Formal Methods for Security and Privacy (SS 2020) Univ-Prof. Dr. Matteo Maffei

Repeat Exam

Date: 23/09/2020 TU Wien

- The exam takes 180 minutes. You can get at most 100 points.
- Your solutions can be handwritten and then scanned or photographed, or directly typed up.
  - Upload a single file (either a single pdf, or a zip archive containing multiple images (jpeg, png)) to the TUWEL assignment.
  - Make sure that the result is legible.
  - Your file may not be bigger than 256MB.
  - Your solution must be handed in **before 13:00 on the day of the exam**.
- **You must work on the exam alone.** You may use available resources (for example lecture slides), but **you must not communicate with anyone** except the lecturers. Everything you hand in must be written and created by you and you must be able to explain your solution. You will be required to confirm this when you upload your file. If plagiarism is detected, all of the parties involved (both commiting and aiding) will be held accountable.
- There will be an oral examination tomorrow (24/09/2020), where we will ask you to explain your solutions. Note however, that **your written solutions must be entirely self-contained**. The oral examination serves to verify that you created your solution on your own.
- In case of questions or uncertainties, you can call the phone number +43 (1) 58801-184864, where one of the lecturers will answer your questions.
  This is the only allowed way of communication during the exam.
- In case we should have an announcement that is relevant to all participants, we will post it in the TUWEL discussion board `https://tuwel.tuwien.ac.at/mod/forum/view.php?id=913285`. Please make sure that you receive these announcements.
- **Write legibly and be concise**. It is in your best interest that we understand your answers. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it.
- Please make sure your name and matriculation number are written on the document you hand in.
- Good luck!

Problem	1	2	3	4	**Total**
Points	30	20	20	30	100

$$
\begin{array}{rcl}
\ell_C, \ell_I & ::= & H \mid L \\
\ell & ::= & \ell_C \ell_I \\
T & ::= & \ell \mid \mathrm{C}^{\ell}[\tilde{T}] \mid \mu\mathrm{K}^{\ell}[\tilde{T}] \\
\mu & ::= & \mathrm{Enc} \mid \mathrm{Dec} \\
L & \sqsubseteq_C & H \\
H & \sqsubseteq_I & L \\
\ell_C^1 \ell_I^1 & \sqsubseteq & \ell_C^2 \ell_I^2 \iff \ell_C^1 \sqsubseteq_C \ell_C^2 \wedge \ell_I^1 \sqsubseteq_I \ell_I^2 \\
\ell_1 & \leq & \ell_2 \text{ whenever } \ell_1 \sqsubseteq \ell_2 \\
LL & \leq & \mathrm{C}^{LL}[LL, \cdots, LL] \\
\mathrm{C}^{\ell}[\tilde{T}] & \leq & \ell \\
LL & \leq & \mu\mathrm{K}^{LL}[LL, \cdots, LL] \\
\mu\mathrm{K}^{\ell}[\tilde{T}] & \leq & \ell
\end{array}
\qquad
\begin{array}{rcl}
\mathcal{L}(\ell) & = & \ell \\
\mathcal{L}(\mathrm{C}^{\ell}[\tilde{T}]) & = & \ell \\
\mathcal{L}(\mu\mathrm{K}^{\ell}[\tilde{T}]) & = & \ell \\
\mathcal{L}_{I,\Gamma}(\{|M|\}^a_{\mathrm{ek}(K)}) & = & \begin{cases} H & \text{if } \Gamma(K) = \mu\mathrm{K}^{HH}[T] \\ & \quad \wedge \mathcal{L}_{I,\Gamma}(M) \sqsubseteq_I \mathcal{L}_I(T) \\ L & \text{otherwise} \end{cases} \\
\mathcal{L}_C(\ell_C \ell_I) & = & \ell_C \\
\mathcal{L}_I(\ell_C \ell_I) & = & \ell_I \\
\mathcal{L}_{\Gamma}(M) & = & \begin{cases} \mathcal{L}(\Gamma(M)) & \text{if } M \in dom(\Gamma) \\ LL & \text{otherwise} \end{cases}
\end{array}
$$

