Exam — Part 2

Xavier Rival

November 19, 2015

1 Part II: Exercise (object sensitive abstraction)

In this problem, we study *context sensitive* and *object sensitive* static analyses of languages with procedures and (a very modest model of) objects.

In this problem, we focus on the values that can be observed for object fields, and do not consider very carefully object updates (although that would be a logical follow up to the work proposed in this problem): due to this, and to keep definitions more concise, we will consider a very simplified language (but sufficient for our purpose).

Syntax. We consider a basic imperative language with procedures (a procedure has no return value, and has a single argument —except for `main` which has none), and with dynamically allocated records (that model objects). Each record has exactly two fields named $\mathbf{a}, \mathbf{b}$. A condition $\mathtt{c}$ is an implication between comparisons of the value of a record field and a constant.

$\bowtie$	$\in$	$\{=, <, >\}$	comparison operators
v	$\in$	$\mathbb{Z}$	values
$\mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{t}, \ldots$	$\in$	$\mathbb{X}$	variables
$\mathtt{i}$	$\in$	$\{\mathbf{a}, \mathbf{b}\}$	fields
$\mathtt{f}_0, \mathtt{f}_1, \ldots$	$\in$	$\mathbb{L}$	control states
$\mathtt{c}$	$::=$	$\mathtt{x} \cdot \mathtt{i} \bowtie v \Rightarrow \mathtt{x} \cdot \mathtt{i} \bowtie v$	conditions (for assertions)
$\mathtt{s}$	$::=$	$\mathtt{x} = \mathbf{new}(v, v)$	creation of an object and initialization
	$\mid$	$\mathtt{x} = \mathtt{y}$	variable assignment
	$\mid$	$\mathbf{if}(?)\mathbf{goto}\ \mathtt{f}_k$	non deterministic branching to $\mathtt{f}_k$
	$\mid$	$\mathbf{goto}\ \mathtt{f}_k$	deterministic branching to $\mathtt{f}_k$
	$\mid$	$\mathtt{f}(\mathtt{x})$	call of a procedure
	$\mid$	$\mathbf{assert}(\mathtt{c})$	assertion
$\mathtt{f}$	$::=$	$\mathbf{fun}\ \mathtt{f}(\mathtt{x})\{\mathtt{f}_0 : \mathtt{s}; \mathtt{f}_1 : \mathtt{s}; \ldots; \mathtt{f}_k \mathtt{s}; \mathtt{f}_{k+1} : \square\}$	a procedure with a single parameter
$\mathtt{p}$	$::=$	$(\mathtt{f}, \ldots, \mathtt{f})$	a set of procedures including a `main`

The body $\mathtt{b}$ of a procedure $\mathtt{f}$ consists of a sequence of statements, and we write $\mathtt{f}_n$ for the control state located right before the n-th statement (after the $(n-1)$-th). A statement is either the allocation of a new record, a branching (conditional or not), a (possibly recursive) procedure call or an assertion. Each procedure has a single argument and a body. A program comprises a set of procedures including a `main` (in the following we always consider it has no argument —though we do not reflect this in the above grammar).

```
        fun f(x){
f_0 :     assert(x · a < 0 ⇒ x · b > 0);
f_1 :     □
        }

        fun main(){
main_0 :  y = new(1, −2);
main_1 :  z = new(−1, 4);
main_2 :  f(y);
main_3 :  f(z);
main_4 :  □
        }
```

(a) Example 1

```
        fun f(x){
f_0 :     t = x;
f_1 :     if(?)goto f_5;
f_2 :     if(?)goto f_6;
f_3 :     assert(x · a < 0 ⇒ x · b > 0);
f_4 :     goto f_7;
f_5 :     t = new(2, −4);
f_6 :     f(t);
f_7 :     □
        }

        fun main(){
main_0 :  y = new(1, −2);
main_1 :  z = new(−1, 4);
main_2 :  f(y);
main_3 :  f(z);
main_4 :  □
        }
```

(b) Example 2

Figure 1: A couple of examples

Semantics. A state is either defined by a stack of pairs made of a control state and a local environment, or an error state:

σ		$(\in \mathbb{S})$	
σ	$::=$	$(\iota, \mu) \cdot \ldots \cdot (\iota, \mu)$	call stack, with current call at the left
	$\mid$	Ω	error state
μ	$\in$	$\mathbb{M} = \mathbb{X} \longrightarrow \mathbb{Z}^2$	
ι	$::=$	$\mathtt{f}_k$	

The semantics $[\![\mathtt{c}]\!] : \mathbb{S} \to \mathbb{B}$ (where $\mathbb{B} = \{\mathbf{true}, \mathbf{false}\}$) of condition $\mathtt{c}$ is a function that returns the boolean value of $\mathtt{c}$ in a state (these semantics are trivial and not given here). The initial configuration is $\sigma_{\mathrm{i}} = (\mathtt{main}_0, \epsilon)$, where ϵ denotes the function with empty domain (entry point of function main, with empty store). We let $\mathbb{S}_{\mathrm{i}} = \{\sigma_{\mathrm{i}}\}$. The transition relation is defined in the table below:

instruction	transition
$\mathtt{f}_k : \mathtt{x} = \mathbf{new}(v_a, v_b)$	$(\mathtt{f}_k, \mu) \cdot \sigma \to (\mathtt{f}_{k+1}, \mu[\mathtt{x} \mapsto (v_a, v_b)]) \cdot \sigma$
$\mathtt{f}_k : \mathtt{x} = \mathtt{y}$	$(\mathtt{f}_k, \mu) \cdot \sigma \to (\mathtt{f}_{k+1}, \mu[\mathtt{x} \mapsto \mu(\mathtt{y})]) \cdot \sigma$
$\mathtt{f}_k : \mathbf{if}(?)\mathbf{goto}\ \mathtt{f}_l$	$(\mathtt{f}_k, \mu) \cdot \sigma \to (\mathtt{f}_{k+1}, \mu) \cdot \sigma$
$\mathtt{f}_k : \mathbf{if}(?)\mathbf{goto}\ \mathtt{f}_l$	$(\mathtt{f}_k, \mu) \cdot \sigma \to (\mathtt{f}_l, \mu) \cdot \sigma$
$\mathtt{f}_k : \mathbf{goto}\ \mathtt{f}_l$	$(\mathtt{f}_k, \mu) \cdot \sigma \to (\mathtt{f}_l, \mu) \cdot \sigma$
$\mathtt{f}_k : \mathtt{g}(\mathtt{x})$	$(\mathtt{f}_k, \mu) \cdot \sigma \to (\mathtt{g}_0, \{\mathtt{t} \mapsto \mu(\mathtt{x})\}) \cdot (\mathtt{f}_k, \mu) \cdot \sigma$ where $\mathtt{t}$ is the parameter of $\mathtt{g}$
$\mathtt{f}_k : \mathbf{assert}(\mathtt{c})$	$(\mathtt{f}_k, \mu) \cdot \sigma \to (\mathtt{f}_{k+1}, \mu) \cdot \sigma$ if $[\![\mathtt{c}]\!](\mu) = \mathbf{true}$
$\mathtt{f}_k : \mathbf{assert}(\mathtt{c})$	$(\mathtt{f}_k, \mu) \cdot \sigma \to \Omega$ if $[\![\mathtt{c}]\!](\mu) = \mathbf{false}$
$\mathtt{f}_k : \square$	$(\mathtt{f}_k, \mu_g) \cdot (\mathtt{g}_l, \mu_f) \cdot \sigma \to (\mathtt{g}_{l+1}, \mu_f) \cdot \sigma$

We will consider the *reachable states semantics* defined by:

$$[\![\mathtt{p}]\!] = \{\sigma \mid \exists \sigma_0 \in \mathbb{S}_{\mathrm{i}},\ \sigma_0 \to^\star \sigma\} = \mathbf{lfp}\, F$$

where $F : \mathbb{S} \to \mathbb{S}$ is such that $F(S) = \mathbb{S}_{\mathrm{i}} \cup \{\sigma' \mid \exists \sigma \in S, \sigma \to \sigma'\}$.

