

# *An abstract domain for separation logic formulae*

**Élodie-Jane Sims**

Elodie-Jane.Sims@polytechnique.fr

# Plan

1. Introduction
2. Separation logic
3. Introduction to the domain : translations logic  $\rightarrow$  the domain
4. About the domain
5. Comparisons

- ❖ Goal: **pointer analysis**: check dereferencing errors, aliases, ...
- ❖  $BI^{\mu\nu}$  a separation logic which permit easy descriptions of the memory, e.g.
  - $x$  points to a list of [1;2;3]
 
$$\exists x_2, x_3. (x \hookrightarrow 1, x_2) * (x_2 \hookrightarrow 2, x_3) * (x_3 \hookrightarrow 3, \text{nil})$$
  - $x$  and  $y$  are aliased pointers
 
$$x = y \wedge \exists x_1, x_2. (x \hookrightarrow x_1, x_2)$$
  - Partitionning:  $x$  and  $y$  belong to two disjoint parts of the heap which have no pointers from one to the other...

## Example of a pointer program with a bug

↑  $\{(\exists z_1, z_2. \text{nil} \hookrightarrow z_1, z_2) \equiv \text{FALSE}\}$   
 $x := \text{nil};$   
 $\{\exists z_1, z_2. x \hookrightarrow z_1, z_2\}$   
 $z := x;$   
 $\{\exists z_1, z_2. z \hookrightarrow z_1, z_2\}$   
 $y := z \cdot 1;$   
 $\{\text{TRUE}\}$

## Point-to analyses: Shape/alias analyses

- **Shape analyses**: the analysis build a graph where
  - the nodes represent locations in the heap
  - the edges represent fields between locations

The analysis usually do approximation (represent more or less nodes/fields than what is in the heap) and computes some more informations about the nodes or edges of the graph.

Recent examples : TVLA (Sagiv, Reps, Wilhelm,...), Smallfoot (O'Hearn, Yang, Berdine, Calcagno, Distefano,...)

- **Alias analyses**: a point-to analysis which computes sets of variables

- ❖ Goal: pointer analysis: check dereferencing errors, aliases, ...
- ❖  $BI^{\mu\nu}$  a separation logic which permit easy descriptions of the memory, e.g.
  - $x$  points to a list of [1;2;3]
 
$$\exists x_2, x_3. (x \hookrightarrow 1, x_2) * (x_2 \hookrightarrow 2, x_3) * (x_3 \hookrightarrow 3, \text{nil})$$
  - $x$  and  $y$  are aliased pointers
 
$$x = y \wedge \exists x_1, x_2. (x \hookrightarrow x_1, x_2)$$
  - Partitionning:  $x$  and  $y$  belong to two disjoint parts of the heap which have no pointers from one to the other...

## Example for a piece of code inserting a cell in a linked list

---

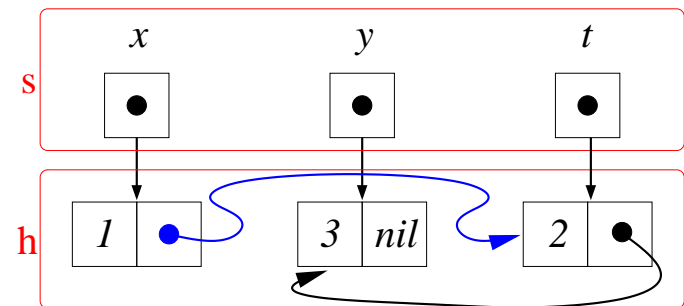
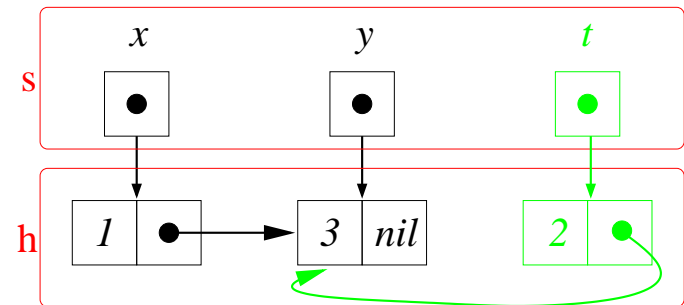
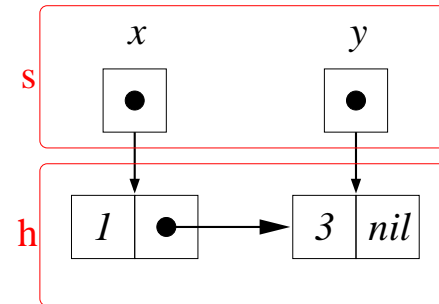
$\{(x \mapsto 1, y) * (y \mapsto 3, \text{nil})\}$

