A Semantic Framework for Direct Information Flows in Hybrid-Dynamic Systems

Sepehr Amir-Mohammadian
University of the Pacific
Stockton, California
samirmohammadian@pacific.edu

ABSTRACT

Hybrid-dynamic models provide an underlying framework to study the evergrowing cyber-physical systems with an emphasis on the integration of their discrete computational steps and the associated continuous physical dynamics. Ubiquity of cyber-physical systems necessitates some level of assurance about the secure flow of information through different discrete and continuous components. In recent years, different logical frameworks have been proposed to analyze indirect information flows in cyber-physical systems. While these frameworks are used to verify secure flow of information in a metalevel, they naturally fall short in support of implementing information flow analyzers that could effectively enforce policies at runtime. This practical limitation has triggered the implementation of direct information flow analyzers in different language settings. In this paper, we focus on direct flows of information confidentiality in hybrid-dynamic environments and propose a semantic framework through which we can judge about such flows. This semantic framework can be used to study the correctness of enforced policies by these analyzers, and in particular taint tracking tools. In this regard, we specify a dynamic taint tracking policy for hybrid dynamic systems and prove its soundness based on the proposed semantic framework. As a case study, we consider the flow of information in a public transportation control system, and the effectiveness of our enforced policy on this system.

KEYWORDS

Cyber-physical systems, information flows, semantics, security

ACM Reference Format:
Sepehr Amir-Mohammadian. 2018. A Semantic Framework for Direct Information Flows in Hybrid-Dynamic Systems. In *CPSS 2021: 7th ACM Cyber-Physical System Security Workshop. June 7th, 2021. Hong Kong, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In today's world, smartening mundane electro-mechanical machinery with computational capabilities has broaden their applicability and improved their services. These systems are known as hybrid or cyber-physical systems (CPSs). Enhanced usability of CPSs has made them ubiquitous. Examples abound, e.g., in medical devices, home and office appliances, transportation systems, smart grids, etc. This is specially important for critical scenarios, where the safety and security of users and/or assets play significant roles, and necessitate some level of correct behavior assurance and resistance against malicious operations.

In order to study the correctness of CPSs, different formal models have been proposed, using automata theory [3, 21], process algebras [12, 19, 25–27], and hybrid-dynamic models [6, 33, 34]. The latter approach to formalization is through programming languages techniques, where the CPS behavior is specified in terms of a hybrid program (HP). Hybridity of an HP refers to the coexistence of the discrete nature of computation and the continuous physical dynamics of a CPS. HPs provide a core programming language for CPSs through which the syntax and semantics of integrated discrete and continuous dynamics is specified. Secure behavior of an HP relies on protecting the flow of information while HP executes. Information flow is already studied in terms of CPS formalizations other than hybrid-dynamic models, e.g., [20, 28, 47]. These studies model physical aspects of CPSs discretely, and thus do not reflect on the true nature of these systems wrt physical continuously-changing quantities at runtime which may be visible to low-level users (attackers). Therefore, in this work we focus on hybrid-dynamic models of CPSs.

In the realm of discrete programs, the semantics of information flow analysis has been well-studied. General flows of information capture both direct and indirect flows. Indirect flows refer to the implicit flows of information through the control structure of the program, e.g., using conditional branches. Direct flows concentrate only on explicit information flows, through assignments and parameter passing for example, and ignore the implicit ones.

It has been shown that the policies associated with the direct and indirect flows of information confidentiality do not induce comparable properties, i.e., neither subsumes the other [37, 38]. There are both programs with direct leakage of secret data to public domain while considered secure based on indirect information flow policies, and programs without such direct leakages that are identified as insecure wrt the same indirect information flow policies. The direct flow of data confidentiality is characterized by a property called explicit secrecy [38]. This property is a semantic framework that is used to specify the notion of correctness for different taint trackers associated with sequential discrete programs.

While there is a rich body of research on the analysis of information flows in discrete programs, less work has been done to study HPs in this respect. A recent contribution in this realm is by Bohrer et al. [7], which provides a logical framework to study indirect flows in hybrid-dynamic systems. However, implementation of indirect

CPSS 2021, June 7th, 2021, Hong Kong, China

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

flow analysis remains challenging in different language settings [36]. This is due to the fact that general flows of information are typically characterized by different flavors of noninterference and/or nondeducibility hyperproperties [11], and thus are not enforceable on single traces of execution. Therefore for practical reasons, we are interested in information flow policies that are restricted to direct flows only. For example, taint tracking has been appealingly effective in identifying direct flows, and consequently incorporated in several programming languages, e.g., in C/C++ [39, 43], Java [5, 23], Javascript [41, 48], and WebAssembly [18, 45].

In this paper, we aim to study the semantics of direct information flows for HPs that underlie CPSs. The proposed semantic framework would then facilitate the study of taint tracking policies for CPSs.

1.1 Contributions

Our contribution is three-fold: 1) First, we demonstrate a semantic framework for direct information flows, general enough to support discrete programs as well as HPs. Earlier work on explicit secrecy restrict the model to structural operational semantics of sequential discrete programs, which is specified formally for deterministic behavior of these programs. HPs cannot be specified neither operationally, nor deterministically due to their hybrid-dynamic nature and the support for continuous programs (reviewed in Section 3). Alternatively, our proposed framework supports denotational and nondeterministic semantics to capture direct flows of information, which is compatible with the nature of HPs' runtime model. 2) Moreover, we specify a sample taint tracking policy for hybrid-dynamic systems that is being enforced at runtime. We prove that the policy is correct wrt our proposed semantic framework for direct information flows, i.e., the enforced dynamic taint tracking policy avoids direct leakage of information from secret domain to public, as long as it satisfies the proposed semantic property. 3) Finally, as an illustrative example, we explore direct information flows in a public transportation control system using our semantic framework, and study whether the enforcement of the sample taint tracking policy protects this system. This enables us to identify the information flows through discrete vs. continuous components of the system at runtime.

1.2 Threat Model

Our specification of semantic framework is defined wrt data confidentiality. We assume that the attacker is able to observe the runtime behavior of the HP, and has access to memory regions corresponding to low confidentiality, but does not have direct access to high confidentiality parts of memory. We also assume that low confidentiality users provide low confidentiality data. We only consider taint for data and data containers and not for code that could be executed as part of the main program. Since this work is focusing on direct information flows, there is an implicit assumption that indirect flows are not being exploited. Moreover, program modification and tracking data trustworthiness are out of scope of this work, as they align with data integrity rather than confidentiality.

1.3 A Motivating Example

In this section, a motivating example is given that illustrates the necessity of studying direct flows in hybrid-dynamic environments. We will revisit this example later in the paper (in Sections 4 and 5) to explain the effectiveness of our formal framework. For this purpose, consider the European Train Control System (ETCS) [13], which can be specified abstractly as a hybrid program [32, 33]. ETCS is a system to manage train commutes. It aims to use the underlying transportation infrastructure more efficiently by sharing the tracks among the commuting trains. This necessitates that the trains do not clash. To this end, a train needs to request and receive permissions for specific blocks of the tracks from certain agents called Radio Block Controllers (RBCs). RBCs grant a block to a train according to the current state of the requested track, the requesting train, as well as the state of other traveling trains. A train needs to negotiate with RBCs for a block extension before reaching to the end of the permitted block. To avoid clashes, a train starts to decrease its speed within a certain distance from that endpoint, while attempting to negotiate for extensions. In the extreme case, the train may stop and wait for the extension to be granted. ETCS can be abstracted as a hybrid-dynamic system with discrete and continuous components. Discrete dynamics refer to controlling operations, e.g., commanding the train to change the acceleration, or initializing the negotiation phase with an RBC. Continuous dynamics refer to the physical behavior of the train specified in terms of its position, speed, acceleration, etc. A low-confidentiality user of this system can compute these physical parameters in realtime, e.g., through third-party tracking devices. Therefore, it is reasonable to assume these parameters to be public. On the other hand, let's assume that controlling parameters are potentially secret information, e.g., as part of protecting the intellectual property. In this paper, we are proposing a framework by which we can study whether such secret information are directly being leaked to the public domain, and whether a direct information flow analyzer prevents such flows.

