Formal Methods for Security and Privacy (SS 2021) Univ-Prof. Dr. Matteo Maffei

Retake Exam

Date: 23/09/2021 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 committing and aiding) will be held accountable.
- During the exam you are required to be connected to the following Zoom call with your camera on : `https://tuwien.zoom.us/j/93560959897?pwd=UlJMRDNMWm1qcldLOWlHbW56bThJQT09` (passcode 2n8kYQKw).
- **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	25	25	30	20	100

Problem 1: Typing for Cryptographic Protocols (25 Points)

$$
\begin{array}{rcl}
\ell_C, \ell_I & ::= & H \mid L \\
\ell & ::= & \ell_C \ell_I \\
\mu & ::= & \mathrm{Sym} \mid \mathrm{Sig} \\
T & ::= & \ell \mid \mathrm{C}^{\ell}[\tilde{T}] \mid \mu\mathrm{K}^{\ell}[\tilde{T}]
\end{array}
$$

$$
\begin{array}{rcl}
L & \sqsubseteq_C & H \\
H & \sqsubseteq_I & L \\
\ell_C^1 \ell_I^1 & \sqsubseteq & \ell_C^2 \ell_I^2 \iff \ell_C^1 \sqsubseteq_C \ell_C^2 \wedge \ell_I^1 \sqsubseteq_I \ell_I^2 \\
\ell_C^1 \sqcup \ell_C^2 & = & \begin{cases} L & \text{if } \ell_C^1 = \ell_C^2 = L \\ , H & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
\ell_1 & \leq & \ell_2 \text{ whenever } \ell_1 \sqsubseteq \ell_2 \\
LL & \leq & \mathrm{C}^{LL}[LL, \cdots, LL] \\
\mathrm{C}^{\ell}[\tilde{T}] & \leq & \ell \\
LL & \leq & \mu\mathrm{K}^{LL}[LL, \cdots, LL] \\
\mu\mathrm{K}^{\ell}[\tilde{T}] & \leq & \ell
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{L}(\ell) & = & \ell \\
\mathcal{L}(\mathrm{C}^{\ell}[\tilde{T}]) & = & \ell \\
\mathcal{L}(\mu\mathrm{K}^{\ell}[\tilde{T}]) & = & \ell \\
\mathcal{L}_{I,\Gamma}(\mathrm{vk}(\mathrm{K})) & = & \mathcal{L}_{I,\Gamma}(\mathrm{K}) \\
\mathcal{L}_{I,\Gamma}(\{|M|\}_K^s) & = & \begin{cases} H & \text{if } \Gamma(K) = \mathrm{SymK}^{HH}[T] \\ & \quad \wedge \mathcal{L}_{i,\Gamma}(m) \sqsubseteq_i \mathcal{L}_i(T) \\ L & \text{otherwise} \end{cases} \\
\mathcal{L}_C(\ell_C \ell_I) & = & \ell_C \\
\mathcal{L}_I(\ell_C \ell_I) & = & \ell_I \\
\mathcal{L}_{\Gamma}(M) & = & \begin{cases} \mathcal{L}(\Gamma(M)) & \text{if } M \in dom(\Gamma) \\ LL & \text{otherwise} \end{cases}
\end{array}
$$

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

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

$$\textsc{Subsumption}\ \frac{\Gamma \vdash M : T' \qquad T' \leq T}{\Gamma \vdash M : T} \qquad \textsc{SymEnc}\ \frac{\Gamma \vdash K : \mathrm{SymK}^{\ell_C \ell_I}[\tilde{T}] \qquad \Gamma \vdash \tilde{M} : \tilde{T}}{\Gamma \vdash \{|\tilde{M}|\}_K^s : L\ell_I}$$

$$\textsc{VerKey}\ \frac{\Gamma \vdash K : \mathrm{SigK}^{\ell_C \ell_I}[\tilde{T}]}{\Gamma \vdash \mathrm{vk}(K) : \mathrm{VerK}^{L\ell_I}[\tilde{T}]} \qquad \textsc{DigSig}\ \frac{\Gamma \vdash K : \mathrm{SigK}^{\ell_C \ell_I}[\tilde{T}] \qquad \Gamma \vdash \tilde{M} : \tilde{T} \qquad \ell'_C = \sqcup_{T \in \tilde{T}} \mathcal{L}_c(T)}{\Gamma \vdash [\tilde{M}]_K : \ell'_C \ell_I}$$