$$
\textsc{Empty}\ \frac{}{\emptyset \vdash \diamond}
\qquad
\textsc{Env}\ \frac{\Gamma \vdash \diamond \qquad M \notin dom(\Gamma) \qquad T \in \{\mathrm{C}^{\ell}[\tilde{T}], \mu\mathrm{K}^{\ell}[\tilde{T}]\} \text{ implies } \ell = HH}{\Gamma, M : T \vdash \diamond}
$$

$$
\textsc{Atom}\ \frac{\Gamma \vdash \diamond \qquad M : T \text{ in } \Gamma}{\Gamma \vdash M : T}
\qquad
\textsc{List}\ \frac{\Gamma \vdash M_1 : T_1 \qquad \ldots \qquad \Gamma \vdash M_n : T_n}{\Gamma \vdash [M_1, \ldots, M_n] : [T_1, \ldots, T_n]}
$$

$$
\textsc{Subsumption}\ \frac{\Gamma \vdash M : T' \qquad T' \leq T}{\Gamma \vdash M : T}
\qquad
\textsc{AsymEnc}\ \frac{\Gamma \vdash K : \mathrm{EncK}^{\ell_C \ell_I}[\tilde{T}] \qquad \Gamma \vdash \tilde{M} : \tilde{T}}{\Gamma \vdash \{|\tilde{M}|\}^a_K : L\ell_I}
$$

$$
\textsc{EncKey}\ \frac{\Gamma \vdash K : \mathrm{DecK}^{\ell_C \ell_I}[\tilde{T}]}{\Gamma \vdash \mathrm{ek}(K) : \mathrm{EncK}^{L\ell_I}[\tilde{T}]}
$$

$$
\textsc{Stop}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0}}
\qquad
\textsc{Par}\ \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}
\qquad
\textsc{Repl}\ \frac{\Gamma \vdash P}{\Gamma \vdash !P}
\qquad
\textsc{Res}\ \frac{\Gamma, a : T \vdash P}{\Gamma \vdash (\nu a : T)\ P}
$$

$$
\textsc{Cond}\ \frac{\Gamma \vdash M : T \qquad \Gamma \vdash N : T' \qquad \Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash \text{if } M = N \text{ then } P \text{ else } Q}
\qquad
\textsc{In}\ \frac{\Gamma, \tilde{x} : \tilde{T} \vdash P \qquad \Gamma \vdash N : \mathrm{C}^{\ell}[\tilde{T}]}{\Gamma \vdash N(\tilde{x}).P}
$$

$$
\textsc{Out}\ \frac{\Gamma \vdash \tilde{M} : \tilde{T} \qquad \Gamma \vdash P \qquad \Gamma \vdash N : \mathrm{C}^{\ell}[\tilde{T}]}{\Gamma \vdash \overline{N}\langle \tilde{M} \rangle.P}
$$

$$
\textsc{AsymDec}\ \frac{\Gamma \vdash M : T \qquad \Gamma \vdash K : \mathrm{DecK}^{\ell}[\tilde{T}] \qquad \Gamma, \tilde{x} : \tilde{T} \vdash P \qquad \mathcal{L}_I(T) = L \Rightarrow \Gamma, \tilde{x} : LL \vdash P}{\Gamma \vdash \text{ case } M \text{ of } \{|\tilde{x}|\}^a_K \text{ in } P}
$$

Definition 1 (Opponent). *A process O is an opponent if any $(\nu a : T)$ ocurring in O are such that $T = LL$.*

Definition 2 (Secrecy). *P preserves secrecy if, for all opponents O, whenever $P \mid O \rightarrow^* (\nu c : T)\ (\nu \tilde{a} : \tilde{T})\ (P' \mid \overline{b}\langle c \rangle.P'')$ we have $\mathcal{L}_C(T) \sqsubseteq_C \mathcal{L}_{C,\Gamma}(b)$, with $\Gamma = c : T, \tilde{a} : \tilde{T}$.*

Definition 3 (Integrity). *P preserves integrity if, for all opponents O, whenever*

$$
\begin{array}{l}
P \mid O \rightarrow^* (\nu b : C^{HH}[T])\ (\nu \tilde{a} : \tilde{T})\ (P' \mid \overline{b}\langle c \rangle.P'') \text{ or} \\
P \mid O \rightarrow^* (\nu k : DecK^{HH}[T])\ (\nu \tilde{a} : \tilde{T})\ (P' \mid \textit{case } \{|c|\}^a_{ek(k)} \textit{ of } \{|x|\}^a_k \textit{ in } P'')
\end{array}
$$

we have $\mathcal{L}_{I,\Gamma}(c) \sqsubseteq \mathcal{L}_I(T)$, with $\Gamma = b : C^{HH}[T], k : DecK^{HH}[T], \tilde{a} : \tilde{T}$.

