Exam

December, 2nd, 2015

1 Part I: Exercise (congruences abstraction)

Consider the following function over $\mathcal{P}(\mathbb{Z})$:

$$\begin{array}{rlll} F: & \mathcal{P}(\mathbb{Z}) & \longrightarrow & \mathcal{P}(\mathbb{Z}) \\ & X & \longmapsto & \{1\} \cup \{x+4 \mid x \in X \wedge x < 10\} \cup \{x-4 \mid x \in X \wedge x < 10\} \\ & & & \cup \{x+2 \mid x \in X \wedge x \geqslant 10\} \end{array}$$

1. Explain why F has a least fixpoint. Show how it can be computed. Give this least-fixpoint (give the result, and prove it is indeed the lfp).
2. We consider the congruence domain. Recall its definition (abstract elements and concretization function).
3. Explain why F has a best over-approximation $F^\sharp$ in the congruence domain lattice, and provide its definition.
4. What about the least-fixpoint of $F^\sharp$? Show the corresponding computation in the abstract.

2 Part II: Problem (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 **a**, **b**. A condition `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
i	$\in$	$\{\mathbf{a},\mathbf{b}\}$	fields
$\mathtt{f}_0,\mathtt{f}_1,\ldots$	$\in$	$\mathbb{L}$	control states of function $\mathtt{f}$
$\mathtt{c}$	$::=$	$\mathtt{x}\cdot\mathrm{i}\bowtie v \Rightarrow \mathtt{x}\cdot\mathrm{i}\bowtie v$	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 `b` of a procedure `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).

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	$\in$	$\mathbb{M} = \mathbb{X} \longrightarrow \mathbb{Z}^2$	memory state, maps each value to a pair of integers
ι	$::=$	$\mathtt{f}_k$	control state

The semantics $[\![\mathtt{c}]\!] : \mathbb{S} \to \mathbb{B}$ (where $\mathbb{B} = \{\mathbf{true},\mathbf{false}\}$) of condition `c` is a function that returns the boolean value of `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 `t` is the parameter of `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 $F(S) = \mathbb{S}_{\mathrm{i}} \cup \{\sigma' \mid \exists\sigma \in S, \sigma \to \sigma'\}$.

```
         fun f(x){
f0 :       assert(x · a < 0 ⇒ x · b > 0);
f1 :       □
         }

         fun main(){
main0 :    y = new(1, −2);
main1 :    z = new(−1, 4);
main2 :    f(y);
main3 :    f(z);
main4 :    □
         }
```

(a) Example 1

```
         fun f(x){
f0 :       t = x;
f1 :       if(?)goto f5;
f2 :       if(?)goto f6;
f3 :       assert(x · a < 0 ⇒ x · b > 0);
f4 :       goto f7;
f5 :       t = new(2, −4);
f6 :       f(t);
f7 :       □
         }

         fun main(){
main0 :    y = new(1, −2);
main1 :    z = new(−1, 4);
main2 :    f(y);
main3 :    f(z);
main4 :    □
         }
```

(b) Example 2

Figure 1: A couple of examples

Question 1 Executions.

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

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).*

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 $\mathrm{Sign}^\sharp$ for this abstract lattice and $\alpha_{\mathrm{Signs}}, \gamma_{\mathrm{Signs}}$ for its abstraction and concretization functions).

Question 3 Sign abstraction.

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

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

We define the following abstract domain and concretization functions:

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

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).*

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:*

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

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.

The imprecision observed in the previous question is not acceptable. Therefore, we propose to address it. Given a program p, we define a second transition relation $\rightsquigarrow$, which is defined just like $\to$ 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 t is the parameter of 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* $\{\sigma \mid \sigma_{\mathrm{i}} \to^{\star} \sigma\}$ *and* $\{\sigma \mid \sigma_{\mathrm{i}} \rightsquigarrow^{\star} \sigma\}$, *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.*

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.

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).*

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.

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).

Question 11 Static analysis.

Describe the changes to the abstract semantics defined in 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).