1.4 Paper Outline

The rest of the paper is organized as follows. In Section 2, we propose the semantic framework for direct information flows that can potentially support HPs. In Section 3, we review the syntax and semantics of HPs, and then instantiate the semantic framework for HPs. Section 4 provides an illustrative example of applying the proposed semantic framework on ETCS transportation system to identify insecure direct information flows. In Section 5, we specify a typical dynamic tainting policy for HPs, and prove its soundness based on our semantic framework. Section 6 reviews the related work, and Section 7 concludes the paper and specifies the potential future work.

2 A NONDETERMINISTIC & DENOTATIONAL MODEL OF DIRECT FLOWS

In this section, we define the semantics of direct flow of information confidentiality, inspired by Schoepe et al. [38]. Our formulation is different in two respects: 1) It is supporting nondeterminism, and 2) it is specified denotationally rather than operationally. These features enable us to discuss direct information flows in a more general context, where the flows can be tracked in discrete programs as well as a combination of discrete and continuous programs, i.e., in hybrid-dynamic systems. Naturally, flow of information integrity can be defined as a dual of the specified semantic framework, and

thus we restrict our specification to flow of information confidentiality.

Let $\mathcal{V}$ be the countably infinite set of variables, and lower-case alphabetic characters (e.g., $x, h, l \in \mathcal{V}$) range over them. Let $\mathcal{S}$ denote the set of states, and $\omega \in \mathcal{S}$ range over states. A state ω is a mapping from $\mathcal{V}$ to values. We deliberately consider the set of values under-specified at this stage. In Section 3.1 we instantiate it with a concrete set for HPs. Let O be the set of observable data, and $\sigma \in O^*$ range over sequence of observable data including empty sequence ε. Observable data denote the information that may be leaked to a user with a certain security clearance.

We posit $\mathfrak{p}$ to range over programs, and its denotation to be defined as a relation $[\![\mathfrak{p}]\!]$ over $\mathcal{S} \times \mathcal{S}$ and O^* (potentially among other sets). That is, a program is interpreted as a relation specifying the reachability between two states, along with the generated observable data. We define $[\![\mathfrak{p}]\!]_\omega$ as the set of pairs (ω', σ) where ω' is reachable by running $\mathfrak{p}$ from ω, and σ is observed.

State transformer is a function that is defined for each program describing how that program changes the state as well as what observable data are generated. In what follows, we deliberately under-specify state transformers using the denotation of programs. In Section 3.2, we instantiate this definition for HPs.

Definition 2.1 (State Transformer). State transformer of program $\mathfrak{p}$ in state ω_0 is some function $f_{\mathfrak{p},\omega_0} : \mathcal{S} \to \mathcal{P}(\mathcal{S} \times O^*)$ that specifies how a state is changed by $\mathfrak{p}$ if it was ω_0. It also specifies the observables if $\mathfrak{p}$ is executed.

We may drop program and state annotations from state transformers for the sake of brevity. Let f and g range over them. In the following we define the composition of state transformers. This notion is employed to define the semantics of sequence of HPs in Section 3.2.

Definition 2.2 (Composition of two state transformers). Composition of two state transformers f and g, $g \odot f$, is defined as $(g \odot f)(\omega) = \{(\omega', \sigma) \mid \exists \omega_0, \sigma_0, \sigma_1.(\omega_0, \sigma_0) \in f(\omega), (\omega', \sigma_1) \in g(\omega_0), \sigma = \sigma_0\sigma_1\}$.

Let $(\mathcal{L}, \preceq, \sqcup)$ be the lattice of security levels including at least two levels L and H denoting public and secret levels, i.e., L $\preceq$ H. These two levels are the infimum and supremum of $\mathcal{L}$, resp. $\sqcup$ is the join operator on $\mathcal{L}$. Assume that $lev : \mathcal{V} \to \mathcal{L}$ is the function that returns the security level of a variable. Two states are low-equivalent if they map public variables to identical values. Throughout the paper, we assume that l and h are public and secret variables, resp., i.e., $lev(l) = \text{L}$ and $lev(h) = \text{H}$.

Definition 2.3 (Low equivalence). Two states ω and ω' are low equivalent, denoted by $\omega =_\text{L} \omega'$, iff for any x, $lev(x) = \text{L}$ implies $\omega(x) = \omega'(x)$.

Let $\mathcal{S}_{init}(\mathfrak{p}) \subseteq \mathcal{S}$ be the set of all states that can be considered as initial states for program $\mathfrak{p}$, i.e., $\mathcal{S}_{init}(\mathfrak{p})$ is the set of valid states from which $\mathfrak{p}$ executes. This set can be defined differently for different language settings. Let π_i be the projection function on set of tuples, returning the set of ith elements of each tuple.

Explicit knowledge is the set of initial states configurable by a low level user (attacker) that generate a particular sequence of observables following the execution that a given state transformer specifies and according to a given initial state of a program. In other words, explicit knowledge is the set of all imaginable initial states by an attacker after observing the execution starting from a given initial state. The smaller this set is, the greater is the knowledge of the attacker.

Definition 2.4 (Explicit Knowledge). Explicit knowledge wrt program $\mathfrak{p}$, initial state ω and state transformer f is defined as

$$k_e(\mathfrak{p}, \omega, f) = \{\omega' \mid \omega =_\text{L} \omega', \omega' \in \mathcal{S}_{init}(\mathfrak{p}), \pi_2(f(\omega)) = \pi_2(f(\omega'))\}.$$

A program enjoys explicit secrecy if the execution specified by its state transformer does not affect the knowledge of the attacker, i.e., all low-equivalent states are still conceivable after running the program. By quantifying on all possible initial states, indirect information flows are ignored. Example 3.2 shows the effect of this quantification after instantiating the framework for HPs.

Definition 2.5 (Explicit Secrecy). Program $\mathfrak{p}$ satisfies explicit secrecy in state ω_0, denoted by $ES \vDash \mathfrak{p}, \omega_0$, iff for any $\omega \in \mathcal{S}_{init}(\mathfrak{p})$, $k_e(\mathfrak{p}, \omega, f_{\mathfrak{p},\omega_0}) = k_e(\mathfrak{p}, \omega, g)$, where g is defined as $g : \hat{\omega} \mapsto \{(\hat{\omega}, \varepsilon)\}$.

Program $\mathfrak{p}$ satisfies explicit secrecy, denoted by $ES \vDash \mathfrak{p}$, iff for any arbitrary state ω_0, we have $ES \vDash \mathfrak{p}, \omega_0$.

Explicit secrecy is the semantic framework which paves the way to study trackers of direct information flows and in particular, dynamic taint trackers. In this regard, let $\tau : \mathcal{V} \to \mathcal{L}$ be the dynamic taint function that returns the security level of a variable in a given state. Let the set of all dynamic taint functions be $\mathcal{T}$. We extend the denotation of program $\mathfrak{p}$ to $[\![\mathfrak{p}]\!]_T \subseteq \mathcal{S} \times \mathcal{T} \times \mathcal{S} \times \mathcal{T} \times O^*$ to capture a dynamic tainting policy, i.e., the relation $(\omega_1, \tau_1, \omega_2, \tau_2, \sigma) \in [\![\mathfrak{p}]\!]_T$ specifies that if $\mathfrak{p}$ is executed in state ω_1, and τ_1 stores the taint of variables in that state, then the resulting state would be ω_2, and the taint of variables would change to τ_2. Moreover, σ would be observed through the execution of $\mathfrak{p}$.

