An abstract domain for separation logic formulae

Élodie-Jane Sims [1]

École Polytechnique, Palaiseau, France
Kansas State University, Manhattan, Kansas, USA

Abstract

We present a separation logic abstract domain for static analysis by abstract interpretation. We consider separation logic with fixpoint formulae. The domain embeds shape and alias information. The main originality compared to usual shape-graphs is that we treat all values (numerical, heap locations, nil,...) the same way, thus we can have numerical summary nodes. To keep the domain as general as possible, it is parameterized by a numerical abstract domain which can be instantiated as needed. We provide a semantics in terms of sets of memory (the usual model for separation logic) and sound functions on the domain, including a widening and a union which precision/cost can be tuned to the specific needs of the context where the domain is used.

Key words: abstract interpretation, separation logic, shape-analysis

1 Introduction

We are interested in doing static analysis of programs [4] that use shared mutable data structures. The final goal of our work is to detect errors in a program (problems of dereferencing, aliasing, etc.) or to prove that a program is correct (with respect to these problems) in an automatic way. Abstract interpretation [2,3,4] provides systematic methods for automatic inference of complex properties. We are interested in shape or alias properties in separation logic (extended with fixpoint) formulae.

Separation logic [5,10] is an extension of Hoare logic whose assertion language is BI logic [5,9,8]. Assertions, ϕ, in separation logic are evaluated with the contents of the local-variable stack, s, and the heap, h, written as $s, h \models \phi$. A new propositional connective, "separating conjunction," $*$, lets one assert $s, h \models \phi_1 * \phi_2$ iff h can be split into *disjoint* regions, h_1 and h_2, such that

[1] Email: Elodie-Jane.Sims@polytechnique.fr

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

$s, h_i \models \phi_i$, for $i \in \{1, 2\}$. Separating conjunction permits concise, flexible, and modular description of properties of data structures with controlled sharing.

There are automatic and semi-automatic tools that use Hoare logic to reason about programs; we are interested in designing static-analysis tools (that address problems of dereferencing, aliasing, etc.) that employ separation logic.

We have made first steps towards this goal: In previous work, we extended separation logic with fixpoints, letting us express finitely separation-logic characterizations of *wlp* and *sp* for `while`-loop programs [12].

Lying at the heart of the abstract domain is its semantic domain of denotations. In this paper, we introduce the domain and its language, giving the latter in both linear and graphical forms. Our graphical representation resembles the graph formats found in pointer analysis and shape analysis [11,6], which are also concerned with describing properties of aliasing and heap data structures, respectively.

To keep our abstract separation-logic domain as general as possible, we *parameterize* it on a numerical abstract domain, which can be instantiated with existing numerical domains, including relational ones. Within our domain language, we treat numerical information in the same way as memory information. We provide a semantics of our domain directly in term of sets of memory.

As part of our domain language, we present some functions on the domain, among with widening and union operators, whose precision and cost can be tuned to the specific needs of the context where the domain is used.

The achievements of the paper are: (i) we define an abstract domain for separation logic that is amenable to static analyis; (ii) we define operations on the domain (iii) we give a rigorous semantics and use it to prove soundness of the operations on the domain. These results provide a solid foundation for implementing efficient static analyzers for separation logic.

2 Examples: Introduction to the domain, translations of formulae

The domain we define is a tuple. Let $(ad, hu, ho, sn, sn^{\infty}, t, d)$ be an element of our domain:

- ad is a function that maps variables (and auxiliary variables) to abstract denotations (graphically represented as a graph)
- hu is a set of auxiliary variables which represent an underapproximation of the locations in the heap
- ho is a set of auxiliary variables which represent an overapproximation of the locations in the heap, it can also take the value `full`

We write _ when the element of the tuple is not pertinent for the example.

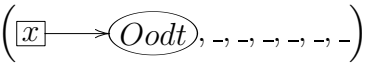
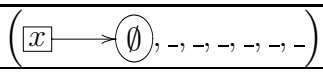
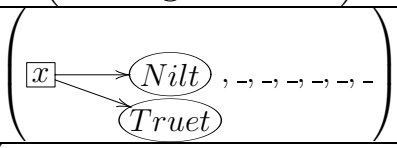
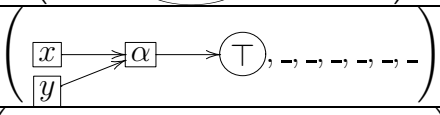
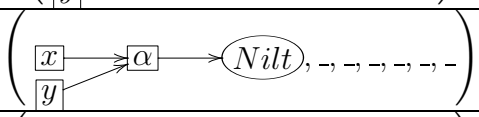
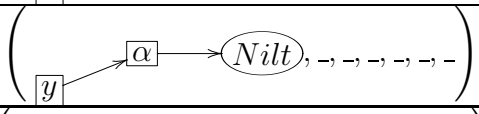
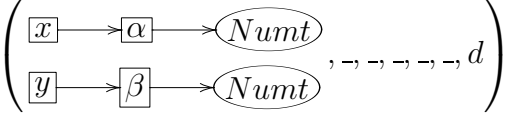
	formula or set states represented	abstract representation
(1)	$(x = \texttt{nil})$	
(2)	$\neg(x = x)$ $\{s, h \mid x \notin dom(s)\}$	
(3)	$\texttt{false}$	
(4)	$(x = \texttt{nil} \vee x = \texttt{true})$	
(5)	$(x = y)$	
(6)	$(x = y \wedge x = \texttt{nil})$	
(7)	$(\exists x.\ x = y \wedge x = \texttt{nil}) \equiv (y = \texttt{nil})$	
(8)	$(x < y + 3)$	 $d \in \mathcal{D}$ encodes that $\alpha < \beta + 3$

Fig. 1. Introduction examples

- sn is a set of auxiliary variables which can represent a final set of concrete values
- sn^{∞} is a set of auxiliary variables which can represent an infinite set of concrete values
- t is a table which compares the values represented between two auxiliary variables.
- d encodes numerical information

In Fig. 1 and 2, we present examples of formulae or sets of memory states and their translations. We do not provide in this paper a complete description of separation logic and its semantics (see [12]).