Theorem 1 (Typing implies Secrecy and Integrity). *Let $\Gamma \vdash P$ with $img(\Gamma) = LL$. Then P preserves both secrecy and integrity.*

Figure 1: Type system for cryptographic protocols

Consider the type system presented in Figure 1, which is a simplified version of the type sytsem presented in the lecture (here we only consider asymmetric encryption).

Problem 1: Typing for Cryptographic Protocols (30 Points)

a) **(15 Points)** Consider the following process:

$$(\nu k_1 : \mathrm{DecK}^{HH}[LL, HL])\ .c(x).\mathbf{case}\ x\ \mathbf{of}\ \{|[y, z]|\}^a_{k_1}\ \mathbf{in}$$
$$(\nu k_2 : \mathrm{DecK}^{HH}[LL, HL, HH]).(\nu v_1 : HH).\overline{c}\langle\{|[y, z, v_1]|\}^a_{\mathrm{ek}(k_2)}\rangle.0$$

Does it preserve secrecy? Does it preserve integrity? If yes, give a complete derivation tree to prove your answer (don't leave any steps implicit!). If not, give a counter example and motivate it.

b) **(5 Points)** Consider the following process:

$$(\nu k_1 : \mathrm{DecK}^{HH}[HH, LH])\ b(x).\mathbf{case}\ x\ \mathbf{of}\ \{|[y, z]|\}^a_{k_1}\ \mathbf{in}\ (\nu v_1 : HL).(\nu k_2 : \mathbf{t})\ \overline{c}\langle\{|v_1, y, z|\}^a_{\mathrm{ek}(k_2)}\rangle.\mathbf{0}$$

Give a concrete instantiation of the type $\mathbf{t}$ such that typing succeeds for the process. You do not have to write down the typing derivation, but do motivate your answer.

c) **(10 Points)** Show an example of a process that preserves secrecy, but not integrity. Motivate your answer.

Problem 2: Observational Equivalence (20 Points)

Preliminaries Recall the following definitions:

Definition 4 (Uniformity)**.** *We say that a biprocess P is* uniform *when* $\mathsf{fst}(P) \to Q_1$ *implies that* $P \to Q$ *for some biprocess* Q *such that* $\mathsf{fst}(Q) \equiv Q_1$ *and symmetrically for* $\mathsf{snd}(P) \to Q_2$*.*

Definition 5 (Observational Equivalence)**.** *Observational equivalence is the largest equivalence relation* $\approx$ *between closed processes such that* $P \approx Q$ *implies:*

- *if* $P \downarrow\downarrow a$ *then* $Q \downarrow\downarrow a$ *(If* P *outputs on channell* a*, then also* Q *outputs on channel* a*)*
- *if* $P \to^* P'$ *then* $Q \to^* Q'$ *and* $P' \approx Q'$*, for some* Q'
- $C[P] \approx C[Q]$ *for all closing evaluation contexts* $C[\ ]$

We say that a biprocess P *satisfies observational equivalence if* $\mathsf{fst}(P) \approx \mathsf{snd}(P)$*.*

Theorem 2 (Uniformity implies observational equivalence)**.** *Let* P_0 *be a closed biprocess. If for all plain evaluation contexts* C *and reductions* $C[P_0] \to^* P$*, the biprocess* P *is uniform, then* P_0 *satisfies observational equivalence (i.e.,* $\mathsf{fst}(P_0) \approx \mathsf{snd}(P_0)$*).*

Remember that we can get $\mathsf{fst}(P)$ by replacing every occurrence of $\mathsf{choice}[x, y]$ in P by x and $\mathsf{snd}(P)$ by replacing every occurrence of $\mathsf{choice}[x, y]$ in P by y, respectively. The internal reduction rules and structural equivalence rules are stated in Figure 2. Also note that we use the syntax (if $N = M$ then P else Q) as syntactic sugar for ($\mathsf{let}\ \_ = \mathrm{eq}(N, M)\ \mathsf{in}\ P$ else Q), where we have $\mathrm{eq}(x, x) = x$. You may also use the if construct in your contexts.

