Written exam
MPRI 2-6, year 2020–2021

30 November 2020

- The exam is individual.
- You are expected to send your copy by email before 30 November 2020 midnight to `antoine.mine@lip6.fr` and `xavier.rival@ens.fr`
- You can either send your answers in a text or PDF file, or as high resolution pictures with legible hand-writing.
- The questions are written in English. You can answer either in English or in French.
- The exercises are independent and can be solved in any order.
- It will not be answered any question during the exam. In case of ambiguity or 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: Cardinal power abstractions

In this exercise, we study the computation of sound abstract post-conditions in cardinal power abstract domains.

Notations. In the following, we use the following notations.

We assume a set of states $\mathbb{S}$, the precise definition of which is left as a parameter for now (several instances will be considered in the following questions). The concrete domain is the powerset of the set of states $(\mathcal{P}(\mathbb{S}), \subseteq)$. Moreover, we will also assume a transition relation over states $(\rightsquigarrow) \subseteq \mathbb{S} \times \mathbb{S}$, which is also left as a parameter for now, and which generally captures computation (e.g., the evaluation of an arbitrary function or one step of execution for a program).

An *abstract domain* is defined by a set $\mathbb{D}$ of abstract elements equipped with an order $\sqsubseteq$ and a monotone concretization function $\gamma : \mathbb{D} \longrightarrow \mathcal{P}(\mathbb{S})$.

A *sound post-condition operator* for abstract domain $(\mathbb{D}, \sqsubseteq, \gamma)$ is a function $\mathbf{post} : \mathbb{D} \longrightarrow \mathbb{D}$ such that:

$$\forall d \in \mathbb{D},\ \forall s \in \gamma(d),\ \forall s' \in \mathbb{S},\ s \rightsquigarrow s' \Longrightarrow s' \in \gamma(\mathbf{post}(d))$$

Cardinal power abstract domain. We let $(\mathbb{D}_0, \sqsubseteq_0, \gamma_0)$ and $(\mathbb{D}_1, \sqsubseteq_1, \gamma_1)$ be two abstract domains. We recall that their *cardinal power* is the abstract domain $(\mathbb{D}_2, \sqsubseteq_2, \gamma_2)$ defined by:

- $\mathbb{D}_2$ is the set of monotone functions $\mathbb{D}_0 \longrightarrow_{\mathcal{M}} \mathbb{D}_1$ from $\mathbb{D}_0$ to $\mathbb{D}_1$;
- $\sqsubseteq_2$ is the pointwise ordering;
- $s \in \gamma_2(d_2)$ if and only if $\forall d_0 \in \mathbb{D}_0,\ s \in \gamma_0(d_0) \Longrightarrow s \in \gamma_1(d_2(d_0))$.

Before we move on to post-conditions, we start with a basic numerical example.

Question 1 (Example cardinal power).

In this question, we make the following assumptions:

- $\mathbb{S} = \mathbb{R}^2$ *where* $\mathbb{R}$ *is the set of real numbers; thus a state may be viewed as a point* (x, y) *in the two-dimensional plane;*
- $(\mathbb{D}_0, \sqsubseteq_0, \gamma_0)$ *defines the interval abstraction applied to the first component of pairs* (x, y)*;*
- $(\mathbb{D}_1, \sqsubseteq_1, \gamma_1)$ *defines the abstract domain of finite conjunctions of linear inequalities.*

Moreover, we consider the set of states S *defined by:*

$$S = \{(x, y) \mid (x \leq 1 \wedge y = -\frac{1}{2}(1 + x)) \vee (x \geq 1 \wedge y = x - 2)\}$$

1. *Define formally the abstractions* $(\mathbb{D}_0, \sqsubseteq_0, \gamma_0)$ *and* $(\mathbb{D}_1, \sqsubseteq_1, \gamma_1)$*.*
2. *Give as precise a characterization of* S *as possible in the cardinal power abstract domain.*
3. *Does this instance of the cardinal power have a best abstraction function? Give a sufficient condition on* $\mathbb{D}_1$ *for the cardinal power abstract domain to have a best abstraction.*

Towards the computation of abstract post-conditions. In the following, we search for ways to define a sound post-condition operator for cardinal power abstract domain $\mathbb{D}_2$ using elementary operators on $\mathbb{D}_0, \mathbb{D}_1$.

