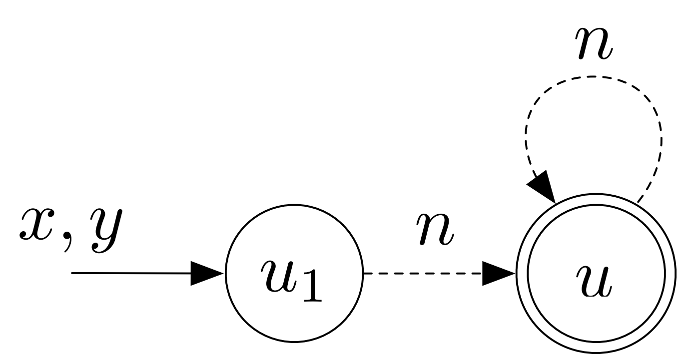
Applying a naive update to a 3-valued structure may give very imprecise results, eg:



To improve precision, define an operation *focus* that forces a given formula φ to a definite value.

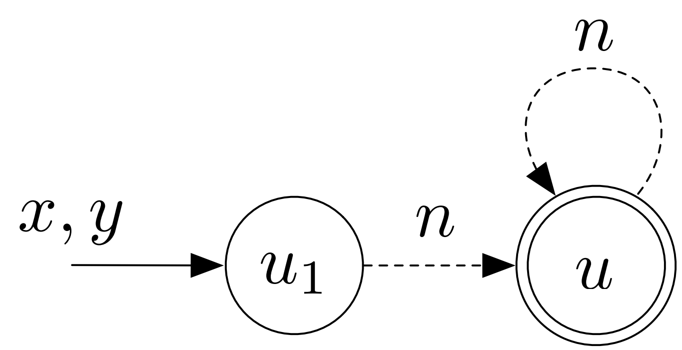
Focussing on a list

Solution to imprecision is to *focus* on a formula, instantiating it with definite values by case-splitting



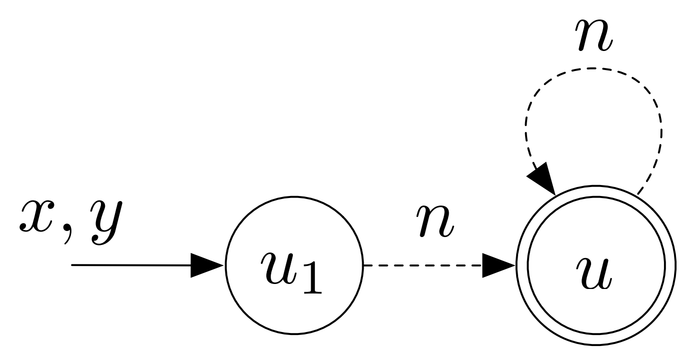
Focus formula: $\varphi_x(v) \stackrel{\mathbf{def}}{=} \exists v_1.\, x(v_1) \wedge n(v_1, v)$

Focussing on a list

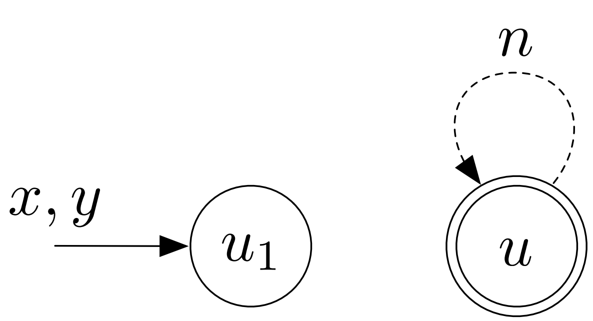


Focus formula: $\varphi_x(v) \stackrel{\text{def}}{=} \exists v_1.\, x(v_1) \wedge n(v_1, v)$

Focussing on a list

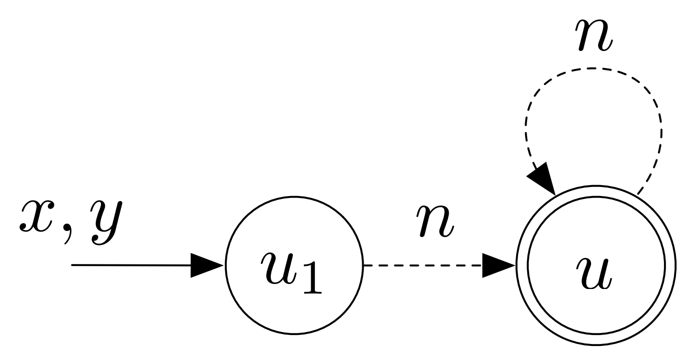


Focus formula: $\varphi_x(v) \stackrel{\mathbf{def}}{=} \exists v_1 .\, x(v_1) \wedge n(v_1, v)$