$t := \text{cons}(2, y);$

$\{(x \mapsto 1, y) * (y \mapsto 3, \text{nil}) * (t \mapsto 2, y)\}$

$x \cdot 2 := t;$

$\{(x \mapsto 1, t) * (t \mapsto 2, y) * (y \mapsto 3, \text{nil})\}$





# Local reasoning

---

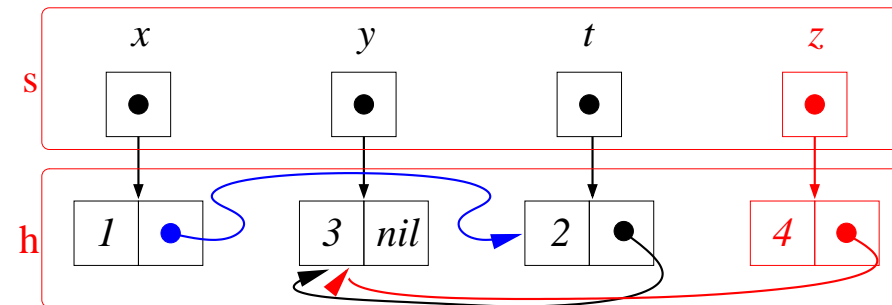
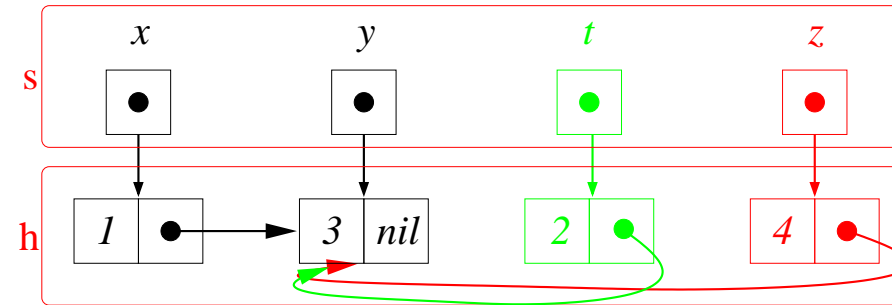
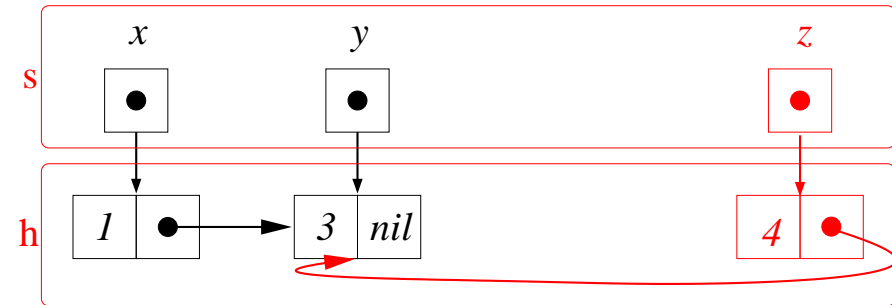
$$\left\{ \begin{array}{l} (x \mapsto 1, y) * (y \mapsto 3, \text{nil}) \\ *(z \mapsto 4, y) \end{array} \right\}$$