Concrete domain Let Loc be an infinite set of heap locations, let $Val \triangleq \mathbb{Z} \uplus Bool \uplus \{\texttt{nil}\} \uplus Loc$ be the set of storable values, let $S \triangleq Var \rightharpoonup Val$ ($\rightharpoonup$ stands for partial function) be the set of temporary-variable stacks, and let $H \triangleq Loc \rightharpoonup (Val \times Val)$ be the set of heaps (partial functions that map locations to cons cells).

We define $M \triangleq S \times H$. The standard model for the logic, which is also the

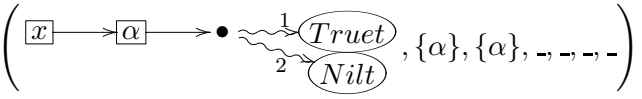
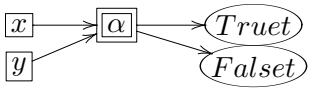
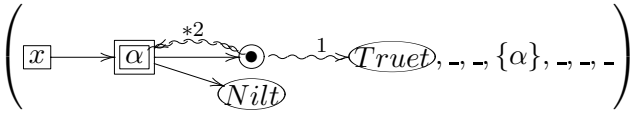
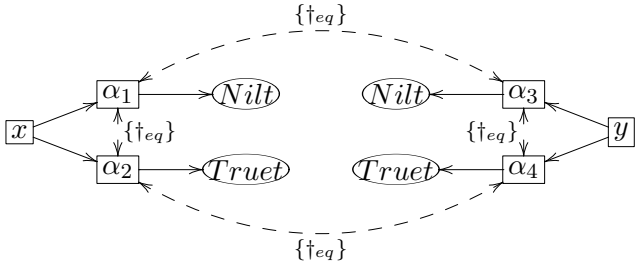
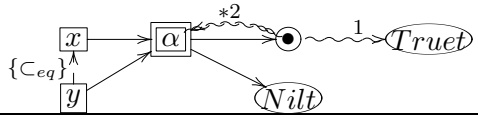
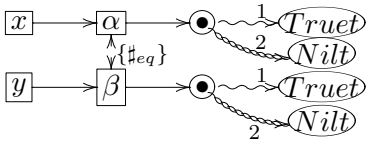
	formula or set states represented	abstract representation
(9)	"x is a location not allocated" $\{s,h \mid s(x) \in Loc \wedge s(x) \notin dom(h)\}$	
(10)	$\texttt{emp}$ $\{s,h \mid dom(h) = \emptyset\}$	$(\_,\_,\emptyset,\_,\_,\_,\_)$
(11)	$(x \mapsto \texttt{true},\texttt{nil})$ $\{s,h\|[s(x) \to \langle True,\texttt{nil}\rangle] = h\}$	
(12)	$(x \hookrightarrow \texttt{true},\texttt{nil})$ $\{s,h\|[s(x) \to \langle True,\texttt{nil}\rangle] \subseteq h\}$	
(13)	approx. of $(x = \texttt{true} \wedge y = \texttt{false})$ $\begin{pmatrix} x = \texttt{true} \\ \vee x = \texttt{false} \end{pmatrix} \wedge \begin{pmatrix} y = \texttt{false} \\ \vee y = \texttt{true} \end{pmatrix}$	
(14)	there is an finite acyclic list of $True$ starting from x $\mu X_v. \begin{pmatrix} (x = \texttt{nil}) \vee \exists x_2. \\ x \hookrightarrow (\texttt{true}, x_2) \ * \ X_v[x_2/x] \end{pmatrix}$	
(15)	$\begin{pmatrix} x = \texttt{nil} \\ \wedge y = \texttt{true} \end{pmatrix} \vee \begin{pmatrix} x = \texttt{true} \\ \wedge y = \texttt{nil} \end{pmatrix}$	
(16)	x and y points to the same acyclic list of $True$ but x comes first	
(17)	$((x \mapsto \texttt{true},\texttt{nil}) * (y \mapsto \texttt{true},\texttt{nil}))$	

Fig. 2. Introduction examples

domain for the concretisation of our abstract domain is $\mathcal{P}(M)$. Given some $s, h \in M$, the denotation of a variable x, is $s(x)$; when $s(x) \in Loc$, its dereferencing is $h(s(x))$. Since separation logic is oriented towards properties of heap structures, such two-step dereferencing dominates one's reasoning about variables, and we make it a key feature of our abstract language.

Simple abstract values One might use separation logic to assert that the value of a variable in the stack is `nil`, thus we have an abstract value $Nilt$ (Ex. 1). As well, we translate `true` by $Truet$, `false` by $Falset$. A variable can be out of the domain of the stack which corresponds to the abstract value $Oodt$ (Ex. 2). Since we have this $Oodt$, when in the abstract domain, a variable is assigned to $\emptyset$, it does not mean that the variable in not in the stack domain but the whole abstract value represent an emptyset or the `false` formula (Ex. 3). The domain assigns to variables a set of abstract values with a disjunctive meaning (Ex. 4).

Auxiliary variables We use an infinite set of auxiliary variables $TVar$ to encode aliasing (Ex. 5). In all our examples, we use $x, y, ... \in Var$ for program variables, and we use $\alpha, \beta, ... \in TVar$ for auxiliary variables that denote values of our abstract domain. We use $v, v_1, v_2, ...$ for variables in $Var \cup TVar$. The use of $TVar$ permits a cheap translation of conjunction, that is to translate $P \wedge Q$ we first translate P then we refine the result while translating Q (Ex. 5 and 6) and it also permits a cheap translation of quantifiers (Ex. 6 and 7).

Auxiliary variables can be used as wanted, for example, $\boxed{x} \longrightarrow (Nilt)$ can also be represented by $\boxed{x} \longrightarrow \boxed{\alpha} \longrightarrow (Nilt)$. But for efficiency, the domain has some constrains (see [1]), for example we forbid $\boxed{x} \longrightarrow \boxed{\alpha} \longrightarrow (\emptyset)$ which should be represented as $\boxed{x} \longrightarrow (\emptyset)$.