Using the notion of explicit secrecy, the soundness of the dynamic tainting policy $[\![\mathfrak{p}]\!]_T$ can be defined. Intuitively, a dynamic tainting policy is sound iff that tainting policy allows execution, only if explicit secrecy is satisfied.

Definition 2.6 (Soundness of Dynamic Tainting). Let $\mathfrak{p}$ be a program. Dynamic tainting policy $[\![\mathfrak{p}]\!]_T$ is sound iff for any state $\omega \in \mathcal{S}_{init}(\mathfrak{p})$ and any initial dynamic taint function τ_{init}, the existence of some ω', τ, and σ such that $(\omega, \tau_{init}, \omega', \tau, \sigma) \in [\![\mathfrak{p}]\!]_T$ implies $ES \vDash \mathfrak{p}, \omega$.

In other words, soundness of $[\![\mathfrak{p}]\!]_T$ means that if $\mathfrak{p}$ does not satisfy explicit secrecy in a given initial state ω and dynamic taint function τ_{init}, then $\mathfrak{p}$ is not allowed to run.

Similar to Schoepe et. al [38], we may relax the model in support of information declassification, using gradual release [4]. Let $\mathcal{R}$ be the set of release events, i.e., the events whose observance do not affect attacker knowledge. In other words, $\mathcal{R}$ is the set of declassified information, and thus their leakage to the attacker is not considered as security breaches. Moreover, let's denote the ith observable in sequence σ with $\sigma^{(i)}$.

Definition 2.7 (Explicit Secrecy Modulo Release). Program $\mathfrak{p}$ satisfies explicit secrecy modulo release iff for any states ω_0, ω and ω', and observables σ, if $\omega \in \mathcal{S}_{init}(\mathfrak{p})$, $(\omega', \sigma) \in f_{\mathfrak{p},\omega_0}(\omega)$, and for all i we have $\sigma^{(i)} \notin \mathcal{R}$, then $ES \vDash \mathfrak{p}, \omega_0$.

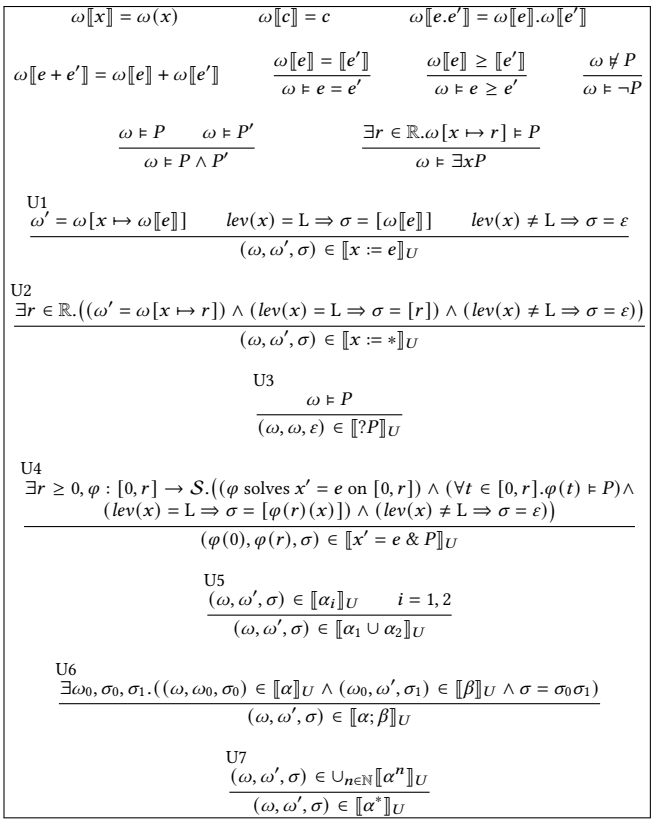

Figure 1: Semantics of 1) polynomial terms: $\omega[\![e]\!]$, 2) FOL of real arithmetic: $\omega \vDash P$, and 3) HPs: $[\![\alpha]\!]_U$.

3 EXPLICIT SECRECY IN HYBRID-DYNAMIC SYSTEMS

In this section, we first review the syntax and semantics of HPs. Next, we instantiate the semantic framework (described in Section 2) for HPs.

3.1 Syntax and Semantics

The syntax and semantics of HPs [33, 35] relies on real arithmetic polynomial terms and first-order logic (FOL) of real arithmetic.

3.1.1 *Polynomial terms.*

Syntax. Polynomial terms e with rational coefficients are defined as $e ::= x \mid c \mid e.e \mid e + e$, where $x \in \mathcal{V}$ and $c \in \mathbb{Q}$.

Semantics. Let $\mathcal{S}$ be the set of states defined as maps from variables to real numbers, i.e., $\omega : \mathcal{V} \to \mathbb{R}$. The semantics of a polynomial term e is defined given a state ω in Figure 1, denoted by $\omega[\![e]\!]$.

3.1.2 *FOL of real arithmetic.*

Syntax. FOL of real arithmetic is defined syntactically by

$$P ::= e = e \mid e \geq e \mid \neg P \mid P \wedge P \mid \exists x P.$$

Other syntactic structures including true, false, disjunction, implication, bidirectional implication, universal quantification, less than, greater than, etc. can be defined using this minimal syntax.

Semantics. State ω models predicate P, denoted by $\omega \vDash P$, according to the definition in Figure 1. Satisfiability and validity can be defined straightforwardly, using $\omega \vDash P$ relation. We define $[\![P]\!] = \{\omega \mid \omega \vDash P\}$.

3.1.3 *Hybrid Programs.*

Syntax. Hybrid programs are defined syntactically as follows:

$$\alpha ::= x := e \mid x := * \mid ?P \mid x' = e \,\&\, P \mid \alpha \cup \alpha \mid \alpha; \alpha \mid \alpha^*.$$

$x := e$ updates x to the value of e. $x := *$ updates x to a nondeterministic value. $?P$ is a test (with boolean result). $x' = e \,\&\, P$ is a continuous program specified in terms of an ordinary differential equation $x' = e$ along with an evolution domain P. x' denotes the time derivative of x. Indeed, continuous programs are in explicit form, i.e., derivatives do not occur in e and P. To simplify the specification of continuous programs, we consider only polynomial differential equations, as syntactically given in $x' = e$. Moreover, we implicitly assume that in general, a continuous program may consist of a vector of differential equations of the form $x_1' = e_1, x_2' = e_2, \cdots, x_n' = e_n \,\&\, P$. We use $\alpha \cup \alpha$ for the nondeterministic choice between two HPs. $\alpha; \alpha$ is a sequence of two HPs. Finally, α^* is the iteration of HP α for nondeterministic number of times.

Semantics. Let lattice of security levels $\mathcal{L}$ to include at least two levels L and H, and *lev* be instantiated for HPs, i.e., for any variable x, $lev(x)$ is defined. $[\![\alpha]\!]_U \subseteq \mathcal{S} \times \mathcal{S} \times \mathbb{R}^*$ is the denotation of α, defined in Figure 1. Note that in the style of Volpano's weak secrecy [46], in case a public variable is updated, that event is visible to the low-confidentiality user in our system. More specifically, $x := e$, $x := *$, and $x' = e \,\&\, P$ generate nontrivial observables if $lev(x) = \mathrm{L}$.

