

Shape analysis abstractions

MPRI — Cours 2.6 “Interprétation abstraite :
application à la vérification et à l’analyse statique”

Xavier Rival

INRIA

Jan, 29th, 2021

Shape analysis

Shape analyses aim at discovering structural invariants of programs that manipulate complex unbounded data-structures

Applications:

- establish **memory safety**
- verify the preservation of **structural properties**
e.g., list, doubly-linked lists, trees, ...
- reason about programs that manipulate **unbounded** memory states

Previous course: separation logic based shape analyses

- **separating conjunction connector** $*$: ties properties that characterize **disjoint memory regions**
- also **many other connectors**:
disjunctions, classical conjunctions, separating implication...
- can be turned into **an abstract domain**

Properties to verify: examples

A program closing a list of file descriptors

```
//l points to a list
c = l;
while(c ≠ NULL){
  close(c → FD);
  c = c → next;
}
```

Correctness properties

- 1 memory safety
- 2 1 is supposed to store all file descriptors at all times
will its structure be preserved ?
yes, no breakage of a next link
- 3 closure of all the descriptors

Examples of structure preservation properties

- algorithms manipulating **trees**, **lists**...
- libraries of algorithms on **balanced trees**
- **not guaranteed by the language** !
e.g., the balancing of Maps in the OCaml standard library was **incorrect** for years (performance bug)

On today's agenda

Another important family of shape analysis abstractions:

- **three valued logic** based abstraction
 - maps predicates into “true”, “false”, “maybe” logical values
- can describe **memory states** (in this course)
 - but also **other objects** (not in this course)
- useful **comparison** with separation logic based abstraction

Combination with value abstraction:

- so far, we have considered **pointer information only**
- real states also include **numerical** and **boolean values**, but also **strings** and others...
- **issue 1**: shape abstractions are very **dynamic**
 - e.g., the scope of summaries varies during the analysis
- **issue 2**: exchange information between shape and value

Outline

- 1 Introduction
- 2 Setup (reminder)
 - Syntax and semantics
 - Basic pointer abstractions
- 3 Shape analysis in Three-Valued Logic (TVL)
- 4 Combining shape and value abstractions
- 5 Conclusion

Assumptions: syntax of programs

l	::= l-values	
	x	($x \in \mathbb{X}$)
	*e	pointer dereference
	l · f	field read
		pointers, array dereference...
e	::= expressions	
	c	($c \in \mathbb{V}$)
	l	(l-value)
	e ⊕ e	(arith operation, comparison)
	&l	"address of" operator
s	::= statements	
	l = e	(assignment)
	s; ... s;	(sequence)
	if(e){s}	(condition)
	while(e){s}	(loop)
	x = malloc(c)	allocation of c bytes
	free(x)	deallocation of the block pointed to by

Semantic domains

No one-to-one relation between memory cells and program variables

- a variable may correspond to **several cells** (structures...)
- **dynamically allocated cells** correspond to no variable at all...

Thus, we distinguish memory contents and variable addresses:

Environment + Heap

- **Addresses** are values: $\mathbb{V}_{\text{addr}} \subseteq \mathbb{V}$
- **Environments** $e \in \mathbb{E}$ map variables into their addresses
- **Heaps** ($h \in \mathbb{H}$) map addresses into values

$$\mathbb{E} = \mathbb{X} \rightarrow \mathbb{V}_{\text{addr}}$$

$$\mathbb{H} = \mathbb{V}_{\text{addr}} \rightarrow \mathbb{V}$$

h is actually only a partial function

- **Memory states** (or **memories**): $\mathbb{M} = \mathbb{E} \times \mathbb{H}$

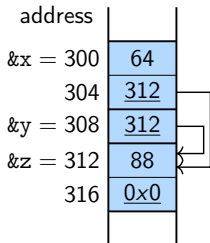
Note: **Avoid confusion between heap (function from addresses to values) and dynamic allocation space (often referred to as “heap”)**

Example of a concrete memory state (variables)

- x and z are two list elements containing values 64 and 88, and where the former points to the latter
- y stores a pointer to z

Memory layout

(pointer values underlined)

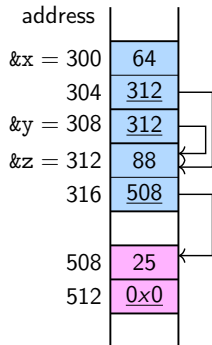


$e :$	x	\mapsto	300
	y	\mapsto	308
	z	\mapsto	312
$h :$	300	\mapsto	64
	304	\mapsto	312
	308	\mapsto	312
	312	\mapsto	88
	316	\mapsto	0

Example of a concrete memory state (variables + dyn. cell)

- same configuration
- + z points to a dynamically allocated list element (in purple)

Memory layout



e :

x	↦	300
y	↦	308
z	↦	312

h :

300	↦	64
304	↦	312
308	↦	312
312	↦	88
316	↦	508
508	↦	25
512	↦	0

Semantics of the pointer operations

Case of l-values: $\llbracket \mathbf{l} \rrbracket : \mathbb{M} \rightarrow \mathbb{V}_{\text{addr}}$

$$\begin{aligned} \llbracket \mathbf{x} \rrbracket(e, h) &= e(\mathbf{x}) \\ \llbracket *e \rrbracket(e, h) &= \begin{cases} h(\llbracket e \rrbracket(e, h)) & \text{if } \llbracket e \rrbracket(e, h) \neq 0 \wedge \llbracket e \rrbracket(e, h) \in \text{Dom}(h) \\ \Omega & \text{otherwise} \end{cases} \\ \llbracket \mathbf{l} \cdot \mathbf{f} \rrbracket(e, h) &= \llbracket \mathbf{l} \rrbracket(e, h) + \mathbf{offset}(\mathbf{f}) \text{ (numeric offset)} \end{aligned}$$

Case of expressions: $\llbracket e \rrbracket : \mathbb{M} \rightarrow \mathbb{V}$, mostly unchanged

$$\begin{aligned} \llbracket \mathbf{l} \rrbracket(e, h) &= h(\llbracket \mathbf{l} \rrbracket(e, h)) && \text{(evaluates into the contents)} \\ \llbracket \&\mathbf{l} \rrbracket(e, h) &= \llbracket \mathbf{l} \rrbracket(e, h) && \text{(evaluates into the address)} \end{aligned}$$

Case of statements that are specific to memory operations:

- **memory allocation** $\mathbf{x} = \mathbf{malloc}(c)$: $(e, h) \rightarrow (e, h')$ where $h' = h[e(\mathbf{x}) \leftarrow k] \uplus \{k \mapsto v_k, k+1 \mapsto v_{k+1}, \dots, k+c-1 \mapsto v_{k+c-1}\}$ and $k, \dots, k+c-1$ are fresh and unused in h
- **memory deallocation** $\mathbf{free}(\mathbf{x})$: $(e, h) \rightarrow (e, h')$ where $k = e(\mathbf{x})$ and $h = h' \uplus \{k \mapsto v_k, k+1 \mapsto v_{k+1}, \dots, k+c-1 \mapsto v_{k+c-1}\}$

Outline

- 1 Introduction
- 2 Setup (reminder)
 - Syntax and semantics
 - Basic pointer abstractions
- 3 Shape analysis in Three-Valued Logic (TVL)
- 4 Combining shape and value abstractions
- 5 Conclusion

Pointer non relational abstractions

Assumption on the set of values:

- $\mathbb{V} = \mathbb{V}_{\text{addr}} \uplus \dots$ and $\mathbb{X} = \mathbb{X}_{\text{addr}} \uplus \dots$
- pointer values (\mathbb{V}_{addr}) describe (either symbolic or numerical) memory addresses
- base values may include integers and other base types
- **abstract cells** \mathbb{C}^\sharp finitely summarize concrete cells, through a **fixed**

$$\phi : \mathbb{V}_{\text{addr}} \longrightarrow \mathbb{C}^\sharp$$

- we apply a **non relational abstraction**:

Non relational pointer abstraction

- Set of **pointer abstract values** $\mathbb{D}_{\text{ptr}}^\sharp$
- **Concretization** $\gamma_{\text{ptr}} : \mathbb{D}_{\text{ptr}}^\sharp \rightarrow \mathcal{P}(\mathbb{V}_{\text{addr}})$ into pointer sets
- **Abstract memory states** of the form $\mathbb{C}^\sharp \rightarrow \mathbb{D}_{\text{ptr}}^\sharp$ with
 $\gamma(m^\sharp) = \{(e, h) \mid \forall p \in \mathbb{X}_{\text{addr}}, h(e(p)) \in \gamma_{\text{ptr}} \circ m^\sharp \circ \phi(e(p))\}$

Pointer non relational abstraction: null pointers

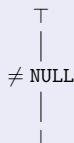
The dereference of a null pointer will cause a crash

To establish **safety**: compute **which pointers may be null**

Null pointer analysis

Abstract domain for addresses:

- $\gamma_{\text{ptr}}(\perp) = \emptyset$
- $\gamma_{\text{ptr}}(\top) = \mathbb{V}_{\text{addr}}$
- $\gamma_{\text{ptr}}(\neq \text{NULL}) = \mathbb{V}_{\text{addr}} \setminus \{0\}$



- we may also use a lattice with a fourth element = **NULL**
exercise: what do we gain using this lattice ?
- very **lightweight**, can typically resolve rather trivial cases
- useful for **C**, but also for **Java**
- we can define very similar abstractions to deal with dangling or invalid pointers

Pointer non relational abstraction: points-to sets

Determine where a pointer may store a reference to

```

1 : int x, y;
2 : int * p;
3 : y = 9;
4 : p = &x;
5 : *p = 0;

```

- what is the final value for x ?
0, since **it is modified at line 5...**
- what is the final value for y ?
9, since **it is not modified at line 5...**

Basic pointer abstraction

- We assume a set of **abstract memory locations** $\mathbb{A}^\#$ is fixed:

$$\mathbb{A}^\# = \{\&x, \&y, \dots, \&t, a_0, a_1, \dots, a_N\}$$

- **Concrete addresses** are **abstracted into** $\mathbb{A}^\#$ by $\phi_{\mathbb{A}} : \mathbb{V}_{\text{addr}} \rightarrow \mathbb{A}^\# \uplus \{\top\}$
- A pointer value is abstracted by the abstraction of the addresses it may point to, *i.e.*, $\mathbb{D}_{\text{ptr}}^\# = \mathcal{P}(\mathbb{A}^\#)$
and $\gamma_{\text{ptr}}(a^\#) = \{a \in \mathbb{V}_{\text{addr}} \mid \phi_{\mathbb{A}}(a) = a^\#\}$

- **example:** p may point to $\{\&x\}$

Points-to sets computation example

Example code:

```

1: int x, y;
2: int * p;
3: y = 9;
4: p = &x;
5: *p = 0;
6: ...