Numerical information To allow translation of numerical values and even numerical relations, we parameterize our domain on a numerical domain, $\mathcal{D}$. The graph will only contain the abstract value $Numt$ and all the numerical information are encoded in the element of the numerical domain (Ex. 8). For example, in Ex. 8, d could be a "difference bound matrix" [7], and we would have $d(\alpha, \beta) = i$ with any $i \geq 2$.

Heap In the logic, one can say that a variable is a dangling pointer thus we have an other abstract value *Dangling_Loc* to represent this information (Ex. 9). To model pointers and the heap, the second and third component of our tuples are *(i) hu*, a set of auxiliary variables that underapproximate the set of locations allocated in the heap, and *(ii) ho* a similar overapproximation (*ho* could also take the value, `full` , to give no information). They are used, for example, to translate the formula `emp` which says that the heap is empty (Ex. 10). In separation logic, one writes assertions that state the exact contents of the heap. For example, the formula, $(x \mapsto \texttt{true}, \texttt{nil})$, not only says that x points to a cell whose *car*-value is $True$ and *cdr*-value is `nil` but also says that the cons-cell pointed to by x is the *only* cell in the heap (Ex. 11). In the translation $\bullet$ represents a heap location, $\overset{1}{\rightsquigarrow}$ / $\overset{2}{\rightsquigarrow}$ represents its car/cdr. The formula $(x \hookrightarrow \texttt{true}, \texttt{nil})$ is like $(x \mapsto \texttt{true}, \texttt{nil})$ except that it allows additional cells in the heap domain, thus *ho* takes the value `full` (Ex. 12).

Summary nodes Notice that we embed shape-graphs: $\bullet$ and $\rightsquigarrow$ being their nodes and edges. To avoid infinite graphs (or to bound the computation

time), our graphs have *summary nodes*, which are nodes that represent multiple concrete values. By default, all variables represent only one value. The forth (sn) and fifth (sn^∞) components are sets of auxiliary variables which are allowed to represent sets of values (possibly infinite for sn^∞). Graphically, normal summary nodes will be doubly circled/squared and infinite summary nodes will be triply circled/squared. The main difference from usual shape-graphs is that we allow to have summary nodes for values which are not locations (Ex. 13). We also use summary nodes to represent a list of any size (Ex. 14). In the formula, μ is a least-fixpoint connective and [ ] is a postponed substitution connective which is used for the recursion [12]. Informally, the formula can be written as $\texttt{nclisttrue}(x) \triangleq (x = \texttt{nil}) \vee \exists x_2.\ x \hookrightarrow (\texttt{true}, x_2) \ * \ \texttt{nclisttrue}(x_2)$.

To allow infinite lists, we would replace μ by ν in the above formula, and in our domain, ◉ would be replaced by ◎

Tables it is obvious that representing union of graphs by a graph with unions (as we do, see Ex. 4) implies some approximations. It can be interesting to do this approximation but we also want the domain to allow additional precision if needed. In Sect. 5.2, we give an explanation on how to do a precise union. The sixth component of the tuple is a 2-dimention table for auxiliary variables which is graphically representated by annoted arrows, $\_ \dashrightarrow$. The annotations are sets properties on the concrete values represented by the auxiliary variables. Details are given in Sec. 3. An arrow labeled with $\{\dagger_{eq}\}$ means that one of the two variables pointed represent no value (see Ex. 15 where α_1 has a value, then α_2 and α_3 do not, and when α_4 has a value, then α_2 and α_3 do not). An arrow labeled with $\{\subset_{eq}\}$ means that the values represented by one variable are included in the ones represented by the other (see Ex. 16).

Separation information There are two ways to encode separation information. The first way is to use $\_ \dashrightarrow$, in particular $\overset{\{\sharp_{eq}\}}{\dashrightarrow}$ says that two variables has no values in common in their concretisations (Ex. 17). The second way is to use $\overset{*1}{\rightsquigarrow}$ / $\overset{*2}{\rightsquigarrow}$ instead of $\overset{1}{\rightsquigarrow}$ / $\overset{2}{\rightsquigarrow}$. In Ex. 14, this insures that the list is not cyclic: α can represent several locations and they can point to one another through their cdr-values but $*2$ says that no path through those locations is cyclic.

In an automatic translation of formulas, the first way would appear when we translate $P * Q$. We would translate P and Q separately with disjoints sets of fresh auxiliary variables. Then, while merging, variables which represent only locations would have $\overset{\{\sharp_{eq}\}}{\dashrightarrow}$ toward the ones of the other component, in particular the variables in hu. The second way would only appear while translating fixpoints and building summary nodes.

3 Definition of the domain : AR

We now formalize the graphs from the previous section as denotations in our abstract separation-logic domain.

$$
\begin{array}{rcl}
VD1 & ::= & Numt \mid Truet \mid Falset \mid Oodt \mid Nilt \mid Dangling\_Loc \mid TVar \\
VD & ::= & VD1 \mid Loc(\mathcal{P}(\{*1, *2\}) \times VD1 \times VD1) \\
PVD^+ & ::= & (\mathcal{P}(VD) \uplus \circledast, \sqcup, \sqcap) \\
AD & ::= & VAR \overset{total}{\rightarrow} PVD^+ \\
CL_{eq} & ::= & \mathcal{P}(\{\ddagger_{eq}, \dagger_{eq}, =_{eq}, \subset_{eq}, \supset_{eq}, \sharp_{eq}, \bigcirc\!\!\bigcirc_{eq}\}) \\
TB & ::= & (TVar \times TVar) \overset{total}{\rightarrow} CL_{eq} \\
AR & ::= & AD \times \mathcal{P}(TVar) \times (\mathcal{P}(TVar) \uplus \texttt{full}) \times \mathcal{P}(TVar) \times \mathcal{P}(TVar) \times TB \\
 & & \times (\mathcal{D}, [\![\cdot]\!]^{\mathcal{D}} : \mathcal{D} \rightarrow (TVar \overset{total}{\rightarrow} \mathcal{P}(\mathbb{Z})))
\end{array}
$$

For the domain PVD^+ we define $\circledast \sqcup S \triangleq S$, $S \sqcup \circledast \triangleq S$,
$\circledast \sqcap S \triangleq S$, $S \sqcap \circledast \triangleq S$, $\forall S_1, S_2 \neq \circledast \; . \; S_1 \sqcup S_2 \triangleq S_1 \cup S_2$, $S_1 \sqcap S_2 \triangleq S_1 \cap S_2$.