According to rule U1, $x := e$ updates the state ω, where x is mapped to the value of e in ω. Moreover, if x is public, the value is observed by the low-level user. In rule U2, ω is updated with x being mapped to a nondeterministic real value. Similar to U1 if x is public, the assigned value becomes observable to the low-level user. $?P$ is only defined for states that model P, without any change to the state (rule U3). According to rule U4, $x' = e \,\&\, P$ is runnable if there exists a solution for the equation $x' = e$ in the domain P, i.e., for some nondeterministic continuous time span $[0, r]$, value of x gets updated according to the equation $x' = e$. In addition, in this time span, every updated state must satisfy the domain condition P. Moreover, observables to lower-level users are available in case x is a public variable. In rule U5, $\alpha_1 \cup \alpha_2$ nondeterministically chooses α_1 or α_2 to run. Rule U6 states that $\alpha; \beta$ is executed by first running α and then β. Finally, according to rule U7, α^* iteratively runs α zero or more times, where the number of iterations is nondeterministic. Note that α^n denotes n iterations of α, defined as $\alpha^0 = ?\top$ and $\alpha^{n+1} = \alpha^n; \alpha$.

3.2 Instantiating Explicit Secrecy for HPs

The proposed semantic framework for direct information flows in Section 2 can be straightforwardly instantiated for HPs: Let the set of observables O be instantiated with the set of real numbers $\mathbb{R}$.

$$\mathcal{S}_{init}(x := e) = \mathcal{S} \qquad \mathcal{S}_{init}(x := *) = \mathcal{S} \qquad \mathcal{S}_{init}(?P) = \{\omega \mid \omega \vDash P\} = [\![P]\!]$$

$$\mathcal{S}_{init}(x' = e \;\&\; P) = [\![P]\!] \qquad \mathcal{S}_{init}(\alpha \cup \beta) = \mathcal{S}_{init}(\alpha) \cup \mathcal{S}_{init}(\beta)$$

$$\mathcal{S}_{init}(\alpha;\beta) = \{\omega \mid \omega \in \mathcal{S}_{init}(\alpha), \exists \omega_0, \sigma_0 \,.\, (\omega, \omega_0, \sigma_0) \in [\![\alpha]\!]_U \wedge \omega_0 \in \mathcal{S}_{init}(\beta)\}$$

$$\mathcal{S}_{init}(\alpha^*) = \mathcal{S}$$

Figure 2: Definition of initial states for HP α.

The set of variables $\mathcal{V}$ and states $\mathcal{S}$ are defined accordingly for HP, where the set of all values is $\mathbb{R}$. Programs $\mathfrak{p}$ are instantiated with HPs whose syntax and semantics are given in the previous section.

We let $\mathcal{S}_{init}(\alpha)$ be defined as the set of states that do not discard the execution of α, i.e., $\mathcal{S}_{init}(\alpha) = \{\omega \mid \exists \omega', \sigma.(\omega, \omega', \sigma) \in [\![\alpha]\!]_U\}$. Accordingly, for each HP α, $\mathcal{S}_{init}(\alpha)$ is defined in Figure 2.

In the following example, we review the initial states for a few HPs, which will be used to review the relation between noninterference and explicit secrecy later in this section.

Example 3.1. Consider the following HPs:

$$\alpha_0 \equiv ?l = 0; l := l + h,$$
$$\alpha_1 \equiv (?l = 0; l := l + h) \cup (?l \neq 0; l := l + 1),$$
$$\alpha_2 \equiv (?h = 0; l := 1) \cup (?h \neq 0; l := 2),$$
$$\alpha_3 \equiv (?h = 0; l := h) \cup (?h \neq 0; l := 0).$$

Then, $\mathcal{S}_{init}(\alpha_0) = [\![l = 0]\!] = \{\omega \mid \omega(l) = 0\}$. We have $\mathcal{S}_{init}(\alpha_1) = [\![l = 0]\!] \cup [\![l \neq 0]\!] = \mathcal{S}$, i.e., all states are acceptable as initial states. Similarly, $\mathcal{S}_{init}(\alpha_2) = \mathcal{S}_{init}(\alpha_3) = \mathcal{S}$.

The lattice of security levels $\mathcal{L}$, as well as *lev* are also considered for HPs. Then, according to Definition 2.1, f_{α,ω_0} is defined using $[\![\alpha]\!]_{\omega_0}$. This gives us the definitions of state transformers for HPs defined in Figure 3. Note that the state transformer of $x := e$, $x := *$, and $x' = e \;\&\; P$ return nontrivial observables in case x is public. In addition, the state transformers of $?P$ and $x' = e \;\&\; P$ in state ω_0 check the satisfiability P in ω_0. In case P is not satisfiable in ω_0, the result is empty.

Using Definition 2.2, the composition of two state transformers of two HPs is instantiated. Low equivalence of two HP states is instantiated according to function *lev*, as described in Definition 2.3. Low equivalence relation facilitates the instantiation of explicit knowledge according to Definition 2.4, which in turn is used to instantiate explicit secrecy (Definition 2.5). For explicit secrecy modulo release (Definition 2.7), any set in one-to-one correspondence to a finite subset of real numbers would suffice as the the set of release events.

In the following example, we review the state transformers and explicit knowledges of the HPs given in Example 3.1. Next, we review whether they satisfy explicit secrecy. This example demonstrates the incomparability of noninterference and explicit secrecy in the context of hybrid-dynamic systems (α_2 and α_3, in particular). As an introductory example, none of the HPs include physical dynamics. In this regard, this example demonstrate the support of our framework for discrete programs. The reader is referred to Section 4 for an illustrative example, where the HP combines discrete and physical dynamics.

$$f_{x:=e,\omega_0}(\omega) = \begin{cases} \{(\omega[x \mapsto \omega[\![e]\!]], [\omega[\![e]\!]])\} & \text{if } lev(x) = \mathrm{L} \\ \{(\omega[x \mapsto \omega[\![e]\!]], \varepsilon)\} & \text{otherwise.} \end{cases}$$

$$f_{x:=*,\omega_0}(\omega) = \begin{cases} \{(\omega[x \mapsto r], [r]) \mid r \in \mathbb{R}\} & \text{if } lev(x) = \mathrm{L} \\ \{(\omega[x \mapsto r], \varepsilon) \mid r \in \mathbb{R}\} & \text{otherwise.} \end{cases}$$

$$f_{?P,\omega_0}(\omega) = \begin{cases} \{(\omega, \varepsilon)\} & \text{if } \omega_0 \vDash P \\ \emptyset & \text{otherwise.} \end{cases} \qquad f_{x'=e \;\&\; P,\omega_0}(\omega) =$$

$$\begin{cases} \{(\varphi(r), [\varphi(r)(x)]) \mid \exists r \geq 0. \exists \varphi : [0,r] \to \mathcal{S}. \\ \varphi \text{ solves } x' = e \text{ on } [0,r], \varphi(0) = \omega, \\ \forall t \in [0,r].\varphi(t) \vDash P\} & \text{if } \omega_0 \vDash P \text{ and } lev(x) = \mathrm{L} \\ \{(\varphi(r), \varepsilon) \mid \exists r \geq 0. \exists \varphi : [0,r] \to \mathcal{S}. \\ \varphi \text{ solves } x' = e \text{ on } [0,r], \varphi(0) = \omega, \\ \forall t \in [0,r].\varphi(t) \vDash P\} & \text{if } \omega_0 \vDash P \text{ and } lev(x) \neq \mathrm{L} \\ \emptyset & \text{otherwise.} \end{cases}$$

$$f_{\alpha\cup\beta,\omega_0}(\omega) = f_{\alpha,\omega_0}(\omega) \cup f_{\beta,\omega_0}(\omega)$$

$$f_{\alpha;\beta,\omega_0}(\omega) = \cup_{\omega_1:(\omega_0,\omega_1,\_)\in[\![\alpha]\!]_U} (f_{\beta,\omega_1} \odot f_{\alpha,\omega_0})(\omega)$$

$$f_{\alpha^*,\omega_0}(\omega) = \cup_{n\in\mathbb{N}} f_{\alpha^n,\omega_0}(\omega)$$

Figure 3: Definition of state transformers for HP α.