Question 1 Executions.

Give a maximal execution trace for Example 1. Comment on the assertion in `f`*.*

Solution 1 *We write* $\mu = \{\mathtt{y} \mapsto (1,-2), \mathtt{z} \mapsto (-1,4)\}$*,* $\mu' = \{\mathtt{x} \mapsto (1,-2)\}$ *and* $\mu'' = \{\mathtt{x} \mapsto (1,4)\}$*. We get the trace:*

$$\begin{array}{l}(\mathtt{main}_0, \epsilon) \quad \to \quad (\mathtt{main}_1, \{\mathtt{y} \mapsto (1,-2)\}) \quad \to \quad (\mathtt{main}_2, \mu) \quad \to \quad (\mathtt{f}_0, \mu') \cdot (\mathtt{main}_2, \mu) \\ \quad \to \quad (\mathtt{f}_1, \mu') \cdot (\mathtt{main}_2, \mu) \quad \to \quad (\mathtt{f}_2, \mu') \cdot (\mathtt{main}_2, \mu) \quad \to \quad (\mathtt{main}_3, \mu) \\ \quad \to \quad (\mathtt{f}_0, \mu'') \cdot (\mathtt{main}_3, \mu) \quad \to \quad (\mathtt{f}_1, \mu'') \cdot (\mathtt{main}_3, \mu) \quad \to \quad (\mathtt{f}_2, \mu'') \cdot (\mathtt{main}_3, \mu) \\ \quad \to \quad (\mathtt{main}_4, \mu)\end{array}$$

Both records created in `main` *eventually reach the assertion, and obviously satisfy it.*

Question 2 Executions.

Comment on the assertion in `f` *in example 2. (as this example is larger, we do not ask to write down any execution trace, but you may do it if that helps your intuition).*

Solution 2 *Three records may reach the assertion (*`y` *and* `z` *defined in* `main` *will reach it for sure and* `t` *defined in* `f` *may reach it depending on the recursive calls, which are non deterministic). They all satisfy the assertion.*

Context sensitive analyses. We first consider several context sensitive analyses as considered in the lecture. In the following, we deliberately assume a very simple, non relational numerical abstraction, where each field is abstracted by its sign (we use the lattice of signs, where $\bot$ represents the empty set, $\top$ any set of integers, $[+]$ represents any set of strictly positive integers and $[-]$ any set of strictly negative integers; we write $\text{Sign}^\sharp$ for this abstract lattice and $\alpha_{\text{Signs}}, \gamma_{\text{Signs}}$ for its abstraction and concretization functions).

Question 3 Sign abstraction.

Give the abstract ordering on $\text{Sign}^\sharp$*, the definition of the* α_{Signs} *and* γ_{Signs} *functions, and show that it defines a Galois connection.*

Solution 3 *The abstract ordering is defined by* $\bot \sqsubseteq [-] \sqsubseteq \top$ *and* $\bot \sqsubseteq [+] \sqsubseteq \top$*. The adjoint functions are defined by:*

$$\begin{array}{lllll}
\alpha_{\text{Signs}} : & \mathcal{P}(\mathbb{Z}) & \longrightarrow & \text{Sign}^\sharp & \\
& \varnothing & \longmapsto & \bot & \\
& X (\neq \varnothing) & \longmapsto & \begin{cases} [-] & \text{if } X \subseteq \{n \in \mathbb{Z} \mid n < 0\} \\ [+] & \text{if } X \subseteq \{n \in \mathbb{Z} \mid n > 0\} \end{cases} & \\
\gamma_{\text{Signs}} : & \text{Sign}^\sharp & \longrightarrow & \mathcal{P}(\mathbb{Z}) & \\
& \bot & \longmapsto & \varnothing & \\
& [-] & \longmapsto & \{n \in \mathbb{Z} \mid n < 0\} & \\
& [+] & \longmapsto & \{n \in \mathbb{Z} \mid n > 0\} & \\
& \top & \longmapsto & \mathbb{Z} &
\end{array}$$

The Galois-connection property can be established by proving that $\alpha_{\text{Signs}}(X) \sqsubseteq X^\sharp \iff X \subseteq \gamma_{\text{Signs}}(X^\sharp)$ *by a simple case analysis on* $X^\sharp$*.*

For concision, if $(s_a, s_b) \in (\text{Sign}^\sharp)^2$, we also write $\gamma_{\text{Signs}}(s_a, s_b)$ for $\{(v_a, v_b) \mid v_a \in \gamma_{\text{Signs}}(s_a) \wedge v_b \in \gamma_{\text{Signs}}(s_b)\}$.