$t := \text{cons}(2, y);$

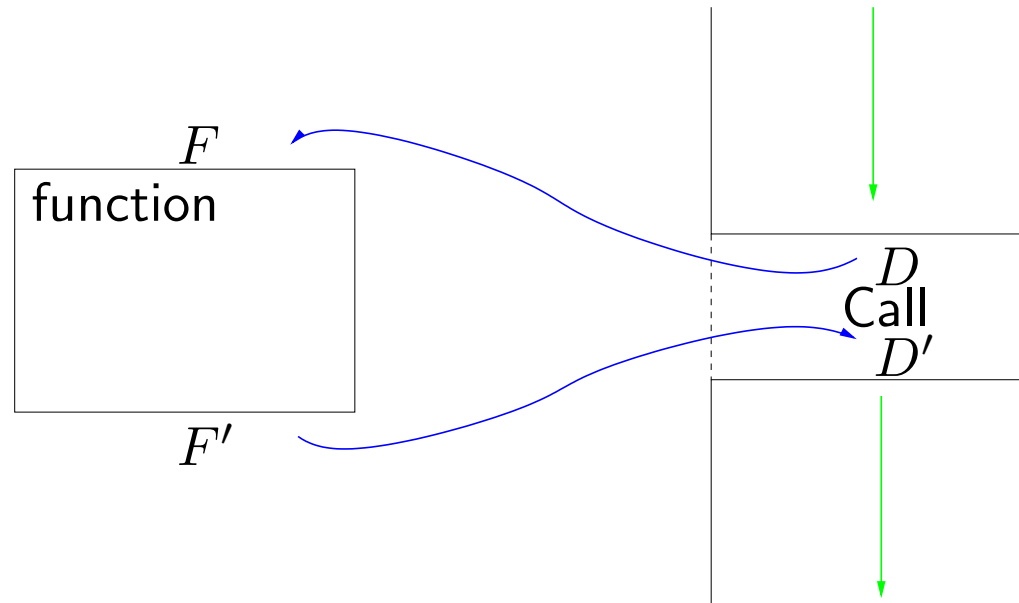
$$\left\{ \begin{array}{l} (x \mapsto 1, y) * (y \mapsto 3, \text{nil}) * (t \mapsto 2, y) \\ *(z \mapsto 4, y) \end{array} \right\}$$

$x \cdot 2 := t;$

$$\left\{ \begin{array}{l} (x \mapsto 1, t) * (t \mapsto 2, y) * (y \mapsto 3, \text{nil}) \\ *(z \mapsto 4, y) \end{array} \right\}$$



- We want to use this logic as an **interface** language for **modular analysis**



Analysis 1  $\rightarrow BI^{\mu\nu} \rightarrow$  Analysis 2  
 Program  $\rightarrow BI^{\mu\nu} \rightarrow$  Analysis 3

We have build an intermediate domain such that:

- it is similar to the existing shape/alias analysis domains to allow translations from/to those domains
- it comes with a concrete semantics in term of sets of states which is the same domain as for the formulae's semantics
- we can translate the formulas into our domain
- it is a cartesian product of different subdomains so that we can cheaply tune the precision depending on the needs
- the domain is parametrised by a numerical domain

# Plan

1. Introduction
2. Separation logic
3. Introduction to the domain : translations logic  $\rightarrow$  the domain
4. About the domain
5. Comparisons

## Domaine of interpretation: *State*

We have a set of variables  $Var$ .

$$\begin{array}{llll} Val & = & Int \cup Bool \cup Atoms \cup Loc & Values \\ S & = & Var \rightarrow Val & Stacks \\ H & = & Loc \rightarrow Val \times Val & Heaps \\ State & = & S \times H & \end{array}$$