Example 3.2. Considering α_0 from Example 3.1 the state transformer is defined as

$$f_{\alpha_0,\omega_0}(\omega) = \cup_{\omega_1:(\omega_0,\omega_1,\_)\in[\![?l=0]\!]_U} (f_{l:=l+h,\omega_1} \odot f_{?l=0,\omega_0})(\omega).$$

Let $\omega_0(l) = 0$. Then, $f_{\alpha_0,\omega_0}(\omega) = (f_{l:=l+h,\omega_0} \odot f_{?l=0,\omega_0})(\omega)$, where

$f_{?l=0,\omega_0}(\omega) = \{(\omega, \varepsilon)\}$, and

$f_{l:=l+h,\omega_0}(\omega) = \{(\omega[l \mapsto \omega(l) + \omega(h)], [\omega(l) + \omega(h)])\}$.

Therefore, $f_{\alpha_0,\omega_0}(\omega) = \{(\omega[l \mapsto \omega(l) + \omega(h)], [\omega(l) + \omega(h)])\}$. Then, the explicit knowledge is defined as

$$\begin{aligned} k_e(\alpha_0, \omega, f_{\alpha_0,\omega_0}) &= \{\omega' \mid \omega =_{\mathrm{L}} \omega', \omega' \in \mathcal{S}_{init}(\alpha_0), \\ &\qquad \pi_2(f_{\alpha_0,\omega_0}(\omega)) = \pi_2(f_{\alpha_0,\omega_0}(\omega'))\} \\ &= \{\omega' \mid \omega =_{\mathrm{L}} \omega', \omega' \in \mathcal{S}_{init}(\alpha_0), \omega(h) = \omega'(h)\} \end{aligned}$$

This entails that $ES \not\vDash \alpha_0, \omega_0$, since ω and ω' may not agree on the value of h. Independent of this result, α_0 is also interfering due to the existence of flow from h to l[1].

Considering α_1 from Example 3.1 the state transformer is defined as

$$\begin{aligned} f_{\alpha_1,\omega_0}(\omega) = &\cup_{\omega_1:(\omega_0,\omega_1,\_)\in[\![?l=0]\!]_U} (f_{l:=l+h,\omega_1} \odot f_{?l=0,\omega_0})(\omega) \\ &\cup_{\omega_1:(\omega_0,\omega_1,\_)\in[\![?l\neq 0]\!]_U} (f_{l:=l+1,\omega_1} \odot f_{?l\neq 0,\omega_0})(\omega). \end{aligned}$$

Let $\omega_0(l) = 0$. Then, $f_{\alpha_1,\omega_0}(\omega) = (f_{l:=l+h,\omega_0} \odot f_{?l=0,\omega_0})(\omega)$, as

$f_{?l=0,\omega_0}(\omega) = \{(\omega, \varepsilon)\}$,

$f_{l:=l+h,\omega_0}(\omega) = \{(\omega[l \mapsto \omega(l) + \omega(h)], [\omega(l) + \omega(h)])\}$, and

$f_{?l\neq 0,\omega_0}(\omega) = \emptyset$.

Therefore, $f_{\alpha_1,\omega_0}(\omega) = \{(\omega[l \mapsto \omega(l)+\omega(h)], [\omega(l)+\omega(h)])\}$. Then explicit knowledge is the same as the one for α_0, and so $ES \not\vDash \alpha_1, \omega_0$.

[1]For the full formalization of indirect flows and the definition of noninterference in the context of hybrid-dynamic systems, the reader is referred to ref. [7]. However, for the sake of comparison in this example, we may define noninterference in a standard way as follows: HP α satisfies noninterference iff $\omega_1 =_{\mathrm{L}} \omega_2$, $(\omega_1, \omega_1', \sigma_1) \in [\![\alpha]\!]_U$, and $(\omega_2, \omega_2', \sigma_2) \in [\![\alpha]\!]_U$ implies $\sigma_1 = \sigma_2$.

Similar to α_0, α_1 is interfering due to the existence of flow from h to l.

Considering α_2 from Example 3.1,

$$f_{\alpha_2,\omega_0}(\omega) = \cup_{\omega_1:(\omega_0,\omega_1,\_)\in[\![?h=0]\!]_U} (f_{l:=1,\omega_1} \odot f_{?h=0,\omega_0})(\omega) \\ \cup_{\omega_1:(\omega_0,\omega_1,\_)\in[\![?h\neq 0]\!]_U} (f_{l:=2,\omega_1} \odot f_{?h\neq 0,\omega_0})(\omega).$$

This can be simplified as

$$f_{\alpha_2,\omega_0}(\omega) = (f_{l:=1,\omega_0} \odot f_{?h=0,\omega_0})(\omega) \cup (f_{l:=2,\omega_0} \odot f_{?h\neq 0,\omega_0})(\omega) \\ = \{(\omega[l \mapsto a], [a])\}.$$

where $a = 1$ if $\omega_0(h) = 0$, and $a = 2$ otherwise. Having explicit knowledge as $k_e(\alpha_2, \omega, f_{\alpha_2,\omega_0}) = \{\omega' \mid \omega =_{\mathrm{L}} \omega', \omega' \in \mathcal{S}_{init}(\alpha_2)\}$ ensures that $ES \vDash \alpha_2, \omega_0$ for any ω_0, and thus $ES \vDash \alpha_2$. This is while α_2 is interfering, since there is indirect flow from h to l. Note that the definition of explicit secrecy (Definition 2.5) quantifies on all initial states ω to satisfy the relation, independent of whether $\omega_0(h) = 0$ or not. This way, indirect flows are successfully ignored.

Considering α_3 from Example 3.1,

$$f_{\alpha_3,\omega_0}(\omega) = \cup_{\omega_1:(\omega_0,\omega_1,\_)\in[\![?h=0]\!]_U} (f_{l:=h,\omega_1} \odot f_{?h=0,\omega_0})(\omega) \\ \cup_{\omega_1:(\omega_0,\omega_1,\_)\in[\![?h\neq 0]\!]_U} (f_{l:=0,\omega_1} \odot f_{?h\neq 0,\omega_0})(\omega).$$

Let $\omega_0(h) = 0$. Then, $f_{\alpha_3,\omega_0}(\omega) = (f_{l:=h,\omega_0} \odot f_{?h=0,\omega_0})(\omega) = \{(\omega[l \mapsto \omega(h)], [\omega(h)])\}$. Having $k_e(\alpha_3, \omega, f_{\alpha_3,\omega_0}) = \{\omega' \mid \omega =_{\mathrm{L}} \omega', \omega' \in \mathcal{S}_{init}(\alpha_3), \omega(h) = \omega'(h)\}$ entails that $ES \nvDash \alpha_3, \omega_0$ as ω and ω' may not agree on h. This is while α_3 is noninterfering.

