Written exam
MPRI 2-6, year 2016–2017

30 November 2016

Duration: 3 hours (8:45–11:45)

The only documents allowed are your own printed copy of the course slides and your personal notes.
The use of electronic devices (computers, phones) is prohibited.
The exam is composed of two independent exercises. Please answer the exercises on different sheets of paper.
The questions are written in English. You can answer either in English or in French.
It will not be answered to any question during the exam. In case of an ambiguity or an error in the definitions or the questions, it is part of the exam to correct them and answer to the best of your abilities.

Exercise 1: Strictness Analysis

Please answer Exercise 1 and Exercise 2 on different sheets of paper.

We consider a first-order language with recursive functions. A program is composed of a set of function names $\mathbb{F} \overset{\text{def}}{=} \{ f_1, \ldots, f_N \}$, a set of formal argument variables $\mathbb{V} \overset{\text{def}}{=} \{ x_1, \ldots, x_n \}$, and a map $\mathit{body} : \mathbb{F} \to e$ associating to each function name $f \in \mathbb{F}$ an expression $\mathit{body}(f)$ in the grammar:

$$\begin{array}{rcll}
e & ::= & c & \textit{(constant, } c \in \mathbb{Z}\textit{)} \\
& | & x & \textit{(variable, } x \in \mathbb{V}\textit{)} \\
& | & e_1 + e_2 & \textit{(addition)} \\
& | & \textbf{if } e_1 = 0 \textbf{ then } e_2 \textbf{ else } e_3 \textbf{ fi} & \textit{(conditional)} \\
& | & f(e_1, \ldots, e_n) & \textit{(function call, } f \in \mathbb{F}\textit{)}
\end{array}$$

To simplify, we assume that there are no global nor local variables, only argument variables, and that all functions have $\mathbb{V}$ as list of formal arguments (i.e., the functions all have arity n).

Let $\mathbb{Z}_\perp \overset{\text{def}}{=} \mathbb{Z} \cup \{\perp\}$ be the set of integers enriched with a special value $\perp$, denoting the absence of a definite value. We denote as $\mathbb{E} \overset{\text{def}}{=} \mathbb{V} \to \mathbb{Z}_\perp$ the set of variable environments, as $\mathbb{D} \overset{\text{def}}{=} \mathbb{E} \overset{m}{\to} \mathbb{Z}_\perp$ the domain of *monotonic* functions, and as $\mathbb{S} \overset{\text{def}}{=} \mathbb{F} \to \mathbb{D}$ the set of function environments, associating a semantics in $\mathbb{D}$ to each function name in $\mathbb{F}$. Given a function environment $S \in \mathbb{S}$, we define the semantics $\mathsf{E}[\![\, e \,]\!]\,(S) \in \mathbb{D}$ of an expression e by induction on its syntax as follows:

$$\begin{array}{lcl}
\mathsf{E}[\![\, c \,]\!]\,(S) & \overset{\text{def}}{=} & \lambda\rho \in \mathbb{E}.c \qquad (1) \\
\mathsf{E}[\![\, x \,]\!]\,(S) & \overset{\text{def}}{=} & \lambda\rho \in \mathbb{E}.\rho(x) \\
\mathsf{E}[\![\, e_1 + e_2 \,]\!]\,(S) & \overset{\text{def}}{=} & \lambda\rho \in \mathbb{E}. \begin{cases} \perp & \text{if } \exists i : \mathsf{E}[\![\, e_i \,]\!]\,(S)(\rho) = \perp \\ v_1 + v_2 & \text{if } \forall i : v_i = \mathsf{E}[\![\, e_i \,]\!]\,(S)(\rho) \neq \perp \end{cases} \\
\mathsf{E}[\![\, \textbf{if } e_1 = 0 \textbf{ then } e_2 \textbf{ else } e_3 \textbf{ fi} \,]\!]\,(S) & \overset{\text{def}}{=} & \lambda\rho \in \mathbb{E}. \begin{cases} \perp & \text{if } \mathsf{E}[\![\, e_1 \,]\!]\,(S)(\rho) = \perp \\ \mathsf{E}[\![\, e_2 \,]\!]\,(S)(\rho) & \text{if } \mathsf{E}[\![\, e_1 \,]\!]\,(S)(\rho) = 0 \\ \mathsf{E}[\![\, e_3 \,]\!]\,(S)(\rho) & \text{otherwise} \end{cases} \\
\mathsf{E}[\![\, f(e_1, \ldots, e_n) \,]\!]\,(S) & \overset{\text{def}}{=} & \lambda\rho \in \mathbb{E}.S(f)(\lambda V_i \in \mathbb{V}.\mathsf{E}[\![\, e_i \,]\!]\,(S)(\rho))
\end{array}$$