```

Abstract locations: $\{\&x, \&y, \&p\}$

Analysis results:

	$\&x$	$\&y$	$\&p$
1	\top	\top	\top
2	\top	\top	\top
3	\top	\top	\top
4	\top	$[9, 9]$	\top
5	\top	$[9, 9]$	$\{\&x\}$
6	$[0, 0]$	$[9, 9]$	$\{\&x\}$

Points-to sets computation and imprecision

```

x ∈ [-10, -5]; y ∈ [5, 10]
1: int * p;
2: if(?) {
3:   p = &x;
4: } else {
5:   p = &y;
6: }
7: *p = 0;
8: ...

```

- What is the final range for x ?
- What is the final range for y ?

Abstract locations: $\{\&x, \&y, \&p\}$

	$\&x$	$\&y$	$\&p$
1	$[-10, -5]$	$[5, 10]$	\top
2	$[-10, -5]$	$[5, 10]$	\top
3	$[-10, -5]$	$[5, 10]$	\top
4	$[-10, -5]$	$[5, 10]$	$\{\&x\}$
5	$[-10, -5]$	$[5, 10]$	\top
6	$[-10, -5]$	$[5, 10]$	$\{\&y\}$
7	$[-10, -5]$	$[5, 10]$	$\{\&x, \&y\}$
8	$[-10, 0]$	$[0, 10]$	$\{\&x, \&y\}$

Imprecise results

- The abstract information about both x and y are weakened
- The fact that $x \neq y$ is lost

Weak-updates

As in array analysis, we encounter:

Weak updates

- **The modified concrete cell cannot be uniquely mapped into a well identified abstract cell that describes only it**
- The resulting abstract information is obtained by **joining the new value and the old information**

Effect in pointer analysis, in the case of an **assignment**:

- if the points-to set contains **exactly one element**, the analysis can perform a **strong update**
- if the points-to set may contain **more than one element**, the analysis needs to perform a **weak-update**

Consequence: **weak updates cause severe losses in precision**

Previous course about memory abstraction: separation logic

Key idea:

Avoid weak updates by localizing memory accesses (read or write) in a very precise manner, and with no ambiguity

Logical items:

- **separating conjunction connector**:
logically, splits the memory into two **disjoint** regions
- **basic predicates**, to describe individual cells
- **inductive summary predicates**, that describe unbounded memory regions

Main algorithms:

- **unfolding**: to refine summary predicates
- **folding**: to synthesize summary predicates

Today: compare separation logic with another shape abstraction and augment shape analysis to describe value properties

Outline

- 1 Introduction
- 2 Setup (reminder)
- 3 Shape analysis in Three-Valued Logic (TVL)
 - Principles of Three-Valued Logic based abstraction
 - Comparing and concretizing Three-Valued Logic abstractions
 - Weakening Three-Valued Logic abstractions
 - Transfer functions
 - Focusing
 - Comparing Separation Logic and Three-Valued logic abstractions
- 4 Combining shape and value abstractions
- 5 Conclusion

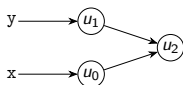
Representation of memory states: memory graphs

Observation: representation of memory states by graphs

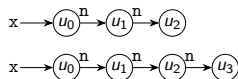
- **Nodes** (aka, atoms) denote **variables, memory locations**
- **Edges** denote **properties of addresses / pointers**, such as:
 - ▶ “field f of location u points to v ”
 - ▶ “variable x is stored at location u ”
- This representation is also relevant in the case of **separation logic** based shape abstraction

A couple of examples:

Two alias pointers:



A list of length 2 or 3:



We need to over-approximate **sets of shape graphs**

Memory graphs and predicates: variables

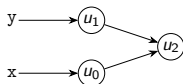
Before we apply some abstraction, we **formalize memory graphs** using some **predicates**, such as:

“Variable content” predicate

We note $x(u) = 1$ if node u represents the contents of x .

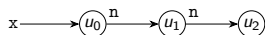
Examples:

- **Two alias pointers:**



Then, we have $x(u_0) = 1$ and $y(u_1) = 1$, and $x(u) = 0$ (*resp.*, $y(u) = 0$) in all the other cases

- **A list of length 2:**



Then, we have $x(u_0) = 1$ and $x(u) = 0$ in all the other cases

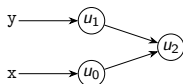
Memory graphs and predicates: (field) pointers

“Field content pointer” predicate

- We note $\mathfrak{n}(u, v)$ if the field \mathfrak{n} of u stores a pointer to v
- We note $\underline{0}(u, v)$ if u stores a pointer to v (base address field is at offset 0)

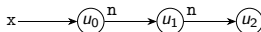
Examples:

- **Two alias pointers:**



Then, we have $\underline{0}(u_0, u_2) = 1$ and $\underline{0}(u_1, u_2) = 1$, and $\underline{0}(u, v) = 0$ in all the other cases

- **A list of length 2:**



Then, we have $\mathfrak{n}(u_0, u_1) = 1$ and $\mathfrak{n}(u_1, u_2) = 1$, and $\mathfrak{n}(u, v) = 0$ in all the other cases

2-structures and concretization

We can represent the memory graphs using **tables of predicate values**:

Two-structures and concretization

We assume a set $\mathcal{P} = \{p_0, p_1, \dots, p_n\}$ of **predicates** (we write k_i for the arity of predicate p_i). A formal representation of a memory graph is a **2-structure** $(\mathcal{U}, \phi) \in \mathbb{D}_2^\sharp$ defined by:

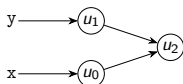
- a set $\mathcal{U} = \{u_0, u_1, \dots, u_m\}$ of **atoms**
- a **truth table** ϕ such that $\phi(p_i, u_{l_1}, \dots, u_{l_{k_i}})$ denotes the truth value of p_i for $u_{l_1}, \dots, u_{l_{k_i}}$ (where arities of predicates are respected)

Then, $\gamma_2(\mathcal{U}, \phi)$ is the set of (e, h, ν) where $\nu : \mathcal{U} \rightarrow \mathbb{V}_{\text{addr}}$ and that satisfy exactly the truth tables defined by ϕ :

- (e, h, ν) satisfies $x(u)$ iff $e(x) = \nu(u)$
- (e, h, ν) satisfies $f(u, v)$ iff $h(\nu(u), f) = \nu(v)$
- the name “two-structure” will become clear (very) soon
- the set of two-structures is parameterized by the data of a set of predicates $x(\cdot), y(\cdot), \underline{0}(\cdot, \cdot), n(\cdot, \cdot)$ (additional predicates will be added soon...)

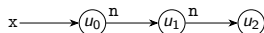
Examples of two-structures

Two alias pointers:



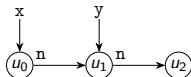
	x	y	\mapsto	u_0	u_1	u_2
u_0	1	0	u_0	0	0	1
u_1	0	1	u_1	0	0	1
u_2	0	0	u_2	0	0	0

A list of length 2:



	x	$\cdot n \mapsto$	u_0	u_1	u_2
u_0	1	u_0	0	1	0
u_1	0	u_1	0	0	1
u_2	0	u_2	0	0	0

A list of length 2:



	x	y	$\cdot n \mapsto$	u_0	u_1	u_2
u_0	1	0	u_0	0	1	0
u_1	0	1	u_1	0	0	1
u_2	0	0	u_2	0	0	0

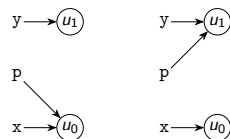
Lists of **arbitrary length** ? More on this **later**

Unknown value: three valued logic

How to abstract away some information ?

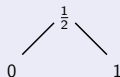
i.e., how to abstract several graphs into one ?

Example: pointer variable p alias with x or y

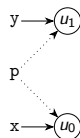


A boolean lattice

- Use **predicate tables**
- Add a \top boolean value; (denoted to by $\frac{1}{2}$ in TVLA papers)



- **Graph representation:**
dotted edges
- **Abstract graph:**



Summary nodes

At this point, we cannot talk about **unbounded memory states** with **finitely many** nodes, since one node represents at most one memory cell

An idea

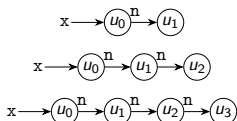
- Choose a node to represent **several** concrete nodes
- Similar to **smashing** of arrays using segments

Definition: summary node

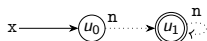
A **summary node** is an atom that may denote several concrete atoms

- intuition: we are using a **non injective function** $\phi_{\Delta} : \mathbb{V}_{\text{addr}} \longrightarrow \mathbb{A}^{\#}$
- representation: double circled nodes

Lists of lengths 1, 2, 3:



Attempt at a **summary** graph:



- Edges to u_1 are dotted

Additional graph predicate: sharing

We now define a few **higher level predicates** based on the previously seen **atomic predicates** describing the graphs.