We define the following abstract domain and concretization functions:

$$\begin{array}{rcl}
\mathbb{M}^\sharp & = & \mathbb{X} \to (\mathrm{Sign}^\sharp)^2 \\
\mathbb{S}^\sharp & = & \mathbb{L} \to \mathbb{M}^\sharp \\
\gamma_{\mathbb{M}} : \mathbb{M}^\sharp & \longrightarrow & \mathcal{P}(\mathbb{M}) \\
M^\sharp & \longmapsto & \{\mu \in \mathbb{M} \mid \forall \mathtt{x},\ \mu(\mathtt{x}) \in \gamma_{\mathrm{Signs}}(M^\sharp(\iota_i)(\mathtt{x}))\} \\
\gamma_{\mathbb{S}} : \mathbb{S}^\sharp & \longrightarrow & \mathcal{P}(\mathbb{S}) \\
S^\sharp & \longmapsto & \{(\iota_0, \mu_0) \cdot \ldots \cdot (\iota_n, \mu_n) \mid \forall i,\ \mu_i \in \gamma_{\mathbb{M}}(S^\sharp(\iota_i))\}
\end{array}$$

Question 4 Non context sensitive analysis (0-CFA).

Show it is possible to define an abstraction function so as to form a Galois connection.
Show the best abstractions of the sets of reachable states for both examples, at point $\mathtt{f}_0$*. Can an analysis based on this abstraction verify that both programs are correct (i.e., that the assertion is never violated).*

Solution 4 *We can prove by equivalence that the function below defines a Galois connection:*

$$\begin{array}{rcl}
\alpha_{\mathbb{M}} : \mathcal{P}(\mathbb{M}) & \longrightarrow & \mathbb{M}^\sharp \\
M & \longmapsto & \lambda \mathtt{x} \cdot (\alpha_{\mathrm{Signs}}(V_a), \alpha_{\mathrm{Signs}}(V_b)) \\
 & & \quad \textit{where } V = \{\mu(\mathtt{x}) \mid \mu \in M\} \\
 & & \quad \textit{and } V_a = \{v_a \mid \exists v_b, (v_a, v_b) \in V\}, V_b = \{v_b \mid \exists v_a, (v_a, v_b) \in V\} \\
\alpha_{\mathbb{S}} : \mathcal{P}(\mathbb{S}) & \longrightarrow & \mathbb{S}^\sharp \\
S & \longmapsto & \lambda \iota \cdot \alpha_{\mathbb{M}}(\{\mu \mid (\ldots \cdot (\iota, \mu) \cdot \ldots) \in S\})
\end{array}$$

In example 1, we get

$$\alpha_{\mathbb{S}}([\![\mathtt{p}]\!])(\mathtt{f}_0)(\mathtt{x}) = (\top, \top)$$

In example 2, we obtain the same abstraction at point $\mathtt{f}_0$*.*

As a consequence, and for both programs, an analysis based on this abstraction will fail to be verify the assertion never crashes.

Before we discuss improvements of this analysis, we propose to formalize partially the computation of abstract invariants.

Question 5 Static analysis.