Question 2 (A naive definition).

As a first attempt, we propose to only assume a sound post-condition for $\mathbb{D}_1$ and to let:

$$\mathbf{post}_2(d_2) = \lambda(d_0 \in \mathbb{D}_0) \cdot \mathbf{post}_1(d_2(d_0))$$

Does this operator define a sound post-condition? If so, prove it correct; otherwise, explain why using a counter example. If it is sound, explain what short-coming it may have.

Projected post-condition operator. We now add some new assumptions, so as to define a new post-condition operation.

Given two abstract domains $\mathbb{D}_0, \mathbb{D}_1$ and two elements d_0, d_0' of $\mathbb{D}_0$, we call a *projected post-condition operator* a function $\mathbf{post}_{\mathbb{D}_1[d_0 \to d_0']}$ such that:

$$\forall d_1 \in \mathbb{D}_1,\ \forall s \in \gamma_1(d_1) \cap \gamma_0(d_0),\ \forall s' \in \gamma_0(d_0'),\ s \rightsquigarrow s' \Longrightarrow s' \in \gamma_1(\mathbf{post}_{\mathbb{D}_1[d_0 \to d_0']}(d_1))$$

In the following, we make the three following assumptions:

1. we assume that $\mathbb{D}_1$ defines projected post-condition operators for any pair of elements of $\mathbb{D}_0$;
2. we assume that $\mathbb{D}_0$ is *covering the concrete domain* in the sense that

$$\forall s \in \mathbb{S},\ \exists d_0 \in \mathbb{D}_0,\ s \in \gamma_0(d_0)$$

3. we assume that $\mathbb{D}_1$ defines a sound over-approximation of unions that we assume to be commutative, associative and defined over any family of abstract elements, so that we note it $\sqcup_1$; moreover, we assume that it satisfies the soundness condition below:

$$\forall E \subseteq \mathbb{D}_1;\ \bigcup_{d_1 \in E} \gamma_1(d_1) \subseteq \gamma_1(\sqcup_1 E)$$

Question 3 (A second post-condition operator proposal).

Under the above assumption, we propose:

$$\mathbf{post}_2(d_2) = \lambda(d_0 \in \mathbb{D}_0) \cdot \sqcup_1 \{\mathbf{post}_{\mathbb{D}_1[d_0' \to d_0]}(d_2(d_0')) \mid d_0' \in \mathbb{D}_0\}$$

Prove that this operator defines a sound post-condition.

Disjunctive pre-conditions. We now add two more assumptions:

- We require $\mathbb{D}_0$ to provide a function $\mathbf{pre}_0 : \mathbb{D}_0 \longrightarrow \mathcal{P}(\mathcal{P}(\mathbb{D}_0))$ that returns all "disjunctive pre-conditions" of any element, in the following sense:

$$\forall d_0 \in \mathbb{D}_0,\ \mathbf{pre}_0(d_0) = \{D_0 \subseteq \mathbb{D}_0 \mid \forall s \in \gamma(d_0),\ \forall s' \in \mathbb{S},\ s' \rightsquigarrow s \Longrightarrow \exists d_0' \in D_0,\ s' \in \gamma_0(d_0')\}$$

- Second, we require $\mathbb{D}_1$ to provide an operator that over-approximates intersection, and that is associative and commutative, so that we note it $\sqcap_1$; it should satisfy the soundness condition below:

$$\forall E \subseteq \mathbb{D}_1;\ \bigcap_{d_1 \in E} \gamma_1(d_1) \subseteq \gamma_1(\sqcap_1 E)$$

Under these assumptions, in addition to the assumptions made in the previous paragraph, we design a new operator:

Question 4 (A third post-condition operator proposal).

We propose a new abstract post-condition operator for the cardinal power

$$\overline{\mathbf{post}}_2(d_2) = \lambda(d_0 \in \mathbb{D}_0) \cdot \sqcap_1 \{\sqcup_1 \{\mathbf{post}_{\mathbb{D}_1[d_0' \to d_0]}(d_2(d_0')) \mid d_0' \in D_0\} \mid D_0 \in \mathbf{pre}_0(d_0)\}$$

- *Prove that this operator is sound.*
- *Give a sufficient condition under which $\mathbb{D}_2$ is strictly more precise than the operator $\mathbf{post}_2$ of the previous question and comment whether this assumption is commonly satisfied.*
- *Compare them from a computational cost point of view. Suggest how to mitigate this difference.*

Question 5 (Comparison).