4 ETCS: AN ILLUSTRATIVE EXAMPLE

In this section, we borrow the specification of the ETCS system from [32, 33] as an HP, which is introduced informally in Example 1.3. We study whether this system satisfies explicit secrecy. Let's denote the position, speed, and acceleration of the train with x, v, and a, resp. Moreover, let's assume that the endpoint position of a track block granted to the train by RBC is m, and the preconfigured maximum distance from position m that necessitates the train to lower its speed is s. Having these notations, ETCS can be defined as $\alpha \equiv (ctrl; drive)^*$, where

$$\begin{aligned} ctrl &\equiv (?m - x < s; a := -b) \cup (?m - x \geq s; a := c) \\ drive &\equiv k := 0; (x' = v, v' = a, k' = 1 \,\&\, v \geq 0 \wedge k \leq d) \end{aligned}$$

ctrl is the digital controller (cyber component) that checks whether the train is within the preconfigured distance s of the end of the block. If so (i.e., $m - x < s$), it applies the brake, by setting the acceleration to the constant negative value $-b$. If the train is not within the distance s from the end of the block ($m - x \geq s$), the controller increases the speed, by setting the acceleration to the constant positive value c.

drive is the plant (physical component) that specifies how the position of the train evolves through time, using the differential equations $x' = v, v' = a$. In addition, *drive* includes a continuous timer k that initially is set to zero and then linearly increases ($k' = 1$). The continuous evolution of the train position, speed, and the timer is constrained by its evolution domain. Evolution domain specifies that 1) train speed cannot be negative, i.e., a train cannot move in the reverse direction, and 2) there is an upper bound on driving duration ($k \leq d$), i.e., for safety purposes, the train cannot drive continuously more than preconfigured d units of time before passing the control to *ctrl*.

One may assume that physical parameters k, x, v, and a are public values, as they can be calculated by any time and movement tracker. This is while, preconfigured controlling parameters b, c, s, and d can be assumed secret. m can also be assumed a non-public value. Having these considerations, let's study whether α satisfies explicit secrecy. According to Figure 2, $\mathcal{S}_{init}(\alpha) = \mathcal{S}$. For the sake of notational brevity, let's consider the following:

$$\begin{aligned} \beta &\equiv ctrl; drive \\ \gamma &\equiv ?m - x < s; a := -b \\ \lambda &\equiv ?m - x \geq s; a := c \\ \theta &\equiv x' = v, v' = a, k' = 1 \,\&\, v \geq 0 \wedge k \leq d \end{aligned}$$

The state transformer is defined as $f_{\alpha,\omega_0}(\omega) = \cup_{n\in\mathbb{N}} f_{\beta^n,\omega_0}(\omega)$. Let's consider the case, where $n = 1$. We have

$$f_{\beta,\omega_0}(\omega) = \cup_{\omega_1:(\omega_0,\omega_1,\_)\in[\![ctrl]\!]_U} (f_{drive,\omega_1} \odot f_{ctrl,\omega_0})(\omega).$$

Let's also assume that $\omega_0 \vDash m - x \geq s$. Then we need to study the state transformer for *ctrl* and *drive*, in ω_0 and ω_1, resp.

The state transformer of *ctrl* in ω_0 is defined as

$$f_{ctrl,\omega_0}(\omega) = f_{\gamma,\omega_0}(\omega) \cup f_{\lambda,\omega_0}(\omega) = f_{\lambda,\omega_0}(\omega) \\ = f_{a:=c,\omega_0} \odot f_{?m-x\geq s,\omega_0}(\omega) = \{(\omega[a \mapsto \omega(c)], [\omega(c)])\}.$$

Note that $lev(a) = \mathrm{L}$, and thus assigning c to a generates observable sequence $[\omega(c)]$.

Next, let's consider the state transformer of *drive* in ω_1.

$$f_{drive,\omega_1}(\omega[a \mapsto \omega(c)]) = \cup_{\omega_2:(\omega_1,\omega_2,\_)\in[\![k:=0]\!]_U} (f_{\theta,\omega_2} \odot \\ f_{k:=0,\omega_1})(\omega[a \mapsto \omega(c)])$$

Considering $f_{k:=0,\omega_1}(\omega[a \mapsto \omega(c)]) = \{(\omega[a \mapsto \omega(c)][k \mapsto 0], [0])\}$, and since $\omega_1[k \mapsto 0] \vDash v \geq 0 \wedge k \leq d$ and $lev(x) = lev(v) = lev(k) = \mathrm{L}$, according to Figure 3 we have

$$f_{\theta,\omega_1[k\mapsto 0]}(\omega[a \mapsto \omega(c)][k \mapsto 0]) = \\ \big\{(\varphi(r), [\varphi(r)(x), \varphi(r)(v), \varphi(r)(k)]) \;\big|\; \exists r \geq 0. \exists \varphi : [0, r] \to \mathcal{S}. \\ \varphi \text{ solves } x' = v, v' = a, k' = 1 \text{ on } [0, r], \\ \varphi(0) = \omega[a \mapsto \omega(c)][k \mapsto 0], \forall t \in [0, r]. \varphi(t) \vDash v \geq 0 \wedge k \leq d\big\}.$$

Indeed, $x' = v, v' = a, k' = 1$ are solvable in a continuous range, i.e., we have state $\varphi(t)$ that maps k to t, maps v to $\omega(c) \cdot t + \omega(v)$, and maps x to $\frac{\omega(c)}{2} t^2 + \omega(v) \cdot t + \omega(x)$. Note that $\omega(v)$ and $\omega(x)$ are the initial values for speed and position when θ starts the execution. Moreover, time is limited by the evolution domain to not exceed d. Therefore, we have

$$f_{\theta,\omega_1[k\mapsto 0]}(\omega[a \mapsto \omega(c)][k \mapsto 0]) = \\ \big\{(\varphi(t), [\varphi(t)(x), \varphi(t)(v), \varphi(t)(k)] \;\big|\; t \in [0, d], \varphi(t)(k) = t, \\ \varphi(t)(x) = \frac{\omega(c)}{2} t^2 + \omega(v) \cdot t + \omega(x), \varphi(t)(v) = \omega(c) \cdot t + \omega(v), \\ \forall y \neq x, v, k. \varphi(t)(y) = \omega[a \mapsto \omega(c)](y)\big\}.$$

Using the results for $f_{k:=0,\omega_1}(\omega[a \mapsto \omega(c)])$ and $f_{\theta,\omega_1[k\mapsto 0]}(\omega[a \mapsto \omega(c)][k \mapsto 0])$, we can simplify the state transformer for *drive* at

state ω_1 as follows.

$$f_{drive,\omega_1}(\omega[a \mapsto \omega(c)]) = \big\{(\varphi(t), [0, \varphi(t)(x), \varphi(t)(v), \varphi(t)(k)] \mid$$
$$t \in [0,d], \varphi(t)(k) = t, \varphi(t)(x) = \frac{\omega(c)}{2}t^2 + \omega(v)\cdot t + \omega(x),$$
$$\varphi(t)(v) = \omega(c)\cdot t + \omega(v),$$
$$\forall y \neq x, v, k.\varphi(t)(y) = \omega[a \mapsto \omega(c)](y)\big\}.$$

Now that we have the state transformers for *ctrl* and *drive* in states ω_0 and ω_1 resp., the state transformer for β in ω_0 is

$$f_{\beta,\omega_0}(\omega) = \big\{(\varphi(t), [\omega(c), 0, \varphi(t)(x), \varphi(t)(v), \varphi(t)(k)] \mid$$
$$t \in [0,d], \varphi(t)(k) = t, \varphi(t)(x) = \frac{\omega(c)}{2}t^2 + \omega(v)\cdot t + \omega(x),$$
$$\varphi(t)(v) = \omega(c)\cdot t + \omega(v),$$
$$\forall y \neq x, v, k.\varphi(t)(y) = \omega[a \mapsto \omega(c)](y)\big\}.$$