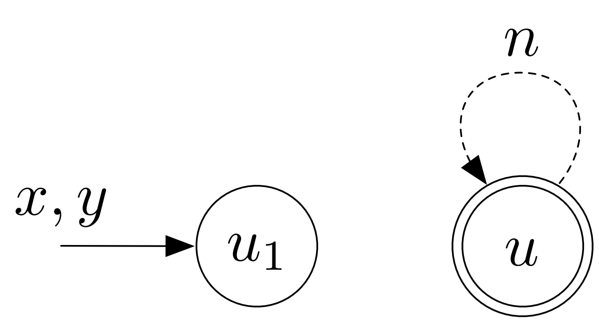


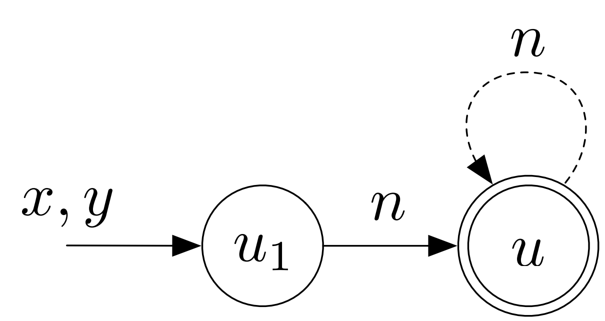
$\varphi_x(u) = 0$

Focussing on a list



Focus formula: $\varphi_x(v) \stackrel{\mathbf{def}}{=} \exists v_1 . \, x(v_1) \wedge n(v_1, v)$

