Verifying Information Flow Properties of Hybrid Systems

Pavithra Prabhakar
IMDEA Software Institute
pavithra.prabhakar@imdea.org

Boris Köpf
IMDEA Software Institute
boris.koepf@imdea.org

ABSTRACT

In this paper, we study the problem of analyzing information flow properties of hybrid systems. We begin by formalizing non-interference – the baseline information flow property – for hybrid systems. We then present a type system for statically enforcing non-interference, together with a proof of soundness. We conclude with discussions on future work towards analyzing more permissive information flow properties.

1. INTRODUCTION

With incidents such as Stuxnet attacking SCADA systems (see, e.g., [2]), the security of embedded control systems has come to the limelight. One of the most fundamental security properties is *confidentiality*, i.e., the requirement that secret information can only be learned by authorized parties. Security mechanisms such as access control, firewalls, and encryption [5] can ensure confidentiality while secrets are stored in databases or transferred over networks—but they cannot provide security guarantees while actual computation is being performed on the secrets. Information flow control [10, 8] protects secret information during computation, with the goal of achieving end-to-end confidentiality guarantees. The baseline information flow property is non-interference [4], which forbids any flow of information from classified (or *high*) to public (or *low*) domains.

A wide range of techniques have been proposed for reasoning about information flow properties in the context of discrete programs (see the survey [8]). Most of these techniques rely on static program analysis, which has the advantage of little or no run-time overhead when compared to techniques based on dynamic execution monitoring. Among the static analysis techniques for enforcing secure information flow, type systems for non-interference are probably the most popular approach. For example, in the type system by Volpano et al. [12], each program variable is annotated with a security type that specifies the domain of the data it stores, and each command is annotated with a security type that specifies the domain of the data it may affect. When type-checking a program, these annotations are combined using typing rules based on program syntax. A successful type check implies that the program will satisfy non-interference at run-time.

Unfortunately, the problem of enforcing information flow has received little attention in the context of *embedded control systems*, namely, systems in which embedded processors are employed for control. These include safety critical systems such as aerospace, automotive and medical devices among others. An inherent characteristic is the interaction of the embedded processors with a continuous environment. Due to the mixed discrete-continuous behavior exhibited by them, they are popularly referred to as hybrid systems.

In this paper, we investigate the use of type systems for enforcing non-interference for hybrid systems. To this end, we first present a syntax for hybrid systems that resembles that of discrete programs, along with a deterministic semantics. We formalize non-interference based on this model. Intuitively, non-interference for hybrid systems requires that for every two input signals that agree on the values of the variables of low security levels, the output signals also agree on the values of the variables of low security levels. Hence, the observable or low part of the output is invariant with respect to the high input.

We then present a static analysis method for inferring non-interference in hybrid systems. The analysis is based on the type system developed in [12] but includes additional typing judgments for the continuous specification of the system. The typing rule for the continuous dynamics needs to capture the interdependent nature of the evolution of the continuous variables. We prove the soundness of the type system, in the sense that a successful type check implies that the hybrid system satisfies non-interference. We conclude with directions for future work.

2. PRELIMINARIES

Let $\mathbb{N}$, $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$ denote the set of natural, real and non-negative real numbers, respectively. Given a function f, we use $dom(f)$ to denote the domain of f.

A *sequence* is a function $\sigma : A \to B$, where A is $\{0, \cdots, n\}$ for some $n \in \mathbb{N}$. Length of a sequence $\sigma : \{0, \cdots, n\} \to B$ is the number of elements in its domain, namely, $n+1$. Given two sequences σ_1 and σ_2, $\sigma_1\sigma_2$ denotes the concatenations of σ_2 after σ_1. More precisely, if $\sigma_1 : \{0, \cdots, n_1\} \to A$ and $\sigma_2 : \{0, \cdots, n_2\} \to A$, $\sigma_1\sigma_2 : \{0, \cdots, n_1 + n_2 + 1\} \to A$ such that $\sigma_1\sigma_2(i) = \sigma_1(i)$ for $0 \leq i \leq n_1$ and $\sigma_1\sigma_2(i) = \sigma_2(i - n_1 - 1)$ for $n_1 < i \leq n_1 + n_2 + 1$. We denote by

$\sigma[i,j]$, the subsequence of σ from the i-th element to the j-th element, that is, $\sigma[i,j] = \sigma(i)\cdots\sigma(j)$.

Given an element $\bar{x} \in \mathbb{R}^n$, we use $\bar{x}|_i$ to denote the projection of $\bar{x}$ to the i-th component, namely, $\bar{x}_i$, where $\bar{x} = (\bar{x}_1, \cdots, \bar{x}_n)$. Given a function $f : A \to \mathbb{R}^n$, the function $f|_i$ represent the projection of the image to the i-th component, that is, $f|_i : A \to \mathbb{R}$ such that for every $a \in A$, $f|_i(a) = f(a)|_i$.

3. DETERMINISTIC HYBRID SYSTEMS

Hybrid systems are systems exhibiting both discrete and continuous behaviors. The continuous behavior is typically modeled by a system of ordinary differential equations, and the discrete behavior corresponding to the control logic is modelled by a finite state automaton (see, for example, [6]). In general, the behavior of a hybrid system can be non-deterministic. In this paper, we focus on hybrid systems with deterministic behaviors, that is, systems which exhibit a unique behavior given an input signal and an initial state. To this end, we present a model which is analogous to a program.

3.1 Syntax

First, we present a simple syntax for expressing deterministic hybrid systems as a programming language.

Let us fix a set of input variables $Var^U = \{u_1, \cdots, u_k\}$ and a set of state variables $Var^X = \{x_1, \cdots, x_n\}$. A hybrid system $\mathcal{H}$ over (Var^U, Var^X) is defined inductively as follows:

$$
\begin{array}{lll}
\mathcal{H} := & \langle \dot{x}_1 = e, \cdots, \dot{x}_n = e, b\rangle & \text{(Flow)} \\
 & \mid [x_1 := e, \cdots, x_n := e] & \text{(Jump)} \\
 & \mid \mathcal{H}; \mathcal{H} & \text{(Sequential Composition)} \\
 & \mid \mathsf{if}\, b\, \mathsf{then}\, \mathcal{H}\, \mathsf{else}\, \mathcal{H} & \text{(Branching)} \\
 & \mid \mathsf{while}\, b\, \mathsf{do}\, \mathcal{H} & \text{(Looping)}
\end{array}
$$

where e is an expression over $Var^U \cup Var^X$ formed from constants and variables using the arithmetic operators of addition and multiplication, and b is a boolean combination of predicates of the form $e > 0$. We will also treat b as an expression returning the boolean values 0 or 1 and as being formed using the binary operators $>$, $\neg$ (essentially $\neg b$ is interpreted as $1 - b$) and $\wedge$. We assume that the variables occurring in b are state variables.