Using the state transformer above, we have the following the explicit knowledge for β

$$k_e(\beta, \omega, f_{\beta,\omega_0}) = \{\omega' \mid \omega =_{\mathrm{L}} \omega', \pi_2(f_{\beta,\omega_0}(\omega)) = \pi_2(f_{\beta,\omega_0}(\omega'))\},$$

where

$$\pi_2(f_{\beta,\omega_0}(\omega)) = \{[\omega(c), 0, \frac{\omega(c)}{2}t^2 + \omega(v)\cdot t + \omega(x),$$
$$\omega(c)\cdot t + \omega(v), t] \mid t \in [0,d]\}$$

Since $lev(c) = \mathrm{H}$, it is not necessarily the case that $\omega(c) = \omega'(c)$. This entails that $\pi_2(f_{\beta,\omega_0}(\omega))$ is not equal to $\pi_2(f_{\beta,\omega_0}(\omega'))$ necessarily for arbitrary low equivalent state ω', and thus $ES \not\models \beta, \omega_0$. Intuitively, secrect data c is directly leaking to public domain. Since β is a single iteration of α, it is intuitive that α does not satisfy explicit secrecy either.

We can follow this same sequence of inferences to deduce that parameter b also directly leaks to the public domain[2]. On the other hand, there is not any direct flow from m, d and s to public variables.

5 A DYNAMIC TAINTING POLICY FOR HYBRID-DYNAMIC SYSTEMS

In this section, we extend HPs with a dynamic tainting policy, and prove its soundness (Definition 2.6) based on explicit secrecy. Let $\tau : \mathcal{V} \to \mathcal{L}$ be the instantiation of the dynamic taint function for HPs with its universe denoted by $\mathcal{T}$. We assume that the initial dynamic taint function τ_{init} is defined as *lev*. Then the taint of a polynomial term, denoted by $\tau[\![e]\!]$, can be defined as demonstrated in Figure 4.

We extend the denotation of HP α to be $[\![\alpha]\!]_T \subseteq \mathcal{S}\times\mathcal{T}\times\mathcal{S}\times\mathcal{T}\times O^*$ to capture a dynamic tainting policy, defined in Figure 4. The policy is standard by disallowing direct flows of highly confidential information to low confidential data containers. This is, in particular, reflected by the rules T1 and T4 through security-level comparisons.

According to this tainting policy, $x := e$ is allowed to run if the taint of e is smaller than the taint of x (rule T1), i.e., e conveys less confidential data than the confidentiality level of x. If this precondition is met, the state and the taint function is updated accordingly. Otherwise, the execution is not allowed. This is while, for $x = *$ (rule T2) such precondition is not enforced, since the nondeterministic real value that is assigned to x is a public value according to

[2]We just need to assume that $\omega_0 \models m - x < s$.

$\tau[\![x]\!] = \tau(x) \qquad \tau[\![c]\!] = \mathrm{L} \qquad \tau[\![e.e']\!] = \tau[\![e]\!] \sqcup \tau[\![e']\!] \qquad \tau[\![e{+}e']\!] = \tau[\![e]\!] \sqcup \tau[\![e']\!]$

T1
$$\frac{\tau[\![e]\!] \preceq \tau[\![x]\!] \qquad \omega' = \omega[x \mapsto \omega[\![e]\!]] \qquad \tau' = \tau[x \mapsto \tau[\![e]\!]] \qquad lev(x) = \mathrm{L} \Rightarrow \sigma = [\omega[\![e]\!]] \qquad lev(x) \neq \mathrm{L} \Rightarrow \sigma = \varepsilon}{(\omega, \tau, \omega', \tau', \sigma) \in [\![x := e]\!]_T}$$

T2
$$\frac{\exists r \in \mathbb{R}.(\omega' = \omega[x \mapsto r] \wedge \tau' = \tau[x \mapsto \mathrm{L}] \wedge (lev(x) = \mathrm{L} \Rightarrow \sigma = [r]) \wedge (lev(x) \neq \mathrm{L} \Rightarrow \sigma = \varepsilon))}{(\omega, \tau, \omega', \tau', \sigma) \in [\![x := *]\!]_T}$$

T3
$$\frac{\omega \models P}{(\omega, \tau, \omega, \tau, \varepsilon) \in [\![?P]\!]_T}$$

T4
$$\frac{\exists r \geq 0, \varphi : [0,r] \to \mathcal{S}.((\varphi \text{ solves } x' = e \text{ on } [0,r]) \wedge (\forall t \in [0,r].\varphi(t) \models P) \wedge (\forall t \in [0,r].\tau[\![\varphi(t)(x)]\!] \preceq \tau[\![x]\!]) \wedge (\tau' = \tau[x \mapsto \tau[\![\varphi(r)(x)]\!]]) \wedge (lev(x) = \mathrm{L} \Rightarrow \sigma = [\varphi(r)(x)]) \wedge (lev(x) \neq \mathrm{L} \Rightarrow \sigma = \varepsilon))}{(\varphi(0), \tau, \varphi(r), \tau', \sigma) \in [\![x' = e \,\&\, P]\!]_T}$$

T5
$$\frac{(\omega, \tau, \omega', \tau', \sigma) \in [\![\alpha_i]\!]_T \qquad i = 1,2}{(\omega, \tau, \omega', \tau', \sigma) \in [\![\alpha_1 \cup \alpha_2]\!]_T}$$

T6
$$\frac{\exists \omega_0, \tau_0, \sigma_0, \sigma_1.((\omega, \tau, \omega_0, \tau_0, \sigma_0) \in [\![\alpha]\!]_T \wedge (\omega_0, \tau_0, \omega', \tau', \sigma_1) \in [\![\beta]\!]_T \wedge \sigma = \sigma_0\sigma_1)}{(\omega, \tau, \omega', \tau', \sigma) \in [\![\alpha;\beta]\!]_T}$$

T7
$$\frac{(\omega, \tau, \omega', \tau', \sigma) \in \cup_{n\in\mathbb{N}}[\![\alpha^n]\!]_T}{(\omega, \tau, \omega', \tau', \sigma) \in [\![\alpha^*]\!]_T}$$

Figure 4: Definition of 1) the taint of polynomial terms: $\tau[\![e]\!]$, and 2) semantics of dynamic tainting policy for HPs: $[\![\alpha]\!]_T$.

the specification of the taint for polynomial terms ($\tau[\![c]\!] = \mathrm{L}$). In this case, taint of x is always set to L. Rule T3 allows the execution for $?P$ in a state as long as that state models P. In this case the taint function does not change. Rule T4 checks whether the taint of the value being assigned to x during the nondeterministic time span $[0, r]$ is smaller than the taint of x. The execution is allowed and the taint function is updated accordingly, only if this precondition is met. For $\alpha_1 \cup \alpha_2$, the tainting policy nondeterministically chooses to run either α_1 or α_2 and accordingly the taint function is updated (rule T5). The sequence $\alpha;\beta$ is allowed to run according to rule T6 if α and β are allowed to run individually one after the other. Finally according to rule T7, iteration of α for nondeterministic number of times is allowed if zero or more iterations of α can be executed in sequence.