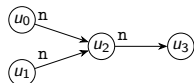
Example: a cell is **shared** if and only if there exists several distinct pointers to it

“Is shared” predicate

The predicate $\underline{\text{sh}}(u)$ holds if and only if

$$\exists v_0, v_1, \left\{ \begin{array}{l} v_0 \neq v_1 \\ \wedge \text{n}(v_0, u) \\ \wedge \text{n}(v_1, u) \end{array} \right.$$

(for concision, we assume only n pointers)



- $\underline{\text{sh}}(u_0) = \underline{\text{sh}}(u_1) = \underline{\text{sh}}(u_3) = 0$
- $\underline{\text{sh}}(u_2) = 1$

Additional graph predicate: reachability

We can also define higher level predicates **using induction**:

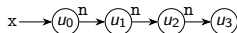
For instance, a cell is **reachable** from u if and only if it is u or it is reachable from a cell pointed to by u .

“Reachability” predicate

The predicate $\underline{r}(u, v)$ holds if and only if:

$$u = v \vee \exists u_0, \mathbf{n}(u, u_0) \wedge \underline{r}(u_0, v)$$

(for concision, we assume only \mathbf{n} pointers)



- $\underline{r}(u_1, u_0) = \underline{r}(u_2, u_0) = \underline{r}(u_3, u_1) = 0$
- $\underline{r}(u_0, u_0) = \underline{r}(u_0, u_2) = \underline{r}(u_0, u_3) = 1$

“Acyclicity” predicate

The predicate $\underline{acy}(u)$ holds iff $\exists v, v \neq u \wedge \underline{r}(u, v) \wedge \underline{r}(v, u)$ does not hold

Three structures

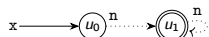
As for 2-structures, we assume a set $\mathcal{P} = \{p_0, p_1, \dots, p_n\}$ of **predicates** fixed and write k_i for the arity of predicate p_i .

Definition: 3-structures

A **3-structure** is a tuple (\mathcal{U}, ϕ) defined by:

- a set $\mathcal{U} = \{u_0, u_1, \dots, u_m\}$ of **atoms**
- a **truth table** ϕ such that $\phi(p_i, u_{l_1}, \dots, u_{l_{k_i}})$ denotes the truth value of p_i for $u_{l_1}, \dots, u_{l_{k_i}}$
note: truth values are elements of the lattice $\{0, \frac{1}{2}, 1\}$

We write \mathbb{D}_3^\sharp for the set of three-structures.



$$\begin{cases} \mathcal{U} = \{u_0, u_1\} \\ \mathcal{P} = \{x(\cdot), n(\cdot, \cdot), \underline{\text{sum}}(\cdot)\} \end{cases}$$

	x	<u>sum</u>	n	u_0	u_1
u_0	1	0	u_0	0	$\frac{1}{2}$
u_1	0	$\frac{1}{2}$	u_1	0	$\frac{1}{2}$

In the following we build up an abstract domain of 3-structures (but a bit more work is need for the concretization)

Main predicates and concretization

We have already seen:

$x(u)$	variable x contains the address of u
$n(u, v)$	field of u points to v
$\text{sum}(u)$	whether u is a summary node (convention: either 0 or $\frac{1}{2}$)
$\text{sh}(u)$	whether there exists several distinct pointers to u
$\text{r}(u, v)$	whether v is reachable starting from u
$\text{acy}(v)$	v may not be on a cycle

Concretization for 2 structures:

$$(e, h, \nu) \in \gamma_2(\mathcal{U}, \phi) \iff \bigwedge_{p \in \mathcal{P}} (env, h, \nu) \text{ evaluates } p \text{ as specified in } \phi$$

Concretization for 3 structures:

- predicates with value $\frac{1}{2}$ may concretize either to true or to false
- but the concretization of summary nodes is still unclear...

Outline

- 1 Introduction
- 2 Setup (reminder)
- 3 Shape analysis in Three-Valued Logic (TVL)
 - Principles of Three-Valued Logic based abstraction
 - Comparing and concretizing Three-Valued Logic abstractions
 - Weakening Three-Valued Logic abstractions
 - Transfer functions
 - Focusing
 - Comparing Separation Logic and Three-Valued logic abstractions
- 4 Combining shape and value abstractions
- 5 Conclusion

Embedding

Reasons why we need to set up a **relation among structures**:

- learn how to **compare** two 3-structures
- describe the **concretization** of 3-structures into 2-structures

The embedding principle

Let $\mathcal{S}_0 = (\mathcal{U}_0, \phi_0)$ and $\mathcal{S}_1 = (\mathcal{U}_1, \phi_1)$ be two three structures, with the same sets of predicates \mathcal{P} . Let $f : \mathcal{U}_0 \rightarrow \mathcal{U}_1$, surjective.

We say that f **embeds** \mathcal{S}_0 **into** \mathcal{S}_1 iff

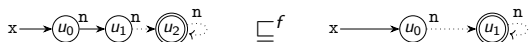
$$\text{for all predicate } p \in \mathcal{P} \text{ of arity } k, \quad \text{for all } u_{l_1}, \dots, u_{l_{k_i}} \in \mathcal{U}_0, \\ \phi_0(u_{l_1}, \dots, u_{l_{k_i}}) \sqsubseteq \phi_1(f(u_{l_1}), \dots, f(u_{l_{k_i}}))$$

Then, **we write** $\mathcal{S}_0 \sqsubseteq^f \mathcal{S}_1$

- Note: we use the order \sqsubseteq of the lattice $\{0, \frac{1}{2}, 1\}$
- Intuition: **embedding** defines an **abstract pre-order**
i.e., when $\mathcal{S}_0 \sqsubseteq^f \mathcal{S}_1$, any property that is satisfied by \mathcal{S}_0 is also satisfied by \mathcal{S}_1

Embedding examples

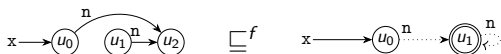
A few examples of the embedding relation:



where $f : u_0 \mapsto u_0; u_1 \mapsto u_1; u_2 \mapsto u_1$



where $f : u_0 \mapsto u_0; u_1 \mapsto u_1; u_2 \mapsto u_1; u_3 \mapsto u_1$



where $f : u_0 \mapsto u_0; u_1 \mapsto u_1; u_2 \mapsto u_1$

The last example shows summary nodes are not enough to capture just lists:

- **reachability** would be necessary to constrain it be a list
- alternatively: list cells **should not be shared**

Concretization of three-structures

Intuitions:

- concrete memory states correspond to 2-structures
- embedding applies uniformly to 2-structures and 3-structures (in fact, 2-structures are a subset of 3-structures)
- 2-structures can be embedded into 3-structures, that abstract them

This suggests **a concretization of 3-structures in two steps:**

- 1 turn it into a set of 2-structures that can be embedded into it
- 2 concretize these 2-structures

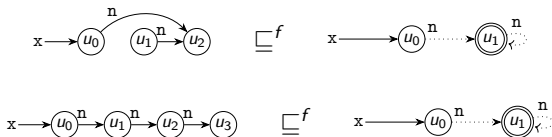
Concretization of 3-structures

Let \mathcal{S} be a 3-structure. Then:

$$\gamma_3(\mathcal{S}) = \bigcup \{ \gamma_2(\mathcal{S}') \mid \mathcal{S}' \text{ 2-structure s.t. } \exists f, \mathcal{S}' \sqsubseteq^f \mathcal{S} \}$$

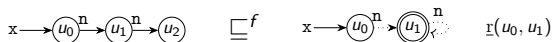
Concretization examples

Without reachability:



where $f : u_0 \mapsto u_0; u_1 \mapsto u_1; u_2 \mapsto u_1; u_3 \mapsto u_1$

With reachability:



where $f : u_0 \mapsto u_0; u_1 \mapsto u_1; u_2 \mapsto u_1$

Disjunctive completion

- Do 3-structures allow for a **sufficient level of precision** ?
- How to **over-approximate a set of 2-structures** ?

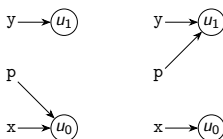
```

int * x; int * y; ...
int * p = NULL;
if(..){
  p = x;
}else{
  p = y;
}
printf("%d", *p);
*p = ...;

```

After the if statement:

abstracting would be imprecise



Abstraction based on disjunctive completion

- In the following, we use **partial disjunctive completion** *i.e.*, TVLA manipulates **finite disjunctions** of 3-structures
We write $\mathbb{D}_{\mathcal{P}(3)}^\sharp$ for the abstract domain made of finite sets of 3-structures in \mathbb{D}_3^\sharp
- How to ensure disjunctions **will not grow infinite** ?
the set of atoms is **unbounded**, so it is not necessarily true!