In the processes below, let h to be a constructor that does not have a destructor (as cryptographic hash functions are typically modeled).

For each of the following four biprocesses, answer the following questions:

(i) Does the biprocess execute uniformly in any plain evaluation context.

(ii) Does the biprocess satisfy observational equivalence.

Motivate your answers! In case one of the properties does not hold, provide a counterexample by giving the context.

a) **(4 Points)** $P_1 := \mathsf{new}\ n.\mathsf{out}(c, \mathsf{choice}[n, h(n)])$

b) **(4 Points)** $P_2 := \mathsf{out}(c, \mathsf{choice}[A, h(A)])$ where we assume that A is a publicly known constant.

c) **(4 Points)** $P_3 := \mathsf{new}\ n.\mathsf{new}\ k_1.\mathsf{new}\ k_2.\mathsf{out}(c, \mathbf{vk}(k_1)).\mathsf{out}(c, \mathsf{choice}[\mathbf{sign}(n, k_1), \mathbf{sign}(n, k_2)])$
where signing is defined as in project 1. That is, $\mathbf{verify}(\mathbf{sign}(m, k), \mathbf{vk}(k))) = \mathit{true}$.

d) **(4 Points)** $P_4 := \mathsf{new}\ r_1.\mathsf{new}\ r_2.\mathsf{new}\ m.\mathsf{new}\ k.\mathsf{out}(c, r_1).\mathsf{out}(c, \mathsf{choice}[\{m\}^{r_1}_{\mathbf{ek}(k)}, \{m\}^{r_2}_{\mathbf{ek}(k)}])$
where encryption and decryption are defined as in project 1. That is, $\mathbf{dec}(\{m\}^{r}_{\mathbf{ek}(k)}, k) = m$.

e) **(4 Points)** $P_5 := \mathsf{out}(c, \mathsf{choice}[A, B]) \mid \mathsf{out}(c, \mathsf{choice}[B, A])$
where we assume that A and B are two publicly known constants.

$$
\begin{array}{llll}
\mathsf{out}(N, M).Q \mid \mathsf{in}(N', x).P & \to & Q \mid P\{M/x\} & \text{if } \Sigma \vdash N = N' \quad \text{(Red I/O)} \\
\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q & \to & P\{M/x\} & \text{if } g(M_1, \ldots, M_n) \Downarrow M \quad \text{(Red Fun 1)} \\
\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q & \to & Q & \text{if } \nexists M.\ g(M_1, \ldots, M_n) \Downarrow M \quad \text{(Red Fun 2)} \\
!P & \to & P \mid !P & \text{(Red Repl)} \\
P \to Q & \Rightarrow & P \mid R \to Q \mid R & \text{(Red Par)} \\
P \to Q & \Rightarrow & \mathsf{new}\ a.P \to \mathsf{new}\ a.Q & \text{(Red Res)} \\
P' \equiv P, P \to Q, Q \equiv Q' & \Rightarrow & P' \to Q' & \text{(Red Eq)}
\end{array}
$$

$$
\begin{array}{lll}
\mathsf{out}(N, M).Q \mid \mathsf{in}(N', x).P \to Q \mid P\{M/x\} & \text{if } \Sigma \vdash \mathsf{fst}(N) = \mathsf{fst}(N') \text{ and } \Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N') & \text{(Red I/O)} \\
\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q \to P\{\mathsf{choice}[M_1, M_2]/x\} & \text{if } \mathsf{fst}(D) \Downarrow M_1 \text{ and } \mathsf{snd}(D) \Downarrow M_2 & \text{(Red Fun 1)} \\
\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q \to Q & \text{if } \nexists M_1.\ \mathsf{fst}(D) \Downarrow M_1 \text{ and } \nexists M_2.\ \mathsf{snd}(D) \Downarrow M_2 & \text{(Red Fun 2)}
\end{array}
$$