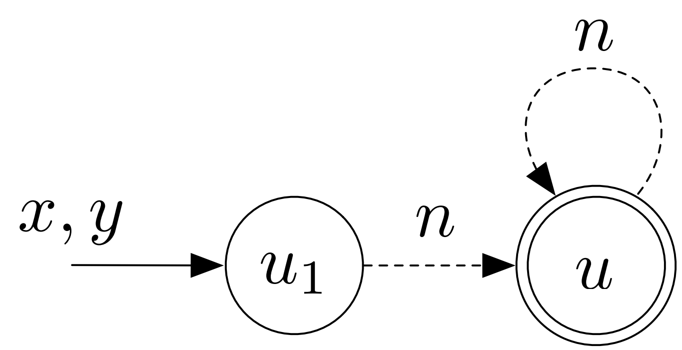




$\varphi_x(u) = 0$

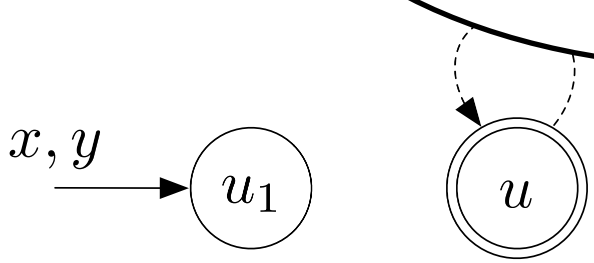
$\varphi_x(u) = 1$

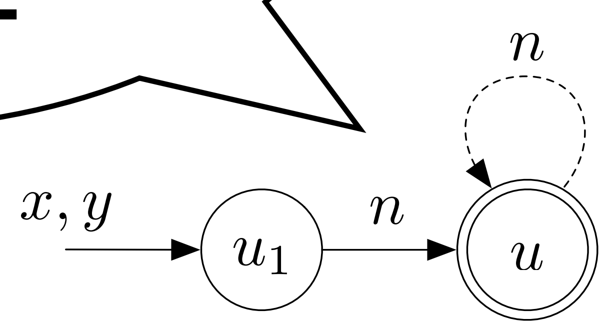
Focussing on a list



note that this only embeds lists of size 2

$\ldots) \stackrel{\mathbf{def}}{=} \exists v_1 .\, x(v_1) \wedge n(v_1, v)$

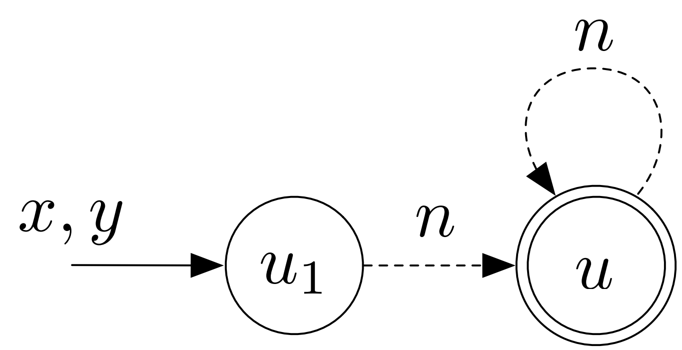




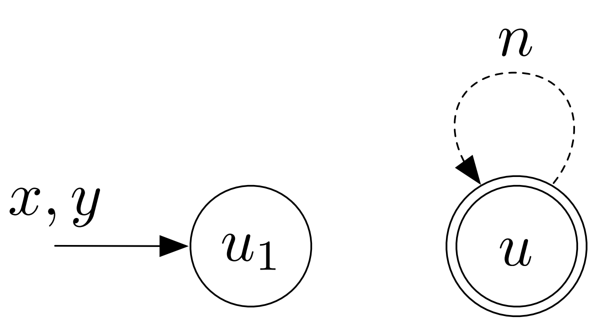
$\varphi_x(u) = 0$

$\varphi_x(u) = 1$

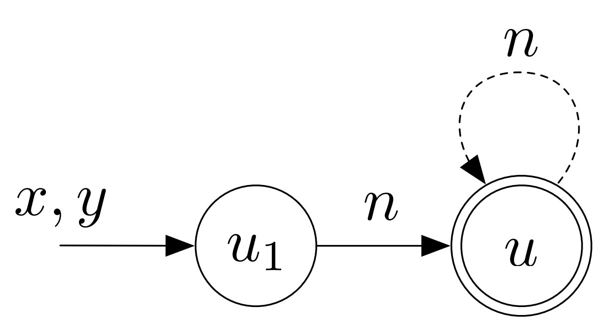
Focussing on a list



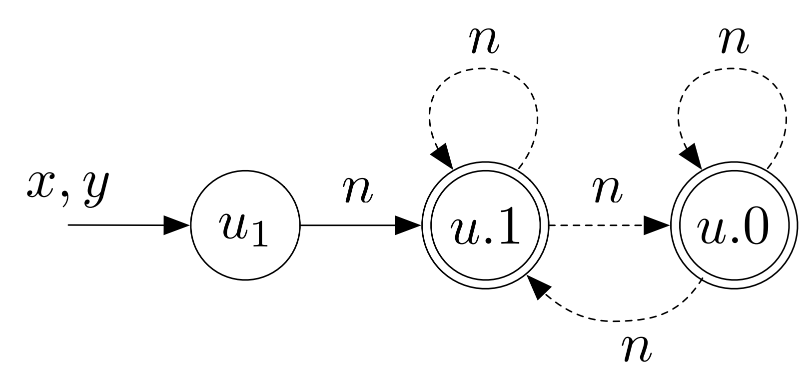
Focus formula: $\varphi_x(v) \stackrel{\mathbf{def}}{=} \exists v_1.\, x(v_1) \wedge n(v_1, v)$