For the domain $\mathcal{P}(TVar) \uplus \texttt{full}$ we define $\texttt{full} \cup S \triangleq \texttt{full}$, $S \cup \texttt{full} \triangleq \texttt{full}$, $\texttt{full} \cap S \triangleq S$, $S \cap \texttt{full} \triangleq S$, $\texttt{full} \setminus S \triangleq \texttt{full}$, $\forall \alpha . \alpha \notin \texttt{full}$.

Fig. 3. Syntax of the domain

Let Var and $TVar$ be two disjoint infinite sets of variables (typically Var will be the set of program/formula variables and $TVar$ will be auxiliary variables). We define $VAR \triangleq Var \uplus TVar$, the set of all variables.

The formal definition of the syntax of the domain are presented in Fig. 3.

$VD1$ is an abstract domain for all values except locations. VD is $VD1$ plus abstract values for locations in the heap. PVD^+ is either a powerset of values in VD either the undefined value noticed $\circledast$.

AD corresponds to the graph drawn in the introduction.

Abstract values in CL_{eq} through elements in TB are associated to pairs (α, β) of auxiliary variables:

- $\ddagger_{eq}$ means both α and β represent an empty set of values
- $\dagger_{eq}$ means exactly one of α and β represents an empty set of values
- $=_{eq}$ means both α and β represent the same non empty set of values
- $\subset_{eq}$ means the non empty set of values represented by α is strictly included in the set represented by β
- $\supset_{eq}$ means the non empty set of values represented by β is strictly included in the set represented by α
- $\sharp_{eq}$ means α and β represent two non empty disjoints sets of values
- $\bigcirc\!\!\bigcirc_{eq}$ means that the three sets of values represented only by α, only by β and by both are non empty

Notice that if α and β are not summary nodes, we have $\subset_{eq}$, $\supset_{eq}$, $\sharp_{eq}$ and $\bigcirc\!\!\bigcirc_{eq}$

have all the same meaning : α and β both have a concrete value and their concrete values are differents, we could write it $\neq_{eq}$.

An implementation could also work with only a sublatice of $\mathcal{P}(\{\ddagger_{eq}, \dagger_{eq}, =_{eq}, \subset_{eq}, \supset_{eq}, \sharp_{eq}, \bigcirc\!\!\!\!\bigcirc_{eq}\})$.

4 Semantics of the domain

The abstractions of separation-logic formulae can be efficiently implemented because we formalize them as a disjunction of eight simple semantic-interpretation functions. This makes a denotation into a tuple of orthogonal elements.
First, we recall the concrete domains:

$$\begin{array}{rlrl} Val &\triangleq \mathbb{Z} \uplus Bool \uplus \texttt{nil} \uplus Loc & Val' &\triangleq Val \cup \{\texttt{ood}\} \\ S &\triangleq Var \rightharpoonup Val & S' &\triangleq Var \overset{total}{\rightarrow} Val' \\ H &\triangleq Var \rightharpoonup (Val \times Val) & F &\triangleq TVar \overset{total}{\rightarrow} \mathcal{P}(Val') \\ & & R &\triangleq Loc \rightharpoonup \mathcal{P}(Loc) \\ M &\triangleq S \times H & MFR &\triangleq \mathcal{P}(S' \times H \times F \times R) \end{array}$$

For simplicity, we work with total functions, so we extend Val with the "out of domain" value $\texttt{ood}$ to define Val'. S' are total stacks, that is we have a bijection between a normal stack and a total stack. We define $^- : S' \to S$ by $\bar{s'} \triangleq s' \restriction_{dom(s') \cap \{x | s'(x) \neq \texttt{ood}\}}$, and $^- : S \to S'$ by $\bar{s} \triangleq [x \in dom(s) \to s(x) \mid x \notin dom(s) \to \texttt{ood}]$.

To lighten the notation, we define a union of stacks in S' and stacks for auxiliary variables in F : $\cdot^{+}\cdot : (S' \times F) \to (VAR \overset{total}{\rightarrow} \mathcal{P}(Val'))$ such that if $x \in Var$ then $s^{+}f(x) \triangleq \{s(x)\}$ and if $\alpha \in TVar$ then $s^{+}f(\alpha) \triangleq f(\alpha)$.

For $(s, h, f, r) \in MFR$, the s corresponds to a (total) stack, h is the heap, f is a stack for the variables in $TVar$, where a variable can map to a set of values (cf. a summary node), and r maps locations to their reachable set of locations, thus encoding separation information.

We define $[\![\cdot]\!]$, the semantics of elements of our domain, in term of $\mathcal{P}(M)$:

$$[\![\cdot]\!] \in \boldsymbol{AR \to \mathcal{P}(M)}$$

$$[\![ar]\!] \triangleq \{\bar{s}, h \mid s, h, f, r \in [\![ar]\!]'\}$$

It uses an intermediate semantics, $[\![\cdot]\!]'$, in MFR which is an intersection of semantics of every part of the domain. (The need for $[\![\cdot]\!]'$ — instead of having directly $[\![\cdot]\!]$ as a conjunction — is explained in [1]):

$$[\![\cdot]\!]' \in \boldsymbol{AR \to MFR}$$

$$[\![(ad, hu, ho, sn, sn^{\infty}, t, d)]\!]' \triangleq [\![ad]\!]^4 \cap [\![hu]\!]^1 \cap [\![ho]\!]^{1'} \cap [\![sn]\!]^2 \cap [\![sn^{\infty}]\!]^{2'} \cap [\![t]\!]^3 \cap [\![d]\!]^7 \cap sem*$$