Note in particular that a function call $f(e_1, \dots, e_n)$ first evaluates recursively the arguments $e_1, \dots, e_n$, then constructs an environment in $\mathbb{E}$ mapping formal arguments $V_i \in \mathbb{V}$ to these values, and finally applies the function $S(f) \in \mathbb{D}$ from the function environment S to these values.

To compute the semantics of the functions while taking their mutually recursive nature into account, the semantics of a program is defined as the fixpoint $\mathcal{F} \in \mathbb{S}$ of a function $F : \mathbb{S} \to \mathbb{S}$:

$$\mathcal{F} \overset{\text{def}}{=} \text{lfp}_{\sqsubseteq_{\mathbb{S}}} F \text{ where } F(S) \overset{\text{def}}{=} \lambda f \in \mathbb{F}.\mathsf{E}[\![\, \mathit{body}(f)\,]\!](S) \tag{2}$$

The posets $(\mathbb{E}, \sqsubseteq_{\mathbb{E}})$, $(\mathbb{D}, \sqsubseteq_{\mathbb{D}})$, and $(\mathbb{S}, \sqsubseteq_{\mathbb{S}})$ are obtained by considering the flat poset $(\mathbb{Z}_\perp, \sqsubseteq_{\mathbb{Z}})$ where:

$$a \sqsubseteq_{\mathbb{Z}} b \overset{\text{def}}{=} a = \perp \text{ or } a = b \tag{3}$$

and extending it pointwise.

Question 1. Prove that the fixpoint in definition (2) indeed exists.

Question 2. Give the semantics, as computed by (2), of the program composed of a single variable x and a single recursive function f where:

$$\mathit{body}(f) \overset{\text{def}}{=} \textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } f(x + (-1)) + 1 \textbf{ fi} \tag{4}$$

Show the iterate of the fixpoint computation.

A function f with a single argument is said to be strict in its argument if, whenever the expression e evaluates to $\perp$, the expression $f(e)$ evaluates to $\perp$. In an eager language (such as C, OCaml), functions employ "call by value" and are always strict (the argument is evaluated just before the function is called). A lazy language (such as Haskell), however, employs "call by name" : a function call $f(e)$ may return a definite value even if the evaluation of the argument e does not terminate, in case f does not actually use its argument.

Question 3. Give the semantics of the following program:

$$\begin{cases} \mathit{body}(f) \overset{\text{def}}{=} g(h(x)) \\ \mathit{body}(g) \overset{\text{def}}{=} 2 \\ \mathit{body}(h) \overset{\text{def}}{=} h(x) \end{cases} \tag{5}$$

and justify that our language has non-strict functions.

Question 4. Show how to modify Definition (1) to obtain an eager language.

We now come back to the original, lazy semantics of (1) and develop a static analysis to determine whether a function is actually strict in its arguments.

Question 5. Let $\mathbb{Z}^\sharp_\perp$ be the abstraction $\mathbb{Z}^\sharp_\perp \overset{\text{def}}{=} \{\perp^\sharp, \top^\sharp\}$ of $\mathcal{P}(\mathbb{Z}_\perp)$, where $\perp^\sharp$ means "always undefined" (only $\perp$) and $\top^\sharp$ means "possibly defined" (any value in $\mathbb{Z}_\perp$). Propose an abstract order $\sqsubseteq^\sharp_{\mathbb{Z}}$ on $\mathbb{Z}^\sharp_\perp$ and show the existence of a Galois connection between the concrete and the abstract domains:

$$(\mathcal{P}(\mathbb{Z}_\perp), \subseteq) \underset{\alpha_{\mathbb{Z}}}{\overset{\gamma_{\mathbb{Z}}}{\leftrightarrows}} (\mathbb{Z}^\sharp_\perp, \sqsubseteq^\sharp_{\mathbb{Z}}) \tag{6}$$

(give $\alpha_{\mathbb{Z}}$, $\gamma_{\mathbb{Z}}$ and prove that they form a Galois connection).