We compare the operators $\mathbf{post}_2$ and $\overline{\mathbf{post}}_2$ based on the example abstract domain and operation shown in question 1.
Can both operators be defined? (are all the required conditions satisfied?).
Explain whether one form may be more precise than the other.

Flow sensitivity and variant as a cardinal power. During the lecture, we have seen that flow sensitivity can be described as a cardinal power. We study the post-condition operator in this case.

We let $\mathbb{L}$ denote a set of control states and $\mathbb{M}$ a set of memory states. We assume that $\mathbb{S} = (\mathbb{L} \times \mathbb{M})^*$ is the set of non empty finite sequences of elements of $\mathbb{L} \times \mathbb{M}$, which denote program executions. Moreover, we let $\rightsquigarrow$ be defined by an execution step relation $(\rightarrow) \subseteq (\mathbb{L} \times \mathbb{M}) \times (\mathbb{L} \times \mathbb{M})$ as:

$$\langle (l_0, m_0), \ldots, (l_n, m_n) \rangle \rightsquigarrow \langle (l_0, m_0), \ldots, (l_n, m_n), (l_{n+1}, m_{n+1}) \rangle \iff (l_n, m_n) \rightarrow (l_{n+1}, m_{n+1})$$

Question 6 (Basic flow sensitive abstraction).

The flow insensitive abstraction maps each control state into the set of memory states that may be observed at that point.

1. *Define formally the abstractions $(\mathbb{D}_0, \sqsubseteq_0, \gamma_0)$ and $(\mathbb{D}_1, \sqsubseteq_1, \gamma_1)$ that allow to define the flow sensitive abstraction as a cardinal power, following the above intution.*
2. *Following the approach from question 3, propose an operator to compute abstract post-conditions in the resulting cardinal power abstract domain and comment on this formula.*
3. *Can we improve this post-condition operator using some inspiration based on question 4?*

We now discuss a more elaborate abstraction that includes some information typically tracked by trace partitioning.

Question 7 (A variant of flow sensitive abstraction that records some history information).

We propose to study another abstraction which maps a pair of control states (l, l') to the memory states observed at l, and at the end of executions that visited l' at some point.

1. *Define formally the abstractions $(\mathbb{D}_0, \sqsubseteq_0, \gamma_0)$ and $(\mathbb{D}_1, \sqsubseteq_1, \gamma_1)$ that allow to define this abstraction as a cardinal power.*
2. *Propose a post-condition operator based on question 3 and question 4, with both precision and cost in mind.*

Exercise 2: Concurrent programs with weak memories

This exercise studies the abstract interpretation of multi-threaded concurrent programs executing on a multi-core processor with a shared memory. A natural execution model, called sequential consistency, considers that the execution of a program is an interleaving of the execution of its threads, and that a thread reads back from a shared memory location the value stored by the previous write to this location by the last thread that wrote to this location. However, in modern processors, due to optimizations, accesses to the shared memory are more complex: a thread may cache a memory write into a buffer for some time before it is dispatched to the memory and becomes visible to other threads; hence, different threads can have different views of the same shared memory location. Under these weaker memory models, the program may exhibit more behaviors, and analyses that are only sound with respect to sequential consistency may no longer be sound.

In this exercise, we will start by constructing a transition relation of sequentially consistent executions and a concrete reachability semantics in equational form. Secondly, we will move on to a weaker memory model, total store ordering, and design a sound operational semantics for this model. Thirdly, we will consider another model, partial store ordering, and express it as an abstraction of total store ordering. Finally, we will consider further abstractions towards effective static analyses.