The expression e represents a polynomial expression over the variables in Var, and the predicate b is a polynomial constraint over the state variables. The continuous dynamics of the system is specified using the system of n differential equations and an invariant, $\langle \dot{x}_1 = e_1, \cdots, \dot{x}_n = e_n, b\rangle$. The variables evolve according to the differential equations and satisfy the constraint specified by b at all times. The expression $[x_1 := e_1, \cdots, x_n := e_n]$ specifies the discrete jumps, namely, the value of a state variable x_i after the jump is obtained by evaluating the expression e_i with the values of the variables before the jump. The operator $\mathcal{H}; \mathcal{H}$ specifies sequential composition, $\mathsf{if}\, b\, \mathsf{then}\, \mathcal{H}\, \mathsf{else}\, \mathcal{H}$ specifies branching and $\mathsf{while}\, b\, \mathsf{do}\, \mathcal{H}$ specifies looping.

3.2 Semantics

Next, we specify the semantics of the hybrid system. Before that, we define the notions of trajectories and transitions.

The continuous evolution of the system is given by a trajectory which is a solution of a system of differential equations given a continuously evolving input trajectory.

Definition. A *trajectory* is a continuous function $\tau : [0,t] \to \mathbb{R}^n$, for some $n \in \mathbb{N}$ and $t \in \mathbb{R}_{\geq 0}$. We denote the set of all trajectories with range $\mathbb{R}^d$ by $Traj(\mathbb{R}^d)$.

A trajectory $\tau' : [0,t'] \to \mathbb{R}^n$ is said to be a *prefix* of a trajectory $\tau : [0,t] \to \mathbb{R}^n$ if $t' \leq t$ and $\tau'(t'') = \tau(t'')$ for every $0 \leq t'' \leq t'$.

The variables in $Var = Var^U \cup Var^X$ take values in $\mathbb{R}$. Hence, we define the input domain U to be $\mathbb{R}^k$, and the state domain X to be $\mathbb{R}^n$. An element $\bar{u} \in U$ is interpreted as a valuation which assigns $\bar{u}|_i$ to the input variable u_i, and similarly, an element $\bar{x} \in X$ is interpreted as a valuation which assigns $\bar{x}|_i$ to the state variable x_i. Hence, given an expression e, we denote by $[\![e]\!](\bar{u}, \bar{x})$, the result obtained by evaluating the expression after substituting each variable by the corresponding value from $\bar{u}$ or $\bar{x}$. Similarly, given a predicate b, we say that $\bar{x}$ satisfies b, denoted $\bar{x} \models b$, if substituting for the variables in b by the corresponding values in $\bar{x}$, the predicate evaluates to true; otherwise $\bar{x} \not\models b$.

We capture the state evolution in response to a continuous input by an input-state trajectory.

Definition. An *input-state trajectory* over (Var^U, Var^X) is an element $(\mathbf{u}, \mathbf{x})$ of $Traj(U) \times Traj(X)$ such that $dom(\mathbf{u}) = dom(\mathbf{x})$.

An input-state trajectory $(\mathbf{u}', \mathbf{x}')$ is a prefix of an input-state trajectory $(\mathbf{u}, \mathbf{x})$ if $\mathbf{u}'$ is a prefix of $\mathbf{u}$ and $\mathbf{x}'$ is a prefix of $\mathbf{x}$.

Assumption. We make the standard assumptions that the functions defined by the polynomial expressions in the right hand sides of the differential equations satisfy the conditions for existence and uniqueness of solutions of the differential equations (see, for example, [7], for details of such conditions).

An input-state trajectories $(\mathbf{u}, \mathbf{x})$ is *consistent* with a system of differential equations $\dot{x}_1 = e_1, \cdots, \dot{x}_n = e_n$, if $\mathbf{x}$ is a solution starting from $\mathbf{x}(0)$ on $\mathbf{u}$, that is, there exists a T with $dom(\mathbf{u}) = dom(\mathbf{x}) = [0,T]$, and for each $t \in [0,T]$ and $1 \leq i \leq n$,

$$d\mathbf{x}|_i/dt = [\![e_i]\!](\mathbf{u}(t), \mathbf{x}(t)).$$

The derivative of the projection of $\mathbf{x}$ to the i-th component at anytime t is equivalent to the value of the expression e_i evaluated at the values of $\mathbf{x}$ and $\mathbf{u}$ at time t.

The first state of $(\mathbf{u}, \mathbf{x})$, namely, $\mathbf{x}(0)$, is denoted as $first((\mathbf{u}, \mathbf{x}))$, and the last state, namely, $\mathbf{x}(T)$, where $dom(\mathbf{x}) = [0,T]$, as $last((\mathbf{u}, \mathbf{x}))$.

The transitions capture the discrete jumps in the systems due to impulse input.

Definition. A *transition* over (Var^U, Var^X) is an element of $X \times U \times X$.

The only prefix of a transition is the transition itself. As before, we define the first and last state of a transition, namely, $first((\bar{x}_1, \bar{u}, \bar{x}_2)) = \bar{x}_1$ and $last((\bar{x}_1, \bar{u}, \bar{x}_2)) = \bar{x}_2$.

An execution of the hybrid system is a sequence of input-state trajectories and transitions such that the last state of an element in the sequence matches with the starting state of the next element in the sequence.

Definition. An *execution* over (Var^U, Var^X) is a finite sequence σ of input-state trajectories and transitions over

(Var^U, Var^X), such that, for every $i > 0$ in $dom(\sigma)$, $last(\sigma(i-1)) = first(\sigma(i))$. Let *Exec* denote the set of executions over (Var^U, Var^X).

Note that an input-state trajectory or a transition is an execution with domain $\{0\}$. An execution σ' of length n' is a prefix of an execution σ of length n if $n' \leq n$ and $\sigma(i) = \sigma'(i)$ for every $0 \leq i < n'$ and $\sigma'(n')$ is a prefix of $\sigma(n')$.