$$
\begin{array}{llll}
P \mid 0 \equiv P & & P \equiv P & \\
P \mid Q \equiv Q \mid P & & Q \equiv P & \Rightarrow\ P \equiv Q \\
(P \mid Q) \mid R \equiv P \mid (Q \mid R) & & P \equiv Q, Q \equiv R & \Rightarrow\ P \equiv R \\
\mathsf{new}\ a.\mathsf{new}\ b.P \equiv \mathsf{new}\ b.\mathsf{new}\ a.P & & P \equiv Q & \Rightarrow\ P \mid R \equiv Q \mid R \\
\mathsf{new}\ a.(P \mid Q) \equiv P \mid \mathsf{new}\ a.Q \quad \text{if } a \notin fn(P) & & P \equiv Q & \Rightarrow\ \mathsf{new}\ a.P \equiv \mathsf{new}\ a.Q
\end{array}
$$

Figure 2: Internal reduction and structural equivalence ules for the applied π-calculus, with and without biprocesses.

Problem 3: Information Flow (20 points)

Rules for typing expressions

$$\textsc{Int}\ \frac{}{n : L} \qquad \textsc{R-Val}\ \frac{\Gamma(x) = \tau\, var}{\Gamma \vdash x : \tau} \qquad \textsc{Op}\ \frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau \qquad \mathbf{op} \in \{+, -, >, <\}}{\Gamma \vdash e_1\ \mathbf{op}\ e_2 : \tau}$$

Rules for typing commands

$$\textsc{Skip}\ \frac{}{\Gamma \vdash \text{skip} : H\, cmd} \qquad \textsc{Assign}\ \frac{\Gamma(x) = \tau\, var \qquad \Gamma \vdash e : \tau}{\Gamma \vdash x := e : \tau\, cmd}$$

$$\textsc{If}\ \frac{\Gamma \vdash e : \tau \qquad \Gamma \vdash c_1 : \tau\, cmd \qquad \Gamma \vdash c_2 : \tau\, cmd}{\Gamma \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 : \tau\, cmd} \qquad \textsc{While}\ \frac{\Gamma \vdash e : \tau \qquad \Gamma \vdash c : \tau\, cmd}{\Gamma \vdash \text{while } e \text{ do } c : \tau\, cmd}$$

$$\textsc{Compose}\ \frac{\Gamma \vdash c_1 : \tau\, cmd \qquad \Gamma \vdash c_2 : \tau\, cmd}{\Gamma \vdash c_1; c_2 : \tau\, cmd}$$

Subtyping rules

$$\textsc{Base}\ \frac{}{L \subseteq H} \qquad \textsc{Cmd}^{-}\ \frac{\tau' \subseteq \tau}{\tau\, cmd \subseteq \tau'\, cmd} \qquad \textsc{Reflex}\ \frac{}{\rho \subseteq \rho} \qquad \textsc{Trans}\ \frac{\rho_1 \subseteq \rho_2 \qquad \rho_2 \subseteq \rho_3}{\rho_1 \subseteq \rho_3}$$

$$\textsc{Subsump}\ \frac{\Gamma \vdash p : \rho_1 \qquad \rho_1 \subseteq \rho_2}{\Gamma \vdash p : \rho_2}$$

Figure 3: Type System for Non-interference.

a) **(12 Points)** For this exercise, use the type system for non-interference reported in Figure 3.

Assume that `l` : L and `h` : H. Try to infer a type for the following program. Show a typing derivation and annotate every step with the applied rule. In case no type exists, point out in the derivation where and why type-checking necessarily fails and argue whether the program is secure or not.

```
while l > 0 do
{
  if h = 0 then
  {
    h := h + 1
  }
  else
  {
    skip
  };
  l := l - 1
};
h := l
```

b) **8 points** Does typing with the type system presented in Figure 3 also imply termination-sensitive non-interference? I.e., does typing prevent an attacker from learning any information about the values labelled as H, if the attacker knows whether the program terminated or not? Either reason why this is the case or give a counterexample.