Concurrent programs. We consider a concurrent program composed of two threads, $\mathcal{T}_1$ and $\mathcal{T}_2$.[1] Each thread $\mathcal{T}_i \stackrel{\text{def}}{=} (\mathcal{N}_i, e_i, \mathcal{E}_i, \mathit{stmt}_i, \mathcal{R}_i)$, for $i \in \{1,2\}$, has its own control-flow graph: a set of nodes $\mathcal{N}_i$, with an entry node $e_i \in \mathcal{N}_i$, and edges $\mathcal{E}_i \subseteq \mathcal{N}_i \times \mathit{stmt}_i \times \mathcal{N}_i$ that are labelled with statements in the language stmt_i. Thread i has a fixed, finite set $\mathcal{R}_i$ of numeric variables local to the thread that we call registers. In addition to registers, the program has a shared memory composed of a fixed, finite set $\mathcal{S}$ of numeric variables. The threads are disjoint: $\mathcal{N}_1 \cap \mathcal{N}_2 = \emptyset$, $\mathcal{R}_1 \cap \mathcal{R}_2 = \emptyset$, $\forall i : \mathcal{R}_i \cap \mathcal{S} = \emptyset$. Statements stmt_i and expressions exp_i for thread $i \in \{1,2\}$ include:

$$
\begin{array}{rcll}
\mathit{stmt}_i & ::= & R \leftarrow \mathit{exp}_i & \textit{(assignment into register } R \in \mathcal{R}_i\textit{)} \\
& | & \mathit{exp}_i \bowtie \mathit{exp}_i? & \textit{(test, } \bowtie \in \{=, \neq, <, >, \leq, \geq\}\textit{)} \\
& | & \mathbf{load}(S \rightarrow R) & \textit{(load from shared memory } S \in \mathcal{S} \textit{ into register } R \in \mathcal{R}_i\textit{)} \\
& | & \mathbf{store}(\mathit{exp}_i \rightarrow S) & \textit{(write to shared memory } S \in \mathcal{S}\textit{)} \\
\\
\mathit{exp}_i & ::= & R & \textit{(register, } R \in \mathcal{R}_i\textit{)} \\
& | & c & \textit{(constant, } c \in \mathbb{Z}\textit{)} \\
& | & \mathit{exp}_i \circ \mathit{exp}_i & \textit{(numeric operation } \circ \in \{+, -, \times\}\textit{)}
\end{array}
$$

As in the course, assignments $R \leftarrow \mathit{exp}_i$ update register values while tests $\mathit{exp}_i \bowtie \mathit{exp}'_i?$ stop execution until the condition is satisfied. Note that expressions exp_i can only contain registers from thread i, and not shared variables. Communication with the shared memory $S \in \mathcal{S}$ is done only through explicit statements: $\mathbf{load}(S \rightarrow R)$ to load into a register and $\mathbf{store}(\mathit{exp}_i \rightarrow S)$ to store the value of an expression.

Sequentially consistent execution model. A program state is a pair $(\ell, m) \in \Sigma \stackrel{\text{def}}{=} \mathcal{C} \times \mathcal{M}$ composed of a control state $\ell \stackrel{\text{def}}{=} (\ell_1, \ell_2) \in \mathcal{C} \stackrel{\text{def}}{=} \mathcal{N}_1 \times \mathcal{N}_2$ and a memory state $m \in \mathcal{M} \stackrel{\text{def}}{=} \mathcal{V} \rightarrow \mathbb{Z}$ where $\mathcal{V} \stackrel{\text{def}}{=} \mathcal{S} \cup \mathcal{R}_1 \cup \mathcal{R}_2$ is the set of all the registers and the shared variables. The execution model is that, in a given a state $((\ell_1, \ell_2), m) \in \Sigma$, a thread $i \in \{1,2\}$ is picked non-deterministcally to execute a statement s along an edge $(\ell_i, s, \ell'_i) \in \mathcal{E}_i$, which updates the memory m and the control state ℓ_i, leaving unchanged the control state of the other thread $\ell_{j \neq i}$. Due to non-determinism, it is possible for a thread to execute an arbitrary number of statements before the other thread executes. The program starts at the entry control point (e_1, e_2) with all the registers and shared variables initialized to 0. In this execution model, the shared memory behaves like regular variables. Hence $\mathbf{load}(S \rightarrow R)$ and $\mathbf{store}(e \rightarrow S)$ are really regular assignments, respectively $R \leftarrow S$ and $S \leftarrow e$.