$\varphi_x(u) = 0$



$\varphi_x(u) = 1$



$\varphi_x(u.1) = 1$

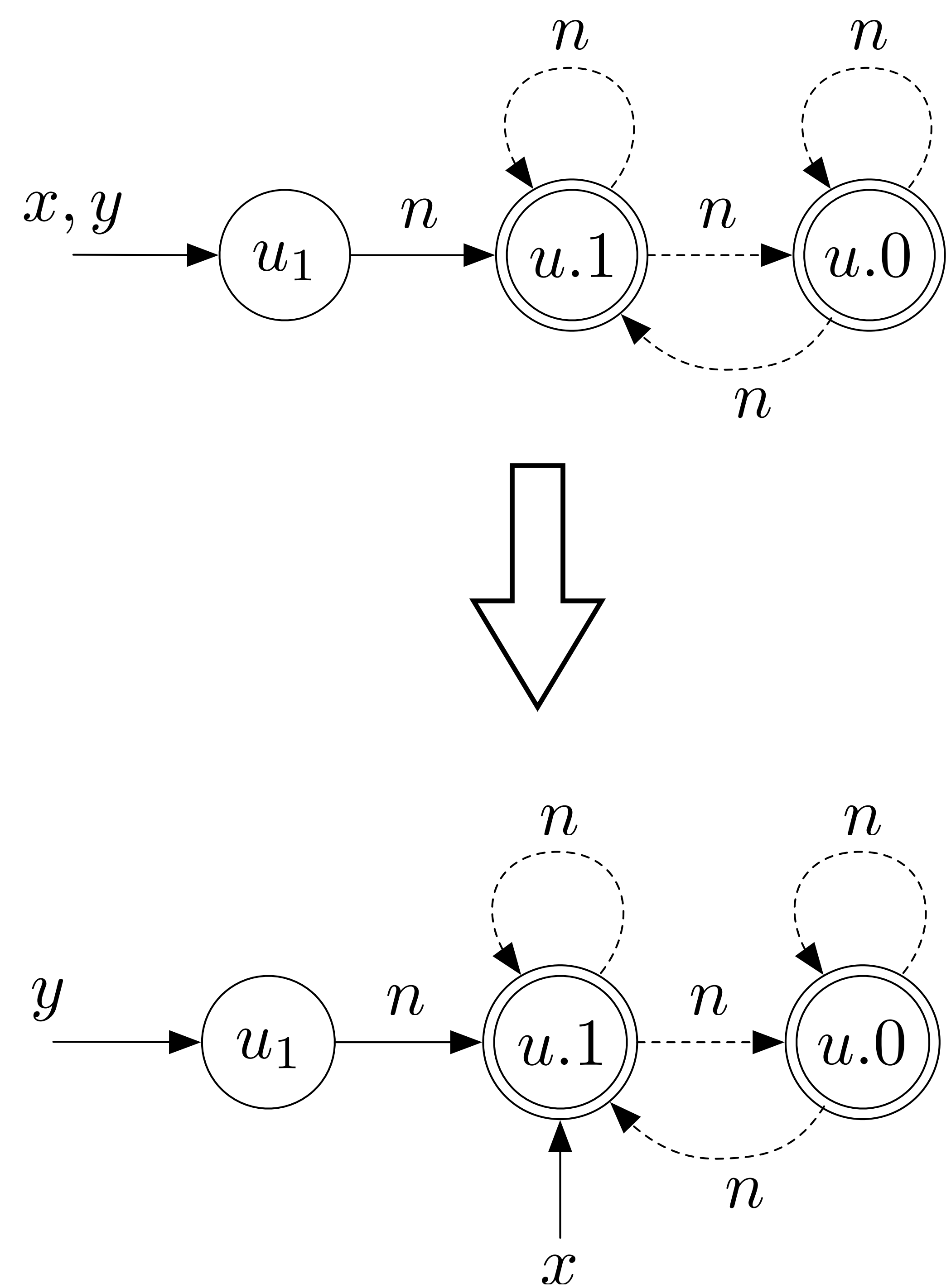
$\varphi_x(u.0) = 0$

Abstract execution

Statement: `x := x->n`

Updates:

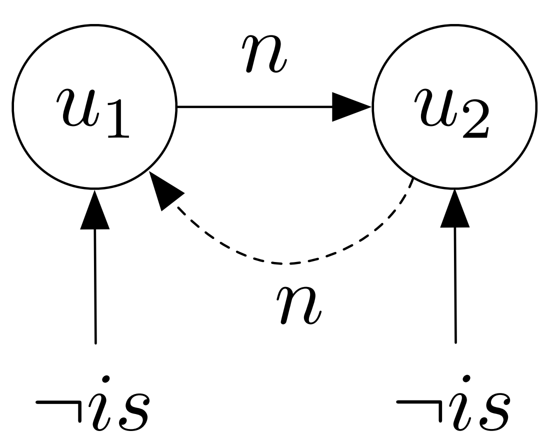
$$
\begin{aligned}
x'(v) &= \exists v_1 .\, x(v_1) \wedge n(v_1, v) \\
y'(v) &= y(v) \\
sm'(v) &= sm(v) \\
n'(v_1, v_2) &= n(v_1, v_2) \\
is'(v) &= is(v)
\end{aligned}
$$