We write $\delta^\sharp_{\iota,\iota'} : \mathbb{M}^\sharp \to \mathbb{M}^\sharp$ *for the abstraction of the transition relation, which meets the soundness condition below and should be as precise as possible:*

$$\forall M^\sharp \in \mathbb{M}^\sharp, \forall ((\iota, \mu) \cdot \sigma), ((\iota', \mu') \cdot \sigma') \in \mathbb{S},$$
$$\left.\begin{array}{rl} & (\iota, \mu) \cdot \sigma \to (\iota', \mu') \cdot \sigma' \\ \wedge & \mu \in \gamma_{\mathbb{M}}(M^\sharp) \end{array}\right\} \Longrightarrow \mu' \in \gamma_{\mathbb{M}}(\delta^\sharp_{\iota,\iota'}(M^\sharp))$$

Define all $\delta^\sharp_{\iota,\iota'}$ *when* ι *is the control state of one of the following statements:*

- $\mathtt{x} = \boldsymbol{new}(v, v)$
- $\mathtt{x} = \mathtt{y}$
- $\boldsymbol{if}(?)\boldsymbol{goto}\ \mathtt{f}_k$
- $\mathtt{f}(\mathtt{x})$
- $\square$ *(end of a function)*

Comment on the precision of the later, and its effect on the analysis.

Solution 5 *In all the following cases, we can determine the following abstract transfer functions are the best sound transformers:*

- *Case of* $\mathtt{f}_k : \mathtt{x} = \textbf{\textit{new}}(v_a, v_b)$*:*

$$\delta^\sharp_{\mathtt{f}_k,\mathtt{f}_{k+1}}(M^\sharp) = M^\sharp[\mathtt{x} \mapsto (\alpha_{\text{Signs}}(\{v_a\}), \alpha_{\text{Signs}}(\{v_b\}))]$$

- *Case of* $\mathtt{f}_k : \mathtt{x} = \mathtt{y}$

$$\delta^\sharp_{\mathtt{f}_k,\mathtt{f}_{k+1}}(M^\sharp) = M^\sharp[\mathtt{x} \mapsto M^\sharp(\mathtt{y})]$$

- *Case of* $\mathtt{f}_k : \textbf{\textit{if}}(?)\textbf{\textit{goto}}\ \mathtt{f}_l$

$$\delta^\sharp_{\mathtt{f}_k,\mathtt{f}_{k+1}}(M^\sharp) = M^\sharp$$
$$\delta^\sharp_{\mathtt{f}_k,\mathtt{f}_l}(M^\sharp) = M^\sharp$$

- *Case of* $\mathtt{f}_k : \mathtt{g}(\mathtt{x})$

$$\delta^\sharp_{\mathtt{f}_k,\mathtt{g}_0}(M^\sharp) = \{\mathtt{t} \mapsto M^\sharp(\mathtt{x})\}$$

  where $\mathtt{t}$ *is the name of the parameter of* $\mathtt{g}$*.*

- *Case of* $\mathtt{f}_k : \square$*:*
  for each $\mathtt{g}_l : \mathtt{f}(\mathtt{t})$*, we have:*

$$\delta^\sharp_{\mathtt{f}_k,\mathtt{g}_{l+1}}(M^\sharp) = \{\mathtt{x} \mapsto \top\}$$

  This transfer function returns a very *imprecise result, as it has no knowledge of the state before the call (for this reason, it is not possible to produce a more precise, sound result).*

The analysis of the return transfer function loses all information. In fact the abstract information about the variables of the callee is lost at the call site.

The imprecision observed in the previous question is not acceptable. Therefore, we propose to address it. Given a program $\mathtt{p}$, we define a second transition relation $\rightsquigarrow$, which is defined just like $\rightarrow$ except for the following cases:

instruction	transition
$\mathtt{f}_k : \mathtt{g}(\mathtt{x})$	$(\mathtt{f}_k, \mu) \cdot \sigma \rightsquigarrow (\mathtt{g}_0, \{\mathtt{t} \mapsto \mu(\mathtt{x})\}) \cdot (\mathtt{f}_k, \mu) \cdot \sigma$ where $\mathtt{t}$ is the parameter of $\mathtt{g}$ $(\mathtt{f}_k, \mu) \cdot \sigma \rightsquigarrow (\mathtt{f}_{k+1}, \mu) \cdot \sigma$
$\mathtt{f}_k : \square$	defines no transition

Note that a call now defines *two* transitions.

Question 6 Improving the analysis of procedure call / procedure return.