Question 1. Transition system.
Show how to derive a transition system $\tau \subseteq \Sigma \times \Sigma$ generated from a concurrent program $(\mathcal{T}_1, \mathcal{T}_2)$. Each transition consists in executing one statement from a thread chosen arbitrarily.

Question 2. Concrete equational semantics.

i) Give the concrete reachability semantics $\mathsf{S}[\![\, s \,]\!] : \mathcal{P}(\mathcal{M}) \rightarrow \mathcal{P}(\mathcal{M})$ of statement $s \in \mathit{stmt}_i$ as a function that, given a set $X \subseteq \mathcal{M}$ of memory states before the statement, provides the set of states $\mathsf{S}[\![\, s \,]\!]\, X$ reachable after the statement.

ii) Deduce an equation system over variables X_ℓ with value in $\mathcal{P}(\mathcal{M})$ for $\ell \in \mathcal{C}$ that computes, in X_ℓ, the set of memory states reachable at control point ℓ.

Example 1. We consider as example a program with shared variables $\mathcal{S} \stackrel{\text{def}}{=} \{S, T\}$, registers $\mathcal{R}_1 \stackrel{\text{def}}{=} \{X\}$ and $\mathcal{R}_2 \stackrel{\text{def}}{=} \{Y\}$, and control-flow graph as follows:

- $\mathcal{T}_1$: $\mathcal{N}_1 \stackrel{\text{def}}{=} \{a, b, c\}$, $e_1 \stackrel{\text{def}}{=} a$, $\mathcal{E}_1 \stackrel{\text{def}}{=} \{\, (a, \mathbf{store}(1 \rightarrow S), b), (b, \mathbf{load}(T \rightarrow X), c), (c, X = 0?, a) \,\}$
- $\mathcal{T}_2$: $\mathcal{N}_2 \stackrel{\text{def}}{=} \{a', b', c'\}$, $e_2 \stackrel{\text{def}}{=} a'$, $\mathcal{E}_2 \stackrel{\text{def}}{=} \{\, (a', \mathbf{store}(1 \rightarrow T), b'), (b', \mathbf{load}(S \rightarrow Y), c'), (c', Y = 0?, a') \,\}$

Question 3.
Give the equation system corresponding to Example 1 and solve it.
Deduce that, at the control point (c, c'), X and Y cannot be both 0.

[1] We consider only two threads to simplify the notations, but the method extends to a finite number of threads.

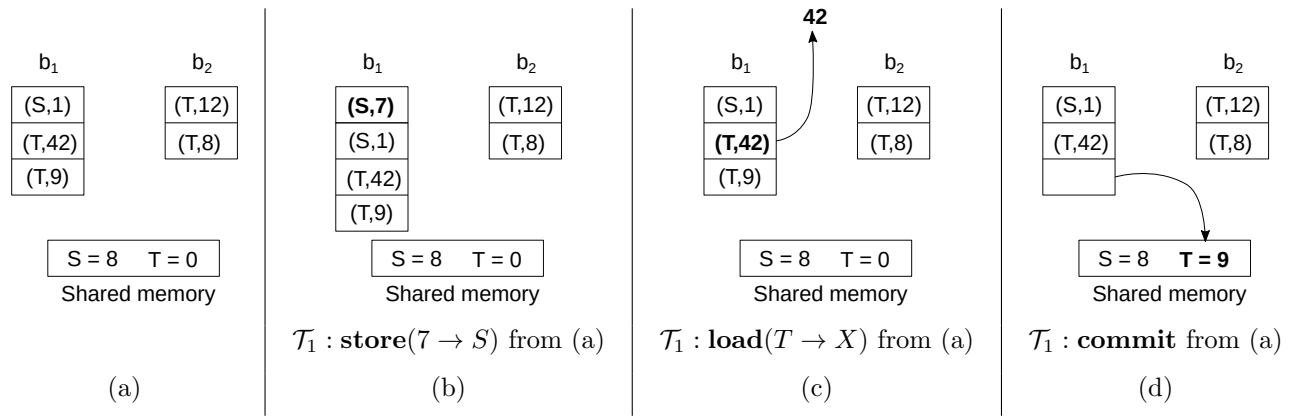

Figure 1: Total store ordering model.