Outline

- 1 Introduction
- 2 Setup (reminder)
- 3 Shape analysis in Three-Valued Logic (TVL)
 - Principles of Three-Valued Logic based abstraction
 - Comparing and concretizing Three-Valued Logic abstractions
 - Weakening Three-Valued Logic abstractions
 - Transfer functions
 - Focusing
 - Comparing Separation Logic and Three-Valued logic abstractions
- 4 Combining shape and value abstractions
- 5 Conclusion

Canonical abstraction

To **prevent disjunctions** from growing infinite, we propose to **normalize (in a precision losing manner)** abstract states:

- the analysis may use all 3-structures at most points
- at selected points (including loop heads), only 3-structures in a finite set $\mathbb{D}_{\text{can}(3)}^\#$ are allowed
- there is a function to coarsen 3-structures into elements of $\mathbb{D}_{\text{can}(3)}^\#$

Canonicalization function

Let \mathcal{L} be a lattice, $\mathcal{L}' \subseteq \mathcal{L}$ be a finite sub-lattice and $\text{can} : \mathcal{L} \rightarrow \mathcal{L}'$:

- operator **can** is called **canonicalization** if and only if it defines an **upper closure operator**
- then it extends into a **canonicalization operator** $\text{can} : \mathcal{P}(\mathcal{L}) \rightarrow \mathcal{P}(\mathcal{L}')$ for the disjunctive completion domain:

$$\text{can}(\mathcal{E}) = \{\text{can}(x) \mid x \in \mathcal{E}\}$$

- proof of the extension two disjunctive completion domains: left as an exercise
- to make the powerset domain work, we simply need a **can** over 3-structures

Canonical abstraction

Definition of a finite lattice $\mathbb{D}_{\text{can}(3)}^\#$

We partition the set of predicates \mathcal{P} into two subsets \mathcal{P}_a and \mathcal{P}_o :

- \mathcal{P}_a defines **abstraction predicates** and should contain only unary predicates and have a finite truth table whatever the number of atoms
- \mathcal{P}_o denotes **non-abstraction predicates**, and may define truth tables of unbounded size

Then, we let $\mathbb{D}_{\text{can}(3)}^\#$ be the set of 3-structures such that **no pair of atoms have the same value of the \mathcal{P}_a predicates**. It defines a finite set of 3-structures.

This sub-lattice defines a clear “canonicalization” algorithm:

Canonical abstraction by truth blurring

- 1 **Identify** nodes that **have different abstraction predicates**
- 2 When several nodes have the **same abstraction predicate** **introduce a summary node**
- 3 **Compute new predicate values** by doing a **join over truth values**

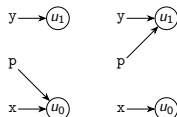
Canonical abstraction examples

Most common TVLA instantiation:

- we assume there are n variables x_1, \dots, x_n
thus the number of unary predicates is finite, and provides a good choice for \mathcal{P}_a
- **sub-lattice**: structures with atoms **distinguished by the values of the unary predicates** x_1, \dots, x_n

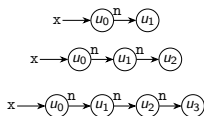
Examples:

Elements not merged:

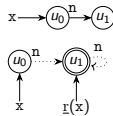


Elements merged:

Lists of lengths 1, 2, 3:



Abstract into:



Outline

- 1 Introduction
- 2 Setup (reminder)
- 3 Shape analysis in Three-Valued Logic (TVL)
 - Principles of Three-Valued Logic based abstraction
 - Comparing and concretizing Three-Valued Logic abstractions
 - Weakening Three-Valued Logic abstractions
 - Transfer functions
 - Focusing
 - Comparing Separation Logic and Three-Valued logic abstractions
- 4 Combining shape and value abstractions
- 5 Conclusion

Principle for the design of sound transfer functions

- Intuitively, **concrete states** correspond to **2-structures**
- The **analysis** should track **3-structures**, thus the analysis and its soundness proof need to **rely on the embedding relation**

Embedding theorem

We assume that

- $\mathcal{S}_0 = (\mathcal{U}_0, \phi_0)$ and $\mathcal{S}_1 = (\mathcal{U}_1, \phi_1)$ define a pair of 3-structures
- $f : \mathcal{U}_0 \rightarrow \mathcal{U}_1$, is such that $\mathcal{S}_0 \sqsubseteq^f \mathcal{S}_1$ (embedding)
- Ψ is a logical formula, with variables in X
- $g : X \rightarrow \mathcal{U}_0$ is an assignment for the variables of Ψ

Then, the semantics (evaluation) of logical formulae is such that

$$\llbracket \Psi \rrbracket_{|g}(\mathcal{S}_0) \sqsubseteq \llbracket \Psi \rrbracket_{|f \circ g}(\mathcal{S}_1)$$

Intuition: this theorem ties the evaluation of conditions in the concrete and in the abstract in a general manner

Principle for the design of sound transfer functions

Transfer functions for static analysis

- **Semantics of concrete statements is encoded into boolean formulas**
- **Evaluation in the abstract is sound (embedding theorem)**

Example: analysis of an assignment $y := x$

- 1 let y' be a new predicate that denotes the *new* value of y
- 2 then we can add the constraint $y'(u) = x(u)$
(using the embedding theorem to prove soundness)
- 3 rename y' into y

Advantages:

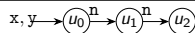
- **abstract transfer functions** derive directly from the concrete transfer functions **(intuition: $\alpha \circ f \circ \gamma \dots$)**
- the same solution works for **weakest pre-conditions**

Disadvantage: precision will require some care, more on this later!

Assignment: a simple case

Statement $l_0 : y = y -> n; l_1 : \dots$

Pre-condition \mathcal{S}

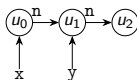


Transfer function computation:

- it should produce an over-approximation of $\{m_1 \in \mathbb{M} \mid (l_0, m_0) \rightarrow (l_1, m_1)\}$
- encoding** using “**primed predicates**” to denote predicates **after** the evaluation of the assignment, to evaluate them in the same structure (non primed variables are removed afterwards and primed variables renamed):

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

- resulting structure:**



This is exactly the expected result

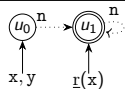
Outline

- 1 Introduction
- 2 Setup (reminder)
- 3 Shape analysis in Three-Valued Logic (TVL)**
 - Principles of Three-Valued Logic based abstraction
 - Comparing and concretizing Three-Valued Logic abstractions
 - Weakening Three-Valued Logic abstractions
 - Transfer functions
 - Focusing
 - Comparing Separation Logic and Three-Valued logic abstractions
- 4 Combining shape and value abstractions
- 5 Conclusion

Assignment: a more involved case

Statement $l_0 : y = y \rightarrow n; l_1 : \dots$

Pre-condition \mathcal{S}



- Let us try to **resolve the update in the same way as before**:

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

- We **cannot resolve y'** :

$$\begin{cases} y'(u_0) = 0 \\ y'(u_1) = \frac{1}{2} \end{cases}$$

Imprecision: after the statement, y may point to anywhere in the list, save for the first element...

- The assignment transfer function **cannot be computed immediately**
- We need to refine the 3-structure first**

Focus

Focusing on a formula

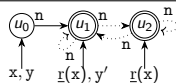
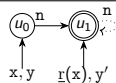
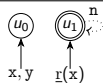
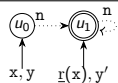
We assume a 3-structure \mathcal{S} and a boolean formula f are given, we call a **focusing** \mathcal{S} on f the generation of a set $\hat{\mathcal{S}}$ of 3-structures such that:

- f **evaluates to 0 or 1** on all elements of $\hat{\mathcal{S}}$
- **precision was gained:** $\forall S' \in \hat{\mathcal{S}}, S' \sqsubseteq \mathcal{S}$ (embedding)
- **soundness is preserved:** $\gamma(\mathcal{S}) = \bigcup \{ \gamma(S') \mid S' \in \hat{\mathcal{S}} \}$

- Details of focusing algorithms are rather complex: not detailed here
- They involve splitting of summary nodes, solving of boolean constraints

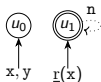
Example: focusing on
 $y'(u) = \exists v, y(v)$
 $\wedge n(v, u)$

We obtain (we show y and y'):

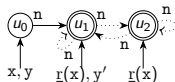


Focus and coerce

Some of the 3-structures generated by focus are not precise



u_1 is reachable from x , but there is no sequence of n fields: this structure has **empty concretization**

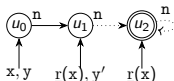
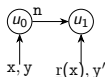


u_0 has an n -field to u_1 so u_1 denotes a unique atom and **cannot be a summary node**

Coerce operation

It **enforces logical constraints** among predicates and discards 3-structures with an empty concretization

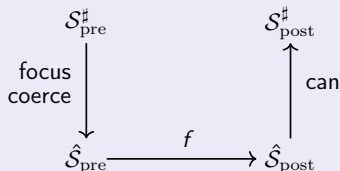
Result: one case removed (bottom), two possibly summary nodes non summary



Focus, transfer, abstract...