Rq: stacks can be partial functions

## The logic: $BI^{\mu\nu}$

	<i>Classical</i>	<i>connectives</i>		
	$E = E'$			false
	$P \Rightarrow Q$			$\exists x.P$
	<i>Spatial</i>	<i>connectives</i>		
	emp	Empty heap		$E \mapsto E_1, E_2$ Points to
	$P * Q$	Spatial conj.		$P \multimap Q$ Spatial imp.
	<i>Fixpoints</i>	<i>connectives</i>		
	$X_v$	Variable for formulae		$P[E/x]$ Posponned substitution
	$\nu X_v.P$	Greatest fixpoint		$\mu X_v.P$ Least fixpoint

$Var_v = \{X_v, Y_v, \dots\}$  infinite set of variables of formulae

## Semantic of $*$

$$\llbracket P * Q \rrbracket_\rho = \left\{ s, h_0 \cdot h_1 \mid \begin{array}{l} \bullet \text{ } \mathit{dom}(h_0) \cap \mathit{dom}(h_1) = \emptyset \\ \bullet \text{ } s, h_0 \in \llbracket P \rrbracket_\rho \\ \bullet \text{ } s, h_1 \in \llbracket Q \rrbracket_\rho \end{array} \right\}$$

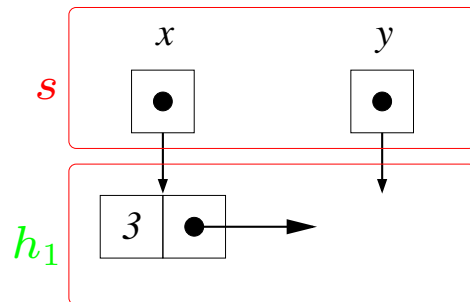


# Examples of formulae

Ex. 1

$$s = [x \rightarrow l_1, y \rightarrow l_2]$$

$$h_1 = [l_1 \rightarrow \langle 3, l_2 \rangle]$$

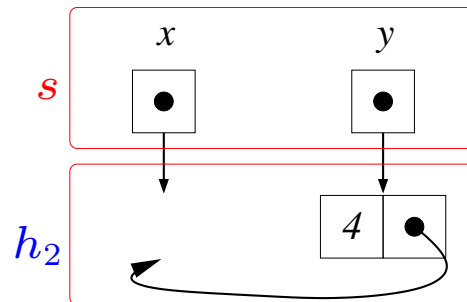


$$\models (x \mapsto 3, y)$$

Ex. 2

$$s = [x \rightarrow l_1, y \rightarrow l_2]$$

$$h_2 = [l_2 \rightarrow \langle 4, l_1 \rangle]$$

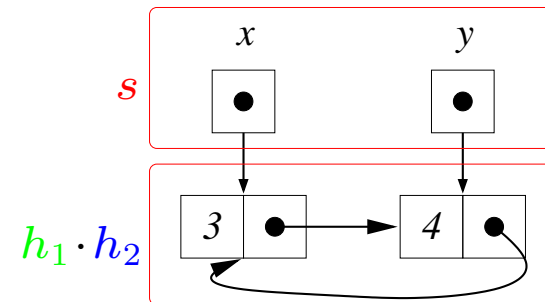


$$\models (y \mapsto 4, x)$$

Ex. 3

$$s = [x \rightarrow l_1, y \rightarrow l_2]$$

$$h_1 \cdot h_2 = \left[ \begin{array}{l} l_1 \rightarrow \langle 3, l_2 \rangle, \\ l_2 \rightarrow \langle 4, l_1 \rangle \end{array} \right]$$



$$\models (x \mapsto 3, y) * (y \mapsto 4, x)$$

$$\not\models (x \mapsto 3, y) \wedge (y \mapsto 4, x)$$

# Plan

1. Introduction
2. Separation logic
3. Introduction to the domain : translations logic  $\rightarrow$  the domain
4. About the domain
5. Comparisons

# Ex1

Formulae	$x = \text{nil}$
Semantics	$\{s, h \mid s(x) = \text{nil}\}, \dots$
Translation	$\left( \boxed{x} \rightarrow \text{Nilt}, \neg, \neg, \neg, \neg, \neg, - \right)$