Again, we use $first(\sigma)$ and $last(\sigma)$ for the first and last states of the execution σ, namely, $first(\sigma(0))$ and $last(\sigma(l))$, where $dom(\sigma) = \{0, \cdots, l\}$, respectively. An execution σ can be interpreted as a pair of sequences by separating the input and the state parts. We represent an execution σ also as a pair (σ_u, σ_x), where

- $dom(\sigma_u) = dom(\sigma_x) = dom(\sigma)$, and
- for each $i \in dom(\sigma)$,
  - if $\sigma(i) = (\mathbf{u}, \mathbf{x}) \in Traj(U) \times Traj(X)$, then $\sigma_u(i) = \mathbf{u}$ and $\sigma_x(i) = \mathbf{x}$; and
  - if $\sigma(i) = (\bar{x}_1, \bar{u}, \bar{x}_2)$, then $\sigma_u(i) = \bar{u}$ and $\sigma_x(i) = (\bar{x}_1, \bar{x}_2)$.

We call σ_u and σ_x, the input and state signal corresponding to σ, respectively.

The semantics of a hybrid system $\mathcal{H}$ over (Var^U, Var^X), denoted $[\![\mathcal{H}]\!]$, is given by a set of executions over (Var^U, Var^X). We define $[\![\mathcal{H}]\!]$ by induction on the structure of $\mathcal{H}$. We also need to define simultaneously the set of *complete* executions, those that execute $\mathcal{H}$ completely, denoted $[\![\mathcal{H}]\!]^c$.

- $[\![\langle \dot{x}_1 = e_1, \cdots, \dot{x}_n = e_n, b\rangle]\!]$ consists of executions which are input-state trajectories $(\mathbf{u}, \mathbf{x})$ consistent with the system of differential equations $\dot{x}_1 = e_1, \cdots, \dot{x}_n = e_n$ such that
  - for every $t \in [0, T]$, $\mathbf{x}(t) \models b$.

  where $dom(\mathbf{u}) = dom(\mathbf{x}) = [0, T]$.

  $[\![\langle \dot{x}_1 = e_1, \cdots, \dot{x}_n = e_n, b\rangle]\!]^c = [\![\langle \dot{x}_1 = e_1, \cdots, \dot{x}_n = e_n, b\rangle]\!]$.

- $[\![[x_1 := e_1, \cdots, x_n := e_n]]\!]$ is the set of transitions $(\bar{x}_1, \bar{u}, \bar{x}_2)$ such that $\bar{x}_2|_i = [\![e_i]\!](\bar{u}, \bar{x}_1)$.

  $[\![[x_1 := e_1, \cdots, x_n := e_n]]\!]^c = [\![[x_1 := e_1, \cdots, x_n := e_n]]\!]$.

- $[\![\mathcal{H}_1; \mathcal{H}_2]\!] = [\![\mathcal{H}_1]\!] \cup \{\sigma \in Exec \,|\, \exists \sigma_1 \in [\![\mathcal{H}_1]\!]^c, \exists \sigma_2 \in [\![\mathcal{H}_2]\!], \sigma = \sigma_1\sigma_2\}$.

  $[\![\mathcal{H}_1; \mathcal{H}_2]\!]^c = \{\sigma \in Exec \,|\, \exists \sigma_1 \in [\![\mathcal{H}_1]\!]^c, \exists \sigma_2 \in [\![\mathcal{H}_2]\!]^c, \sigma = \sigma_1\sigma_2\}$.

- $[\![\mathsf{if}\, b\, \mathsf{then}\, \mathcal{H}_1\, \mathsf{else}\, \mathcal{H}_2]\!] = \{\sigma \,|\, (first(\sigma) \models b \text{ and } \sigma \in [\![\mathcal{H}_1]\!]) \text{ or } (first(\sigma) \not\models b \text{ and } \sigma \in [\![\mathcal{H}_2]\!])\}$.

  $[\![\mathsf{if}\, b\, \mathsf{then}\, \mathcal{H}_1\, \mathsf{else}\, \mathcal{H}_2]\!]^c = \{\sigma \,|\, (first(\sigma) \models b \text{ and } \sigma \in [\![\mathcal{H}_1]\!]^c) \text{ or } (first(\sigma) \not\models b \text{ and } \sigma \in [\![\mathcal{H}_2]\!]^c)\}$.

- $[\![\mathsf{while}\, b\, \mathsf{do}\, \mathcal{H}]\!] = \{\sigma \in Exec \,|\, \exists \sigma_0, \sigma_1, \cdots, \sigma_l, \sigma = \sigma_0 \sigma_1 \cdots \sigma_l, \forall 0 \leq i \leq l, first(\sigma_i) \models b, \forall 0 \leq i < l, \sigma_i \in [\![\mathcal{H}]\!]^c, \sigma_l \in [\![\mathcal{H}]\!]\}$.

  $[\![\mathsf{while}\, b\, \mathsf{do}\, \mathcal{H}]\!]^c = \{\sigma \in Exec \,|\, \exists \sigma_0, \sigma_1, \cdots, \sigma_l, \sigma = \sigma_0 \sigma_1 \cdots \sigma_l, \forall 0 \leq i \leq l, first(\sigma_i) \models b, \forall 0 \leq i \leq l, \sigma_i \in [\![\mathcal{H}]\!]^c, last(\sigma_l) \not\models b\}$.

Remark. Note that the set of complete executions associated with a hybrid system is a subset of the set of all executions associated with it, that is, $[\![\mathcal{H}]\!]^c \subseteq [\![\mathcal{H}]\!]$.

Remark. The set of executions $[\![\mathcal{H}]\!]$ is prefix-closed, that is, if $\sigma \in [\![\mathcal{H}]\!]$ and σ' is a prefix of σ, then $\sigma' \in [\![\mathcal{H}]\!]$.

3.3 Determinism

A set of executions is deterministic if corresponding to every input signal and initial state, there is at most one state signal.

Definition. A set of executions $E \subseteq Exec$ is said to be *deterministic* if for any pair of executions $\sigma = (\sigma_u, \sigma_x)$ and $\sigma' = (\sigma'_u, \sigma'_x)$ with $\sigma_u = \sigma'_u$ and $first(\sigma) = first(\sigma')$, $\sigma_x = \sigma'_x$.

The next proposition states that the semantics of a hybrid system is deterministic.

PROPOSITION 1. *Given a hybrid system $\mathcal{H}$ over (Var^X, Var^U), the set of executions $[\![\mathcal{H}]\!]$ is deterministic.*

Note that given an input trajectory and an initial state, the state trajectory if it exists corresponds to the unique solution guaranteed by the conditions of existence and uniqueness on the system of differential equations. The discrete jumps are correspond to a function from the initial state and input to a final state (as opposed to a relation), and hence is deterministic. The rest are standard constructs of a programming language which do not introduce any non-determinism. Details of the proof can be found in the Appendix.