Computation of a transfer function

We consider a transfer function encoded into boolean formula f



Soundness proof steps:

- ① **sound encoding of the semantics of program statements into formulas** (typically, no loss of precision at this stage)
- ② **focusing** produces a **refined** over-approximation (disjunction)
- ③ **canonicalization over-approximates graphs** (truth blurring)

A common picture in shape analysis

Shape analysis with three valued logic

Abstract states; two abstract domains are used:

- **infinite domain** $\mathbb{D}_{\mathcal{P}(3)}^\#$: finite disjunctions of 3-structures in $\mathbb{D}_3^\#$ for general abstract computations
- **finite domain** $\mathbb{D}_{\mathcal{P}(\text{can}(3))}^\#$: disjunctions of finite domain $\mathbb{D}_{\text{can}(3)}^\#$ to simplify abstract states and for loop iteration
- **concretization** via $\mathbb{D}_2^\#$

Abstract post-conditions:

- 1 start from $\mathbb{D}_{\mathcal{P}(3)}^\#$ or $\mathbb{D}_{\text{can}(3)}^\#$
- 2 focus and coerce when needed
- 3 apply the concrete transformation
- 4 apply can to weaken abstract states; result in $\mathbb{D}_{\mathcal{P}(\text{can}(3))}^\#$

Analysis of loops:

- iterations in $\mathbb{D}_{\mathcal{P}(\text{can}(3))}^\#$ terminate, as it is finite

Outline

1 Introduction

2 Setup (reminder)

3 Shape analysis in Three-Valued Logic (TVL)

- Principles of Three-Valued Logic based abstraction
- Comparing and concretizing Three-Valued Logic abstractions
- Weakening Three-Valued Logic abstractions
- Transfer functions
- Focusing
- Comparing Separation Logic and Three-Valued logic abstractions

4 Combining shape and value abstractions

5 Conclusion

Separation logic

Separation logic formulas (main connectors only)

$$\begin{array}{l}
 F ::= \text{emp} \\
 \quad | \text{TRUE} \\
 \quad | l \mapsto l \\
 \quad | F_0 * F_1 \\
 \quad | F_0 \wedge F_1 \\
 \quad | F_0 \text{---} * F_1
 \end{array}$$

Concretization:

$$\begin{aligned}
 \gamma(\text{emp}) &= \mathbb{E} \times \{\emptyset\} \\
 \gamma(\text{TRUE}) &= \mathbb{E} \times \mathbb{H} \\
 \gamma(l \mapsto v) &= \{(e, \llbracket l \rrbracket(e, h) \mapsto v) \mid e \in \mathbb{E}\} \\
 \gamma(F_0 * F_1) &= \{(e, h_0 \otimes h_1) \mid (e, h_0) \in \gamma(F_0) \wedge (e, h_1) \in \gamma(F_1)\} \\
 \gamma(F_0 \wedge F_1) &= \gamma(F_0) \cap \gamma(F_1) \\
 \gamma(F_0 \text{---} * F_1) &= \text{exercise}
 \end{aligned}$$

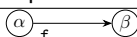

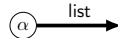
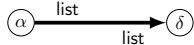
Program reasoning: **frame rule** and **strong updates**

Shape graphs and separation logic

Shape graphs: provide an efficient data-structure to describe a **subset** of separation logic predicates, and do static analysis with them.

Important addition: **inductive predicates.**

Semantic preserving translation Π of graphs into separation logic formulas:

Graph $S^\# \in \mathbb{D}_{sh}^\#$	Translated formula $\Pi(S^\#)$
	$\alpha \cdot f \mapsto \beta$
	$\Pi(S_0^\#) * \Pi(S_1^\#)$
	$\alpha \cdot list$
	$\alpha \cdot list_endp(\delta)$
other inductives and segments	similar

Note that:

- shape graphs can be encoded into separation logic formula
- the opposite is usually not true

Comparing the structure of abstract formulae

Separation logic:

$$F_0 * F_1 * \dots * F_n$$

- first the heap is partitioned
- each region is described separately
- some of the F_i components may be summary predicates, describing unbounded regions
- reachability is implicit
- allows local reasoning

Three valued logic:

$$p_0 \wedge p_1 \wedge \dots \wedge p_n$$

- first a conjunction of properties
- each predicate p_i may talk about any heap region
- no direct heap partitioning
- reachability can be expressed (natively)
- no local reasoning

Two very different sets of predicates

- one allows local reasoning, the other not
- the other way for reachability predicates

Summarization: one abstract cell, many concrete cells

Large / unbounded numbers of concrete cells need to be abstracted

- **Dynamic structures** (lists, trees) have an unknown and unbounded number of cells, hence require summarization
- We also needed summaries to deal with **arrays**

Summary

A **summary predicate** allows to describe an **unbounded number** of memory locations using a fixed, finite set of predicates

Principles underlying summarization:

- in **separation logic**:
using inductive definitions for lists, trees...
unbounded size of the summarized region is hidden in the **recursion**
- in **three-valued logic**:
summary nodes + high level predicates (such as reachability)
one summary node **carries the properties** of an unbounded number of cells

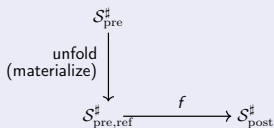
Concretize partially, update, abstract

For precise analysis, summaries need to be (temporarily) refined

Separation logic:

Local (partial) concretization

For **materialization**:

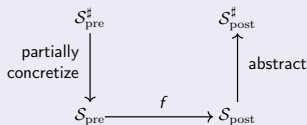


In both cases, two mechanisms are needed:

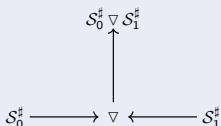
- 1 refine summaries
- 2 synthesize summaries

TVLA:

Focus, analyze, canonicalize



Global abstraction: widening



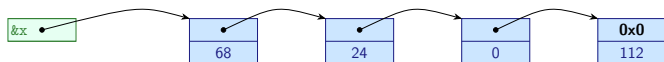
Outline

- 1 Introduction
- 2 Setup (reminder)
- 3 Shape analysis in Three-Valued Logic (TVL)
- 4 Combining shape and value abstractions**
 - Shape and value properties
 - Combined abstraction with cofibered abstract domain
 - Combined analysis algorithms
- 5 Conclusion

Shape and value properties

Common data-structures require to reason both about shape and data:

- **hybrid stores:** data stored next to inductive structures
- **list of even elements:**



- **sorted list:**



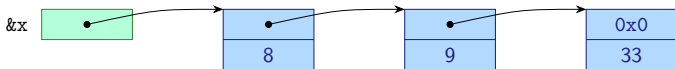
- **list with a length constraint**
- **tries:** binary trees with paths labelled with sequences of “0” and “1”
- **balanced trees:** red-black, AVL...

This part of the course:

- how to **express both shape and numerical properties ?**
- how to **extend shape analysis algorithms**

Description of a sorted list

- **Example:** sorted list



Inductive definition

- Each element should be greater than the previous one
- The first element simply needs be greater than $-\infty$...
- We need to propagate **the lower bound**, using a scalar parameter

$$\alpha \cdot \text{l-sort}_{\text{aux}}(n) := \begin{array}{l} \alpha = 0 \wedge \text{emp} \\ \vee \alpha \neq 0 \wedge n \leq \beta \wedge \alpha \cdot \text{next} \mapsto \delta \\ * \alpha \cdot \text{data} \mapsto \beta * \delta \cdot \text{l-sort}_{\text{aux}}(\beta) \end{array}$$

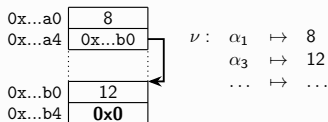
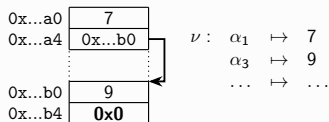
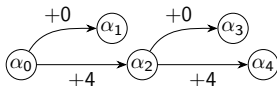
$$\alpha \cdot \text{l-sort}() := \alpha \cdot \text{l-sort}_{\text{aux}}(-\infty)$$

Adding value information (here, numeric)

**Concrete numeric values appear in the valuation
thus the abstracting contents boils down to abstracting ν !**

Example: all lists of length 2, sorted in the increasing order of data fields

Memory abstraction:



Abstraction of valuations: $\nu(\alpha_1) < \nu(\alpha_3)$, can be described by the constraint $\alpha_1 < \alpha_3$

A first step towards a combined domain

Domains and their concretization:

- **shape abstract domain** $\mathbb{D}_{sh}^\#$ of graphs
 abstract stores together with a **physical mapping** of nodes

$$\gamma_{sh} : \mathbb{D}_{sh}^\# \rightarrow \mathcal{P}((\mathbb{D}_{sh}^\# \rightarrow \mathbb{M}) \times (\mathbb{V}^\# \rightarrow \mathbb{V}))$$
- **numerical abstract domain** $\mathbb{D}_{num}^\#$, abstracts physical mapping of nodes

$$\gamma_{num} : \mathbb{D}_{num}^\# \rightarrow \mathcal{P}(\mathbb{V}^\# \rightarrow \mathbb{V})$$

Combined domain [CR]

- **Set of abstract values:** $\mathbb{D}^\# = \mathbb{D}_{sh}^\# \times \mathbb{D}_{num}^\#$
- **Concretization:**

$$\gamma(S^\#, N^\#) = \{(\ell, \nu) \in \mathbb{M} \mid \nu \in \gamma_{num}(N^\#) \wedge (\ell, \nu) \in \gamma_{sh}(S^\#)\}$$

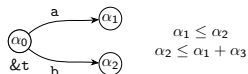
Can it be described as a reduced product ?