Problem 4: Static Analysis (30 points)

Assume the tiny goto language with the following instructions:

$$\mathsf{PUSH}\ v,\ \mathsf{DUP},\ \mathsf{ADD},\ \mathsf{GOTOif}\ i \qquad\qquad i \in \mathbb{N}, v \in \mathbb{Z}$$

You can think of programs in this language as simple imperative programs that can modify a stack and otherwise can just alter the control flow by conditional jumps. More precisely, the instruction $\mathsf{PUSH}\ v$ pushes the value v to the stack, the instruction DUP duplicates the top stack value, the instruction ADD adds the two top values from the stack, pops them, and writes the result to the stack, and finally $\mathsf{GOTOif}\ i$ reads the top value from the stack, pops it, and jumps to the instruction at program counter i only if the top stack value is 0.

More formally, we describe the state of a program by a configuration of the form (pc, s). Intuitively, such a configuration says that pc is the current program counter and the current stack is s where s is a list of integer numbers. Note that we write ϵ for the empty stack and $v :: s'$ for a stack with top element v and substack s'.

Next, we can define the semantics of the goto language in terms of a small-step relation. More precisely, $code \vdash (pc, s) \to (pc', s')$ means that given a program $code$ (which is just a list of instructions), the configuration changes from (pc, s) to (pc', s') within processing one instruction. Formally, the small-step relation is defined by the following inference rules:

$$\text{PUSH}\ \frac{code[pc] = \mathsf{PUSH}\ v}{code \vdash (pc, s) \to (pc + 1, v :: s)} \qquad \text{DUP}\ \frac{code[pc] = \mathsf{DUP}}{code \vdash (pc, v :: s) \to (pc + 1, v :: (v :: s))}$$

$$\text{ADD}\ \frac{code[pc] = \mathsf{ADD}}{code \vdash (pc, v_1 :: (v_2 :: s)) \to (pc + 1, (v_1 + v_2) :: s)} \qquad \text{GOTOif-true}\ \frac{code[pc] = \mathsf{GOTOif}\ i \qquad v = 0}{code \vdash (pc, v :: s) \to (i, s)}$$

$$\text{GOTOif-false}\ \frac{code[pc] = \mathsf{GOTOif}\ i \qquad v \neq 0}{code \vdash (pc, v :: s) \to (pc + 1, s)}$$

a) **(6 Points)** Assume the following simple program:

```
1 PUSH 1
2 DUP
3 GOTOifz 8
4 PUSH -1
5 ADD
6 PUSH 0
7 GOTOifz 2
8 ADD
```

Starting in the configuration $(1, \epsilon)$ which configurations are reachable using the small step semantics? Name the rules (in the right order) that need to be applied to reach the corresponding configurations and give all intermediate configurations.

b) **(8 + 8 + 8 Points)** In the following, we will present different approaches to defining an abstract analysis for the presented language. As the versions of static analysis that were presented in the lecture, these approaches are based on Horn clause resolution. For each of the presented analysis approaches we define an abstraction function α that maps configurations into a state predicate (S) which models the abstract configuration. Analogously, we define a function h that defines the Horn clauses describing the abstract semantics.

Finally, we state soundness claims of the form

$$code \vdash c \to^* c' \implies \forall \Delta \geq \alpha(c).\ \exists \Delta' \geq \alpha(c').\ \Delta, h(code) \vdash \Delta'$$

which states that whenever configuration c executes $code$ and reaches configuration c' (within some number of steps) then it also holds that from any abstract configuration Δ that is at least as abstract as $\alpha(c)$ $(\Delta \geq \alpha(c))$ one can logically derive $(\vdash)$ an abstract configuration Δ' that is at least as abstract as $\alpha(c')$ $(\Delta' \geq \alpha(c'))$ using the Horn clauses $h(code)$.

Intuitively, an abstract configuration Δ is at least as abstract as an abstract configuration Δ' if for every fact (predicate application) in Δ' one can find one in Δ that is at least as abstract. Formally $\geq$ on abstract configurations is defined as follows:

$$\Delta \geq \Delta' := \quad \forall f' \in \Delta'.\ \exists f \in \Delta. f \geq_F f'$$

Note that we also need to define what it means for a fact (predicate application) to be as abstract as another ($\geq_F$). Since this notion of abstraction can be specific to the analysis, we will give the abstraction relation $\geq_F$ individually for each of the presented analysis approaches.

