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
- \( \iota^S \) associates predicates with values

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

Also require a variable interpretation \( Z : \{v_1, v_2, \ldots\} \rightarrow U^S \)
2-valued logic

Write formulas $\varphi$ with the following operators:

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

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

$$[[\varphi]]^S_2(Z)$$
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:

\[ x(u_1) \]

\[ x \rightarrow u_1 \]

Edges are recorded by binary predicates:

\[ n(u_1, u_2) \]

\[ u_1 \rightarrow u_2 \]
2-valued list

Unary predicates $x$ and $y$:

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$u_2$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$u_3$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Binary predicate $n$:

<table>
<thead>
<tr>
<th></th>
<th>$n$</th>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$u_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$u_2$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$u_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

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

\[ \forall v_1, v_2: (\exists v_3 : n(v_3, v_1) \land n(v_3, v_2)) \Rightarrow v_1 = v_2 \]

- Every variable points to at most one individual

for each $x \in PVar, \forall v_1, v_2 : x(v_1) \land x(v_2) \Rightarrow v_1 = v_2$

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 \( y := y \rightarrow n \)
...we have update: \( y'(v) = \exists v_1. y(v_1) \land n(v_1, v) \)

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

Statement: \( y := y \rightarrow n \)
Updating the List

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$u_2$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$u_3$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$n$</th>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$u_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$u_2$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$u_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Statement: \( y := y \rightarrow n \)

Updates:

\[
\begin{align*}
    x'(v) & = x(v) \\
    y'(v) & = \exists v_1. y(v_1) \land n(v_1, v) \\
    n'(v_1, v_2) & = n(v_1, v_2)
\end{align*}
\]
Updating the List

\[ u_1 \quad u_2 \quad u_3 \]
\[
\begin{array}{c|c|c|c}
\hline
& x & y \\
\hline
u_1 & 1 & 1 \\
u_2 & 0 & 0 \\
u_3 & 0 & 0 \\
\hline
\end{array}
\]

only \( y \) is updated according to the semantics

\[ n'_{u_1, u_2} = n(u_1, u_2) \]

Statement: \( y := y \rightarrow n \)

Updates:

\[
x'(v) = x(v)
\]
\[
y'(v) = \exists v_1. y(v_1) \land n(v_1, v)
\]
\[
n'(v_1, v_2) = n(v_1, v_2)
\]
Statement:  \( y := y \rightarrow n \)

**Updates:**

\[
\begin{align*}
  x'(v) & = x(v) \\
y'(v) & = \exists v_1. y(v_1) \land n(v_1, v) \\
n'(v_1, v_2) & = n(v_1, v_2)
\end{align*}
\]
Updating the List

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1</td>
</tr>
<tr>
<td>$u_2$</td>
<td>0</td>
</tr>
<tr>
<td>$u_3$</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$u_3$</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
</tr>
<tr>
<td>$u_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Statement: $y := y \rightarrow n$
3-valued structures

We call 1 and 0 definite values, and $\frac{1}{2}$ the indefinite value.

In a 3-value structure $\iota^S$ maps each arity-$k$ predicate and tuple $(u_1, \ldots, u_k)$ to 0, 1, or $\frac{1}{2}$.
3-valued logic

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

\[
\begin{align*}
1 \land \frac{1}{2} &= \frac{1}{2} \\
0 \lor \frac{1}{2} &= \frac{1}{2} \\
\ldots
\end{align*}
\]

Given a variable interpretation \( Z \) we denote the 3-valued meaning of a formula \( \varphi \) by:

\[
\llbracket \varphi \rrbracket_3^S(Z)
\]
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 $\varphi$ and complete assignment $Z$:

$$[\varphi]^S_3(Z) \sqsubseteq [\varphi]^{S'}_3(Z)$$

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

\[
\begin{array}{c|c|c}
\hline
x & y \\
\hline
u_1 & 1 & 1 \\
\hline
u_2 & 0 & 0 \\
\hline
u_3 & 0 & 0 \\
\hline
u_4 & 0 & 0 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
\hline
n & u_1 & u_2 & u_3 & u_4 \\
\hline
\hline
u_1 & 0 & 1 & 0 & 0 \\
\hline
u_2 & 0 & 0 & 1 & 0 \\
\hline
u_3 & 0 & 0 & 0 & 1 \\
\hline
u_4 & 0 & 0 & 0 & 0 \\
\hline
\end{array}
\]
List abstraction

\[
\begin{array}{cc}
\begin{array}{c|c}
\hline
x & y \\
\hline
u_1 & 1 & 1 \\
u_2 & 0 & 0 \\
u_3 & 0 & 0 \\
u_4 & 0 & 0 \\
\hline
\end{array}
\end{array}
\begin{array}{cc}
\begin{array}{ccccc}
\hline
n & u_1 & u_2 & u_3 & u_4 \\
\hline
u_1 & 0 & 1 & 0 & 0 \\
u_2 & 0 & 0 & 1 & 0 \\
u_3 & 0 & 0 & 0 & 1 \\
u_4 & 0 & 0 & 0 & 0 \\
\hline
\end{array}
\end{array}
\]