The semantics of the graph ad is also a conjunction for all assignments:

$$[\![\cdot]\!]^{\mathbf{4}} \in \boldsymbol{AD} \to \boldsymbol{MFR}$$

$$[\![ad]\!]^4 \triangleq \bigcap_{v \in VAR} [\![v, ad(v)]\!]^5$$

An assignment to a set is a disjunction of assignments:

$$[\![\cdot]\!]^{\mathbf{5}} \in (\boldsymbol{VAR} \times \boldsymbol{PVD}^+) \to \boldsymbol{MFR}$$

$$[\![v, \top]\!]^5 \triangleq MFR$$

$$[\![v, \circledast]\!]^5 \triangleq \{s, h, f, r \mid s^+f(v) = \emptyset\}$$

$$S \neq \top, \circledast \quad [\![v, S]\!]^5 \triangleq \{s, h, f, r \mid s^+f(v) \subseteq \bigcup_{vd \in S} [\![vd, (h, f, r)]\!]^8\}$$

We use an auxiliary function, $[\![\cdot]\!]^8$, which gives the sets of concrete values in Val' that correspond to an abstract value in VD for a particular element of $H \times F \times R$:

$$[\![\cdot]\!]^{\mathbf{8}} \in (\boldsymbol{VD} \times (\boldsymbol{H} \times \boldsymbol{F} \times \boldsymbol{R})) \to \boldsymbol{\mathcal{P}(Val')}$$

$$[\![Nilt, \_]\!]^8 \triangleq \{\texttt{nil}\} \qquad [\![Truet, \_]\!]^8 \triangleq \{True\}$$

$$[\![Falset, \_]\!]^8 \triangleq \{False\} \qquad [\![Oodt, \_]\!]^8 \triangleq \{\texttt{ood}\}$$

$$[\![Numt, \_]\!]^8 \triangleq \mathbb{Z} \qquad [\![Dangling\_Loc, (h, \_, \_)]\!]^8 \triangleq Loc \setminus dom(h)$$

$$[\![v, (\_, f, \_)]\!]^8 \triangleq f(v) \text{ when } v \in TVar$$

$$[\![Loc(A, vd1, vd2), (h, f, r)]\!]^8$$

$$\triangleq \left\{ l \in dom(h) \;\middle|\; \left[\begin{array}{l} \bullet\ \Pi_1(h(l)) \in [\![vd1, (h, f, r)]\!]^8 \\ \bullet\ \Pi_2(h(l)) \in [\![vd2, (h, f, r)]\!]^8 \\ \bullet\ *1 \in A \wedge \Pi_1(h(l)) \in Loc \Rightarrow \Pi_1(h(l)) \in r(l) \\ \bullet\ *2 \in A \wedge \Pi_2(h(l)) \in Loc \Rightarrow \Pi_2(h(l)) \in r(l) \end{array} \right. \right\}$$

The semantics of the separation information in the abstract domain ($\mathcal{P}(\{*1, *2\})$) is given by this last rule ($[\![Loc(A, vd1, vd2), (h, f, r)]\!]^8$) and $sem*$:

$$\boldsymbol{sem* \in MFR}$$

$$sem* \triangleq \left\{ s, h, f, r \;\middle|\; \forall l \in dom(r). \left[\begin{array}{l} \bullet\ l \notin r(l) \\ \bullet\ \forall l' \in r(l) \cap dom(r). r(l') \subseteq r(l) \end{array} \right. \right\}$$

The semantics of the lower bound of heap's domain is given by $[\![\cdot]\!]^1$:

$$[\![\cdot]\!]^{\mathbf{1}} \in \boldsymbol{\mathcal{P}(TVar) \to MFR}$$

$$[\![hu]\!]^1 \triangleq \bigcap_{\alpha \in hu} \{s, h, f, r \mid f(\alpha) \cap dom(h) \neq \emptyset\}$$

and the semantics of the upper bound of heap's domain is given by $[\![\cdot]\!]^{1'}$:

$$[\![\cdot]\!]^{1'} \in (\boldsymbol{\mathcal{P}(TVar)} \uplus \texttt{full}) \to \boldsymbol{MFR}$$

$$[\![\texttt{full}]\!]^{1'} \triangleq MFR$$

$$[\![ho]\!]^{1'} \triangleq \{s,h,f,r \mid dom(h) \subseteq \bigcup_{\alpha \in ho} f(\alpha)\}$$

The sets of summary nodes say which nodes can represent several values, their semantics are given by $[\![\cdot]\!]^2$ and $[\![\cdot]\!]^{2'}$

$$[\![\cdot]\!]^{\mathbf{2}} \in \boldsymbol{\mathcal{P}(TVar)} \to \boldsymbol{MFR}$$

$$[\![sn]\!]^2 \triangleq \{s,h,f,r \mid \forall \alpha \in TVar \setminus sn. |f(\alpha)| \leq 1\}$$

$$[\![\cdot]\!]^{\mathbf{2'}} \in \boldsymbol{\mathcal{P}(TVar)} \to \boldsymbol{MFR}$$

$$[\![sn^\infty]\!]^2 \triangleq \{s,h,f,r \mid \forall \alpha \in TVar \setminus sn^\infty.\ f(\alpha) \text{ is finite}\}$$

The semantics of the table is a conjunction of all semantics of its assignments:

$$[\![\cdot]\!]^{\mathbf{3}} \in \boldsymbol{TB} \to \boldsymbol{MFR}$$

$$[\![t]\!]^3 \triangleq \bigcap_{(\alpha_1,\alpha_2) \in TVar \times TVar} \left\{ s,h,f,r \;\middle|\; [\![t(\alpha_1,\alpha_2), f(\alpha_1), f(\alpha_2)]\!]^{6'} \right\}$$

An assignment of the table is a disjunction of assignments:

$$[\![\cdot]\!]^{\mathbf{6'}} \in (\boldsymbol{CL_{eq}} \times \boldsymbol{\mathcal{P}(Val')} \times \boldsymbol{\mathcal{P}(Val')}) \to \boldsymbol{Bool}$$

$$[\![S,A,B]\!]^{6'} \triangleq \bigvee_{l \in S} [\![\ l,A,B]\!]^6$$

The table encodes aliasing, non-aliasing information, and mutual exclusion. (For example, $\dagger_{eq}$ insures that two variables cannot live at the same time, and $\ddagger_{eq}$ insures that two variables are not aliased.)