Total store ordering (TSO). We now consider a more realistic memory model. Each thread $\mathcal{T}_i$ is equipped with a *write buffer* $b_i \in \mathcal{B} \stackrel{\text{def}}{=} (\mathcal{S} \times \mathbb{Z})^*$ which is a (possibly empty) finite sequence of pairs composed of a shared variable and its value, as show in Fig. 1.(a). It behaves as a FIFO (first-in first-out) queue.

1. A $\mathbf{store}(e \to S)$ executed by thread $\mathcal{T}_i$ does not access the shared memory m; instead, it adds a pair (S, v), where v is the value of e, to its buffer b_i (Fig. 1.(b)).
2. For a $\mathbf{load}(S \to R)$, $\mathcal{T}_i$ first looks for the most recent pair (S, v) matching the loaded variable S in its buffer b_i and stores v into R (Fig. 1.(c)); in case there is no pair of the form (S, v) in b_i, the thread reads the value of the variable from the shared memory $m(S)$ and stores it into R (as in the classic semantics).
3. Independently from the execution of the threads, the oldest pair (S, v) of any buffer b_i, $i \in \{1, 2\}$ can be committed into the shared memory: the pair is removed from b_i and the memory m is replaced with $m[S \mapsto v]$ (Fig. 1.(d)). This process happens in parallel to the execution of the threads, and there might be an arbitrary (possibly null) number of commits from b_1 and from b_2 between the execution of two thread statements.

To model this, we enrich the memory state $\mathcal{M}$ with two buffers and define: $\mathcal{M}_{TSO} \stackrel{\text{def}}{=} \mathcal{M} \times \mathcal{B} \times \mathcal{B}$, while the control state $\mathcal{C}$ remains intact. The reachability semantics is now an operator $\mathsf{S}_{TSO}[\![s]\!] : \mathcal{P}(\mathcal{M}_{TSO}) \to \mathcal{P}(\mathcal{M}_{TSO})$.

Question 4.

i) Give the new semantic operators $\mathsf{S}_{TSO}[\![\mathbf{store}(e \to S)]\!]$ *and* $\mathsf{S}_{TSO}[\![\mathbf{load}(S \to R)]\!]$.

ii) Give the semantics $\mathsf{S}_{TSO}[\![\mathbf{commit}]\!]$ *corresponding to a single commit.*

iii) Show how to derive a new equational semantics over variables X_ℓ, $\ell \in \mathcal{C}$, *with value in* $\mathcal{P}(\mathcal{M}_{TSO})$, *that takes into account the semantics of commits.*

iv) Give the reachable memory states for Example 1. Is it still true that X *and* Y *cannot be both 0 at control point* (c, c')*?*

Memory barriers. To avoid the effect of write buffers, processors offer memory barrier operations. We add to our language a **fence** statement with the following semantics: the thread executing **fence** is blocked until its write buffer is fully committed (i.e., become empty).

Question 5.

i) Give the concrete semantics of memory barriers $\mathsf{S}_{TSO}[\![\mathbf{fence}]\!]$.

ii) Show that, in Example 1, if we insert a **fence** *statement just after both* $\mathbf{store}(1 \to S)$ *and* $\mathbf{store}(1 \to T)$ *statements, then we retrieve the property that* X *and* Y *cannot be both 0 at control point* (c, c').

Partial store ordering (PSO). The previous memory model can be further relaxed by forgetting in write buffers the order between pairs (S, v) and (S', v') for different variables $S \neq S'$, while the order between pairs (S, v) and (S, v') on the same variable is maintained. We propose a new semantics which replaces, for each thread, its buffer $b_i \in \mathcal{B} \stackrel{\text{def}}{=} (\mathcal{S} \times \mathbb{Z})^*$ with a per-variable buffer $\overline{b_i} \in \overline{\mathcal{B}} \stackrel{\text{def}}{=} \mathcal{S} \to \mathbb{Z}^*$, associating to each variable a list of values. $\mathcal{M}_{TSO}$ is replaced with $\mathcal{M}_{PSO} \stackrel{\text{def}}{=} \mathcal{M} \times \overline{\mathcal{B}} \times \overline{\mathcal{B}}$.