Describe informally the effect of this change. Compare $[\![\mathtt{p}]\!]$ *with the set of states that are reachable from* σ_{i}*, and using* $\rightsquigarrow$ *(compare sets* $\{\sigma \mid \sigma_{\text{i}} \rightarrow^\star \sigma\}$ *and* $\{\sigma \mid \sigma_{\text{i}} \rightsquigarrow^\star \sigma\}$*, and explain how to prove this property —the proof may be long, thus it is fine to only provide the skeleton of the proof). Deduce a way to derive precise abstract information after a procedure return, from information at the call site.*

Solution 6 *This new transition relation computes directly the states observed after a procedure return from the states observed before the call; it preserves the intuitive meaning of* $\rightarrow$ *as procedures cannot modify local variables of other procedures (if they could do that, we would need a more complex* $\rightsquigarrow$ *relation in order to integrate information both at the call site, and at the procedure return site).*

We can actually prove that:

$$\{\sigma \mid \sigma_{\text{i}} \rightarrow^\star \sigma\} \subseteq \{\sigma \mid \sigma_{\text{i}} \rightsquigarrow^\star \sigma\}$$

The inclusion may be strict when an assertion in a procedure fails, as this means the return point would not be reachable with relation $\rightarrow$ *whereas it would be reachable for* $\rightsquigarrow$*. The proof proceeds by double induction on the maximal call depth and on the length of a derivation that* $\sigma_0 \rightarrow^\star \sigma_1$*.*

We deduce the following abstract transition relation for procedure calls and returns:

- *Case of* $\mathtt{f}_k : \mathtt{g}(\mathtt{x})$

$$\delta^\sharp_{\mathtt{f}_k,\mathtt{g}_0}(M^\sharp) = \{\mathtt{t} \mapsto M^\sharp(\mathtt{x})\}$$
$$\delta^\sharp_{\mathtt{f}_k,\mathtt{f}_{k+1}}(M^\sharp) = M^\sharp$$

  where $\mathtt{t}$ *is the name of the parameter of* $\mathtt{g}$.

- *Case of* $\mathtt{f}_k : \square$*:*
  it now defines no transition for $\rightsquigarrow$*, thus, for each* $\mathtt{g}_l : \mathtt{f}(\mathtt{t})$*, we have:*

$$\delta^\sharp_{\mathtt{f}_k,\mathtt{g}_{l+1}}(M^\sharp) = \{\mathtt{x} \mapsto \bot\}$$

Question 7 Definition of the static analysis.

Describe how we can obtain an abstract join operator.
Define how the abstract semantics can be computed. Show its soundness and its termination.

Solution 7 *A sound abstract join operator* $\sqcup^\sharp$ *can be obtained by applying pointwisely the abstract join operator of the constant lattice to abstract elements viewed as functions.*

Finally, a static analysis function can be obtained by a classic approximate fixpoint transfer, using abstraction of initial states $I^\sharp_0$ *and abstract semantic function defined below:*

$$I^\sharp_0 : \left\{ \begin{array}{rcl} \mathtt{main}_0 & \mapsto & (\mathtt{x} \mapsto \top) \\ \iota(\neq \mathtt{main}_0) & \mapsto & (\mathtt{x} \mapsto \bot) \end{array} \right.$$

$$\begin{array}{rccl} F^\sharp : & \mathbb{S}^\sharp & \longrightarrow & \mathbb{S}^\sharp \\ & I^\sharp & \longmapsto & I^\sharp_0 \sqcup^\sharp (\sqcup^\sharp \{\delta^\sharp_{\iota,\iota'}(I^\sharp(\iota)) \mid \iota, \iota' \in \mathbb{L}\}) \end{array}$$

The computation of the least-fixpoint of $F^\sharp$ *terminates as the abstract lattice is of finite height, and it can be proved to approximate the concrete semantics by fixpoint transfer.*

Context sensitive analyses refine the abstraction using calling contexts: for instance 1-CFA will discriminate states depending on the call-site of the current procedure, whereas k-CFA discriminates them according to the k ongoing procedure calls on top of the call stack.

Question 8 Partially context sensitive analysis (k-CFA).