$$[\![\cdot]\!]^{\mathbf{6}} \in (\top_{\boldsymbol{eq}} \times \boldsymbol{\mathcal{P}(Val')} \times \boldsymbol{\mathcal{P}(Val')}) \to \boldsymbol{Bool}$$

$$[\![\ddagger_{eq},A,B]\!]^6 \triangleq A \cup B = \emptyset \qquad [\![\dagger_{eq},A,B]\!]^6 \triangleq A = \emptyset \text{ xor } B = \emptyset$$

$$[\![=_{eq},A,B]\!]^6 \triangleq A = B \wedge A \neq \emptyset \quad [\![\subset_{eq},A,B]\!]^6 \triangleq A \subset B \wedge A \neq \emptyset$$

$$[\![\supset_{eq},A,B]\!]^6 \triangleq A \supset B \wedge B \neq \emptyset \quad [\![\sharp_{eq},A,B]\!]^6 \triangleq A \cap B = \emptyset \wedge A \neq \emptyset \wedge B \neq \emptyset$$

$$[\![\bigcirc\!\!\!\bigcirc_{eq},A,B]\!]^6 \triangleq A \cap B \neq \emptyset \wedge A \setminus B \neq \emptyset \wedge B \setminus A \neq \emptyset$$

The semantics of the numerical part is given by $[\![\cdot]\!]^7$:

$$[\![\cdot]\!]^{\mathbf{7}} \in \boldsymbol{\mathcal{D}} \to \boldsymbol{MFR}$$

$$[\![d]\!]^7 \triangleq \bigcup_{g \in [\![d]\!]^{\mathcal{D}}} \bigcap_{\alpha \in TVar} \{s,h,f,r \mid f(\alpha) \cap \mathbb{Z} \subseteq g(\alpha)\}$$

5 Operations on the domain

We now present four key operations and describe how they are computed within the abstract domain. All the operations have been proved sound with

respect to the formal semantics, but due to lack of space, formal definitions of the functions, the theorems and proofs are found in [1].

Below, P is a formula in separation logic (with recursion) and $T(P)$ is the transformation of the formula.

For all the transformations T in our domain, we give theorems of this form: $\forall s, h, f, r \in MFR. \exists g.\ s, h, f, r \in [\![ar]\!]' \Rightarrow s, h, g(f), r \in [\![T(ar)]\!]'$

To explain, we should say that we want to translate formulae in *separation logic with fixpoints* into the domain. If T is a translation of separation-logic formula P to the domain, we want $[\![P]\!] \subseteq [\![T(P)]\!]$. For efficiency on the computation of conjunction, we will define another translation, T', from an element of the domain and a formula to an element of the domain. For example, we have $T'(ar, P_1 \wedge P_2) = T'(T'(ar, P_1), P_2)$.

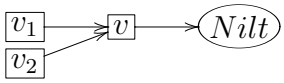
We do the proofs by induction on the syntax of the formula. We do the proofs at the level of $[\![\cdot]\!]'$ and need to keep the properties at this level and not $[\![\cdot]\!]$ because $[\![ar_1]\!] \subseteq [\![ar_2]\!] \not\Rightarrow [\![ar_1]\!]' \subseteq [\![ar_2]\!]'$. Then at the end we define $T(P) = T'(ar_i, P)$. Since we have $\forall s, h, f, r \in MFR.\ \exists g.\ s, h, f, r \in ar_i \wedge \bar{s}, h \in [\![P]\!] \Rightarrow s, h, g(f), r \in [\![T(P)]\!]'$ and $[\![ar_i]\!]' = MFR$, we get $[\![P]\!] \subseteq [\![T(P)]\!]$.

The $\exists g.$ part of the theorems is there because we want to allow changes for auxiliary variables (like for the function *merge*).

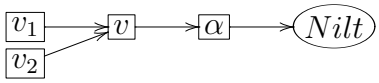
5.1 Extension

We present two extension functions, which add a "fresh variable" (a variable whose value is $\circledast$) as an intermediary between a variable and its assignment. They have the same properties. Those functions will be combined with union [2] and will improve precision of the union. The first one is cheaper but the second one contributes better to the precision of the union than the first.

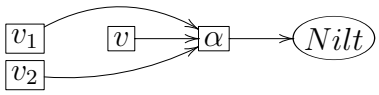
The first version of extension, named $extend(v, \alpha, \_)$ (which is cheaper) will replace



by



but the second, will replace it by



Proposition 5.1 $\forall v \in VAR. \alpha \in TVar. [ar \mid \alpha \rightarrow \circledast] \in AR. (s, h, f, r) \in MFR.$
$s, h, f, r \in [\![[ar \mid \alpha \rightarrow \circledast]]\!]' \Leftrightarrow s, h, [f \mid \alpha \rightarrow s^{+} f(v)] \in [\![extend(v, \alpha, [ar \mid \alpha \rightarrow \circledast])]\!]'$

5.2 Union

The basic union is almost a simple union of all nodes and edges except that if a variable in Var has no outgoing edges (that is the variables is assign to $\top$)

[2] those functions are what one could do instead of doing variable renaming before union

the union graph does not have edges from this variables.