- **product abstraction:** $\mathbb{D}^\# = \mathbb{D}_0^\# \times \mathbb{D}_1^\#$ (componentwise ordering)
- **concretization:** $\gamma(x_0, x_1) = \gamma(x_0) \cap \gamma(x_1)$
- **reduction:** $\mathbb{D}_r^\#$ is the quotient of $\mathbb{D}^\#$ by the equivalence relation \equiv defined by
 $(x_0, x_1) \equiv (x'_0, x'_1) \iff \gamma(x_0, x_1) = \gamma(x'_0, x'_1)$

Formalizing the product domain

The use of a simple reduced product raises several issues

Elements without a clear meaning:



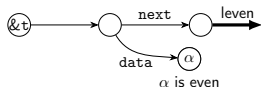
$$\alpha_1 \leq \alpha_2$$

$$\alpha_2 \leq \alpha_1 + \alpha_3$$

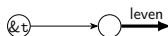
- this element exists in the reduced product domain (independent components)
- but, ... **what is α_3 ?**

Unclear comparison:

How can we compare the two elements below ?



and



- in the reduced product domain, they are **not comparable**:
nodes do not match, so componentwise comparison does not make sense
- when concretizing them, there is **clear inclusion**

Towards a more adapted combination operator

Reason why the reduced product construction does not work well:

- the set of nodes / symbolic variables **is not fixed**
 - the set of dimensions in the numerical domain depends on the shape abstraction
- ⇒ **thus the product is not symmetric**
 however, the reduced product construction is symmetric

Intuitions

- Graphs form a **shape domain** $\mathbb{D}_{\text{sh}}^{\#}$
- For **each** graph $S^{\#} \in \mathbb{D}_{\text{sh}}^{\#}$, we have a **numerical lattice** $\mathbb{D}_{\text{num}\langle S^{\#} \rangle}^{\#}$
 - ▶ example: if graph $S^{\#}$ contains nodes $\alpha_0, \alpha_1, \alpha_2$, $\mathbb{D}_{\text{num}\langle S^{\#} \rangle}^{\#}$ should abstract $\{\alpha_0, \alpha_1, \alpha_2\} \rightarrow \mathbb{V}$
- **An abstract value is a pair** $(S^{\#}, N^{\#})$, such that $N^{\#} \in \mathbb{D}_{\text{num}\langle N^{\#} \rangle}^{\#}$

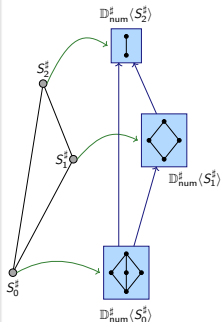
Outline

- 1 Introduction
- 2 Setup (reminder)
- 3 Shape analysis in Three-Valued Logic (TVL)
- 4 Combining shape and value abstractions**
 - Shape and value properties
 - Combined abstraction with cofibered abstract domain
 - Combined analysis algorithms
- 5 Conclusion

Cofibered domain

Definition, for shape + num

- **Basis:** abstract domain $(\mathbb{D}_{\text{sh}}^{\#}, \sqsubseteq^{\#})$, with concretization $\gamma_{\text{sh}} : \mathbb{D}_{\text{sh}}^{\#} \rightarrow \mathbb{D}$
- **Function:** $\phi : \mathbb{D}_{\text{sh}}^{\#} \rightarrow \mathcal{D}$, where each element of \mathcal{D} is an abstract domain instance $(\mathbb{D}_{\text{num}}^{\#}, \sqsubseteq_{\text{num}}^{\#})$, with a concretization $\gamma_{\text{num}} : \mathbb{D}_{\text{num}}^{\#} \rightarrow \mathbb{D}$ (**tied to a shape graph**)
- **Domain $\mathbb{D}^{\#}$:** set of **pairs $(S^{\#}, N^{\#})$ where $N^{\#} \in \phi(S^{\#})$**
- **Concretization:** $\gamma(S^{\#}, N^{\#}) = \gamma(S^{\#}) \cap \gamma(N^{\#})$
- **Lift functions:** $\forall S_0^{\#}, S_1^{\#} \in \mathbb{D}_{\text{sh}}^{\#}$, such that $S_0^{\#} \sqsubseteq^{\#} S_1^{\#}$, there exists a function $\Pi_{S_0^{\#}, S_1^{\#}} : \phi(S_0^{\#}) \rightarrow \phi(S_1^{\#})$, that is monotone for $\gamma_{S_0^{\#}}$ and $\gamma_{S_1^{\#}}$

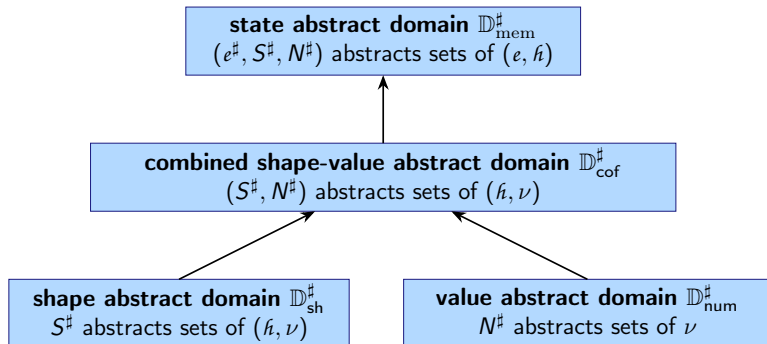


- General construction presented in **[AV]** (Arnaud Venet)
- Intuition: a **dependent domain product**

Overall abstract domain structure

Implementation exploiting the modular structure

- **Each layer** accounts for one **aspect of the concrete states**
- **Each layer** boils down to a **module or functor in ML**



How about operations, transfer functions? Also to be modularly defined

Domain operations

The cofibered structure allows to define **standard domain operations**:

- **ift functions** allow to **switch domain when needed**
- computations first done in the basis, then in the numerical domains, after lifting, when needed

Comparison of $(S_0^\#, N_0^\#)$ and $(S_1^\#, N_1^\#)$

- 1 First, **compare** $S_0^\#$ and $S_1^\#$ in $\mathbb{D}_{sh}^\#$
- 2 If $S_0^\# \sqsubseteq^\# S_1^\#$, **compare** $\Pi_{S_0^\#, S_1^\#}(N_0^\#)$ and $N_1^\#$

Widening of $(S_0^\#, N_0^\#)$ and $(S_1^\#, N_1^\#)$

- 1 First, compute the **widening in the basis** $S^\# = S_0^\# \nabla S_1^\#$
- 2 Then **move to** $\phi(S^\#)$, by computing $N_{0c}^\# = \Pi_{S_0^\#, S^\#}(N_0^\#)$ and $N_{1c}^\# = \Pi_{S_1^\#, S^\#}(N_1^\#)$
- 3 Last **widen in** $\phi(S^\#)$: $N^\# = N_{0c}^\# \nabla_{S^\#} N_{1c}^\#$
- 4 Return $(S_0^\#, N_0^\#) \nabla (S_1^\#, N_1^\#) = (S^\#, N^\#)$

Outline

1 Introduction

2 Setup (reminder)

3 Shape analysis in Three-Valued Logic (TVL)

4 **Combining shape and value abstractions**

- Shape and value properties
- Combined abstraction with cofibered abstract domain
- Combined analysis algorithms

5 Conclusion

Domain operations and transfer functions

Abstract assignments, condition tests:

- need to modify both the shape abstraction and the value abstraction
- both modification are interdependent

Typical process to compute abstract post-conditions

- 1 compute the post in the shape abstract domain and update the basis
- 2 update the value abstraction (numerics) to model dimensions additions and removals
- 3 compute the post in the value abstract domain

Proofs of soundness of transfer functions rely on:

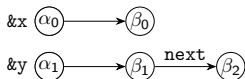
- the soundness of the lift functions
- the soundness of both domain transfer functions

Analysis of an assignment in the graph domain

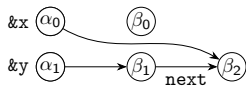
Steps for analyzing $x = y \rightarrow \text{next}$ (local reasoning)

- 1 Evaluate **l-value** x into **points-to edge** $\alpha \mapsto \beta$
- 2 Evaluate **r-value** $y \rightarrow \text{next}$ into **node** β'
- 3 Replace points-to edge $\alpha \mapsto \beta$ with **points-to edge** $\alpha \mapsto \beta'$

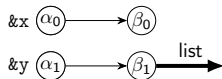
With pre-condition:



- Step 1 produces $\alpha_0 \mapsto \beta_0$
- Step 2 produces β_2
- End result:

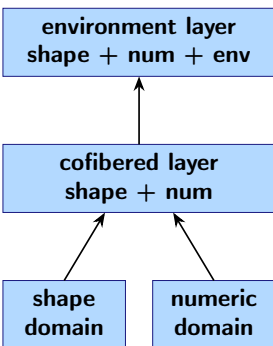


With pre-condition:



- Step 1 produces $\alpha_0 \mapsto \beta_0$
- **Step 2 can succeed only after unfolding is performed**

Analysis of an assignment in the combined domain



$$\&x \ (\alpha_0) \longrightarrow (\alpha_1)$$

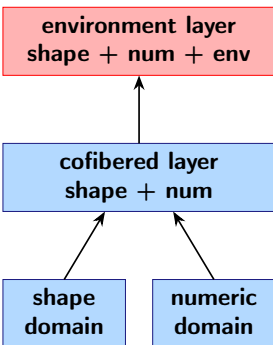
$$\&y \ (\alpha_2) \longrightarrow (\alpha_3) \xrightarrow{\text{lpos}}$$

$$N^\# = \alpha_1 \geq 0 \wedge \alpha_3 \neq \mathbf{0x0}$$

$$y \rightarrow d = x + 1$$

Abstract post-condition ?

Analysis of an assignment in the combined domain



$$\&x \quad (\alpha_0) \longrightarrow (\alpha_1)$$

$$\&y \quad (\alpha_2) \longrightarrow (\alpha_3) \xrightarrow{\text{pos}}$$

$$N^\# = \alpha_1 \geq 0 \wedge \alpha_3 \neq \mathbf{0x0}$$

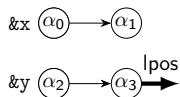
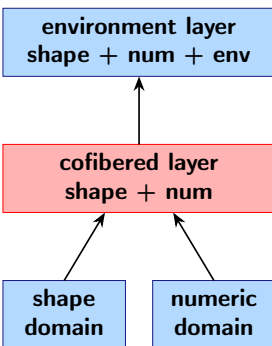
$$y \rightarrow d = x + 1 \quad \Rightarrow \quad (*\alpha_2) \cdot d = (*\alpha_0) + 1$$

Abstract post-condition ?