Coercion

Increase precision by collapsing indefinite to definite values

Consider the following 3-value structure using the sharing predicate is

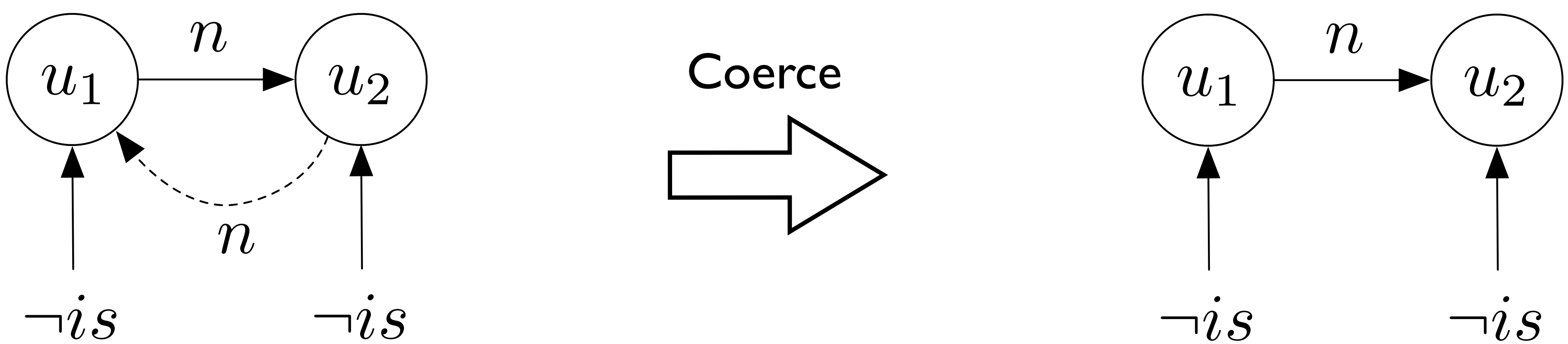


The prohibition on sharing implies that the indefinite edge doesn't exist.

Coercion

Increase precision by collapsing indefinite to definite values

Consider the following 3-value structure using the sharing predicate is

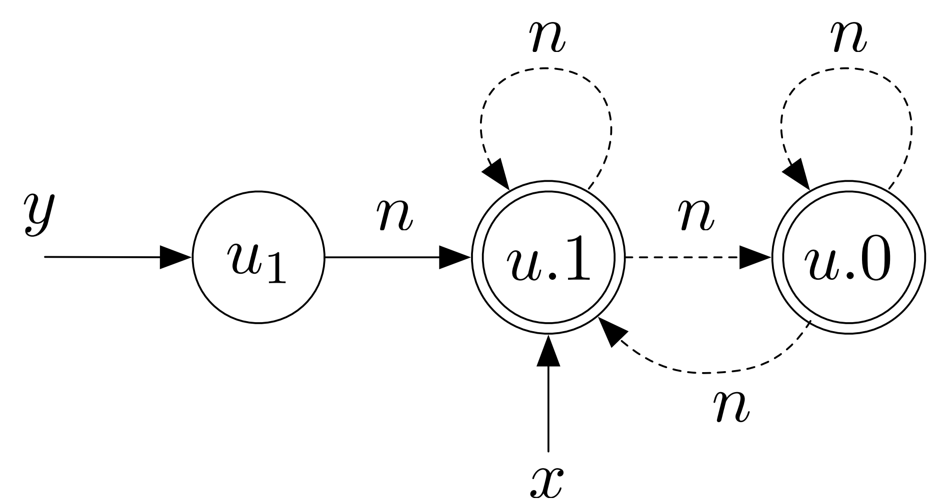


The prohibition on sharing implies that the indefinite edge doesn't exist.

Coercing a list

Coerce into a more precise representation

Recall that
$is(u_1) = is(u.1) = is(u.0) = 0$



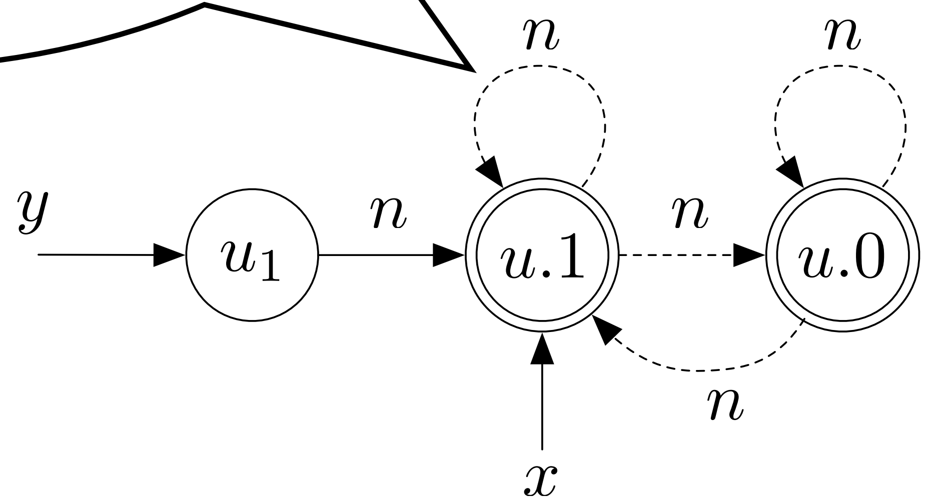
Node $u.1$ consequently must be a definite node in order to fit with semantics of is