Formulae	$(x = \text{nil} \vee x = \text{true})$
Translation	$\left( \boxed{x} \rightarrow \text{Nilt}, \neg, \neg, \neg, \neg, \neg, - \right)$ $\left( \boxed{x} \rightarrow \text{Truet} \right)$

## Ex2

Formula	$A \wedge B$
Constraints	cheap translation of $\wedge$
Translation	$T(A \wedge B) \triangleq T'(T'(\top, A), B)$

$\top$  is the empty graph, representing no information

## Ex3, Ex4

Formula	$x = y$
Constraints	refine the information for one variables while also refining the information of the second one in a cheap way
Adds	infinite set of auxiliary variables $TVar$ $VAR \triangleq Var \uplus TVar$
Translation	
Formula	$x = y \wedge x = \mathbf{nil}$
Translation	

## Ex5

Formula	$x = y \wedge x = \mathbf{nil}$
Translation	
Formula	$(\exists x. x = y \wedge x = \mathbf{nil}) \equiv (y = \mathbf{nil})$
Translation	

## Ex6

Formula	$(x < y + 3)$
Translation	$\left( \begin{array}{l} \boxed{x} \rightarrow \boxed{\alpha} \rightarrow \text{Numt} \\ \boxed{y} \rightarrow \boxed{\beta} \rightarrow \text{Numt} \end{array} \right), \neg, \neg, \neg, \neg, \neg, d$ <p><math>d \in \mathcal{D}</math> encodes that <math>\alpha &lt; \beta + 3</math></p>



## Ex7

Formula	“x is a location not allocated”
Semantic	$\{s, h \mid s(x) \in Loc \wedge s(x) \notin dom(h)\}$
Translation	$\left( \boxed{x} \rightarrow \text{Dangling\_Loc}, -, -, -, -, -, - \right)$

## Ex8

Formula	emp
Semantic	$\{s, h \mid \text{dom}(h) = \emptyset\}$
Adds	$HU \triangleq \mathcal{P}(TVar)$ $HO \triangleq \mathcal{P}(TVar) \uplus \text{full}$
Translation	$(\top, -, \emptyset, -, -, -, -)$

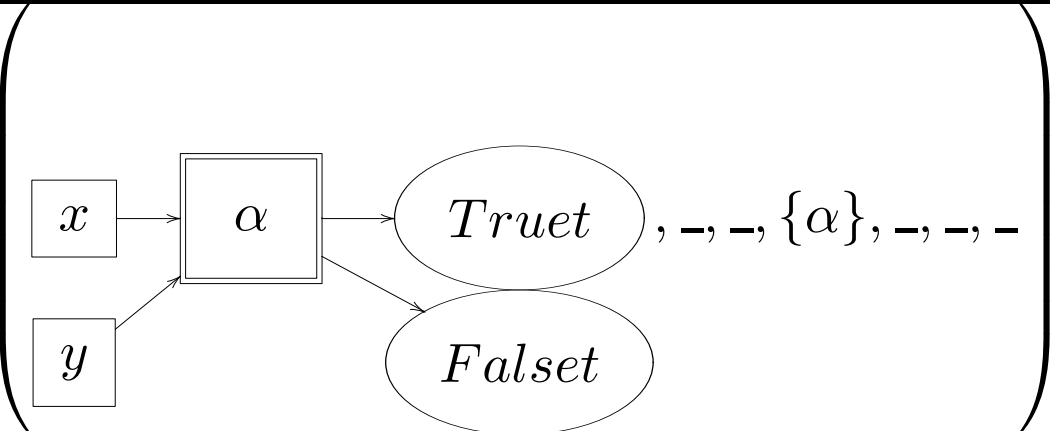
# Ex9

Formula	$(x \mapsto \text{true}, \text{nil})$
Semantic	$\{s, h \mid [s(x) \rightarrow \langle \text{True}, \text{nil} \rangle] = h\}$
Translation	

Formula	$(x \hookrightarrow \text{true}, \text{nil})$
Semantic	$\{s, h \mid [s(x) \rightarrow \langle \text{True}, \text{nil} \rangle] \subseteq h\}$
Translation	

## Ex10

Variables represent at most one value. To allow approximation we introduce summary nodes which can represent several values.