Given an input signal and initial state, there might not always exist a corresponding state signal. For instance, for the flow construct, the solution of the differential equations corresponding to an input signal and an initial state may not satisfy the invariant. The other reason why a state signal might not exists is if the input is not of the expected type, that is, a discrete transition expects an impulse input and a continuous execution an input trajectory. Hence, we define the notion of a valid input for a given state.

Definition. An input signal σ_u is said to be *valid* for $\mathcal{H}$ and an initial state $\bar{x}_0$, if there exists an execution (σ_u, σ_x) in $\mathcal{H}$ with $first(\sigma_x) = \bar{x}_0$. In this case, we will denote the unique state signal corresponding to an input signal σ_u and an initial state $\bar{x}_0$ as $\Phi_{\mathcal{H}}(\sigma_u, \bar{x}_0)$.

When $\mathcal{H}$ is clear from the context, we drop the subscript $\mathcal{H}$ in $\Phi_{\mathcal{H}}$.

4. NON-INTERFERENCE

In this section, we present a notion of non-interference [4] for hybrid systems. There are various formalizations of non-interference on different system models; in the context of program semantics one typically requires that the values of high security variables do not affect those of low security variables [12, 8].

As is common, we consider a simple scenario with only two security levels H and L, for high and low security data, respectively. Non-interference requires that, for any two input signals that agree on their low security components, the corresponding state signals, if any, also agree on the low components. Hence, the effects of changes in the high security components is not observable.

More formally, let Var^U_L and Var^U_H be a partition of Var^U into low security and high security input variables, and similarly, Var^X_L and Var^X_H a partition of Var^X into low and high security state variables. The variable τ will be use to range over the security types, namely, H and L, and we assume the ordering $L < H$. We use $\bar{x}|_\tau$ and $\bar{u}|_\tau$ to represent the projection of the state $\bar{x}$ and the input $\bar{u}$ to the components corresponding to the variables with security type τ or lower, respectively; and similarly, we use $\sigma_u|_\tau$ and $\sigma_x|_\tau$ to denote the projection of the input signal and the state signal to the variables with security type τ or lower, respectively.

Definition. A hybrid system $\mathcal{H}$ over $(Var_L^U \cup Var_H^U, Var_L^X \cup Var_H^X)$ satisfies *non-interference* if for every pair of initial states $\bar{x}_0$ and $\bar{x}_0'$ and every pair of input signals σ_u and σ_u' such that $\bar{x}_0|_L = \bar{x}_0'|_L$ and $\sigma_u|_L = \sigma_u'|_L$, the following hold:

- σ_u is valid for $\bar{x}_0$ if and only if σ_u' is valid for $\bar{x}_0'$, and
- If σ_u is valid for $\bar{x}_0$ and σ_u' is valid for $\bar{x}_0'$, then

$$\Phi(\sigma_u, \bar{x}_0)|_L = \Phi(\sigma_u', \bar{x}_0')|_L.$$

We assume that the output variables are the same as the state variables, however, not all the state variables are observable, only the low security state variables are public.

Remark. A well-studied problem in control theory is observability, wherein, the question is to estimate the current state of the system by observing its output. Non-interference, on the other hand, refers to the problem of inferring the high input or the state by observing the low input or the state.

5. A SOLUTION BASED ON TYPE SYSTEMS

In this section, we present a security type system for statically reasoning about information flow in hybrid systems. The type system is sound, i.e., it guarantees that every well-typed hybrid system satisfies non-interference. Our presentation of the type system closely follows that of [12], but is extended with typing rules for the continuous specification of the system. The typing rules are given in Figures 1, 2, and 3 and are explained below.

$$\text{(Int)} \qquad \lambda \vdash n : \tau \quad n = 0, 1$$

$$\text{(Var)} \qquad \lambda \vdash x : \tau\ var \quad \lambda(x) = \tau$$

$$\text{(R-Var)} \qquad \frac{\lambda \vdash x : \tau\ var}{\lambda \vdash x : \tau}$$

$$\text{(Op)} \qquad \frac{\begin{array}{l}\lambda \vdash e_1 : \tau \\ \lambda \vdash e_2 : \tau\end{array}}{\lambda \vdash e_1 \circ e_2 : \tau} \qquad \circ \text{ binary operator}$$

Figure 1: Typing rules for expressions

A type system is used to derive judgments of the form $\lambda \vdash p : \rho$. Here, λ is an environment that assigns a security type H or L to each variable, p is either an expression, a variable, or a hybrid system, and ρ is either τ (corresponding to an expression), $\tau\ var$ (corresponding to a variable), or $\tau\ cmd$ (corresponding to a hybrid system).

The statements without a horizontal line are axioms. Statements with a horizontal line are inference rules, with the premise above and the conclusion below the line. A typing judgment is inferred from the type system, if it is the last statement in a sequence of statements such that each statement is either an axiom, or is inferred by an inference rule of the type system from previous statements in the sequence.

$$\text{(Jump)} \qquad \frac{\begin{array}{ll}\lambda \vdash x_i : \tau_i\ var & \text{for } 1 \le i \le n \\ \lambda \vdash e_i : \tau_i & \text{for } 1 \le i \le n \\ \tau \le \tau_i & \text{for } 1 \le i \le n\end{array}}{\lambda \vdash [x_1 := e_1, \cdots, x_n := e_n] : \tau\ cmd}$$

$$\text{(Flow)} \qquad \frac{\begin{array}{ll}\lambda \vdash x_i : \tau_i\ var & \text{for } 1 \le i \le n \\ \lambda \vdash e_i : \tau_i & \text{for } 1 \le i \le n \\ \lambda \vdash b : \tau & \\ \tau \le \tau_i & \text{for } 1 \le i \le n\end{array}}{\lambda \vdash \langle \dot{x}_1 := e_1, \cdots, \dot{x}_n := e_n, b \rangle : \tau\ cmd}$$

$$\text{(Comp)} \qquad \frac{\begin{array}{l}\lambda \vdash \mathcal{H}_1 : \tau\ cmd \\ \lambda \vdash \mathcal{H}_2 : \tau\ cmd\end{array}}{\lambda \vdash \mathcal{H}_1; \mathcal{H}_2 : \tau\ cmd}$$

$$\text{(If)} \qquad \frac{\begin{array}{l}\lambda \vdash b : \tau \\ \lambda \vdash \mathcal{H}_1 : \tau\ cmd \\ \lambda \vdash \mathcal{H}_2 : \tau\ cmd\end{array}}{\lambda \vdash \mathsf{if}\, b\, \mathsf{then}\, \mathcal{H}_1\, \mathsf{else}\, \mathcal{H}_2 : \tau\ cmd}$$

$$\text{(While)} \qquad \frac{\begin{array}{l}\lambda \vdash b : \tau \\ \lambda \vdash \mathcal{H} : \tau\ cmd\end{array}}{\lambda \vdash \mathsf{while}\, b\, \mathsf{do}\, \mathcal{H} : \tau\ cmd}$$