Coercing a list

$u.1$ can't be shared as $is(u.1) = 0$, so this edge definitely doesn't exist

Coerce into a more p
representation

Recall that
$is(u_1) = is(u.1) = is(u.0) = 0$



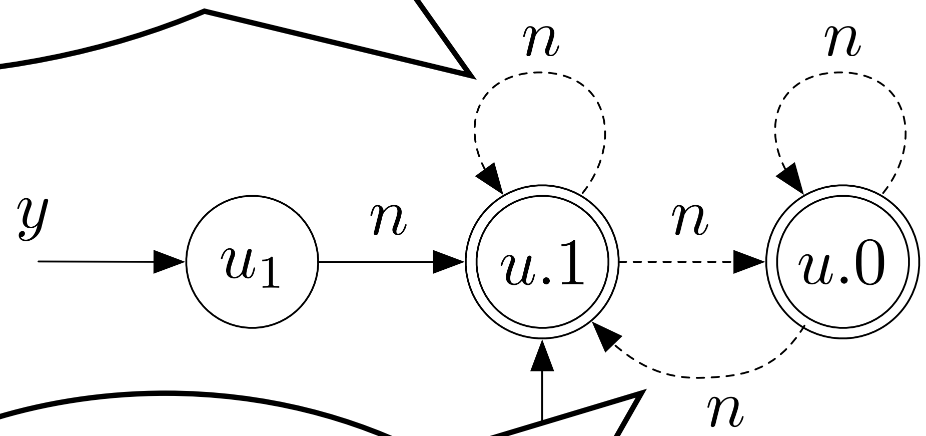
Node $u.1$ consequently must be a definite node in order to fit with semantics of *is*

Coercing a list

$u.1$ can't be shared as $is(u.1) = 0$, so this edge definitely doesn't exist

Coerce into a more pr
representation

Recall that
$is(u_1) = is(u.1) = is(u.0) = 0$



Node $u.1$ consequently must be a definite node in order to fit with semantics of is

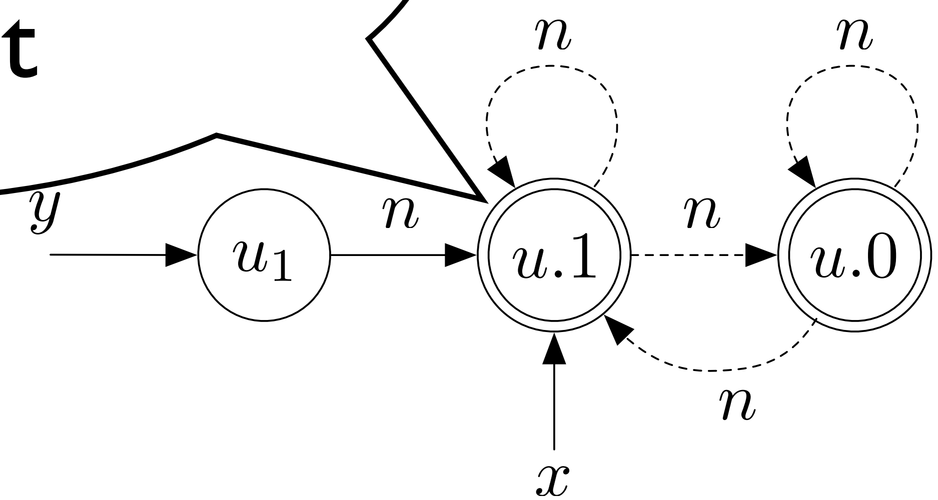
this edge definitely doesn't exist for the same reason

Coercing a list

This node is the target of a definite edge, therefore must exist

Coerce into a more p
representation

Recall that
$is(u_1) = is(u.1) = is(u.0) = 0$



Node $u.1$ consequently must be a definite node in order to fit with semantics of is