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

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

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

$$\textsc{SymDec}\ \frac{\Gamma \vdash M : T \qquad \Gamma \vdash K : \mathrm{SymK}^{\ell}[\tilde{T}] \qquad \Gamma, \tilde{x} : \tilde{T} \vdash P}{\Gamma \vdash \text{ case } M \text{ of } \{|\tilde{x}|\}_K^s \text{ in } P}$$

$$\textsc{SignCheck}\ \frac{\Gamma \vdash M : T \qquad \Gamma \vdash K : \mathrm{VerK}^{\ell_C, \ell_I}[\tilde{T}] \qquad \Gamma, \tilde{x} : \tilde{T} \vdash P \qquad \mathcal{L}_C(T) = H \Rightarrow \ell_i = H}{\Gamma \vdash \text{ case } M \text{ of } [\tilde{x}]_K \text{ in } P}$$

Figure 1: Type system for cryptographic protocols

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

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*

$$P \mid O \rightarrow^* (\nu b : C^{HH}[T])\ (\nu \tilde{a} : \tilde{T})\ (P' \mid \overline{b}\langle c\rangle.P'') \text{ or}$$
$$P \mid O \rightarrow^* (\nu k : \mu K^{HH}[T])\ (\nu \tilde{a} : \tilde{T})\ (P' \mid \text{case } \{|c|\}^s_k \text{ of } \{|x|\}^s_k \text{ in } P'')$$

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

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

Figure 2: Definitions and theorems

Consider the following protocol, where n is symmetrically encrypted with key k_1 and the pair $\{[n]\}_{k_1}, m$ is signed with key k_2.

$$A \qquad\qquad B$$
$$\longrightarrow^{[\{[n]\}_{k_1}, m]_{k_2}}$$

a) **(10 points)** Model this protocol in the spi-calculus, where

- n is a fresh nonce of type HH
- m is a fresh nonce type LL
- k_1 and k_2 are fresh keys with appropriate types (chosen by you), such that the protocol can be successfully typed with the rules from Figure 1.

b) **(15 points)** Prove that this process preserves secrecy and integrity, using Theorem 1 from Figure 2. Provide a complete typing derivation.

Problem 2: Bana-Comon Logic (25 points)

Predicates

$t \sqsubseteq t_1, \cdots t_n$	iff t is a subterm of some t_i
$t \sqsubseteq_{\not d} t_1, \cdots t_n$	iff any occurrence of t in any t_i is in a decryption key position (i.e., $dec(\cdot, t)$)
$RandGen(t)$	iff $t \in \mathcal{N}$
$Fresh(x; \vec{x})$	iff $RandGen(x) \wedge x \not\sqsubseteq \vec{x}$

Axioms

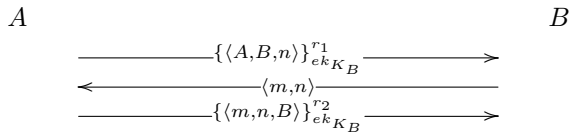
Self derivability	$\vec{x}, x \vdash x$
Increasing capabilities	$\vec{x} \vdash y \rightarrow \vec{x}, x \vdash y$
Commutativity	$\vec{x} \vdash y \rightarrow \vec{x'} \vdash y$ if $\vec{x'}$ is a permutation of $\vec{x}$
Transitivity	$\vec{x} \vdash y \wedge \vec{x}, y \vdash z \rightarrow \vec{x} \vdash z$
Function derivability	$\vec{x} \vdash f(\vec{x})$
No telepathy	$Fresh(x; \vec{x}) \rightarrow \vec{x} \not\rightarrow x$
Secrecy	$RandGen(K) \wedge ek_K \sqsubseteq \phi \wedge Fresh(R; \phi, x, y) \wedge \phi, \{x\}^R_{ek_K} \vdash y$ $\rightarrow dk_K \sqsubseteq_{\not d} \phi, x \vee \phi \vdash y$

Equations

$$
\begin{aligned}
dec(\{x\}^R_{ek_K}, dk_K) &= x \\
\pi_1(\langle x, y \rangle) &= x \\
\pi_2(\langle x, y \rangle) &= y \\
\pi_1(\langle x, y, z \rangle) &= x \\
\pi_2(\langle x, y, z \rangle) &= y \\
\pi_3(\langle x, y, z \rangle) &= z
\end{aligned}
$$

Figure 3: Rules for the Bana-Comon Logic

Consider the following protocol:



Here $n, r_1, r_2 \in \mathcal{N}$ are names known only to A and $k_B, m \in \mathcal{N}$ are names only known to B. A, B are public constants, and ek_{K_B} is public.

a) **(10 points)** Model the protocol as a transition system. Each party should check the correctness of incoming messages as precisely as possible with its knowledge. You can use the predicate $ID(\cdot)$ to check that some term is a valid identifier and you can assume that all names are of length η.

b) **(3 + 3 + 9 points)** Let ϕ be the frame that results from the execution of the first step (A sending the initial message to B) from the initial state. Prove formally that the following statements are inconsistent with the axioms:

(i) $\phi \not\vdash ek_{k_B}$

(ii) $\phi \vdash m$

(iii) $\phi \vdash n$

Justify each of the steps in your proof using the rules from Figure 3.