Define the abstract domain that would be used by a 1*-CFA analysis.*
How do the analysis transfer functions need be updated ?
Explain whether it allows to analyze successfully (i.e., compute invariants that are precise enough to establish none of the assertions will be violated) example 1 and example 2. In case one of the examples cannot be analyzed successfully with 1*-CFA, would a* k*-CFA work ? (if so, indicate for which* k*).*

Solution 8 *An "abstract memory" (function from variables to abstraction of the records they may store) should be associated to each* pair *of control states* $(\mathtt{f}_k, \mathtt{g}_l)$ *where* $\mathtt{f}_k$ *denotes the current control state and* $\mathtt{g}_l$ *the control state at which the current procedure has been called. A side exception needs to be made for* `main` *which is not called from any other context. The memory abstraction part is not modified. Therefore, we define the following abstract domain as follows:*

$$\mathbb{S}^\sharp = (\mathbb{L}^2 \to \mathbb{M}^\sharp) \times (\{\mathtt{main}_0, \ldots, \mathtt{main}_k\} \to \mathbb{M}^\sharp)$$

If $S^\sharp_0 \in \mathbb{L}^2 \to \mathbb{M}^\sharp$ *and* $S^\sharp_1 \in \{\mathtt{main}_0, \ldots, \mathtt{main}_k\} \to \mathbb{M}^\sharp$*, then:*

$$\begin{array}{rcl} \gamma_{\mathbb{S}}(S^\sharp_0, S^\sharp_1) & = & \{(\iota_0, \mu_0) \cdot \ldots \cdot (\iota_n, \mu_n) \mid \forall i,\ \mu_i \in \gamma_{\mathbb{M}}(S^\sharp_0(\iota_i, \iota_{i+1}))\} \\ & & \uplus \{(\mathtt{main}_k, \mu) \mid \forall \mathtt{x},\ \mu \in \gamma_{\mathbb{M}}(S^\sharp_1(\mathtt{main}_k))\} \end{array}$$

The transfer functions corresponding to statements internal to procedures do not change much (only the signature of $\delta^\sharp_{\_,\_,\_}$ becomes a bit more complex. Procedure calls need be updated to reflect call stack updates.

The 1-CFA analysis will work in the case of example 1, as it will distinguish the calls on the two records created in `main`*.*

The case of example 2 is less favorable. The 1-CFA analysis will fail to achieve the property, as the call at point $\mathtt{f}_6$ *may be on a pair defined in* `main`*, or on the pair* $(2, -4)$ *created at* $\mathtt{f}_5$*: therefore, the 1-CFA analysis will consider that both fields may have either positive or negative values at the entry into* `f`*, which prevents the verification of the assertion. The situation is the same for any k-CFA analysis because any pair can be propagated along an unbounded sequence of calls to* `f` *before it reaches the assertion.*

Object sensitive abstraction. We observe that in both examples, the properties of the records can be fully determined by the site at which their values where first set ($\mathtt{main}_0, \mathtt{main}_1$ in example 1). Thus we propose to drop any context sensivity, but to use *object sensitivity*, where each record will be related to its creation site in the abstract level. For example, in example 1 and at the entry of $\mathtt{f}_0$:

- if the record pointed to by `x` was created at point $\mathtt{main}_0$, its contents can be abstracted by $([+], [-])$;
- if it was created at point $\mathtt{main}_1$, its contents can be abstracted by $([-], [+])$.

In both cases, the information allows to show the condition of the assertion is satisfied.

Question 9 Example 2.

Show intuitively that this approach can also cope with example 2, and give the number of cases that need be considered during the analysis.