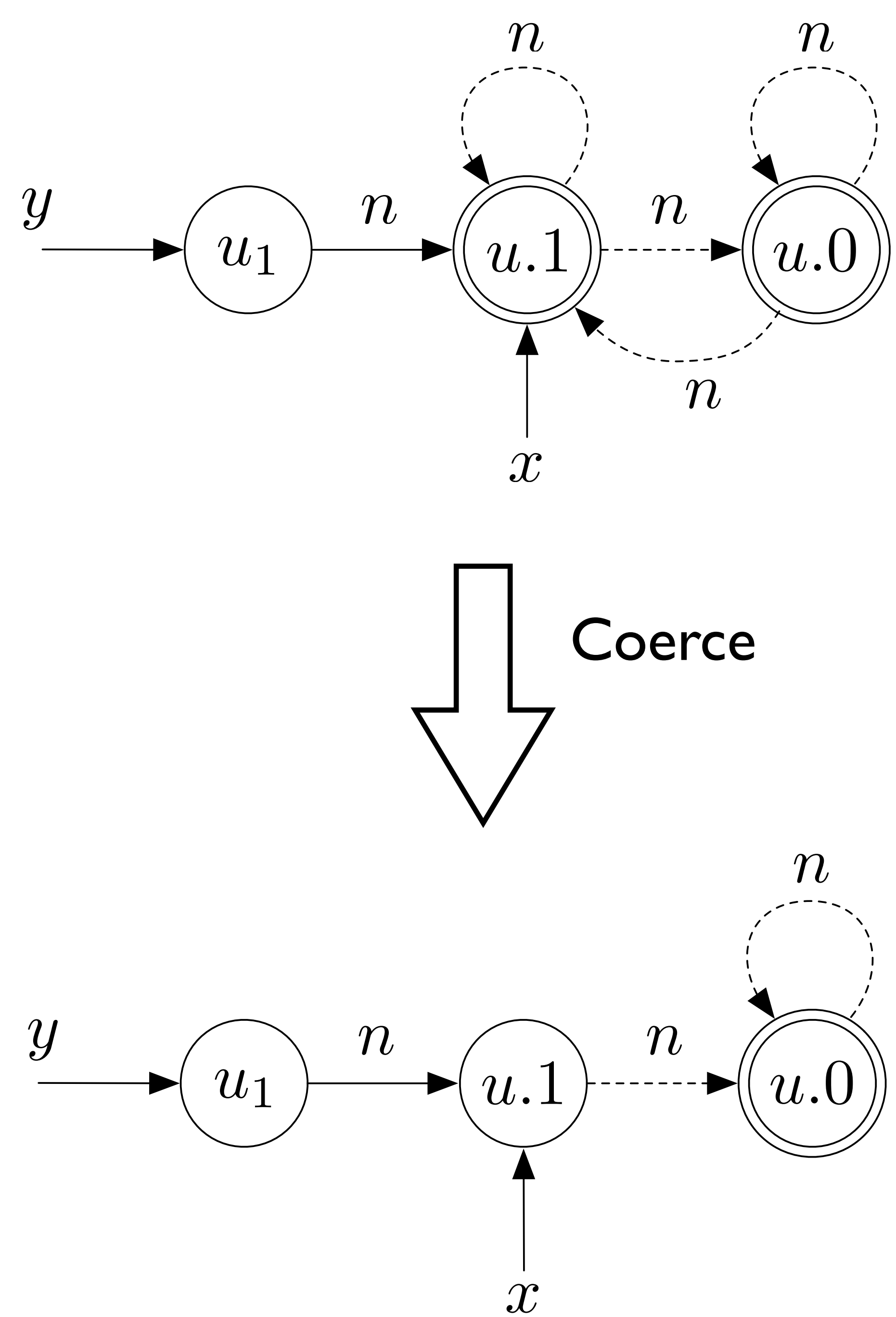
Coercing a list

Coerce into a more precise representation

Recall that

$is(u_1) = is(u.1) = is(u.0) = 0$

Node $u.1$ consequently must be a definite node in order to fit with semantics of is