Problem 3: Information Flow (30 points)

Assume the tiny register-based language with the following instructions:

$$\mathsf{CONST}\ n\ r,\ \mathsf{ADD}\ r\ r\ r,\ \mathsf{SUB}\ r\ r\ r,\ \mathsf{JUMPifz}\ n\ r \qquad\qquad n \in \mathbb{N},\ r \in \mathbb{N}$$

Informally, the instruction $\mathsf{CONST}\ n\ r$ put the integer n in the register r, the instruction $\mathsf{ADD}\ r_0\ r_1\ r_2$ (resp. $\mathsf{SUB}\ r_0\ r_1\ r_2$) adds (resp. subtracts) values in registers r_0 and r_1 and puts the result in the register r_2, and the instruction $\mathsf{JUMPifz}\ l\ r$ jumps at the instruction at program counter n if the value in register r is equal to zero.

More formally, we describe the state of a program by a configuration of the form $\{pc, \pi\}$, where $pc \in \mathbb{N}$ is the current program counter and $\pi \in \mathbb{N} \mapsto \mathbb{N}$ is a mapping from registers to contained values. We assume an infinite number of register with zero as default value. For example, $\{0, [r_1 \mapsto 2, r_3 \mapsto 4]\}$ denotes the state where the program counter is 0, register r_1 contains value 2, register r_3 contains value 4 and all other registers contain value 0. Next, we can define the semantics of our register-based language in terms of a small-step relation, in which $code \vdash \{pc, \pi\} \to \{pc', \pi'\}$ means that, for given a program $code$ (which is just a list of instructions), the configuration changes from $\{pc, \pi\}$ to $\{pc', \pi'\}$ within processing one instruction:

$$\textsc{Const}\ \frac{code[pc] = \mathsf{CONST}\ n\ r}{code \vdash \{pc, \pi\} \to \{pc + 1, \pi[r \leftarrow n]\}}$$

$$\textsc{Add}\ \frac{code[pc] = \mathsf{ADD}\ r_0\ r_1\ r_2}{code \vdash \{pc, \pi\} \to \{pc + 1, \pi[r_2 \leftarrow \pi[r_0] + \pi[r_1]]\}} \qquad \textsc{JumpIfz-true}\ \frac{code[pc] = \mathsf{JUMPifz}\ n\ r \qquad \pi[r] = 0}{code \vdash \{pc, \pi\} \to \{n, \pi\}}$$

$$\textsc{Sub}\ \frac{code[pc] = \mathsf{SUB}\ r_0\ r_1\ r_2}{code \vdash \{pc, \pi\} \to \{pc + 1, \pi[r_2 \leftarrow \pi[r_0] - \pi[r_1]]\}} \qquad \textsc{JumpIfz-false}\ \frac{code[pc] = \mathsf{JUMPifz}\ n\ r \qquad \pi[r] \neq 0}{code \vdash \{pc, \pi\} \to \{pc + 1, \pi\}}$$

Figure 4: Concrete semantics

In this exercice, we would like to enforce constant-time non-interference on programs written in our register-based language. Constant-time non-interference is a stronger version of the non-interference presented during the lecture which prevents, among other things, leaking information by branching on secrets. To this end, we define a type system which manipulates judgments of the form $\Gamma \vdash pc : \gamma_1 \Rightarrow \gamma_2$. In this judgement, $\Gamma \in \mathbb{N} \mapsto (\mathbb{N} \mapsto \{L, H\})$ associates to every program counter a map from registers to security types, γ_1 is the map $\Gamma(pc)$, and γ_2 is the map we obtain after executing instruction pc over γ_1.