...abstracts to

\[
\begin{array}{ccc}
\begin{array}{c|ccc}
\hline
x & y & sm \\
\hline
u_1 & 1 & 1 & 0 \\
u_{234} & 0 & 0 & 1/2 \\
\hline
\end{array}
\end{array}
\begin{array}{cc}
\begin{array}{cc}
\hline
n & u_1 & u_{234} \\
\hline
u_1 & 0 & 1/2 \\
u_{234} & 0 & 1/2 \\
\hline
\end{array}
\end{array}
\]

15
### List abstraction

**Table 1:**

<table>
<thead>
<tr>
<th></th>
<th>(x)</th>
<th>(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u_1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(u_2)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(u_3)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(u_4)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 2:**

<table>
<thead>
<tr>
<th></th>
<th>(u_1)</th>
<th>(u_2)</th>
<th>(u_3)</th>
<th>(u_4)</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>

---

**Diagram:**

- **Node \(u_{234}\) summarises nodes \(u_2, u_3, u_4\)**
List abstraction

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$u_2$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$u_3$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$u_4$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$u_3$</th>
<th>$u_4$</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>$u_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**predicate** $sm$

records that a node has been summarised
List abstraction

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
 & $x$ & $y$ & $sm$ \\
\hline
$u_1$ & 1 & 1 & 0 \\
$u_{234}$ & 0 & 0 & $\frac{1}{2}$ \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
 & $u_1$ & $u_{234}$ \\
\hline
$n$ & 0 & $\frac{1}{2}$ \\
$u_1$ & 0 & $\frac{1}{2}$ \\
$u_{234}$ & 0 & $\frac{1}{2}$ \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tikzpicture}
    \node (u1) at (0,0) {$u_1$};
    \node (u234) at (1,0) {$u_{234}$};
    \node (n) at (0.5,1) {$n$};
    \node (xyn) at (-1,-0.5) {$x, y$};
    \draw[->] (xyn) -- (u1);
    \draw[->] (u1) -- (u234);
    \draw[->, dashed] (n) -- (u234);
\end{tikzpicture}
\end{center}
List abstraction

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>sm</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$u_{234}$</td>
<td>0</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>$u_1$</th>
<th>$u_{234}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>0</td>
<td></td>
<td>$\frac{1}{2}$</td>
</tr>
<tr>
<td>$u_{234}$</td>
<td>0</td>
<td></td>
<td>$\frac{1}{2}$</td>
</tr>
</tbody>
</table>

represent summary nodes by double circles
List abstraction

represent summary nodes by double circles

represent indefinite values by dotted edges
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

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

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

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

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

\( \langle \emptyset, \lambda p. \lambda u_1, \ldots, u_k, \frac{1}{2} \rangle \) 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: \[ x := x \rightarrow n \]

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

\[ x'(v) = \exists v_1. x(v_1) \land n(v_1, v) \]
Naively updating structures

Statement: \( x := x \rightarrow n \)

\[ x'(v) = \exists v_1. x(v_1) \land n(v_1, v) \]
Naively updating structures

\[ x := x \rightarrow n \]

\( x \) is indefinite because the list may be of length one - so \( x \) may not have a valid value.
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
This three-value structure also summarises lists with cycles, such as:

\[
\begin{array}{ccc}
  x & y & sm \\
  u_1 & 1 & 1 & 0 \\
  u_{234} & 0 & 0 & 1/2 \\
\end{array}
\]

\[
\begin{array}{ccc}
  n & u_1 & u_{234} \\
  u_1 & 0 & 1/2 \\
  u_{234} & 0 & 1/2 \\
\end{array}
\]
Add a sharing predicate

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

Acyclic list:

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
<th>$sm$</th>
<th>$is$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$u_{234}$</td>
<td>0</td>
<td>0</td>
<td>1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$n$</th>
<th>$u_1$</th>
<th>$u_{234}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>0</td>
<td>1/2</td>
<td></td>
</tr>
<tr>
<td>$u_{234}$</td>
<td>0</td>
<td>1/2</td>
<td></td>
</tr>
</tbody>
</table>
Add a sharing predicate

Acyclic list:

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>sm</th>
<th>is</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$u_{234}$</td>
<td>0</td>
<td>0</td>
<td>1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>$u_1$</th>
<th>$u_{234}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>0</td>
<td>1/2</td>
</tr>
<tr>
<td>$u_{234}$</td>
<td>0</td>
<td>1/2</td>
</tr>
</tbody>
</table>

Cyclic list:

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>sm</th>
<th>is</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$u_{234}$</td>
<td>0</td>
<td>0</td>
<td>1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>$u_1$</th>
<th>$u_{234}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>0</td>
<td>1/2</td>
</tr>
<tr>
<td>$u_{234}$</td>
<td>0</td>
<td>1/2</td>
</tr>
</tbody>
</table>
Updating the \( is \) predicates

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

Statement: \( x \rightarrow n = y \)

Update formula for \( is(u) \):

\[
is'(v) \defeq \left( is(v) \land \exists v_1, v_2. v_1 \neq v_2 \right) \\
\land n(v_1, v) \land n(v_2, v) \\
\land \neg x(v_1) \land \neg x(v_2) \\
\lor (y(v) \land \exists v_1. n(v_1, v) \land \neg x(v_1))
\]
Updating the \( is \) predicates

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

\[ \text{Update formula for } is \quad (u) \]

\begin{align*}
\text{Statement:} & \quad x \rightarrow n = y \\
\text{v is} & \quad \text{shared between two elements that aren't pointed to by } x \\
\text{Update formula:} & \quad is'(v) \overset{\text{def}}{=} \\
& \quad \left( is(v) \land \exists v_1, v_2. v_1 \neq v_2 \right) \\
& \quad \land n(v_1, v) \land n(v_2, v) \\
& \quad \land \neg x(v_1) \land \neg x(v_2) \\
& \quad \lor (y(v) \land \exists v_1. n(v_1, v) \land \neg x(v_1))
\end{align*}
Updating the *is* predicates

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

**Statement:**

\[ x \rightarrow n = y \]

\[ v \text{ is shared between two elements that aren’t pointed to by } x \]

\[ \text{Update formula for } \text{is} \]

\[ \text{is}'(v) \overset{\text{def}}{=} (\text{is}(v) \land \exists v_1, v_2. \ v_1 \land n(v_1, v) \land n(v_2, v) \land \neg x(v_1) \land \neg x(v_2) \land \neg \exists v_1 . n(v_1, v) \land \neg x(v_1)) \]

\[ \lor (y(v) \land \exists v_1 . n(v_1, v) \land \neg x(v_1)) \]

\[ v \text{ is now shared between } x \text{ and } y \]
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 \textit{focus} that forces a given formula $\varphi$ to a definite value.
Solution to imprecision is to *focus* on a formula, instantiating it with definite values by case-splitting

Focus formula: $\varphi_x(v) \overset{\text{def}}{=} \exists v_1. x(v_1) \land n(v_1, v)$
Focussing on a list

Focus formula: \( \varphi_x(v) \overset{\text{def}}{=} \exists v_1. x(v_1) \land n(v_1, v) \)
Focussing on a list

Focus formula: \( \varphi_x(v) \overset{\text{def}}{=} \exists v_1. x(v_1) \land n(v_1, v) \)

\( \varphi_x(u) = 0 \)
Focussing on a list

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

$\varphi_x(u_1) = 0$

$\varphi_x(u) = 1$
Focussing on a list

Note that this only embeds lists of size 2

\[ \varphi_x(u) = 0 \]

\[ \varphi_x(u) = 1 \]
Focussing on a list

Focus formula: $\phi_x(v) \overset{\text{def}}{=} \exists v_1 . x(v_1) \land n(v_1, v)$

$\varphi_x(u) = 0$

$\varphi_x(u) = 1$

$\varphi_x(u.1) = 1$

$\varphi_x(u.0) = 0$
Abstract execution

Statement: \[ x := x \mapsto n \]

Updates:
\[
\begin{align*}
x'(v) &= \exists v_1. x(v_1) \wedge n(v_1, v) \\
y'(v) &= y(v) \\
sm'(v) &= sm(v) \\
n'(v_1, v_2) &= n(v_1, v_2) \\
is'(v) &= is(v)
\end{align*}
\]
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 \( is \)

\[
\begin{array}{c}
\text{\( u_1 \)} \quad \text{\( n \)} \quad \text{\( u_2 \)} \\
\downarrow \quad \downarrow \quad \downarrow \\
\lnot is \quad \lnot is \\
\end{array}
\]

\[
\begin{array}{c}
\text{Coerce} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\( u_1 \)} \quad \text{\( n \)} \quad \text{\( u_2 \)} \\
\downarrow \quad \downarrow \quad \downarrow \\
\lnot is \quad \lnot is \\
\end{array}
\]

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 \( is \)
Recall that

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

Coerce into a more precise representation

\[
is(u_1) = is(u.1) = is(u.0) = 0
\]

Node \( u.1 \) consequently must be a definite node in order to fit with semantics of \( is \)
Coercing a list

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 \( is \)

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

this edge definitely doesn’t exist for the same reason
Coercing a list

Recall that

\[ is(u_1) = is(u.1) = is(u.0) = 0 \]

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

Coerce into a more precise representation

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
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