$union{:}(\boldsymbol{AR} \times \boldsymbol{AR}) \to \boldsymbol{AR}$

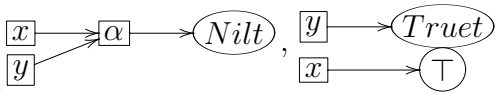
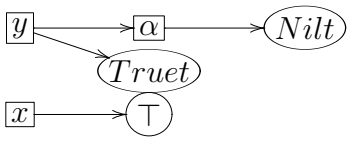
$union\left(\left(ad_1, hu_1, ho_1, sn_1, sn_1^\infty, t_1, d_1\right), \left(ad_2, hu_2, ho_2, sn_2, sn_2^\infty, t_2, d_2\right)\right)$

$\triangleq \left(ad_1 \dot{\sqcup}\, ad_2, hu_1 \cap hu_2, ho_1 \cup ho_2, sn_1 \cup sn_2, sn_1^\infty \cup sn_2^\infty, t_1 \dot{\cup}\, t_2, d_1 \sqcup^{\mathcal{D}} d_2\right)$

$\dot{\sqcup}/\dot{\cup}$ being the point to point applications of $\sqcup/\cup$

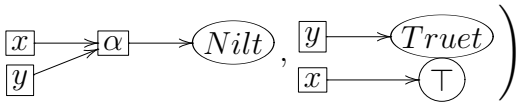
Proposition 5.2 $\forall ar_1, ar_2 \in AR.\ [\![ar_1]\!]' \cup [\![ar_2]\!]' \subseteq [\![union(ar_1, ar_2)]\!]'$

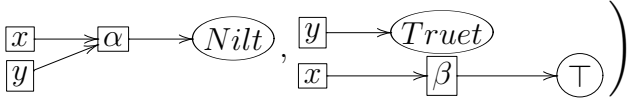
But this is not precise, for example it translates $(x = y \wedge y = \texttt{nil}) \vee (y = \texttt{true})$ as

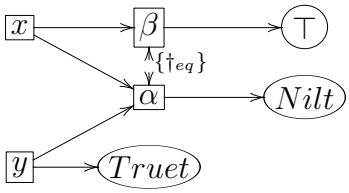
$union\Big($  $\Big) =$ 

which is $(y = \texttt{nil}) \vee (y = \texttt{true})$.

So we will combine the basic union with extend. For the same example

$union\Big($  $\Big)$

$= union\Big($  $\Big)$

$=$ 

This is exactly $(x = \texttt{nil} \wedge y = \texttt{nil}) \vee (y = \texttt{true})$

You can see this example with the non-graphical presentation in [1].

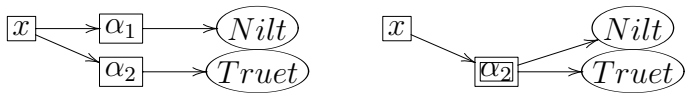
So one can tune the precision of the union by combining it with *extend* but it is more expensive (using it everywhere would be comparable to renaming all variables before union). We would suggest to apply *extend* only when we have to union $\top$ with $S \neq \top$ such that $\exists \alpha \in TVar.\ \alpha \in S$.

We do not suggest to first do several *extend*, then do the normal *union*. The *extend* could be done on the fly, we would just have to be careful to pick *fresh* variables for both sides.

5.3 Merging nodes

We define polymorphic functions for merging two nodes. The first node information gets included in the second one and the first node is removed. This function will be used to reduce the number of auxiliary variables which are used and will create summary nodes.

For example, after merging α_1 into α_2 in we get



5.4 Widening

This widening does not insure stabilization but that the widening will take a finite number of values, to have termination, the user should apply it to a trace which have some information which is monotonic.

The widening operator combines the widening on the numerical domain and a strategy to bound the number of used variables. When applying the widening operator, we will apply the widening for the numerical element and we will apply the function *merge* to merge nodes to keep the number of nodes bounded.

Remember that we have as a condition on $\mathcal{D}$, $\nabla^{\mathcal{D}} : (\bigcup_{n\in\mathbb{Z}} [0,n] \overset{total}{\rightarrow} \mathcal{D}) \rightarrow \mathcal{D}$ such that

- $\forall w \in \mathbb{Z} \overset{total}{\rightarrow} \mathcal{D}.\ \exists i \in \mathbb{Z}. \forall i' \geq i.\ \nabla^{\mathcal{D}}(w \restriction_{[0,i']^{\overset{total}{\rightarrow}_{\mathcal{D}}}}) = \nabla^{\mathcal{D}}(w \restriction_{[0,i]^{\overset{total}{\rightarrow}_{\mathcal{D}}}})$
- $\forall w \in \mathbb{Z} \overset{total}{\rightarrow} \mathcal{D}.\ \forall i \in \mathbb{Z}.\ \forall g_1 \in [\![w(i)]\!]^{\mathcal{D}}. \exists g_2 \in [\![\nabla^{\mathcal{D}}\left(w \restriction_{[0,i]^{\overset{total}{\rightarrow}_{\mathcal{D}}}}\right)]\!]^{\mathcal{D}}.$
  $\forall \alpha \in dom(g_1) \cap dom(g_2).\ g_1(\alpha) \subseteq g_2(\alpha)$

We define $give\_d : AR \rightarrow \mathcal{D}, (ad, hu, ho, sn, sn^{\infty}, t, d) \mapsto d$ and $give\_d : (\mathbb{Z} \overset{total}{\rightarrow} AR) \rightarrow (\mathbb{Z} \overset{total}{\rightarrow} \mathcal{D}), w \mapsto (i \mapsto give\_d(w(i)))$.
$set\_d : (AR \times \mathcal{D}) \rightarrow AR, ((ad, hu, ho, sn, sn^{\infty}, t, d), d') \mapsto (ad, hu, ho, sn, sn^{\infty}, t, d')$

We can extend $\nabla^{\mathcal{D}}$ to be applied with elements of AR:

$$\nabla^{\mathcal{D}}_{AR} : (\bigcup_{n\in\mathbb{Z}} [0,n] \overset{total}{\rightarrow} AR) \rightarrow AR$$

such that

$$\forall w \in \mathbb{Z} \overset{total}{\rightarrow} AR.\ \forall i \in \mathbb{Z}. \nabla^{\mathcal{D}}_{AR}(w \restriction_{[0,i]^{\overset{total}{\rightarrow}_{\mathcal{D}}}}) = set\_d(w(i), \nabla^{\mathcal{D}}(give\_d(w \restriction_{[0,i]^{\overset{total}{\rightarrow}_{\mathcal{D}}}})))$$

We can easily see that we have:

- $\forall w \in \mathbb{Z} \overset{total}{\rightarrow} AR.\ \exists i \in \mathbb{Z}. \forall i' \geq i.\ give\_d(\nabla^{\mathcal{D}}_{AR}(w \restriction_{[0,i']^{\overset{total}{\rightarrow}_{\mathcal{D}}}})) = give\_d(\nabla^{\mathcal{D}}_{AR}(w \restriction_{[0,i]^{\overset{total}{\rightarrow}_{\mathcal{D}}}}))$
- $\forall w \in \mathbb{Z} \overset{total}{\rightarrow} AR.\ \forall i \in \mathbb{Z}.\ \forall g_1 \in [\![give\_d(w(i))]\!]^{\mathcal{D}}. \exists g_2 \in [\![give\_d(\nabla^{\mathcal{D}}_{AR}\left(w \restriction_{[0,i]^{\overset{total}{\rightarrow}_{AR}}}\right))]\!]^{\mathcal{D}}.$
  $\forall \alpha \in dom(g_1) \cap dom(g_2).\ g_1(\alpha) \subseteq g_2(\alpha)$

Suppose that we have a strategy for merging:

$$\nabla^{merge} : (\bigcup_{n\in\mathbb{Z}} [0,n] \overset{total}{\rightarrow} AR) \rightarrow (\bigcup_{n\in\mathbb{Z}} [0,n] \overset{total}{\rightarrow} (TVar \times TVar))$$

such that:

$$\begin{array}{l} \forall w \in \mathbb{Z} \overset{total}{\rightarrow} AR.\ \exists A \in \mathcal{P}(TVar).\ (A \text{ is finite}) \wedge \exists i \in \mathbb{Z}. \forall i' \geq i. \\ (used(merge(\nabla^{merge}(w \restriction_{[0,i']^{total}_{\rightarrow} AR}), w(i'))) \subseteq A) \end{array} \quad (1)$$

This stategy could for example consist on merging variables which have been used while analysing the same program point or the same part of a formula (typically the variable build for translating $\exists$).

We can now define :

Definition 5.3 $\nabla^{AR} : (\bigcup_{n\in\mathbb{Z}} [0,n] \overset{total}{\rightarrow} AR) \rightarrow AR)$ such that

$$\begin{array}{l} \forall w \in \mathbb{Z} \overset{total}{\rightarrow} AR.\ \forall i \in \mathbb{Z}. \\ \nabla^{AR}(w \restriction_{[0,i]^{total}_{\rightarrow} AR}) \triangleq \nabla^{\mathcal{D}}_{AR} \left(\left[w \restriction_{[0,i]^{total}_{\rightarrow} AR} |\ i \rightarrow merge(\nabla^{merge}(w \restriction_{[0,i]^{total}_{\rightarrow} AR}), w(i)) \right] \right) \end{array} \quad (2)$$

Proposition 5.4
$$\begin{array}{l} \forall w \in \mathbb{Z} \overset{total}{\rightarrow} AR.\ \exists A \in \mathcal{P}(AR).\ (A \textit{ is finite}) \wedge \exists i \in \mathbb{Z}. \forall i' \geq i. \\ \nabla^{AR}(w \restriction_{[0,i']^{total}_{\rightarrow} AR}) \in A \end{array}$$

Proposition 5.5 $\forall w \in \mathbb{Z} \overset{total}{\rightarrow} AR.\ \forall i \in \mathbb{Z}.\ \forall s,h,f,r. \exists g.\ s,h,f,r \in [\![w(i)]\!]' \Rightarrow s,h,g(f),r \in [\![\nabla^{AR}(w \restriction_{[0,i]^{total}_{\rightarrow} AR})]\!]'$

6 Conclusion

In this paper, we presented a new abstract domain for separation logic whose denotations resemble shape graphs. The improvement is that numericals and locations are treated the same way, thus we can have numerical summary nodes. This domain is designed for the abstraction of separation logic with fixpoint's formulae. To keep the domain as general as possible, it is parameterized by a numerical abstract domain which can be instantiated as needed with existing ones including relational ones.

The originality was to design a semantics in terms of sets of memory (the usual model for separation logic). This is very suitable to prove the correctness of functions on the domain. In particular, the semantics of a graph is the disjunction of the semantics of its edges. Our domain being a tuple, the semantics is the disjunction of each elements of the tuple, thus one can drop some elements of the tuples losing precision but not correctness.

Therefore, we provide a widening and a union (along with their correctness proofs) where precision/cost can be tuned to the specific needs of the context where the domain is used.

Here we only presented the domain, however we have designed the translation of the formulae into the domain and most of them are already proven which convince that the domain together with its semantics is suitable.

The domain was designed with the goal of building a translation toward/from existing shape-graphs, so we believe that the domain along with its semantics will prove useful for both separation-logic and also heap-shape analysis. Please remark that 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).

References

[1] *Appendix for* A*n abstract domain for separation logic formulae: http://www.enseignement.polytechnique.fr/profs/informatique/Elodie-Jane.Sims/publications/EAAI06/annexes/.*

[2] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, France, March 1978.

[3] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL'77*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[4] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL'79*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY.

[5] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *POPL'01*, pages 14–26, 2001.

[6] Patrick Lam, Viktor Kuncak, and Martin Rinard. Generalized typestate checking for data structure consistency. In *6th International Conference on Verification, Model Checking and Abstract Interpretation*, 2005.

[7] A. Miné. A new numerical abstract domain based on difference-bound matrices. In *PADO II*, volume 2053 of *LNCS*, pages 155–172. Springer-Verlag, May 2001.

[8] D.J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, 2002.

[9] D.J. Pym, P.W. O'Hearn, and H. Yang. Possible worlds and resources: The semantics of BI. *Theoretical Computer Science*, 315(1):257–305, 2004. Erratum: p. 285, l. -12: ", for some $P', Q \equiv P; P'$ " should be "$P \vdash Q$".

[10] J. C. Reynolds. Separation logic : A logic for shared mutable data structures. In *LICS'02*, pages 55–74, Denmark, 2002. IEEE Computer Society.

[11] M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. In *ACM Transactions on Programming Languages and Systems*, 2002.

[12] Élodie-Jane Sims. Extending separation logic with fixpoints and postponed substitution. *Theoretical Computer Science*, 5775, 2005.