Your task is to state for each of the presented analysis approaches if they satisfy the soundness claim.

- If yes, argue why the soundness claim holds and name (and explain) at least one point where the analysis looses precision. More concretely, give an example program *code* and initial configuration c such that that there is an abstract configuration Δ' that is derivable from $\alpha(c)$, but there is no concrete configuration c' that is reachable from c such that $\alpha(c') = \Delta'$. Provide also all reachable configurations c' explicitly, as well as $h(code)$.
- If not, give a counter example to the soundness claim. More precisely give a program *code* and initial configuration c, and an abstract configuration $\Delta \geq \alpha(c)$ such that there is a configuration c' that is reachable from c, but there is no $\Delta' \geq \alpha(c')$ that is derivable from Δ. Provide also c' and $h(code)$ explicitly and explain why no such Δ' is derivable.

(i) Predicates:

$$\{\mathsf{S}_i(size, a, b) \mid i, size \in \mathbb{N} \wedge a, b \in \mathbb{Z}\}$$

Abstraction relation:

$$\begin{aligned}\mathsf{S}_i(size_1, a_1, b_1) \geq_F \mathsf{S}_i(size_2, a_2, b_2) := size_1 = size_2 \wedge (&size_1 = 0\\ &\vee size_1 = 1 \wedge a_1 = a_2\\ &\vee size_1 \geq 2 \wedge a_1 = a_2 \wedge b_1 = b_2)\end{aligned}$$

Abstraction of configurations:

$$\alpha_1((pc, s)) = \begin{cases}\{\mathsf{S}_{pc}(0,0,0)\} & s = \epsilon\\ \{\mathsf{S}_{pc}(1,v,0)\} & s = v :: \epsilon\\ \{\mathsf{S}_{pc}(|s|, v_1, v_2)\} & s = v_1 :: (v_2 :: s')\end{cases}$$

Abstract rules:

$$\begin{aligned}h_1(code) = \\ &\{\mathsf{S}_{pc}(size, a, b) \Rightarrow \mathsf{S}_{pc+1}(size+1, v, a) \mid code[pc] = \mathsf{PUSH}\ v\}\\ &\cup \{\mathsf{S}_{pc}(size, a, b) \wedge size > 0 \Rightarrow \mathsf{S}_{pc+1}(size+1, a, a) \mid code[pc] = \mathsf{DUP}\}\\ &\cup \{\mathsf{S}_{pc}(size, a, b) \wedge size > 1 \Rightarrow \mathsf{S}_{pc+1}(size-1, a+b, 0) \mid code[pc] = \mathsf{ADD}\}\\ &\cup \{\mathsf{S}_{pc}(size, a, b) \wedge size > 0 \wedge a = 0 \Rightarrow \mathsf{S}_i(size-1, b, 0) \mid code[pc] = \mathsf{GOTOif}\ i\}\\ &\cup \{\mathsf{S}_{pc}(size, a, b) \wedge size > 0 \wedge a \neq 0 \Rightarrow \mathsf{S}_{pc+1}(size-1, b, 0) \mid code[pc] = \mathsf{GOTOif}\ i\}\end{aligned}$$

(ii) Predicates:

$$\{\mathsf{S}_i(a, b) \mid i \in \mathbb{N} \wedge a, b \in \mathbb{Z} \cup \{\clubsuit\}\}$$

Abstraction relation:

$$\mathsf{S}_i(a_1, b_1) \geq_F \mathsf{S}_i(a_2, b_2) := (a_1 = \clubsuit \vee a_1 = a_2) \wedge (b_1 = \clubsuit \vee b_1 = b_2)$$

Abstraction of configurations:

$$\alpha_2((pc, s)) = \begin{cases}\{\mathsf{S}_{pc}(\clubsuit, \clubsuit)\} & s = \epsilon\\ \{\mathsf{S}_{pc}(v, \clubsuit)\} & s = v :: \epsilon\\ \{\mathsf{S}_{pc}(v_1, v_2)\} & s = v_1 :: (v_2 :: s')\end{cases}$$