Question 6. Galois connection.

i) Give a Galois insertion $\mathcal{P}(\mathcal{M}_{TSO}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{P}(\mathcal{M}_{PSO})$ between the two semantic domains (prove that it is a Galois insertion).

ii) Justify the claim that the abstraction indeed corresponds to forgetting the order of pairs over different variables.

Question 7. Abstract semantics.

i) Show how to derive the PSO semantics $\mathsf{S}_{PSO}[\![\, \mathbf{store}(e \to S) \,]\!]$, $\mathsf{S}_{PSO}[\![\, \mathbf{load}(S \to R) \,]\!]$, $\mathsf{S}_{PSO}[\![\, \mathbf{commit} \,]\!]$, and $\mathsf{S}_{PSO}[\![\, \mathbf{fence} \,]\!]$ from the TSO semantics by abstraction.

ii) Give an example of statement with a non-exact abstraction in PSO.

iii) Does this abstraction change the result of the semantics of Example 1 without the fences? of Example 1 with fences added (Question 5)?

Uniform buffer abstraction. We wish to analyze our programs in PSO semantics using a non-relational domain such as the interval domain. However, numeric domains only abstract a finite number of variables, while elements in $(m, \overline{b_1}, \overline{b_2}) \in \mathcal{M}_{PSO}$ have an unbounded number of integer values as each buffer $\overline{b_i}(V)$ can have an arbitrary size. To solve this problem, we will maintain only one piece of information per buffer. We consider a uniform, non-relational abstract domain $\mathcal{D}_U \stackrel{\text{def}}{=} \mathcal{V}_U \to \mathcal{P}(\mathbb{Z})$ with $\mathcal{V}_U \stackrel{\text{def}}{=} \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{\, S, S_1, S_2 \mid S \in \mathcal{S} \,\}$: $\rho \in \mathcal{D}_U$ maintains a set of values $\rho(R)$ for each register $R \in \mathcal{R}_1 \cup \mathcal{R}_2$ and $\rho(S)$ for each shared variable $S \in \mathcal{S}$; while $\rho(S_1)$ and $\rho(S_2)$ denote the set of possible values in the buffer associated to variable S in, respectively, thread 1 and thread 2.

Question 8. Galois connection.

Give a Galois connection $\mathcal{P}(\mathcal{M}_{PSO}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}_U$ between $\mathcal{P}(\mathcal{M}_{PSO})$ and $\mathcal{D}_U$ (prove that it is a Galois connection).
Discuss carefully the abstraction of states containing empty buffers, and the definition of $\gamma(x)$ when $x(V) = \emptyset$ for some $V \in \mathcal{V}_U$.
Is it a Galois insertion? (justify your answer).

Question 9. Abstract semantics.

i) Give a sound abstract semantics in $\mathcal{D}_U$ for $\mathsf{S}_U[\![\, \mathbf{store}(e \to S) \,]\!]$, $\mathsf{S}_U[\![\, \mathbf{load}(S \to R) \,]\!]$, $\mathsf{S}_U[\![\, \mathbf{commit} \,]\!]$, $\mathsf{S}_U[\![\, \mathbf{fence} \,]\!]$.

ii) Give a sound abstraction in $\mathcal{D}_U$ of the union $\cup$ and the intersection $\cap$ and explain how the intersection differs from the one we saw in the course for non-relational domains (give an example).

iii) Is $\mathcal{D}_U$ sufficiently precise to handle Example 1 (with or without fences)?

It would now be a simple matter to replace, in $\mathcal{D}_U$, $\mathcal{P}(\mathbb{Z})$ with a non-relational domain basis, such as intervals, to obtain an effective analysis. We will not pursue this avenue further.

Improving the precision. Consider a thread that executes $\mathbf{store}(1 \to S); \mathbf{store}(2 \to S); \mathbf{load}(S \to X)$ while the other thread does not write into S. Then, under the PSO semantics, X will equal 2.

Question 10.

i) Show that the uniform semantics in $\mathcal{D}_U$ loses precision on this example.

*ii) Show how to recover the lost precision by abstracting separately, in each buffer, the element added by the last store from the elements added by previous stores. You will provide the new abstract domain, its Galois connection, and sound operators for $\mathbf{store}(e \to S)$, $\mathbf{load}(S \to R)$, **commit**, and **fence**.*