Appendix

This document contains additional Appendix for the submission titled "*Runtime Verification of Hyperproperties for Deterministic Programs*". In this Appendix, we present how other security and information-flow policies such as non-interference and software doping can be expressed using Hyper$_{2S}$.

A Properties expressed in Hyper$_{2S}$

In this section, we consider some security and information-flow policies, and present how they can be formulated as safety hyperproperties using Hyper$_{2S}$ (a subset of HyperLTL).

A.1 Noninterference

A program satisfies noninterference if every pair of traces with the same (initial) low observation remain indistinguishable for low users. The outputs observed by the low security users (publicly visible output) should not depend on secret (high) information.

Let us recall that we consider program $\mathcal{P} : I \times O$, a single execution of $\mathcal{P}$ is an input-output event $(i, o) \in I \times O$. Consider alphabet $\Sigma = I \times O$. Consider an input-output trace π, where projection on inputs is denoted as π_I, and projection on outputs is denoted using π_O. Projection of high inputs is denoted as $\pi_{I,H}$, and projection on high outputs is denoted as $\pi_{O,H}$.

Definition 1 (Noninterference). *Given program $\mathcal{P} : I \to O$, noninterference of $\mathcal{P}$ is expressed as a hyperproperty in Hyper$_{2S}$ as:*

$$\forall \pi, \forall \pi' : (\pi_{I,L} = \pi'_{I,L}) \implies (\pi_{O,L} = \pi'_{O,L})$$

Hyper$_{2S}$formula in Definition 1 is a safety hyperproperty. It expresses that for any two traces π and π', if their projection on low inputs are equal, then their projection on low outputs should be equal.

A.2 Integrity

Integrity expresses that high behaviour of a system cannot be altered maliciously. More precisely, high behaviour of a system should not be influenced by low inputs, that can be potentially altered by a malicious user. Traces having the same high inputs but possibly different low inputs should have the same high outputs.

Definition 2 (Integrity). *Given program $\mathcal{P} : I \to O$, integrity of $\mathcal{P}$ is expressed in Hyper$_{2S}$ as:*

$$\forall \pi, \forall \pi' : (\pi_{I,H} = \pi'_{I,H}) \implies (\pi_{O,H} = \pi'_{O,H})$$

Hyper$_{2S}$ formula in definition 2 is a safety hyperproperty. It expresses that for any two traces π and π', if their projection on high inputs are equal, then their projection on high outputs should be equal.

A.3 Software doping

A software system can be considered as doped if some *hidden* functionality is incorporated by the software manufacturer in a way that the resulting behaviour of the system is against the interest of the software licensee or of the society and intentionally favours a designated party [1,2]. A doped software thus includes behaviour that serves some hidden interest favouring a certain manufacturer, or vendor which cannot be justified by the interest of the licensee.

In [1], some real cases of software doping were described, and [2] describes formal definitions that enable to identify whether a program is clean or doped.

Different characterizations of doping-free software were described in [2]. We consider one simple definition of characterization of a doping-free software from [2] and formulate it using Hyper$_{2S}$.

Let us recall some notations. I denotes a finite set of inputs, and $I' \subset I$ is a set of standard inputs. O denotes a finite set of outputs, $Parm$ denotes a finite set of parameters, and $PIntrs \subset Parm$ denotes parameters of interest. The alphabet $\Sigma = (I \times Parm) \times O$, and each execution of program $\mathcal{P}$ is an event $((i,p),o) \in \Sigma$, where $i \in I$, $p \in Parm$, and $o \in O$.

A parameterised program $\mathcal{P}$ is clean (doping-free) if for all pairs of parameters of interest $p, p' \in PIntrs$, and input $i \in I$, if $i \in Stdin$, then $\mathcal{P}(p)(i) = \mathcal{P}(p')(i)$. Consider a trace $\pi \in \Sigma$. π_{Parm} denotes projection on parameter values, π_I denotes projection on inputs and π_O denotes projection on outputs.

It can be expressed in Hyper$_{2S}$ as follows:

Definition 3 (Clean (doping-free) program). *A parametrized program $\mathcal{P} : I \times Parm \to O$, is clean (doping-free) if*

$$\begin{aligned}\forall\pi, \forall\pi' : ((\pi_{Parm} \in PIntrs) \wedge (\pi'_{Parm} \in PIntrs) \wedge (\pi_I = \pi'_I))\\ \implies (\pi_O = \pi'_O)\end{aligned}$$

Hyper$_{2S}$ formula in definition 3 is a safety hyperproperty. It expresses that for any two traces π and π', if the parameter values in both traces belong to the set of parameters of interest $PIntrs$, and if the inputs in both traces π and π' are equal, then the outputs in both traces π and π' should be also equal.