Figure 2: Typing rules for commands

$$\text{(Subtype-1)} \qquad \frac{\begin{array}{l}\lambda \vdash p : \tau\ cmd \\ \tau' \le \tau\end{array}}{\lambda \vdash p : \tau'\ cmd}$$

$$\text{(Subtype-2)} \qquad \frac{\begin{array}{l}\lambda \vdash p : \tau \\ \tau \le \tau'\end{array}}{\lambda \vdash p : \tau'}$$

Figure 3: Subtyping rules

We say that a hybrid system $\mathcal{H}$ is *well-typed* according to a variable typing λ if there exists a τ such that $\lambda \vdash \mathcal{H} : \tau\ cmd$ can be inferred.

A judgment with security type τ for an expression e implies that the information contained in e is of type τ or lower. A type $\tau\ cmd$ for a hybrid system $\mathcal{H}$ indicates that every variable which $\mathcal{H}$ assigns to has type τ or greater.

This information is used to prevent explicit and implicit flows of information. *Explicit* flows are prevented, e.g., by the typing rules for (Jump) in Figure 2: the rule requires equality between the security types of the variable on the left and the expression on the right, preventing the explicit flow of high data into a low variable. Note however that the rule (Subtype-2) in Figure 3 allows to change the security type of right-hand expression to a higher level, thereby allowing flow from low to high. *Implicit* flows are prevented, e.g., by the typing rule (If) in Figure 2: the rule again requires equality between the security types of the guard and that of the branches, which prevents typing of programs of the kind

$$\mathsf{if}\, h = 1\, \mathsf{then}\, [l := 1]\, \mathsf{else}\, [l := 0] \ ,$$

in which the control flow (rather than an assignment) leaks information from high to low. The rule (Subtype-1) in Figure 3 allows to use programs with assignments to high variables as branches under a low guard (but not vice versa). Note that we do not need the (Subtype-1) rule in our setting, since all the state variables are present in the (Jump) and (Flow) equations, which in turn sets the type of every command to be the lowest type of the state variables. However, in a more general setting, where the jump and flow equations do not contain all the state variables, one will need subtyping.

The remainder of the typing rules are standard, with the exception of the rule (Flow) in Figure 2 that deals with the continuous behavior of the hybrid system. Again, non-interference is enforced in a system of differential equations by local static syntax checks on each of the differential equations (as if they were assignments).

The following theorem shows that a successful syntactic type check implies the semantic notion of non-interference.

THEOREM 1. *Let $\mathcal{H}$ be a hybrid system over the variables $(Var_L^U \cup Var_H^U,\ Var_L^X \cup Var_H^X)$. Let λ be a mapping that assigns L to every variable in $Var_L^U \cup Var_L^X$, and H to the remaining variables. Then a successful type check $\lambda \vdash \mathcal{H}\colon \rho$ implies that $\mathcal{H}$ satisfies non-interference.*

PROOF. We will prove the following claims:

1. If $\lambda \vdash e\colon \tau$ can be inferred, then for any two states $\bar{x}_1, \bar{x}_2 \in X$ and any two inputs $\bar{u}_1, \bar{u}_2 \in U$ such that $\bar{x}_1|_\tau = \bar{x}_2|_\tau$ and $\bar{u}_1|_\tau = \bar{u}_2|_\tau$, then $[\![e]\!](\bar{u}_1, \bar{x}_1) = [\![e]\!](\bar{x}_2, \bar{u}_2)$.

2. If $\lambda \vdash \mathcal{H}\colon \rho$, then for every pair of initial states $\bar{x}_0$ and $\bar{x}'_0$ and every pair of input signals σ_u and σ'_u such that $\bar{x}_0|_L = \bar{x}'_0|_L$ and $\sigma_u|_L = \sigma'_u|_L$, the following hold:

   (a) σ_u is valid for $\bar{x}_0$ if and only if σ'_u is valid for $\bar{x}'_0$, and

   (b) if σ_u is valid for $\bar{x}_0$ and σ'_u is valid for $\bar{x}'_0$, then $\Phi(\sigma_u, \bar{x}_0)|_L = \Phi(\sigma'_u, \bar{x}'_0)|_L$.

   (c) if σ_u is valid for $\bar{x}_0$ and σ'_u is valid for $\bar{x}'_0$, then $(\sigma_u, \Phi(\sigma_u, \bar{x}_0))$ is a complete execution of $\mathcal{H}$ if and only if $(\sigma'_u, \Phi(\sigma'_u, \bar{x}'_0))$ is a complete execution of $\mathcal{H}$.

Proof by induction on the structure of the inference. More precisely, it suffices to show that each of the inference rules preserves the claims. First we show that the first claim holds for the inference rules corresponding to the expressions.

Case (Int): Here $e = n$, where $n = 0$ or 1. Since $[\![e]\!](\bar{u}, \bar{x}) = n$ for every $\bar{u}$ and $\bar{x}$, the claim is trivially satisfied.

Case (R-Var): Here $e = x$, where x is a variable. Note that $\lambda \vdash x\colon \tau\ var$ implies that $\lambda(x) = \tau$. Again, the claim is trivially satisfied.

Case (Op): Let $\bar{x}_1, \bar{x}_2 \in X$ be two states and $\bar{u}_1, \bar{u}_2 \in U$ be two inputs such that $\bar{x}_1|_\tau = \bar{x}_2|_\tau$ and $\bar{u}_1|_\tau = \bar{u}_2|_\tau$. Then $[\![e_1 \circ e_2]\!](\bar{u}_1, \bar{x}_1) = [\![e_1]\!](\bar{u}_1, \bar{x}_1) \circ [\![e_2]\!](\bar{u}_1, \bar{x}_1) = [\![e_1]\!](\bar{u}_2, \bar{x}_2) \circ [\![e_2]\!](\bar{u}_2, \bar{x}_2)$ (induction hypothesis) $= [\![e_1 \circ e_2]\!](\bar{u}_2, \bar{x}_2)$.

