

Precise Set Sharing Analysis for Java Programs

Mario Méndez-Lojo¹ Manuel Hermenegildo^{1,2}

¹University of New Mexico (USA)

²Technical University of Madrid and IMDEA-Software (Spain)

VMCAI'08, January 8

Motivation

- Computing, at compile time, precise approximations of memory state at any program point —classic static analysis problem.
- Sharing representations proved very useful in several applications (e.g., parallelization in logic programming [JL89, MH91]), but not used to date in OO programming.
- An exception is [Spoto05], which uses *pair* sharing as abstraction.

Goals

- Design an analysis based on a *set* sharing heap representation.
- Study its precision (specially in comparison to *pair* sharing).
- We develop an interprocedural, context-sensitive, multivariant analysis for *Java bytecode* based on Abstract Interp. [CC77].

Standard Domain Σ and Basic Definitions

- The set of distinct identifiers defined in the program is \mathcal{V} ; any element in \mathcal{V} has a type included in \mathcal{K} , the program-defined classes.
- Concrete states $\delta : \Sigma$ are pairs (frame,memory) that represent a valid heap configuration.
- A *frame* maps elements of \mathcal{V} to `null` or a location, which is linked to an object by the *memory* function.
- Objects consist of a type and a frame, the latter representing the fields.
- The set of locations reachable from $v : \mathcal{V}$ in state $\delta : \Sigma$ is obtained by applying the reachability function $R(\delta, v)$.

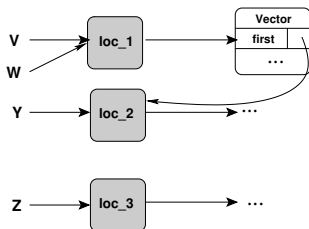
Sharing in the Concrete Domain

Variables $V = \{v_1, \dots, v_n\}$ share in state δ if there is at least one common location in their reachability sets:

$$share(\delta, V) \Leftrightarrow \bigcap_{i=1}^n R(\delta, v_i) \neq \emptyset$$

Example:

```
v = new Vector();
w = v;
y = v.first;
z = new Vector();
```



It holds that $share(\delta, \{v, w, y\}) \wedge share(\delta, \{z\}) \wedge share(\delta, \{v\}) \wedge \dots$

Abstract Domain

An abstract state σ is an element of $\mathcal{D} = \mathcal{DS} \times \mathcal{DN} \times \mathcal{DT}$.

- $\mathcal{DS} = \mathcal{P}(\mathcal{P}(\mathcal{V}))$ is the sharing component.
- $\mathcal{DN} = \mathcal{P}(\mathcal{V} \mapsto \{null, nnull, unk\})$ tells whether a given variable is definitely null, non null, or of unknown nullness.
- $\mathcal{DT} = \mathcal{P}(\mathcal{V} \mapsto \mathcal{P}(\mathcal{K}))$ contains a list with all the possible types of the variable.

Concretization function $\gamma : \mathcal{D} \mapsto \mathcal{P}(\Sigma)$

$$\gamma(sh, nl, \tau) = \{\delta \in \Sigma \mid \forall V \subseteq \mathcal{V}, share(\delta, V)\}$$

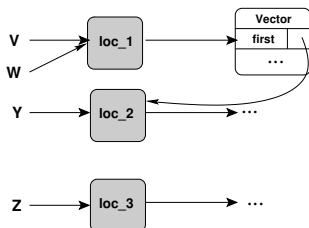
and $\nexists W, V \subset W \subseteq \mathcal{V}$ s.t. $share(\delta, W) \Rightarrow V \in sh$

, and $R(\delta, v) = \emptyset$ if $nl(v) = null$, and $R(\delta, v) \neq \emptyset$ if $nl(v) = nnull$

Sharing in the Abstract Domain

For the example already shown:

```
v = new Vector();
w = v;
y = v.first;
z = new Vector();
```



Given an abstract state

$$\begin{aligned}
 & (\{ \{v, w, y\}, \{z\} \}, \\
 & \{v/[nnull], w/[nnull], y/[unk], z/[nnull]\}, \\
 & \{v/[Vector], w/[Vector], y/[Element], z/[Vector]\})
 \end{aligned}$$

, δ is an element of its concretization.

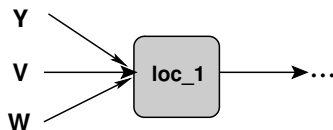
Concretization Function γ — Examples

(We omitted the type information for simplicity.)

abstract state σ

$\{\{v, w\}, \{v, y\}, \{w, y\}\}$
 $\{v/[nnull], w/[nnull], y/[nnull]\}$

concrete state δ



$\delta \in \gamma(\sigma)?$

?