Formula	approx. of $(x = \text{true} \wedge y = \text{false})$ by $\left( \begin{array}{c} x = \text{true} \\ \vee x = \text{false} \end{array} \right) \wedge \left( \begin{array}{c} y = \text{false} \\ \vee y = \text{true} \end{array} \right)$
Translation	

## Ex11: finite acyclic list of *True* starting from *x*

Formula	$\mu X_v. \left( \begin{array}{l} (x = \text{nil}) \vee \exists x_2. \\ x \hookrightarrow (\text{true}, x_2) * X_v[x_2/x] \end{array} \right)$
Translation	

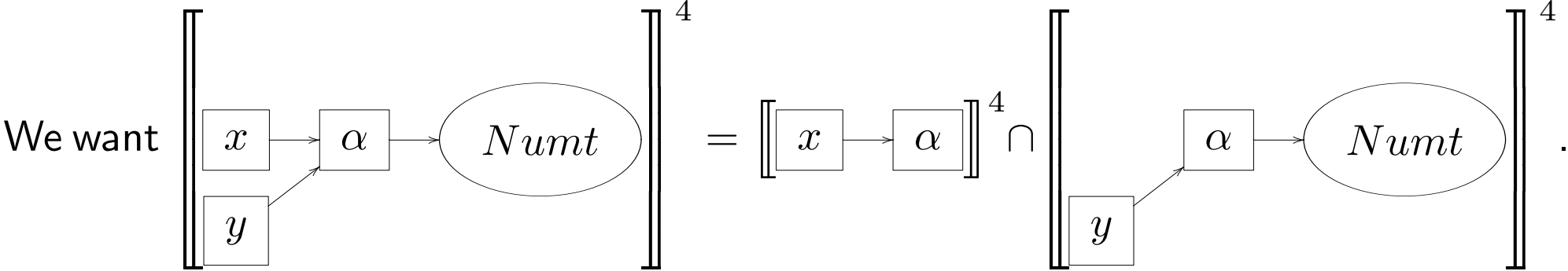
$\emptyset$  is the set of infinite summary nodes, for infinite list  $\mu$  would be replaced by  $\nu$  and  $\emptyset$  by  $\{\alpha\}$ ).

## Ex12: increase precision of union

Formula	$\left( \begin{array}{l} x = \text{nil} \\ \wedge y = \text{true} \end{array} \right) \vee \left( \begin{array}{l} x = \text{true} \\ \wedge y = \text{nil} \end{array} \right)$
Translation	<p>The diagram illustrates the translation of the formula into an abstract domain. It consists of four abstract nodes: <math>\alpha_1</math> and <math>\alpha_2</math> on the left, and <math>\alpha_3</math> and <math>\alpha_4</math> on the right. Node <math>\alpha_1</math> is connected to a <i>Nilt</i> oval, and <math>\alpha_2</math> is connected to a <i>Truet</i> oval. Node <math>\alpha_3</math> is connected to a <i>Nilt</i> oval, and <math>\alpha_4</math> is connected to a <i>Truet</i> oval. Variable <math>x</math> points to <math>\alpha_1</math> and <math>\alpha_2</math>, and variable <math>y</math> points to <math>\alpha_3</math> and <math>\alpha_4</math>. Dashed arrows labeled <math>\{\dagger_{eq}\}</math> connect <math>\alpha_1</math> to <math>\alpha_2</math>, <math>\alpha_3</math> to <math>\alpha_4</math>, <math>\alpha_1</math> to <math>\alpha_3</math>, and <math>\alpha_2</math> to <math>\alpha_4</math>.</p>