Question 6. Abstract values $\mathbb{Z}^\sharp_\perp$ are lifted to abstract environments $\mathbb{E}^\sharp \overset{\text{def}}{=} \mathbb{V} \to \mathbb{Z}^\sharp_\perp$. We denote as $\sqsubseteq^\sharp_\mathbb{E}$ the point-wise order. Show the existence of a Galois connection between the concrete and the abstract environments:

$$(\mathcal{P}(\mathbb{E}), \subseteq) \underset{\alpha_\mathbb{E}}{\overset{\gamma_\mathbb{E}}{\leftrightarrows}} (\mathbb{E}^\sharp, \sqsubseteq^\sharp_\mathbb{E}) \tag{7}$$

(*Hint:* you may start by abstracting $\mathcal{P}(\mathbb{E}) = \mathcal{P}(\mathbb{V} \to \mathbb{Z}_\perp)$ into $\mathbb{V} \to \mathcal{P}(\mathbb{Z}_\perp)$.)

The domains of abstract values $\mathbb{Z}^\sharp_\perp$ and abstract environments $\mathbb{E}^\sharp$ are lifted to the domain of monotonic abstract functions $\mathbb{D}^\sharp \overset{\text{def}}{=} \mathbb{E}^\sharp \overset{m}{\to} \mathbb{Z}^\sharp_\perp$, with the point-wise order $\sqsubseteq^\sharp_\mathbb{D}$. Hence, $f^\sharp \in \mathbb{D}^\sharp$ such that $f^\sharp(\perp^\sharp) = \perp^\sharp$ denotes a function that is strict in its single argument, while $f^\sharp(\top^\sharp) = \perp^\sharp$ denotes additionally that the function never terminates, whatever its argument.

Question 7. Prove the following general result: if $A \underset{\alpha_A}{\overset{\gamma_A}{\leftrightarrows}} A^\sharp$ and $B \underset{\alpha_B}{\overset{\gamma_B}{\leftrightarrows}} B^\sharp$ are Galois connections, then we have a Galois connection between the monotonic function spaces $A \overset{m}{\to} B$ and $A^\sharp \overset{m}{\to} B^\sharp$ as follows:

$$\begin{array}{l} (A \overset{m}{\to} B) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A^\sharp \overset{m}{\to} B^\sharp) \\ \text{where:} \\ \alpha(f) \overset{\text{def}}{=} \alpha_B \circ f \circ \gamma_A \\ \gamma(f^\sharp) \overset{\text{def}}{=} \gamma_B \circ f^\sharp \circ \alpha_A \end{array} \tag{8}$$

Use this result to prove the existence of a Galois connection:

$$(\mathcal{P}(\mathbb{D}), \subseteq) \underset{\alpha_\mathbb{D}}{\overset{\gamma_\mathbb{D}}{\leftrightarrows}} (\mathbb{D}^\sharp, \sqsubseteq^\sharp_\mathbb{D}) \tag{9}$$

Question 8. Finally, we lift $\mathbb{D}^\sharp$ to $\mathbb{S}^\sharp \overset{\text{def}}{=} \mathbb{F} \to \mathbb{D}^\sharp$ point-wise. Propose an abstract version $\mathsf{E}^\sharp[\![e]\!] : \mathbb{S}^\sharp \to \mathbb{D}^\sharp$ of $\mathsf{E}[\![e]\!]$ for each syntactic construction in (1). Discuss the exactness and the optimality of each abstract operator.

Question 9. Propose an abstract version of the fixpoint (2) in $\mathbb{S}^\sharp$ and show that this provides a sound and effective static analysis. Discuss the complexity in memory and time. Propose an example program where the abstract analysis is able to prove that a strict function is strict. Propose also an example program where the abstract analysis is not able to prove that a strict function is strict.

Exercise 2: Arrays and Materialization

Please answer Exercise 1 and Exercise 2 on different sheets of paper.

The purpose of this exercise is to study the notion of *materialization* on array segment predicates, how it operates, and when it improves precision. For our study, we use a very basic array language.