Stage 1: environment resolution

- replaces x with $*e^\#(x)$

Analysis of an assignment in the combined domain



$$N^\# = \alpha_1 \geq 0 \wedge \alpha_3 \neq \mathbf{0x0}$$

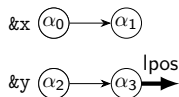
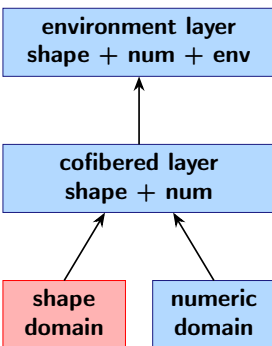
$$(*\alpha_2) \cdot \mathbf{d} = (*\alpha_0) + 1$$

Abstract post-condition ?

Stage 2: propagate into the shape + numerics domain

- only symbolic nodes appear

Analysis of an assignment in the combined domain



$$N^\# = \alpha_1 \geq 0 \wedge \alpha_3 \neq \mathbf{0x0}$$

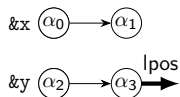
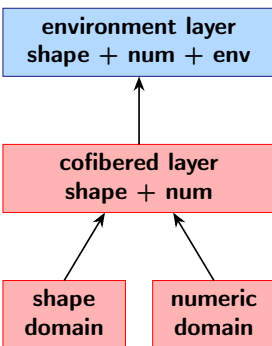
$$(*\alpha_2) \cdot \mathbf{d} = (*\alpha_0) + 1$$

Abstract post-condition ?

Stage 3: resolve cells in the shape graph abstract domain

- $*\alpha_0$ evaluates to α_1 ; $*\alpha_2$ evaluates to α_3
- $(*\alpha_2) \cdot \mathbf{d}$ **fails to evaluate: no points-to out of α_3**

Analysis of an assignment in the combined domain



$$N^\# = \alpha_1 \geq 0 \wedge \alpha_3 \neq \mathbf{0x0}$$

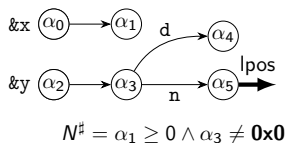
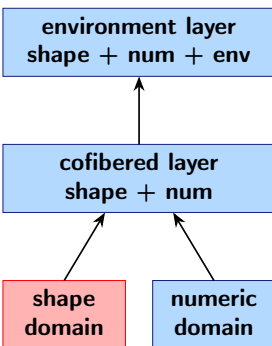
$$(*\alpha_2) \cdot \mathbf{d} = (*\alpha_0) + 1$$

Abstract post-condition ?

Stage 4 (a): unfolding triggered

- the analysis needs to locally materialize $\alpha_3 \cdot \mathbf{lpos}$...
- thus, unfolding starts at symbolic variable α_3

Analysis of an assignment in the combined domain



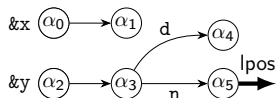
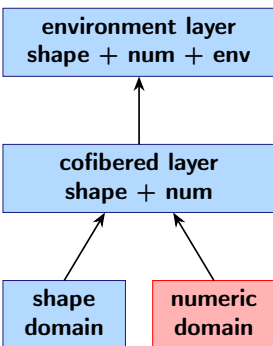
$$(*\alpha_2) \cdot d = (*\alpha_0) + 1$$

Abstract post-condition ?

Stage 4 (b): unfolding, shape part

- unfolding of the memory predicate part
- numerical predicates still need be taken into account

Analysis of an assignment in the combined domain



$$N^\# = \alpha_1 \geq 0 \wedge \alpha_3 \neq \mathbf{0x0} \wedge \alpha_4 \geq 0$$

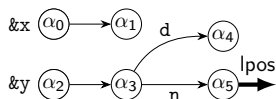
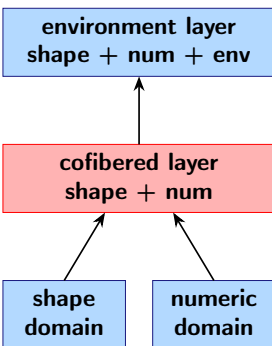
$$(*\alpha_2) \cdot d = (*\alpha_0) + 1$$

Abstract post-condition ?

Stage 4 (c): unfolding, numeric part

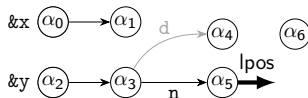
- numerical predicates taken into account
- l-value $\alpha_3 \cdot d$ **now evaluates into edge** $\alpha_3 \cdot d \mapsto \alpha_4$

Analysis of an assignment in the combined domain



$$N^\# = \alpha_1 \geq 0 \wedge \alpha_3 \neq \mathbf{0x0} \wedge \alpha_4 \geq 0$$

create node α_6

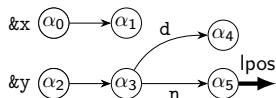
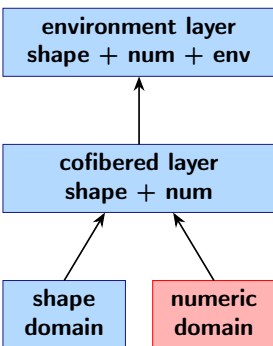


$$N^\# = \alpha_1 \geq 0 \wedge \alpha_3 \neq \mathbf{0x0} \wedge \alpha_4 \geq 0$$

Stage 5: create a new node

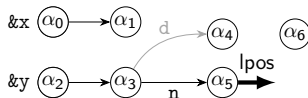
- new node α_6 denotes a new value
will store the new value

Analysis of an assignment in the combined domain



$$N^\# = \alpha_1 \geq 0 \wedge \alpha_3 \neq \mathbf{0x0} \wedge \alpha_4 \geq 0$$

$\alpha_6 \leftarrow \alpha_1 + 1$ in numerics

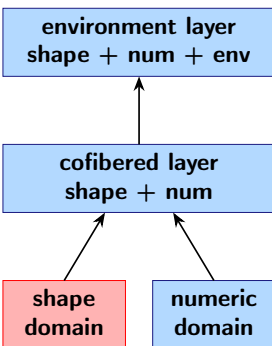


$$N^\# = \alpha_1 \geq 0 \wedge \alpha_3 \neq \mathbf{0x0} \wedge \alpha_4 \geq 0 \wedge \alpha_6 \geq 1$$

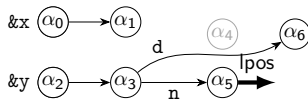
Stage 6: perform numeric assignment

- numeric assignment **completely ignores pointer structures** to the new node

Analysis of an assignment in the combined domain



mutate $(\alpha_3 \cdot d) \mapsto \alpha_4$ into α_6



$$N^\# = \alpha_1 \geq 0 \wedge \alpha_3 \neq \mathbf{0x0} \wedge \alpha_4 \geq 0 \wedge \alpha_6 \geq 1$$

Stage 7: perform the update in the graph

- classic **strong update** in a pointer aware domain
- symbolic node α_4 becomes redundant and can be removed

Shape graph weakening: definition (reminder)

To design **inclusion test**, **join** and **widening** algorithms, we first study a more general notion of **weakening**:

Weakening

We say that S_0^\sharp **can be weakened into** S_1^\sharp if and only if

$$\forall (h, \nu) \in \gamma_{\text{sh}}(S_0^\sharp), \exists \nu' \in \text{Val}, (h, \nu') \in \gamma_{\text{sh}}(S_1^\sharp)$$

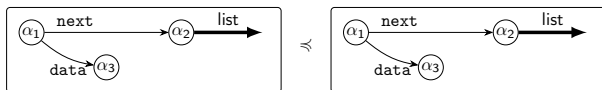
We then note $S_0^\sharp \preceq S_1^\sharp$

Applications:

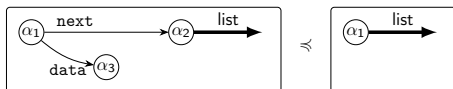
- **inclusion test** (comparison) inputs S_0^\sharp, S_1^\sharp ; if returns true $S_0^\sharp \preceq S_1^\sharp$
- **canonicalization** (unary weakening) inputs S_0^\sharp and returns $\rho(S_0^\sharp)$ such that $S_0^\sharp \preceq \rho(S_0^\sharp)$
- **widening / join** (binary weakening ensuring termination or not) inputs S_0^\sharp, S_1^\sharp and returns S_{up}^\sharp such that $S_i^\sharp \preceq S_{\text{up}}^\sharp$

Shape graph weakening weakening based on local rules (reminder)

By **rule** (\approx_{Id}):



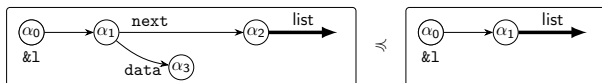
Thus, by **rule** ($\approx_{\mathcal{U}}$):



Additionally, by **rule** (\approx_{Id}):



Thus, by **rule** (\approx_{*}):



Shape graph abstract union

The principle of join and widening algorithm **is similar to that of \sqsubseteq^\sharp** :

- It can be computed **region by region**, as for weakening in general:
If $\forall i \in \{0, 1\}, \forall s \in \{\text{lft}, \text{rgh}\}, S_{i,s}^\sharp \preceq S_s^\sharp$,

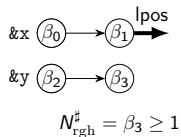
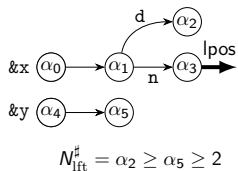
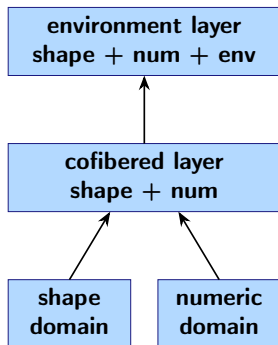


The partitioning of inputs / different nodes sets requires a **node correspondence function**