Case (Subtype-1): Let $\bar{x}_1, \bar{x}_2 \in X$ be two states and $\bar{u}_1, \bar{u}_2 \in U$ be two inputs such that $\bar{x}_1|_{\tau'} = \bar{x}_2|_{\tau'}$ and $\bar{u}_1|_{\tau'} = \bar{u}_2|_{\tau'}$. Then, $\bar{x}_1|_\tau = \bar{x}_2|_\tau$ and $\bar{u}_1|_\tau = \bar{u}_2|_\tau$, since $\tau \leq \tau'$ implies that the variables with type τ is a subset of variables of type τ'. Therefore, $[\![e]\!](\bar{u}_1, \bar{x}_1) = [\![e]\!](\bar{x}_2, \bar{u}_2)$.

Next, we show that the second claim holds for the inference rules corresponding to the hybrid system. Let $\bar{x}_0$ and $\bar{x}'_0$ be states and σ_u and σ'_u be a pair of input signals that are valid for $\bar{x}_0$ and $\bar{x}'_0$, respectively, such that $\bar{x}_0|_L = \bar{x}'_0|_L$ and $\sigma_u|_L = \sigma'_u|_L$.

Case (Jump): The inputs signals corresponding to a jump are just elements of the input. The claim 2(a) is trivially true since every input for a discrete transition is valid for every state since $[\![e_i]\!]$ is a function. For claim 2(b), we need to show that for each variable x_i such that $\lambda(x_i) = L$, $[\![e_i]\!](\sigma_u, \bar{x}_0) = [\![e_i]\!](\sigma'_u, \bar{x}'_0)$. Since τ_i is L in this case, the hypothesis for the first part of the claim corresponding to e_i holds and hence $[\![e_i]\!](\sigma_u, \bar{x}_0) = [\![e_i]\!](\sigma'_u, \bar{x}'_0)$.

Case (Flow): First, we show that the solutions of the differential equations on $(\sigma_u, \bar{x}_0)$ and $(\sigma'_u, \bar{x}'_0)$ agree on the low variables. Let us denote by $\hat{\Phi}(\sigma_u, \bar{x}_0)$, the solution of the differential equations $\dot{x_i} = e_i$, $1 \leq i \leq n$. ($\hat{\Phi}$ differs from Φ in that it does not need to satisfy the invariant b).

For each $x_i \in Var_L^X$, $\tau_i = L$. Hence, e_i is independent of the variables in Var_H^X and Var_H^U. Hence, the set of differential equations corresponding to Var_L^X form a system of differential equations with Var_L^U as the input variables. Let us call the system S. Hence, $\hat{\Phi}(\sigma_u, \bar{x}_0)|_L$ (with initial state $\bar{x}_0|_L$) and $\sigma_u|_L$, and $\hat{\Phi}(\sigma'_u, \bar{x}'_0)|_L$ (with initial state $\bar{x}'_0|_L$) and $\sigma'_u|_L$, also satisfy system S. Since $\bar{x}_0|_L = \bar{x}'_0|_L$ and $\sigma_u|_L = \sigma'_u|_L$, and the solutions of the differential equations are unique (due to standard conditions of existence and uniqueness of differential equations), we obtain that $\hat{\Phi}(\sigma_u, \bar{x}_0)|_L = \hat{\Phi}(\sigma'_u, \bar{x}'_0)|_L$.

Suppose σ_u is valid for $\bar{x}_0$. Then $\hat{\Phi}(\sigma_u, \bar{x}_0)$ satisfies the invariant b. Since b has type $\tau \leq \tau_i$ for every i, b is of type low. Since $\hat{\Phi}(\sigma_u, \bar{x}_0)|_L = \hat{\Phi}(\sigma'_u, \bar{x}'_0)|_L$ and b is of type low, $\hat{\Phi}(\sigma'_u, \bar{x}'_0)$ also satisfies b and hence is valid. Therefore σ'_u is valid for $\bar{x}'_0$. The other direction is similar.

If σ_u is valid for $\bar{x}_0$ and σ'_u is valid for $\bar{x}'_0$, then $\hat{\Phi}(\sigma_u, \bar{x}_0)|_L = \hat{\Phi}(\sigma'_u, \bar{x}'_0)|_L$ implies that $\Phi(\sigma_u, \bar{x}_0)|_L = \Phi(\sigma'_u, \bar{x}'_0)|_L$.s

Case (Comp): Suppose σ_u is valid for $\bar{x}_0$ in $\mathcal{H}$. There exists an execution $\sigma = (\sigma_u, \sigma_x) \in Exec(\mathcal{H})$. Then there exist executions σ^f and σ^s such that $\sigma = \sigma^f \sigma^s$, where σ^f is a complete execution or σ^s is empty. We can apply the induction hypothesis on $\mathcal{H}_1$ to infer that the prefix σ'_u corresponding to σ_u^f, namely, $\sigma_u^{f'}$ is valid as well, that is, there exists an execution $\sigma'_f = (\sigma_u^{f'}, \sigma_x^{f'})$ of $\mathcal{H}_1$. Further, either σ^f and $\sigma^{f'}$ are both complete or are both not complete. Also, the lower part of last states of the executions coincide. Hence, we can apply the induction hypothesis again on the remaining input signal $\sigma^{s'}$ to obtain an execution for $\mathcal{H}_2$. Concatenating the two gives an execution for $\mathcal{H}$ which is valid for σ'_u and $\bar{x}'_0$. Further, the executions corresponding to both σ_u and σ'_u are either both complete or incomplete follows directly from the induction hypotheses corresponding to Claim 2(c) for $\mathcal{H}_1$ and $\mathcal{H}_2$. Finally, $\Phi(\sigma_u, \bar{x}_0)|_L = \Phi_{\mathcal{H}_2}(\sigma_u^s, (\Phi_{\mathcal{H}_1}(\sigma_u^f, \bar{x}_0)))|_L = \Phi_{\mathcal{H}_2}(\sigma_u^{s'}, (\Phi_{\mathcal{H}_1}(\sigma_u^{f'}, \bar{x}'_0)))|_L = \Phi(\sigma'_u, \bar{x}'_0)|_L$.