Typing rules for instructions, where $\gamma := \Gamma(pc)$

$$\textsc{Const}\ \frac{code[pc] = \mathsf{CONST}\ n\ r}{\Gamma \vdash pc : \gamma \Rightarrow \gamma[r \leftarrow L]} \qquad \textsc{JumpIfz}\ \frac{code[pc] = \mathsf{JUMPifz}\ n\ r \qquad \gamma[r] = L}{\Gamma \vdash pc : \gamma \Rightarrow \gamma}$$

$$\textsc{Add}\ \frac{code[pc] = \mathsf{ADD}\ r_0\ r_1\ r_2 \qquad \gamma[r_0] \subseteq \tau \qquad \gamma[r_1] \subseteq \tau}{\Gamma \vdash pc : \gamma \Rightarrow \gamma[r_2 \leftarrow \tau]}$$

$$\textsc{Sub}\ \frac{code[pc] = \mathsf{SUB}\ r_0\ r_1\ r_2 \qquad \gamma[r_0] \subseteq \tau \qquad \gamma[r_1] \subseteq \tau}{\Gamma \vdash pc : \gamma \Rightarrow \gamma[r_2 \leftarrow \tau]}$$

Subtyping rules

$$\textsc{Base}\ \frac{}{L \subseteq H} \qquad \textsc{Reflex}\ \frac{}{\rho \subseteq \rho} \qquad \textsc{Trans}\ \frac{\rho_1 \subseteq \rho_2 \qquad \rho_2 \subseteq \rho_3}{\rho_1 \subseteq \rho_3} \qquad \textsc{Subsump}\ \frac{\Gamma \vdash p : \rho_1 \qquad \rho_1 \subseteq \rho_2}{\Gamma \vdash p : \rho_2}$$

Figure 5: Type system for constant-time.

Definition 4 (Constant-Time Well-Typed Programs)**.** *A program p is constant-time well-typed with respect to Γ if, for every instruction pc and for all of its successors pc', there exists γ such that*

$$\Gamma \vdash pc : \Gamma(pc) \Rightarrow \gamma\ \wedge\ \gamma \sqsubseteq \Gamma(pc') \qquad \textit{where} \qquad \gamma \sqsubseteq \gamma'\ \ \textit{iff}\ \ \forall n \in \mathbb{N}. \gamma(n) \subseteq \gamma'(n)\,.$$

a) **(4+4 points)** Using the semantics given in Figure 4:

- Execute Program 1 starting from configuration $\{0, [r_0 \mapsto 0, r_1 \mapsto 1, r_2 \mapsto 2, r_3 \mapsto 3]\}$.
- Execute Program 2 starting from configuration $\{0, [r_0 \mapsto 0, r_1 \mapsto 3, r_2 \mapsto 1, r_3 \mapsto 2]\}$.

Name the rules (in the right order) that need to be applied and give all intermediate states.

b) **(6+6 points)** According to Definition 4, Figure 5, and environment given in Figure 6:

- Is Program 1 well-typed in Γ_1? In Γ_2?
- Is Program 2 well-typed in Γ_3? In Γ_4?

Show a type derivation or point out where and why type-checking necessarily fails.

c) **(8+2 points)** Using a case-by-case argument, explain how Definition 4 with Figure 5 ensures constant-time non-interference, i.e. that well-typed programs do not leak values in registers of type H into registers of type L, and that branching on a register of type H is prevented. What are the conditions that Γ needs to satisfy?

Program 1:

```
0 JUMPifz 2 r3
1 CONST   0 r0
2 ADD r1 r2 r3
3 JUMPifz 5 r0
4 SUB r2 r1 r0
5 // THE END //
```

Program 2:

```
0 CONST   1 r1
1 JUMPifz 5 r2
2 ADD r2 r3 r3
3 SUB r2 r1 r2
4 JUMPifz 1 r0
5 // THE END //
```

$$\Gamma_1 := \left\{ \begin{array}{l} \gamma_0 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto L] \\ \gamma_1 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto L] \\ \gamma_2 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto L] \\ \gamma_3 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto H] \\ \gamma_4 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto H] \\ \gamma_5 = [r_0 \mapsto H, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto H] \end{array} \right\}$$

$$\Gamma_2 := \left\{ \begin{array}{l} \gamma_0 = [r_0 \mapsto H, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto L] \\ \gamma_1 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto L] \\ \gamma_2 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto L] \\ \gamma_3 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto H] \\ \gamma_4 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto H] \\ \gamma_5 = [r_0 \mapsto H, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto H] \end{array} \right\}$$