A program manipulates a finite set of integer variables $\mathbb{X}$ and a single array `a` of length `s`. Thus, an l-value is either a variable or an array cell. An expression is either a constant, or an l-value.

Last, a program is either an assignment, or a sequence, or a condition (without an else branch).

$$\begin{array}{rcll}
n & \in & \mathbb{Z} & \text{(mathematical) integer values} \\
\mathtt{x}, \mathtt{y}, \ldots & \in & \mathbb{X} & \text{integer variables} \\
\mathtt{a} & & & \text{array variable} \\
\circledcirc & ::= & \leq \mid == \mid \ldots & \text{comparison operators} \\
\mathtt{l} & ::= & \mathtt{v} \mid \mathtt{a}[\mathtt{v}] & \text{lvalues} \\
\mathtt{e} & ::= & n \mid \mathtt{l} & \text{expressions} \\
\mathtt{C} & ::= & \mathtt{l} = \mathtt{e} \mid \mathtt{C}; \mathtt{C} \mid \mathbf{if}(\mathtt{e} \circledcirc n)\{\mathtt{C}\} & \text{commands}
\end{array}$$

A *memory state* is a function $\sigma : \mathbb{X} \uplus \{0, 1, \ldots, \mathtt{s} - 1\} \longrightarrow \mathbb{Z}$, that maps each variable and each array cell (denoted by its index) to its value. We let $\mathbb{M}$ denote the set of memory states.

Question 1 (Concrete semantics). We let the concrete semantics $[\![\mathtt{e}]\!]$ of an expression $\mathtt{e}$ be a function that maps a memory state to a value or an error. Recall the definition of this semantics by induction on the syntax of expressions. We let the concrete semantics $[\![\mathtt{C}]\!]$ of a command be a function that maps a set of input memory states to the corresponding set of output memory states (we ignore executions that lead to an error). Define this semantics by induction on the syntax of commands.

We now define a set of abstract predicates to be used in static analysis. A *symbolic bound* $\mathtt{b}$ is either an integer or a variable plus an integer constant (such as $\mathtt{x} + 1$ or $\mathtt{x} - 1$). A *constraint* $\mathtt{c}^\sharp$ is either an inequality between two bounds or a range constraint $\mathbf{range}(\mathtt{b}_0, \mathtt{b}_1, \mathtt{I})$, where $\mathtt{b}_0, \mathtt{b}_1$ are symbolic bounds and $\mathtt{I}$ is an integer interval (with possibly infinite bounds —i.e., intervals bounds are either infinities or integer constants), which means that all the array cells with an index between $\mathtt{b}_0$ and $\mathtt{b}_1$ contain values in the interval $\mathtt{I}$. An *abstract state* $\sigma^\sharp$ is either $\perp$ or a finite conjunction of constraints. We let $\mathbb{M}^\sharp$ denote the set of abstract states. Abstract states are defined by the grammar below:

$$\begin{array}{rcll}
\mathtt{b} & ::= & n \mid \mathtt{x} + n & (n \in \mathbb{Z}, \mathtt{x} \in \mathbb{X}) \\
\mathtt{c}^\sharp & ::= & \mathtt{b} \leq \mathtt{b} \mid \mathbf{range}(\mathtt{b}, \mathtt{b}, \mathtt{I}) & \\
\sigma^\sharp & ::= & \perp \mid \mathtt{c}^\sharp \wedge \ldots \wedge \mathtt{c}^\sharp &
\end{array}$$

We do not discuss the machine representation of abstract predicates, as the purpose of the exercise is to study the precision of abstract post-conditions, and not the actual analysis algorithms. Thus, we consider abstract states equivalent modulo commutativity and associativity of $\wedge$.

Question 2 (Concretization). Formalize the concretization function $\gamma : \mathbb{M}^\sharp \longrightarrow \mathcal{P}(\mathbb{M})$. Based on this, compare the **range** constraints with the array abstraction based on segment predicates discussed during the course. Discuss the similarity between the **range** constraints and the cardinal power construction presented during the course, and comment on the need for an internal reduction operation on abstract states (in this exercise, we will not ask to provide a reduction).