Case (If): If σ_u is valid for $\bar{x}_0$, then either $b \models \bar{x}_0$ and σ_u is valid for $\mathcal{H}_1$ or $b \not\models \bar{x}_0$ and σ_u is valid for $\mathcal{H}_2$. First, we note that if there is at least one low variable in the system, then τcmd is always L (one can prove this by induction). Assuming $Var_L^X \neq \emptyset$ (otherwise there is nothing to prove), we infer that $\tau = L$. Since $\bar{x}_0|_L = \bar{x}'_0|_L$, and the type of b is L, $b \models \bar{x}'_0$ if and only if $b \models \bar{x}_0$. We obtain from induction hypothesis of $\mathcal{H}_1$ and $\mathcal{H}_2$ that either $b \models \bar{x}'_0$ and σ'_u is valid for $\mathcal{H}_1$ or $b \not\models \bar{x}'_0$ and σ'_u is valid for $\mathcal{H}_2$. There fore σ'_u is valid for $\mathcal{H}$ with $\bar{x}'_0$ as the initial state. Also, note that state signals agree on the low variable values and on the completeness of the execution.

Case (While): The argument is similar to the case for (Comp). Here an execution is split into a finite number of sub-executions into two. If an input signal σ_u is valid for $\bar{x}_0$, then it can be split into sub-signals corresponding to the body of the while loop, namely, $\mathcal{H}$. Then induction hypothesis can be applied to each of these parts to obtain a valid execution for each of the sub-signals of σ'_u. These can then be concatenated to obtain an execution to the while loop, using the fact that the sub-executions agree on the low variable values and b only depends on the low variable values. $\square$

Remark. The above theorem establishes the soundness of the type inference system. In fact, we can also show that every hybrid system, in which the expressions on the right hand sides have variables whose type is not higher than the type of the variable on the left, and the variables in the guards (of the if and while statements) is low, has a type inference using the type system.

Discussions.

Though the rules in the type system appear similar to that in [12], there are some fundamental differences between their interpretation in a hybrid system versus that in a discrete program. The variable assignments to all the variables happen concurrently in the hybrid system, and so does the evolution of the continuous variables. One can safely interpret a variable update as a sequence of updates to a fresh set of variables followed by a copy-back to the original variables.

However, such a sequential interpretation is not possible for the continuous evolution. In particular, one not only needs to ensure that in each of the individual differential equations $\dot{x}_i = e_i$, there is no unauthorized flow from the variables in e_i to x_i, but also needs to recursively ensure that there is no indirect path from a higher typed variable to x_i in the set of differential equations. However, it turns out that this is equivalent to ensuring that each individual differential equation is well-typed. More precisely, the typing rules impose a structure on the system of differential equations which decompose the system into two subsystems: (1) a subsystem whose statespace corresponds to only the low state variables, and inputs given by only low input variables; and (2) a subsystem whose statespace corresponds to the high state variables with inputs consisting of all the input variables and also the low state variables. It is clear that the solutions projected to the low state variables in such a system depend only the low variable values. For example, for a linear system $\dot{x} = Ax + Bu$, this implies that A and B have upper block triangular structure (for appropriate ordering of the state and input variables).

Example. Consider the simple hybrid system:

$$\mathcal{H} := \textsf{if}\, u > 0\, \textsf{then}\, \langle \dot{x} = 1, x > 3 \rangle\, \textsf{else}\, \langle \dot{x} = 2, x < 0 \rangle,$$

where u is a high input variable and x is a low state variable. There is no direct flow of information from from u to x, but there is an implicit flow due to the condition $u > 0$. That is, by observing x, one can deduce whether $u > 0$ or not. We will not be able to deduce $\gamma \vdash \mathcal{H} : \tau\ cmd$ for any τ.

Now suppose u is a low input variable and x is a high state variable. The following is a partial derivation of a judgment for $\mathcal{H}$.

1. $\lambda \vdash \langle \dot{x} = 1, x > 3 \rangle : H\ cmd$
2. $\lambda \vdash \langle \dot{x} = 2, x < 0 \rangle : H\ cmd$
3. $\lambda \vdash u > 0 : L$
4. $L \leq H$
5. (Subtype -2) $\lambda \vdash u > 0 : H$ (using 3 and 4)
6. (If) $\lambda \vdash \textsf{if}\, u > 0\, \textsf{then}\, \langle \dot{x} = 1, x > 3 \rangle\, \textsf{else}\, \langle \dot{x} = 2, x < 0 \rangle : H\ cmd$ (using 1, 2 and 5)

Here, the derivations corresponding to 1, 2 and 3 are not given.

6. CONCLUSIONS AND FUTURE DIRECTIONS

We formalized a notion of non-interference for hybrid systems and proposed a type system-based static analysis for verifying non-interference. Non-interference is often considered to be too restrictive for practical scenarios. In the context of discrete programs, this restriction is typically circumvented by declassification [9] (which allows to specify which parts of the secret may be leaked) or by quantification [1] (which allows to specify bounds on the amount of leaked information). As ongoing work, we are investigating formalizations and analysis techniques for such relaxed notions in the context of hybrid systems.

Moreover, we will investigate hybrid systems counterpart to the interpretation of information-flow properties as a safety property over pairs of program runs [11]. Progress along those lines will allows us to use existing safety analysis tools for reasoning about information-flow properties in hybrid systems [3].

7. ACKNOWLEDGEMENTS

The authors would like to thank Richard Murray for useful inputs.

8. REFERENCES

[1] M. Backes, B. Köpf, and A. Rybalchenko. Automatic Discovery and Quantification of Information Leaks. In *Proc. 30th IEEE Symposium on Security and Privacy (SSP '09)*, pages 141–153. IEEE, 2009.

[2] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry. Attacks against process control systems: risk assessment, detection, and response. In B. S. N. Cheung, L. C. K. Hui, R. S. Sandhu, and D. S. Wong, editors, *ASIACCS*, pages 355–366. ACM, 2011.

[3] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In G. Gopalakrishnan and S. Qadeer, editors, *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 379–395. Springer, 2011.

[4] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[5] G. S. Graham and P. J. Denning. Protection: principles and practice. In *Proceedings of the May 16-18, 1972, spring joint computer conference*, AFIPS '72 (Spring), pages 417–429, New York, NY, USA, 1972. ACM.

[6] T. A. Henzinger. The theory of hybrid automata. In *LICS*, pages 278–292, 1996.

[7] H. K. Khalil. *Nonlinear Systems*. Prentice-Hall, Upper Saddle River, NJ, 1996.

[8] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21:2003, 2003.