$$\Gamma_3 := \left\{ \begin{array}{l} \gamma_0 = [r_0 \mapsto L, r_1 \mapsto H, r_2 \mapsto L, r_3 \mapsto H] \\ \gamma_1 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto L, r_3 \mapsto H] \\ \gamma_2 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto L, r_3 \mapsto H] \\ \gamma_3 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto L, r_3 \mapsto H] \\ \gamma_4 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto L, r_3 \mapsto H] \\ \gamma_5 = [r_0 \mapsto L, r_1 \mapsto H, r_2 \mapsto L, r_3 \mapsto H] \end{array} \right\}$$

$$\Gamma_4 := \left\{ \begin{array}{l} \gamma_0 = [r_0 \mapsto L, r_1 \mapsto H, r_2 \mapsto H, r_3 \mapsto L] \\ \gamma_1 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto L] \\ \gamma_2 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto H] \\ \gamma_3 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto H] \\ \gamma_4 = [r_0 \mapsto L, r_1 \mapsto L, r_2 \mapsto H, r_3 \mapsto H] \\ \gamma_5 = [r_0 \mapsto L, r_1 \mapsto H, r_2 \mapsto H, r_3 \mapsto H] \end{array} \right\}$$

Figure 6: Typing environments

Problem 4: Static Analysis (20 points)

In the previous exercise we defined in Definition 4 and Figure 5 a type system for constant-time non-interference parameterized by $\Gamma \in \mathbb{N} \mapsto (\mathbb{N} \mapsto \{L, H\})$. In this exercise we present a static analysis which computes such Γ for a given program. As the versions of static analysis that were presented in the lecture, this static analysis is base on Horn clause resolution.

Let α be an abstraction function that maps configurations into a state predicate (S) which models the abstract configuration. Analogously, let h be a function that defines the Horn clauses describing the abstract semantics. Then we define the following soundness claim

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

which states that whenever configuration c executes *code* and reaches a configuration c' 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 $h(\mathit{code})$.

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:

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

where $\geq_F$ is an abstract relation which states that a fact (predicate application) is as abstract as another.

Predicates:

$$\{S_i(\gamma) \mid \gamma \in \mathbb{N} \mapsto \{L, H\}\}$$

Abstraction relation:

$$S_i(\gamma) \geq_F S_i(\gamma') \quad \text{iff} \quad \forall n \in \mathbb{N}.\gamma'(n) \subseteq \gamma(n)$$

Abstraction of configurations:

$$\alpha(\{pc, \pi\}) = \{S_{pc}(\gamma)\} \quad \text{where } \gamma \text{ is a map from references to types.}$$

Abstract rules:

$$\begin{aligned} h(\mathit{code}) = {} & \{S_{pc}(\gamma) \Rightarrow S_{pc+1}(\gamma) \mid \mathit{code}[pc] = \mathsf{CONST}\ n\ r\} \\ & \cup \{S_{pc}(\gamma) \Rightarrow S_{pc+1}(\gamma[r_2 \leftarrow \gamma[r_0] \cup \gamma[r_1] \cup \gamma[r_2]]) \mid \mathit{code}[pc] = \mathsf{ADD}\ r_0\ r_1\ r_2\} \\ & \cup \{S_{pc}(\gamma) \Rightarrow S_{pc+1}(\gamma[r_2 \leftarrow \gamma[r_0] \cup \gamma[r_1] \cup \gamma[r_2]]) \mid \mathit{code}[pc] = \mathsf{SUB}\ r_0\ r_1\ r_2\} \\ & \cup \{S_{pc}(\gamma) \Rightarrow S_n(\gamma) \mid \mathit{code}[pc] = \mathsf{JUMPifz}\ n\ r\} \\ & \cup \{S_{pc}(\gamma) \Rightarrow S_{pc+1}(\gamma) \mid \mathit{code}[pc] = \mathsf{JUMPifz}\ n\ r\} \end{aligned}$$

Figure 7: Predicate, abstract relation, configuration and rules.

a) **(5+5 points)** Using Figure 7:

- Apply the abstract rules on Program 1 starting from abstract configuration $S_0([r_0 \mapsto L, r_1 \mapsto H, r_2 \mapsto L, r_3 \mapsto L])$.
- Apply the abstract rules on Program 2 starting from abstract configuration $S_0([r_0 \mapsto L, r_1 \mapsto H, r_2 \mapsto H, r_3 \mapsto L])$.

Name the rules (in the right order) that need to be applied and give all intermediate states.

b) **(6+4 points)** Argue why the soundness claim holds with predicates, abstraction relation, abstraction of configuration and abstract rules given in Figure 7. What is the connection between the result of this analysis and the Γ used in Definition 4 and Figure 5 for constant-time non-interference?

This is the last page.