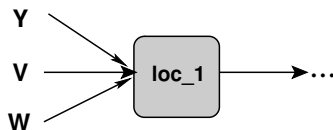
Concretization Function γ — Examples

(We omitted the type information for simplicity.)

abstract state σ

$\{\{v, w\}, \{v, y\}, \{w, y\}\}$
 $\{v/[nnull], w/[nnull], y/[nnull]\}$

concrete state δ



$\delta \in \gamma(\sigma)?$

no

Concretization Function γ — Examples

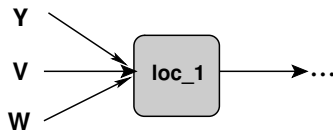
(We omitted the type information for simplicity.)

abstract state σ

concrete state δ

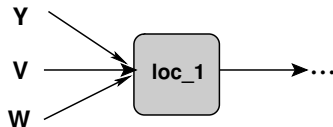
$\delta \in \gamma(\sigma)$?

$\{\{v, w\}, \{v, y\}, \{w, y\}\}$
 $\{v/[nnull], w/[nnull], y/[nnull]\}$



no

$\{\{v, w\}, \{v, w, y\}\}$
 $\{v/[nnull], w/[nnull], y/[nnull]\}$



?

Concretization Function γ — Examples

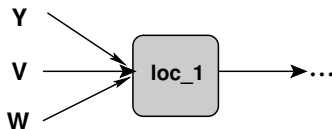
(We omitted the type information for simplicity.)

abstract state σ

concrete state δ

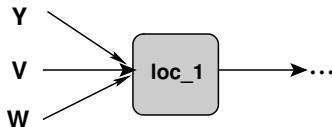
$\delta \in \gamma(\sigma)$?

$\{\{v, w\}, \{v, y\}, \{w, y\}\}$
 $\{v/[nnull], w/[nnull], y/[nnull]\}$



no

$\{\{v, w\}, \{v, w, y\}\}$
 $\{v/[nnull], w/[nnull], y/[nnull]\}$



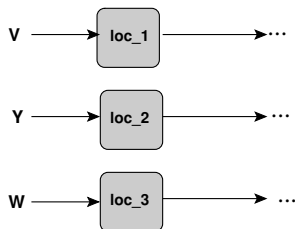
yes

Concretization Function γ — Examples

abstract state σ

$\{\{v\}, \{w\}, \{y\}, \{v, w\},$
 $\{v, y\}, \{w, y\}, \{v, w, y\}\}$
 $\{v/[nnull], w/[nnull], y/[nnull]\}$

concrete state δ



$\delta \in \gamma(\sigma)?$

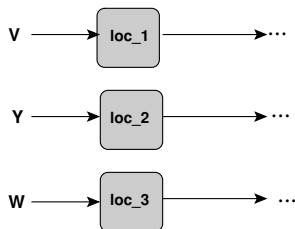
?

Concretization Function γ — Examples

abstract state σ

$\{\{v\}, \{w\}, \{y\}, \{v, w\},$
 $\{v, y\}, \{w, y\}, \{v, w, y\}\}$
 $\{v/[nnull], w/[nnull], y/[nnull]\}$

concrete state δ



$\delta \in \gamma(\sigma)?$

yes

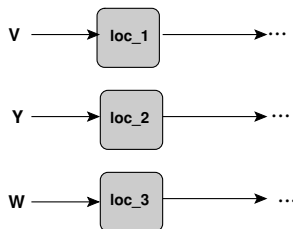
Concretization Function γ — Examples

abstract state σ

concrete state δ

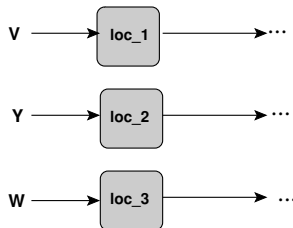
$\delta \in \gamma(\sigma)$?

$\{\{v\}, \{w\}, \{y\}, \{v, w\},$
 $\{v, y\}, \{w, y\}, \{v, w, y\}\}$
 $\{v/[nnull], w/[nnull], y/[nnull]\}$



yes

$\{\{v\}, \{w\}, \{y\}, \{v, w\},$
 $\{v, y\}, \{w, y\}, \{v, w, y\}\}$
 $\{v/[nnull], w/[null], y/[nnull]\}$



?