Solution 9 *The intuitive proof is very similar. At point* $\mathtt{f}_0$*, the value stored in* `x` *may have been created at only three points:* $\mathtt{main}_0, \mathtt{main}_1, \mathtt{f}_5$*.*

- *if the record pointed to by* `x` *was created at point* $\mathtt{main}_0$*, its contents can be abstracted by* $([+], [-])$*;*
- *if it was created at point* $\mathtt{main}_1$*, its contents can be abstracted by* $([-], [+])$*;*
- *last, if it was created at point* $\mathtt{f}_5$*, its contents can be abstracted by* $([+], [-])$*.*

Again, in all cases, the assertion at $\mathtt{f}_3$ *can be verified.*

Before we can formalize this abstraction, we need to actually extend the concrete semantics, so that, for each state, each record also contains the information about the point at which it was created. For this, we need to extend the definition of μ into:

$$\mu \in \mathbb{X} \to \mathbb{L} \times \mathbb{Z} \times \mathbb{Z}$$

Now, $\mu(\mathtt{x}) = (\iota, v_a, v_b)$ means that `x` stores pair (v_a, v_b) and that this pair was created at control state ι or copied from a pair created at control state ι (possibly indirectly, as this may be the result of a chain of copies).

Only one of the rules in the definition of the transition relation needs to be changed:

instruction	transition
$\mathtt{f}_k : \mathtt{x} = \mathbf{new}(v_a, v_b)$	$(\mathtt{f}_k, \mu) \cdot \sigma \to (\mathtt{f}_{k+1}, \mu[\mathtt{x} \mapsto (\mathtt{f}_k, v_a, v_b)]) \cdot \sigma$
$\vdots$	$\vdots$

Question 10 Formalize the abstract domain.

Define the abstract domain for object sensitive analysis that allows to treat the above example, and give the corresponding concretization function (as shown in the example in the beginning of this part, the object sensitive abstraction should abstract separately pairs that stem from distinct allocation sites).

Solution 10 *The new abstract domain will maintain, for each control state and each variable, a function from control states into signs, which will ensure pairs that stem from distinct call sites are never abstracted together into a same predicate in the lattice of signs:*

$$\mathbb{S}^\sharp = \mathbb{L} \to (\mathbb{X} \to (\mathbb{L} \to (\mathrm{Sign}^\sharp)^2))$$

Likewise, we need to define the concretization function below:

$$\begin{array}{rccl} \gamma_{\mathbb{S}} : & \mathbb{S}^\sharp & \longrightarrow & \mathcal{P}(\mathbb{S}) \\ & S^\sharp & \longmapsto & \{(\iota_0, \mu_0) \cdot \ldots \cdot (\iota_n, \mu_n) \mid \forall i, \forall \mathtt{x}, \mu_i(\mathtt{x}) = (\bar{\iota}, v_a, v_b) \Rightarrow (v_a, v_b) \in \gamma_{\mathrm{Signs}}(S^\sharp(\iota_i)(\mathtt{x})(\bar{\iota}))\} \end{array}$$

We note that this definition captures the relation between $\bar{\iota}$ (the creation site) and the contents of the record.

Question 11 Static analysis.

Describe the changes to the abstract semantics defined in question 5.

Solution 11 *The changes to the abstract semantics reflect the changes to the concrete semantics:*

- *case of* $\mathtt{f}_k : \mathtt{x} = \boldsymbol{new}(v_a, v_b)$*:*

$$\delta^\sharp_{\mathtt{f}_k, \mathtt{f}_{k+1}}(M^\sharp) = M^\sharp[\mathtt{x} \mapsto \{\mathtt{f}_k \mapsto (\alpha_{\mathrm{Signs}}(\{v_a\}), \alpha_{\mathrm{Signs}}(\{v_b\}))\}]$$

- *other cases are similar to the question 5.*

Question 12 Advantages of the object sensitive approach.

Informally explain when the object sensitive abstraction is appropriate and when it is unlikely to give better precision (beyond the language studied in this problem).

Solution 12 *It allows to distinguish record fields values based on their creation site, so it is particularly useful when the records created in different sites have properties that (1) should be distinguished to guarantee the success of the analysis and (2) that would not be discriminated in the base domain (this is the case here as we chose a deliberatly simplified numerical abstract domain; with a more realistic numeric domain, object sensitivity will still help to verify more complex properties).*

By contrast, it will most likely not to very precise when objects get modified often after their creation site, and it is not possible to easily related field properties and call sites due to these updates. In that case, context sensitive approaches may improve the analysis precision, if the field updates can be related to certain sets of call stacks.