Examples illustrating Definition 3 are given in [2]. Other notions of software doping are also described in [2].

A.4 Strong distributed minimality

We briefly discussed in Section 2.4 about data minimisation in the monolithic case (programs with single input source). Further details are given in [3]. In [3], we also detailed about data minimisation properties in the distributed case for programs with multiple (independent) input sources and their monitoring.

A program in the distributed case with n input sources can be considered as a function, denoted as $\mathcal{P} : I_1 \times \cdots \times I_n \to O$. We consider monitoring input-output behavior of multiple executions of program $\mathcal{P}$. Repeated execution of $\mathcal{P}$ is denoted as $\mathcal{P}l$, where $\mathcal{P}l : I^* \to O^*$, with $I = I_1 \times \cdots \times I_n$.

Monolithic minimality is in general not suitable (too strong a notion) for programs with multiple input sources. Let us consider the following examples. Let $\mathcal{P}$ be the program $\mathrm{XOR} : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ that takes two Boolean inputs and returns a Boolean as output. Since $\mathrm{XOR}(0,0) = \mathrm{XOR}(1,1)$, it follows that XOR is not monolithic minimal. The program $\mathrm{OR} : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ is also not monolithic minimal since $\mathrm{OR}(0,1) = \mathrm{OR}(1,0)$.

Here, we briefly present a variant of distributed minimality (called as strong distributed minimality in [3]), how it can be expressed as hyperproperty.

Definition 4 (Strong distributed minimality of program $\mathcal{P}$). *Program $\mathcal{P} : I_1 \times \cdots \times I_n \to O$, where for all input sources $id \in [1,n]$, I_{id} is the set of possible inputs from source id, $I' \subseteq I$, and O is the set of possible outputs, $\mathcal{P}$ is* strongly distributed minimal *for I' iff:*
for any two input events $(i_1, \cdots, i_n)$ and $(i'_1, \cdots, i'_n)$ belonging to I' that differ exactly in one element, the output that the program $\mathcal{P}$ produces for input $(i'_1, \cdots, i'_n)$ is different from the output that it produces for input $(i_1, \cdots, i_n)$. Formally,

$$\begin{aligned}
&\forall (i_1, \cdots, i_n), (i'_1, \cdots, i'_n) \in I' : \\
&\quad (\exists j \in [1,n] : i_j \neq i'_j \wedge \forall k \in [1,n] : k \neq j \implies i_k = i'_k) \implies \\
&\qquad \mathcal{DF}((i_1, \cdots, i_n)) \neq \mathcal{DF}((i'_1, \cdots, i'_n))
\end{aligned}$$

Note that when the number of input sources is one, strong distributed minimality (Definition 4) reduces to the monolithic minimality (Definition of monolithic minimality is given in Section 2.4 in the paper).

Strong distributed minimality can be expressed as hyperproperty as follows:

Definition 5 (Strong distributed minimality property). *A program $\mathcal{P} : I \to O$ (where $I = I_1 \times \cdots \times I_n$) is strong distributed minimal if,*

for any two executions π and π' of $\mathcal{P}$, where $(i_1, \cdots, i_n)$ and $(i'_1, \cdots, i'_n)$ are the input events corresponding to π and π', If only one input source value differ in $(i_1, \cdots, i_n)$ and $(i'_1, \cdots, i'_n)$, then the projection on outputs of π and π' should differ. Formally,

$$\begin{aligned}
&\forall \pi, \forall \pi', \\
&\quad \textit{let } \pi_i = (i_1, \cdots, i_n), \pi'_i = (i'_1, \cdots, i'_n). \\
&\quad (\exists x \in [1,n] : i_x \neq i'_x \wedge \forall y \in [1,n] : y \neq x \implies i_y = i'_y) \implies \pi_o \neq \pi'_o.
\end{aligned}$$

Runtime verification monitor for strong distributed minimality can also be synthesised using the approach presented in this paper. Details regarding monitoring for data minimality (monolithic ans distributed cases) are given in [3].

References

1. Barthe, G., D'Argenio, P.R., Finkbeiner, B., Hermanns, H. In: Facets of Software Doping. Springer International Publishing, Cham (2016) 601–608
2. D'Argenio, P.R., Barthe, G., Biewer, S., Finkbeiner, B., Hermanns, H.: Is your software on dope? In: ESOP, New York, NY, USA, Springer-Verlag New York, Inc. (2017) 83–110
3. Pinisetty, S., Antignac, T., Sands, D., Schneider, G.: Monitoring data minimisation. CoRR **abs/1801.02484** (2018)