# Plan

1. Introduction
2. Separation logic
3. Introduction to the domain : translations logic  $\rightarrow$  the domain
4. About the domain
5. Translation of  $*$
6. Comparisons





## Semantic

$$\begin{array}{ll} Val \triangleq \mathbb{Z} \uplus Bool \uplus \mathbf{nil} \uplus Loc & Val' \triangleq Val \cup \{\mathbf{ood}\} \\ S \triangleq Var \rightarrow Val & S' \triangleq Var \xrightarrow{total} Val' \\ H \triangleq Loc \rightarrow (Val \times Val) & F \triangleq TVar \xrightarrow{total} \mathcal{P}(Val') \\ & R \triangleq Loc \rightarrow \mathcal{P}(Loc) \\ State \triangleq S \times H & MFR \triangleq \mathcal{P}(S' \times H \times F \times R) \end{array}$$

$$\begin{aligned} \llbracket \cdot \rrbracket &\in \mathbf{AR} \rightarrow \mathcal{P}(\mathbf{State}) \\ \llbracket ar \rrbracket &\triangleq \{\bar{s}, h \mid s, h, f, r \in \llbracket ar \rrbracket'\} \end{aligned}$$

$$\begin{aligned} \llbracket \cdot \rrbracket' &\in \mathbf{AR} \rightarrow \mathbf{MFR} \\ \llbracket (ad, hu, ho, sn, sn^\infty, t, d) \rrbracket' &\triangleq \llbracket ad \rrbracket^4 \cap \llbracket hu \rrbracket^1 \cap \llbracket ho \rrbracket^{1'} \cap \llbracket sn \rrbracket^2 \cap \llbracket sn^\infty \rrbracket^{2'} \\ &\quad \cap \llbracket t \rrbracket^3 \cap \llbracket d \rrbracket^7 \cap \mathit{sem}^* \end{aligned}$$

$$\begin{aligned} \llbracket \cdot \rrbracket^4 &\in \mathbf{AD} \rightarrow \mathbf{MFR} \\ \llbracket ad \rrbracket^4 &\triangleq \bigcap_{v \in \mathbf{VAR}} \llbracket v, ad(v) \rrbracket^5 \end{aligned}$$

## Operations

- union, intersection
- extension (replace  $[v \rightarrow S]$  by  $[v \rightarrow \{v'\} | v' \rightarrow S]$  with a fresh  $v'$  )  
used to tune the precision of the union
- merging (replace  $[v_1 \rightarrow S_1 | v_2 \rightarrow S_2]$  by  $[v_2 \rightarrow (S_1 \cup S_2)]$ )  
used with the widening/narrowing
- translations from formulae to the domain

# Plan

1. Introduction
2. Separation logic
3. Introduction to the domain : translations logic  $\rightarrow$  the domain
4. About the domain
5. Translation of  $*$
6. Comparisons

## Comparisons

- the  $\bullet \begin{array}{c} 1 \\ \rightsquigarrow \\ 2 \end{array}$  represent nodes in the usual shape graphs
- summary nodes as for other shape graphs, seems to give more possibilities than predicate abstraction (with each time a specific predicate for list, etc...) but the technics of predicate and their algorithm/heuristics (like folding/unfolding) could probably also be use on our graphs
- a lonely outgoing edge can be seen as a “must” arrow (or valued 1), several outgoing edges from a variable can be seen as a “may” arrow (or valued  $1/2$ , but it is a bit more precise because we know that one of them should exist), and an edge to  $\emptyset$  can be seen as a “must not” arrow (or valued 0)

- we deal with numerical (*Magill & al.* also do)
- we have a formal semantic of our domain, the semantics of auxiliary variables are formally defined and formally used in the proofs
- we don't have to check for equalities of variables
- the domain is a cartesian product, we can add or remove some parts depending on the precision we want
- we directly have in the domain the “Dangling” information which is suitable for cleaning checking

End