[9] A. Sabelfeld and D. Sands. Dimensions and Principles of Declassification. In *Proc. IEEE Workshop on Computer Security Foundations (CSFW '05)*, pages 255–269. IEEE Computer Society, 2005.

[10] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, Nov. 1984.

[11] T. Terauchi and A. Aiken. Secure information flow as a safety problem. In *Proceedings of the 12th international conference on Static Analysis*, SAS'05, pages 352–367, Berlin, Heidelberg, 2005. Springer-Verlag.

[12] D. M. Volpano, C. E. Irvine, and G. Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996.

APPENDIX

Proof of determinism of the hybrid system.

Proof of Proposition 1. We will prove the following statements by induction on $\mathcal{H}$.

- $[\![\mathcal{H}]\!]$ is deterministic.
- Given any execution $\sigma = \sigma_0\sigma_1 \cdots \sigma_l$, there exists at most one i such that $\sigma[0, i] = \sigma_0\sigma_1 \cdots \sigma_i$ is a complete execution of $\mathcal{H}$, that is, $\sigma[0, i] \in [\![\mathcal{H}]\!]^c$.

Proof by induction on the structure of $\mathcal{H}$.

- $\mathcal{H} = \langle \dot{x}_1 = e_1, \cdots, \dot{x}_n = e_n, b\rangle$: The continuous dynamics is deterministic by the assumptions of existence and uniqueness of the solutions of the differential equations.

  Further, given any execution $\sigma = \sigma_0\sigma_1 \cdots \sigma_l$, if there exists an i such that $\sigma[0, i] \in [\![\mathcal{H}]\!]^c$, then $i = 0$. This is obvious from the definition of $[\![\mathcal{H}]\!]^c$.
- $\mathcal{H} = [x_1 := e_1, \cdots, x_n := e_n]$: Since $[\![e_i]\!]$ is a function mapping the input $\bar{u}$ and the initial state $\bar{x}_1$ to a unique state $\bar{x}_2$, $[\![\mathcal{H}]\!]$ is deterministic. Again, any complete execution of $\mathcal{H}$ has exactly one element.
- $\mathcal{H} = \mathcal{H}_1; \mathcal{H}_2$: By induction hypothesis, we have that both $\mathcal{H}_1$ and $\mathcal{H}_2$ are deterministic. Suppose $\mathcal{H}$ is not deterministic. Then there exist executions $\sigma = (\sigma_u, \sigma_x)$ and $\sigma' = (\sigma'_u, \sigma'_x)$ with $\sigma_u = \sigma'_u$ and $\mathit{first}(\sigma) = \mathit{first}(\sigma')$, and $\sigma_x \neq \sigma'_x$. Then there exist i and i' such that $\sigma[0, i] \in [\![\mathcal{H}_1]\!]^c$ and $\sigma'[0, i'] \in [\![\mathcal{H}_1]\!]^c$. Note that the length of σ and σ' are the same. Let us say that the length is l.

  We claim that $i = i'$. Otherwise, w.l.o.g, we can assume that $i < i'$. Since $[\![\mathcal{H}_1]\!]$ is prefix closed $\sigma'[0, i] \in [\![\mathcal{H}_1]\!]$. If $\sigma'[0, i] \neq \sigma[0, i]$, then it contradicts the determinism of $\mathcal{H}_1$. If $\sigma'[0, i] = \sigma[0, i]$, then $\sigma'[0, i]$ is complete, and it contradicts the second part of the induction hypothesis. Therefore $i = i'$. Further $\sigma[0, i] = \sigma'[0, i]$, otherwise, the determinism of $\mathcal{H}_1$ is contradicted. Note that since $\sigma[0, i] = \sigma'[0, i]$, the initial states of $\sigma[i, l]$ and $\sigma'[i, l]$ are the same. And the inputs are the same. Hence, determinism of $\mathcal{H}_2$ implies that $\sigma_x[i, l] = \sigma'_x[i, l]$. This contradicts our assumption that $\sigma_x \neq \sigma'_x$.

  The second part of the claim follows directly from the fact that if there exist two distinct prefixes of σ corresponding to $\mathcal{H}$, then there exists two distinct prefixes corresponding to either $\mathcal{H}_1$ or $\mathcal{H}_2$ contradicting the induction hypothesis.
- $\mathcal{H} = \mathsf{if}\, b\, \mathsf{then}\, \mathcal{H}_1\, \mathsf{else}\, \mathcal{H}_2$: Given σ and σ' be executions of $\mathcal{H}$ with the same initial states and input signal, then both the execution belong to $\mathcal{H}_1$ or to $\mathcal{H}_2$ depending on whether the initial states satisfy b or not, respectively. Hence, the determinism of $\mathcal{H}_1$ and $\mathcal{H}_2$ imply that $\sigma_x = \sigma'_x$.

  Again, if the second part of the claim does not hold for $\mathcal{H}$, then it does not hold for either $\mathcal{H}_1$ or $\mathcal{H}_2$.
- $\mathcal{H} = \mathsf{while}\, b\, \mathsf{do}\, \mathcal{H}^s$: Suppose there exist σ and σ' in $[\![\mathcal{H}]\!]$ with the same initial states and input signal. Suppose $\sigma = \sigma_0 \cdots \sigma_l$ and $\sigma' = \sigma'_0 \cdots \sigma'_{l'}$, where each $\sigma_i \in [\![\mathcal{H}^s]\!]$ and $\sigma'_j \in [\![\mathcal{H}^s]\!]$. We can show using the arguments similar to the case where $\mathcal{H} = \mathcal{H}_1; \mathcal{H}_2$ repeatedly, that $l = l'$ and the length of σ_i is the same as that of σ'_i for every i. Then, the determinism of $\mathcal{H}^s$ implies that $\sigma_i = \sigma'_i$ for each i.

  For the second part of the claim, assume that $\sigma[0, i]$ and $\sigma[0, i']$ are two prefixes of σ which belong to $[\![\mathcal{H}]\!]$. Then $\mathit{last}\sigma[0, i] \not\models b$. However, since $\sigma[0, i']$ contains a sequence of complete executions of $\mathcal{H}^s$ and this execution sequence coincides with that in the part $\sigma[0, i]$, hence $\sigma[i, i']$ also contains a sequence of complete executions of $\mathcal{H}^s$. Further, since all the executions in the sequence satisfy b at the initial states, $\mathit{first}\sigma[i, i'] \models b$. This is contradictions since $\mathit{last}\sigma[0, i] = \mathit{first}\sigma[i, i']$.