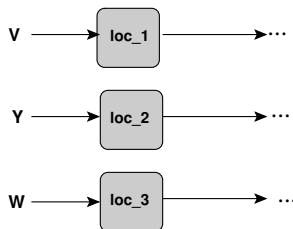
Concretization Function γ — Examples

abstract state σ

concrete state δ

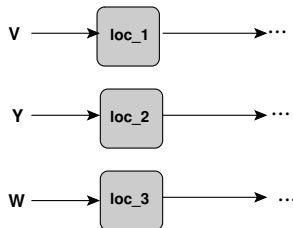
$\delta \in \gamma(\sigma)$?

$\{\{v\}, \{w\}, \{y\}, \{v, w\},$
 $\{v, y\}, \{w, y\}, \{v, w, y\}\}$
 $\{v/[nnull], w/[nnull], y/[nnull]\}$



yes

$\{\{v\}, \{w\}, \{y\}, \{v, w\},$
 $\{v, y\}, \{w, y\}, \{v, w, y\}\}$
 $\{v/[nnull], w/[null], y/[nnull]\}$



invalid σ

Set Sharing vs. Pair Sharing

Set sharing allows the representation of more complex abstractions than pair sharing.

- *Example:* assume an initial abstract state whose sharing component is $sh_0 = \{\{v, w\}\}$, which has the same representation in pair and set sharing. What is the resulting set sharing state after evaluating $z=v$?
 - ▶ In **pair sharing** [SS05], $\mathcal{SC}^{PS} \llbracket z=v \rrbracket sh_0 = \{\{v, w\}, \{v, z\}, \{w, z\}\}$, representable in set sharing as $\{\{v, w\}, \{v, z\}, \{w, z\}, \{v, w, z\}\} = sh_1$.
 - ▶ In **set sharing**, $\mathcal{SC}^{SS} \llbracket z=v \rrbracket sh_0 = \{\{v, w, z\}\} = sh_2$, where $sh_2 \subset sh_1$.

Set Sharing vs. Pair Sharing: Results

The experiments compare pair and set sharing domains with no nullity or type information.

$$\%sh = 100 * \left(1 - \frac{|sh|}{2^{|v|-1}}\right)$$

	PS		SS	
	<i>#sh</i>	<i>%sh</i>	<i>#sh</i>	<i>%sh</i>
dyndisp*	640	60.37	435	73.07
clone	174	53.10	151	60.16
dfs	1573	96.46	1109	97.51
passau*	5828	94.56	3492	96.74
qsort	1481	67.41	1082	76.34
intqsort	2413	66.47	1874	75.65
pollet01*	793	89.81	1043	91.81
zipvector*	6161	68.71	5064	80.28
cleanness*	1300	63.63	1189	70.61
overall	20363	73.39	15439	80.24

Domain Combination, Multivariance

- An abstract state contains not only sharing, but also nullity and type information.
- The analysis is context sensitive and multivariant.
- The combination results in increased precision.

```

public void append(Vector v) {
   $\sigma_{10} = (\{\{this\}, \{this, v\}, \{v\}\}, \{this/\{nnull\}, v/\{unk\}\})$ 
   $\sigma_{20} = (\{\{this\}, \{v\}\}, \{this/\{nnull\}, v/\{unk\}\})$ 
  if (this != v) { //append v to the end of this

  } else {

  }

}

```

Domain Combination, Multivariance

- An abstract state contains not only sharing, but also nullity and type information.
- The analysis is context sensitive and multivariant.
- The combination results in increased precision.

```

public void append(Vector v) {
   $\sigma_{10} = (\{\{this\}, \{this, v\}, \{v\}\}, \{this / \{nnull\}, v / \{unk\}\})$ 
   $\sigma_{20} = (\{\{this\}, \{v\}\}, \{this / \{nnull\}, v / \{unk\}\})$ 
  if (this != v) { //append v to the end of this
     $\sigma_{1i} = \sigma_{2i} = (\{\{this, v\}\}, \{this / \{nnull\}, v / \{nnull\}\})$ 
  } else {

  }

}

```

Domain Combination, Multivariance

- An abstract state contains not only sharing, but also nullity and type information.
- The analysis is context sensitive and multivariant.
- The combination results in increased precision.

```

public void append(Vector v) {
   $\sigma_{10} = (\{\{this\}, \{this, v\}, \{v\}\}, \{this / \{nnull\}, v / \{unk\}\})$ 
   $\sigma_{20} = (\{\{this\}, \{v\}\}, \{this / \{nnull\}, v / \{unk\}\})$ 
  if (this != v) { //append v to the end of this
     $\sigma_{1i} = \sigma_{2i} = (\{\{this, v\}\}, \{this / \{nnull\}, v / \{nnull\}\})$ 
  } else {
     $\sigma_{1e} = \sigma_{10}$  and  $\sigma_{2e} = \perp$ 
  }
}
}