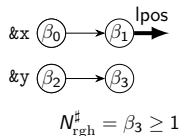
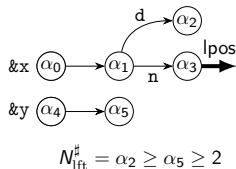
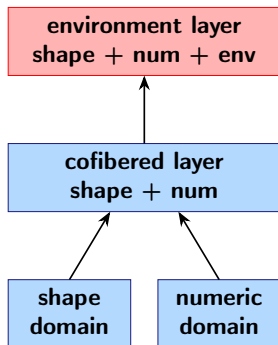
$$\Psi : \mathbb{V}^\sharp(S_{\text{lft}}^\sharp) \times \mathbb{V}^\sharp(S_{\text{rgh}}^\sharp) \longrightarrow \mathbb{V}^\sharp(S^\sharp)$$

- The computation of the shape join progresses by the application of **local join rules**, that produce a **new (output) shape graph, that weakens both inputs**

Widening / join in the combined domain



Widening / join in the combined domain

 $\&x$ δ_0 $\&y$ δ_1

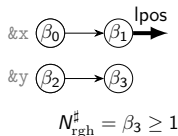
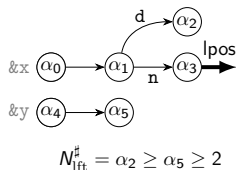
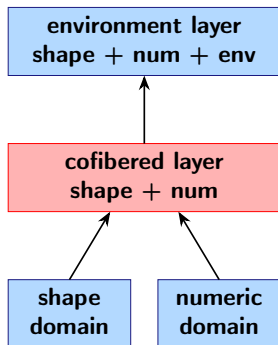
$$\Psi(\alpha_0, \beta_0) = \delta_0$$

$$\Psi(\alpha_4, \beta_2) = \delta_1$$

Stage 1: abstract environment

- compute new abstract environment and initial node relation
e.g., α_0, β_0 both denote $\&x$

Widening / join in the combined domain

 $\&x \delta_0$ $\&y \delta_1$

$$\Psi(\alpha_0, \beta_0) = \delta_0$$

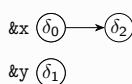
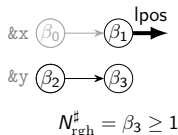
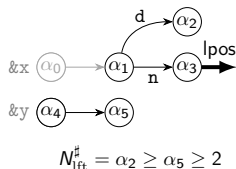
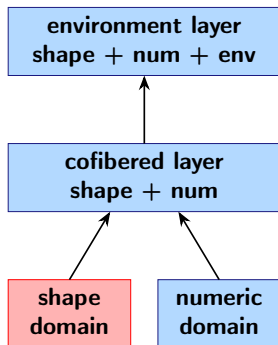
$$\Psi(\alpha_4, \beta_2) = \delta_1$$

Stage 2: join in the “cofibered” layer

operations to perform:

- 1 compute the join in the graph
- 2 convert value abstractions, and join the resulting lattice

Widening / join in the combined domain



$$\Psi(\alpha_0, \beta_0) = \delta_0$$

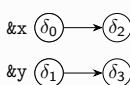
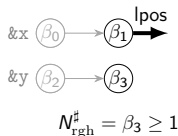
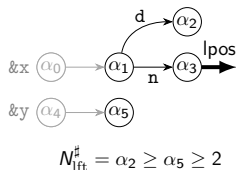
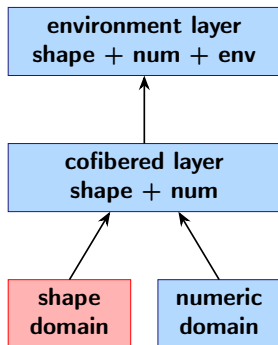
$$\Psi(\alpha_4, \beta_2) = \delta_1$$

$$\Psi(\alpha_1, \beta_1) = \delta_2$$

Stage 2: graph join

- apply local join rules
ex: **points-to matching, weakening to inductive...**
- incremental algorithm

Widening / join in the combined domain

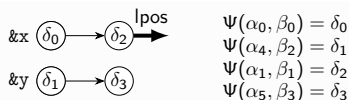
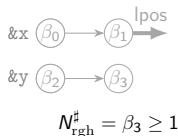
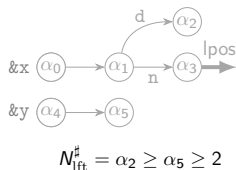
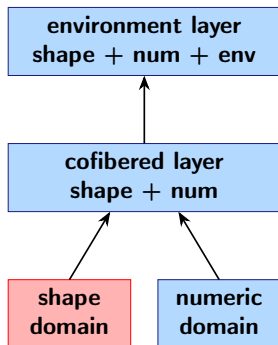


$$\begin{aligned} \Psi(\alpha_0, \beta_0) &= \delta_0 \\ \Psi(\alpha_4, \beta_2) &= \delta_1 \\ \Psi(\alpha_1, \beta_1) &= \delta_2 \\ \Psi(\alpha_5, \beta_3) &= \delta_3 \end{aligned}$$

Stage 2: graph join

- apply local join rules
ex: **points-to matching, weakening to inductive...**
- incremental algorithm

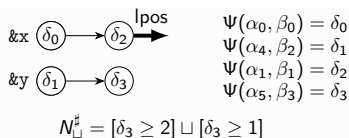
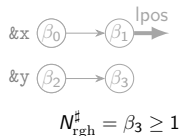
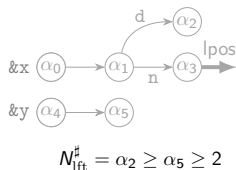
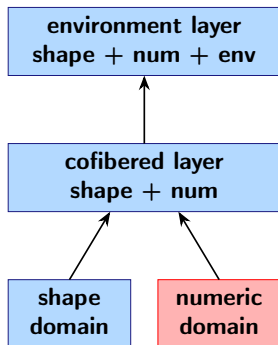
Widening / join in the combined domain



Stage 2: graph join

- apply local join rules
ex: **points-to matching, weakening to inductive...**
- incremental algorithm

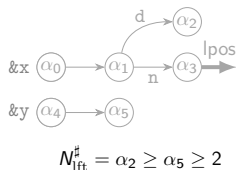
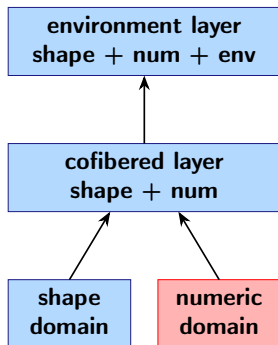
Widening / join in the combined domain



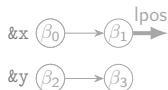
Stage 3: conversion function application in numerics

- remove nodes that were abstracted away
- rename other nodes

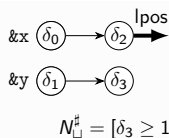
Widening / join in the combined domain



$$N_{\text{lit}}^{\#} = \alpha_2 \geq \alpha_5 \geq 2$$



$$N_{\text{rgh}}^{\#} = \beta_3 \geq 1$$



$$N_{\sqcup}^{\#} = [\delta_3 \geq 1]$$

$$\begin{aligned} \Psi(\alpha_0, \beta_0) &= \delta_0 \\ \Psi(\alpha_4, \beta_2) &= \delta_1 \\ \Psi(\alpha_1, \beta_1) &= \delta_2 \\ \Psi(\alpha_5, \beta_3) &= \delta_3 \end{aligned}$$

Stage 4: join in the numeric domain

- apply \sqcup for regular join, ∇ for a widening

Outline

- 1 Introduction
- 2 Setup (reminder)
- 3 Shape analysis in Three-Valued Logic (TVL)
- 4 Combining shape and value abstractions
- 5 Conclusion**

Shape analysis and summarization

Summaries:

- describe **unbounded** memory regions, with general predicates e.g., list or tree structures, local and global sharing (doubly-linked lists)
- **summary nodes** + associated predicates in TVLA, **inductive predicates** in separation logic

Local refinement (concretization):

- **focus** in TVLA, **unfolding** in separation logic based analysis
- required to **analyze precisely post-conditions** that touch summaries

Global abstraction:

- **ensure termination** despite unbounded, infinite domain
- in TVLA, **canonical abstraction** into a finite domain

In all cases, analysis algorithms aim at avoiding **weak updates** (that would cause a severe precision loss over the whole memory)

Shape analysis and value abstraction

Main issue: the support of the shape abstraction is **always changing**

- summaries appear at canonicalization/widening points
- new atoms/nodes appear at focus/materialization points

Cofibered domain

an abstract form of dependent product

assymmetric version of $\mathbb{D}_{\text{sh}}^{\#} \times \mathbb{D}_{\text{num}}^{\#}$

- the shape abstraction “controls” the value abstraction
- information can still be exchanged in both directions (reduction)
- slightly more complex lattice structure
but standard definitions for widening, inclusion test...

Bibliography

- **[SRW]: Parametric Shape Analysis via 3-Valued Logic.**
Shmuel Sagiv, Thomas W. Reps et Reinhard Wilhelm. In POPL'99, pages 105–118, 1999.
- **[AV]: Abstract Cofibered Domains: Application to the Alias Analysis of Untyped Programs.**
Arnaud Venet.
In SAS'96, pages 366–382.
- **[CR]: Relational inductive shape analysis.**
Bor-Yuh Evan Chang et Xavier Rival.
In POPL'08, pages 247–260, 2008.

Assignment: formalization and paper reading

Formalization of the concretization of 2-structures:

- describe the concretization formula, assuming that we consider the predicates discussed in the course
- run it on the list abstraction example (from the 3-structure to a few select 2-structures, and down to memory states)
- prove the correctness and termination of the widening of the cofibered abstract domain

Reading:

Parametric Shape Analysis via 3-Valued Logic.

Shmuel Sagiv, Thomas W. Reps et Reinhard Wilhelm.

In POPL'99, pages 105–118, 1999.