Abstract rules:

$$
\begin{aligned}
h_2(\mathit{code}) = \\
&\{\mathsf{S}_{pc}(\mathit{size}, a, b) \Rightarrow \mathsf{S}_{pc+1}(v, a) \mid \mathit{code}[pc] = \mathsf{PUSH}\ v\} \\
&\cup \{\mathsf{S}_{pc}(a, b) \Rightarrow \mathsf{S}_{pc+1}(a, a) \mid \mathit{code}[pc] = \mathsf{DUP}\} \\
&\cup \{\mathsf{S}_{pc}(a, b) \wedge a \in \mathbb{Z} \wedge b \in \mathbb{Z} \Rightarrow \mathsf{S}_{pc+1}(a + b, \clubsuit) \mid \mathit{code}[pc] = \mathsf{ADD}\} \\
&\cup \{\mathsf{S}_{pc}(\clubsuit, b) \Rightarrow \mathsf{S}_{pc+1}(\clubsuit, \clubsuit) \mid \mathit{code}[pc] = \mathsf{ADD}\} \\
&\cup \{\mathsf{S}_{pc}(a, \clubsuit) \Rightarrow \mathsf{S}_{pc+1}(\clubsuit, \clubsuit) \mid \mathit{code}[pc] = \mathsf{ADD}\} \\
&\cup \{\mathsf{S}_{pc}(a, b) \wedge a = 0 \Rightarrow \mathsf{S}_{i}(b, \clubsuit) \mid \mathit{code}[pc] = \mathsf{GOTOif}\ i\} \\
&\cup \{\mathsf{S}_{pc}(a, b) \wedge a = \clubsuit \Rightarrow \mathsf{S}_{i}(b, \clubsuit) \mid \mathit{code}[pc] = \mathsf{GOTOif}\ i\} \\
&\cup \{\mathsf{S}_{pc}(a, b) \wedge a \neq 0 \Rightarrow \mathsf{S}_{pc+1}(b, \clubsuit) \mid \mathit{code}[pc] = \mathsf{GOTOif}\ i\}
\end{aligned}
$$

(iii) Predicates:

$$\{\mathsf{S}_i(s) \mid i \in \mathbb{N} \wedge s \in \mathcal{L}(\mathbb{Z})\}$$

where $\mathcal{L}(\mathbb{Z})$ denotes the set of lists over integer numbers.

Abstraction of configurations:

$$\alpha_3((pc, s)) = \{\mathsf{S}_{pc}(s)\}$$

Abstraction relation:

$$\mathsf{S}_i(s_1) \geq_F \mathsf{S}_i(s_2) := s_1 = s_2$$

Abstraction of configurations:

$$\alpha_3((pc, s)) = \mathsf{S}_{pc}(s)$$

Abstract rules:

$$
\begin{aligned}
h_2(\mathit{code}) = \\
&\{\mathsf{S}_{pc}(s) \Rightarrow \mathsf{S}_{pc+1}(v :: s) \mid \mathit{code}[pc] = \mathsf{PUSH}\ v\} \\
&\cup \{\mathsf{S}_{pc}(s) \Rightarrow \mathsf{S}_{pc+1}(v :: s) \mid \mathit{code}[pc] = \mathsf{DUP} \wedge v \in \mathbb{Z}\} \\
&\cup \{\mathsf{S}_{pc}(v_1 :: (v_2 :: s)) \Rightarrow \mathsf{S}_{pc+1}((v_1 + v_2) :: s) \mid \mathit{code}[pc] = \mathsf{ADD}\} \\
&\cup \{\mathsf{S}_{pc}(v :: s) \wedge v = 0 \Rightarrow \mathsf{S}_{i}(s) \mid \mathit{code}[pc] = \mathsf{GOTOif}\ i\} \\
&\cup \{\mathsf{S}_{pc}(v :: s) \wedge v \neq 0 \Rightarrow \mathsf{S}_{pc+1}(s) \mid \mathit{code}[pc] = \mathsf{GOTOif}\ i\}
\end{aligned}
$$

This is the last page.