```

Domain Combination, Multivariance

- An abstract state contains not only sharing, but also nullity and type information.
- The analysis is context sensitive and multivariant.
- The combination results in increased precision.

```

public void append(Vector v) {
 $\sigma_{10} = (\{\{this\}, \{this, v\}, \{v\}\}, \{this / \{nnull\}, v / \{unk\}\})$ 
 $\sigma_{20} = (\{\{this\}, \{v\}\}, \{this / \{nnull\}, v / \{unk\}\})$ 
    if (this != v) { //append v to the end of this
 $\sigma_{1i} = \sigma_{2i} = (\{\{this, v\}\}, \{this / \{nnull\}, v / \{nnull\}\})$ 
    } else {
 $\sigma_{1e} = \sigma_{10}$  and  $\sigma_{2e} = \perp$ 
    }
 $\sigma_{1f} = \sigma_{1i} \sqcup \sigma_{1e} = (\{\{this\}, \{this, v\}, \{v\}\}, \{this / \{nnull\}, v / \{unk\}\})$ 
 $\sigma_{2f} = \sigma_{2i} \sqcup \perp = (\{\{this, v\}\}, \{this / \{nnull\}, v / \{nnull\}\})$ 
}

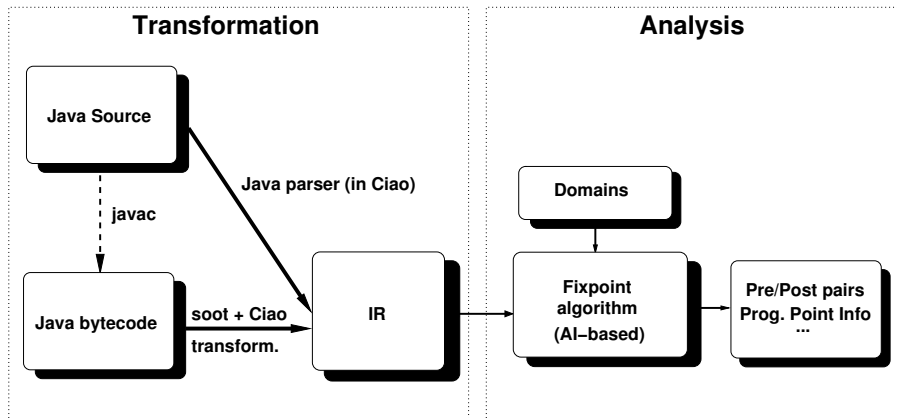
```

Precision Improvements due to Combining Domains

The table measures the percentage of program points that are deemed to be unreachable by analysis. Since the results are correct, a larger number for %up indicates better precision.

	PS	SS	SSNITau
	%up	%up	%up
dyndisp*	4.22	4.22	14.08
clone	7.31	7.31	24.39
dfs	3.92	3.92	10.78
passau*	1.19	1.19	5.98
qsort	23.24	23.24	23.24
intqsort	22.51	22.51	22.51
pollet01*	18.18	18.18	36.36
zipvector*	1.10	1.10	9.92
cleanness*	11.78	11.78	15.28
overall	10.38	10.38	18.06

Framework Pipeline



Analysis Framework

- Java (bytecode) programs are compiled to a generic intermediate representation [LOPSTR'07].
- The framework derives the semantics of a program given a particular abstract domain.
- A context-sensitive, multivariant, efficient fixpoint algorithm [FTfJP'07] is at the core of the system.
- Abstract domains, like set sharing, are plugins that the analysis designer adds to the framework.
- The definition of a domain contains:
 - ▶ The abstract domain operations (partial order, least upper bound, project, extend, etc.).
 - ▶ Abstract transfer functions for the primitive operations of the language (builtins).

Q & A

Check

<http://www.cliplab.org/~mario/>
for related publications



P. Cousot and R. Cousot.

Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.

In *Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.



D. Jacobs and A. Langen.

Accurate and Efficient Approximation of Variable Aliasing in Logic Programs.

In *1989 North American Conference on Logic Programming*. MIT Press, October 1989.



K. Muthukumar and M. Hermenegildo.

Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation.

In *1991 International Conference on Logic Programming*, pages 49–63. MIT Press, June 1991.



S. Secci and F. Spoto.

Pair-sharing analysis of object-oriented programs.

In *SAS*, pages 320–335, 2005.