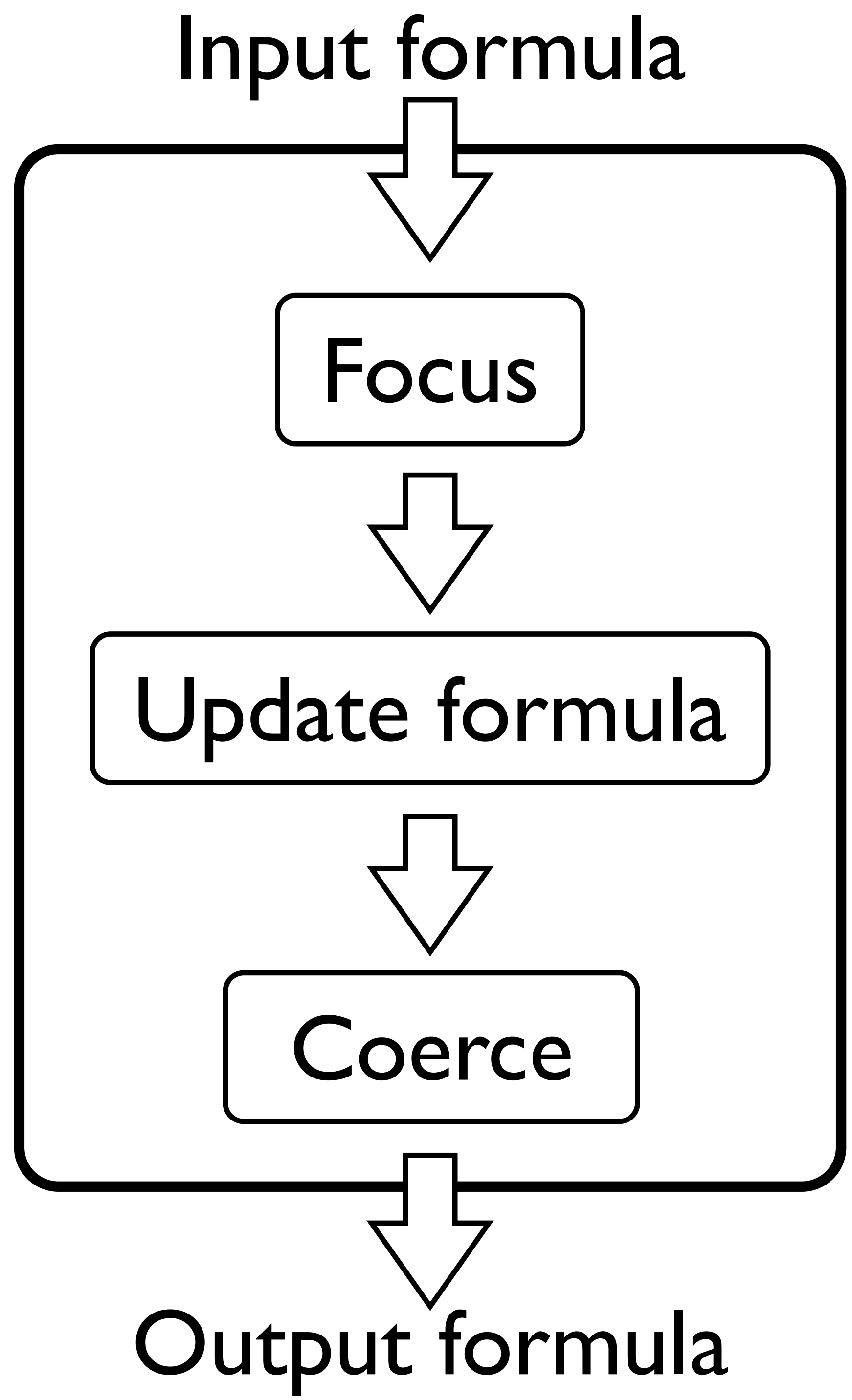


Update structure

Analysis uses the focus and coercion operations to improve the precision of analysis

Both take a set of structures and construct an equivalent set of more precise structures.

Collapse output formulas to bounded structures to ensure termination.



Summary

Analysis based on 3-valued structures

- Definite values are used to represent definite heap element; indefinite values represent possible heap elements
- 2-valued structures are *embedded* in representative 3-valued structures

3-valued structures are attached to a control-flow graph

- Abstract semantics of C statements based on logical updates
- Termination is ensured by a finite representation

Summary (2)

Simple abstract execution is extremely imprecise, so several strategies are needed to improve precision:

- *Instrumentation predicates* record explicit information about large-scale properties
- *Focussing* splits structures into sets of smaller, more precise structures
- *Coercion* makes structures more precise by collapsing indefinite values to definite values