In the following, we search for an analysis function $[\![.]\!]^\sharp$ for commands, that should be sound, that is such that, for all commands $\mathtt{C}$, $[\![\mathtt{C}]\!] \circ \gamma \subseteq \gamma \circ [\![\mathtt{C}]\!]^\sharp$. In general, the definition of such an analysis function is not unique.

Question 3 (Computation of abstract post-conditions for array assignments). We consider the assignment $\mathtt{a}[\mathtt{x}] = n$ (where $0 \leq \mathtt{x} \leq \mathtt{s}{-}1$) and the abstract states $\sigma_0^\sharp = (\mathbf{range}(\mathtt{y}, \mathtt{z}, [n', n'']))$ and $\sigma_1^\sharp = (\mathtt{y} \leq \mathtt{x} \wedge \mathtt{x} \leq \mathtt{z} \wedge \mathbf{range}(\mathtt{y}, \mathtt{z}, [n', n'']))$. Propose definitions of $[\![\mathtt{a}[\mathtt{x}] = n]\!]^\sharp(\sigma_0^\sharp)$ and of

$[\![\mathtt{a}[\mathtt{x}] = n]\!]^\sharp(\sigma_1^\sharp)$, using *at most one* **range** constraint. Observe that there are at least two incomparable choices in the case of $\sigma_0^\sharp$, and four in the case of $\sigma_1^\sharp$.

To improve the precision of the analysis of assignments $\mathtt{a}[\mathtt{x}] = n$, we introduce the materialization function **mat** defined by:

$$\begin{array}{l}\mathbf{mat}((\mathtt{y} \leq \mathtt{x} \wedge \mathtt{x} \leq \mathtt{z} \wedge \mathbf{range}(\mathtt{y}, \mathtt{z}, \mathtt{I})), \mathtt{x}) \\ \quad ::= (\mathtt{y} \leq \mathtt{x} \wedge \mathtt{x} \leq \mathtt{z} \wedge \mathbf{range}(\mathtt{y}, \mathtt{x} - 1, \mathtt{I}) \wedge \mathbf{range}(\mathtt{x}, \mathtt{x}, \mathtt{I}) \wedge \mathbf{range}(\mathtt{x} + 1, \mathtt{z}, \mathtt{I}))\end{array}$$

Question 4 (Soundness of materialization). Prove that the materialization is sound, that is $\gamma(\mathtt{y} \leq \mathtt{x} \wedge \mathtt{x} \leq \mathtt{z} \wedge \mathbf{range}(\mathtt{y}, \mathtt{z}, \mathtt{I})) \subseteq \gamma(\mathbf{mat}((\mathtt{y} \leq \mathtt{x} \wedge \mathtt{x} \leq \mathtt{z} \wedge \mathbf{range}(\mathtt{y}, \mathtt{z}, \mathtt{I})), \mathtt{x}))$. Is the materialization exact? Extend this materialization function to other abstract states than the conjunction of two inequalities and one range constraint. We suggest to decompose all **range** constraints that can be materialized.

We now consider the analysis of assignments (Question 3).

Question 5 (Materialization and analysis of array assignment). Propose an improvement of the analysis function for $\mathtt{a}[\mathtt{x}] = n$, and then prove that it is sound, and at least as precise as the previous analysis function. Hint: first, consider the abstract state $\mathtt{y} \leq \mathtt{x} \wedge \mathtt{x} \leq \mathtt{z} \wedge \mathbf{range}(\mathtt{y}, \mathtt{z}, \mathtt{I})$ used above for the definition of materialization, and then generalize. Is this new analysis function exact?

Question 6 (Abstract join). Propose an algorithm to compute a **join** operator that over-approximates the union of sets of states: it is such that, for all abstract states $\sigma_0^\sharp, \sigma_1^\sharp$, we have $\gamma(\sigma_0^\sharp) \cup \gamma(\sigma_1^\sharp) \subseteq \gamma(\mathbf{join}(\sigma_0^\sharp, \sigma_1^\sharp))$. Discuss its strengths and its limits. Such an operator is not unique, which is why the discussion on limitations is almost as important as the algorithm.

Question 7 (Computation of other abstract post-conditions). Propose a definition of $[\![\mathtt{C}]\!]^\sharp$ for the condition statements and the sequences, and prove their soundness. In the case of conditions, propose an analysis function that does not resort to materialization for tests and an analysis function that does. Discuss the difference.