Example 5.1. As described in Section 4, the ETCS hybrid-dynamic system does not satisfy explicit secrecy due to the existence of direct flows from secret parameters b and c to the public domain. Applying the dynamic tainting policy of Figure 4 prevents the execution of the HP. According to the rule T1, $\tau[\![c]\!] \preceq \tau[\![a]\!]$ and $\tau[\![-b]\!] \preceq \tau[\![a]\!]$ are prerequisites for the execution of $a := c$ and $a := -b$, resp., whereas neither of these prerequisites holds. For the sake of discussion, let's assume that acceleration is also considered secret data. Then, these assignments do not violate the aforementioned tainting policy. However, the execution of $x' = v, v' = a, k' = 1 \,\&\, v \geq 0 \wedge k \leq \epsilon$ in the *drive* component is rejected based on rule T4, since $\tau[\![\varphi(t)(x)]\!] = \tau[\![\varphi(t)(v)]\!] = \mathrm{H}$ as $\tau[\![c]\!] = \mathrm{H}$, and thus $\tau[\![\varphi(t)(x)]\!] \preceq \tau[\![x]\!]$ and $\tau[\![\varphi(t)(v)]\!] \preceq \tau[\![v]\!]$ do not hold.

Using the proposed semantic framework for direct flow of information confidentiality, we can study the soundness of taint tracking

system as a property, where a given HP is successfully executed starting from an initial state, only if that HP satisfies explicit secrecy in that initial state. The proof of the theorem is given in Appendix A.

Theorem 5.2 (Soundness of $[\![\alpha]\!]_T$). *The dynamic tainting policy $[\![\alpha]\!]_T$ of Figure 4 is sound (Definition 2.6).*

Declassification. Indeed the depicted tainting policy of Figure 4 does not support information declassification. For this purpose, Definition 2.7 provides the semantic specification of direct flows in the presence of declassified data. A dynamic tainting policy that supports gradual release [4] may include HPs that enable declassification.

Let HPs be extended with a declassifier: $\alpha ::= \cdots \mid dc(e)$. The semantics of $dc(e)$ can be defined as $(\omega, \omega, rel(\omega[\![e]\!])) \in [\![dc(e)]\!]_U$, where $rel(\cdot)$ is used to mark a released observable, and the set of release events is defined as $\mathcal{R} \subset \{rel(r) \mid r \in \mathbb{R}\}$. The tainting policy allows to declassify data and generates release events accordingly: $(\omega, \tau, \omega, \tau, rel(\omega[\![e]\!])) \in [\![dc(e)]\!]_T$.

6 RELATED WORK

Formal models of CPSs. There are three major approaches to model CPSs in a formal fashion: 1) Hybrid automata [3, 21] are one of the earliest attempts to formally model CPSs, where a finite state transition system describes a hybrid system by capturing discrete (cyber) and continuous (physical) variables in each state. 2) Hybrid process calculi [19, 25–27] model CPSs in terms of agents that represent physical plants and cyber components communicating through named channels. For example, in CCPS [26] a labeled transition system is defined that demonstrates how such agents execute using shared channels. 3) Hybrid-dynamic models [6, 33–35] rely on programming languages techniques, and in particular hybrid programs to specify CPSs. A hybrid program is a sequence of statements, where a statement is either a discrete step of execution (e.g., an assignment, or a branch), or a continuous evolution of dynamic variables in a certain domain.

Indirect information flows in CPSs. Indirect (general) information flow analysis has been explored in CPSs in several lines of work. Akella et al. [2] rely on process algebra to specify a semantic model for information flows in CPSs, including a gas pipeline system. However, their model ignores physical plants of the CPS altogether. Akella et al. [1] have formalized the flow of information confidentiality in FREEDM smart grid [22] using a process algebraic model of the system, describing nondeducibility and noninterference properties. Gamage et al. [20] use a finite state machine to analyze the flow of information in CPSs. Lanotte et al. [28] propose CCPSA, a process calculus that models indirect information flows in CPSs. Castellanos et al. [8] use data flow graphs and propose a reachability algorithm to analyze PLC programs that capture the interaction among different components of a CPS. Morris et al. [31] use an information-theoretic model to quantify the susceptibility of a CPS to attacks that target flow of information integrity. Liu et al. [29] propose an architecture to secure CPSs, which includes input analysis, information flow analysis and hardware verification. All these lines of work, however, treat the physical components of CPSs discretely rather than continuously. Bohrer et al. [7] have addressed this issue by proposing a logic to verify indirect information flows in hybrid programs. Using this logic a variants of nondeducibility property is specified for a smart grid system and information leakage is demonstrated.

Semantics of direct information flows. While all the aforementioned lines of work propose frameworks to verify the general flow of information in CPSs, they do not support any implementation of flow analysis in such systems. In contrast, direct information flow policies can be enforced on single traces of execution. Schoepe et al. [38] have proposed an underlying semantic framework through which direct information flow can be studied in single threaded low-level programs with a single-step operational semantics. Explicit secrecy is defined as a property of a program, where the execution does not change the knowledge of a low confidentiality user. Knowledge [4] is defined as the set of initial states that a low confidentiality user is able to consider to generate a given sequence of observables. Explicit knowledge restricts attacker knowledge for direct confidentiality flows only. Explicit secrecy models direct flow of information confidentiality. This provides a semantic framework by which correctness of different confidentiality taint trackers can be studied. In the same line of work, Skalka et al. [42] have recently proposed the semantic framework for direct flow of information integrity in high-level functional settings, with small-step operational semantics. These formalizations, however, are not applicable to hybrid-dynamic settings, where due to their nature, the semantics is expressible denotationally, mapping an initial state to a set of final states through the execution of an HP.

Taint tracking in CPSs. Both static and dynamic taint tracking have been used to analyze direct flow of information in CPSs. CPAC [14] introduces dynamic taint analysis for programmable logic controllers. MISMO [44] is a reverse engineering framework for CPSs that includes a dynamic taint tracker, where sensor measurements introduce the tainted data, and actuator input functionalities are the taint sinks. LUCON [40] is a policy framework for CPSs that uses dynamic taint analysis to enforce security policies at runtime. It assigns taint to the messages being communicated between different components of the CPS. Klingensmith et al. [24] propose a privacy agent for mobile and IoT devices that embeds a dynamic taint tracker to study the direct flow of data from input channels to the storage APIs. Cherupalli et al. [10] provide direct information flow analysis by tracking the taints at the logical gate level. To this end, they have developed a tool that simulates gate-level behavior of the IoT applications. Recently, Ferrara et al. [16] have extended a Julia-based static taint analyzer [17] for IoT privacy. FlowFence [15] uses sandboxing to identify the flow of sensitive taint-labeled data in and out of different IoT application functions. Saint [9] is a static taint tracker for IoT applications that translates application source code into an intermediate-level code to locate the taint sources and sinks. Another recent work [30] studies static taint analysis in the the context of multiple programming languages being used to deploy a single CPS.

7 CONCLUSION AND FUTURE WORK

In this paper, we have proposed the semantics of direct information flows in CPSs. This framework is a variant of explicit secrecy

that supports nondeterminism and denotational semantics, and thus appropriate for hybrid-dynamic environments in which discrete computational steps (cyber) are combined with continuous dynamics (physical). The proposed semantics provides an underlying framework to study the effectiveness of direct information flow analyzers in hybrid-dynamic systems. This notion is formulated as a soundness property. As an example, we have defined a dynamic confidentiality taint tracking policy for hybrid programs and proven its soundness.

The proposed semantic framework paves the way to study static and dynamic taint trackers that are used for cyber-physical systems, some of which are explored in the related work. As a future work, we are planning to model a subset of these systems in hybrid-dynamic settings and study their effectiveness using our semantic framework.

Along with hybrid programs, there are alternative approaches to specify CPSs. One such alternative is to use process algebraic models. A potential future work is to study direct flow of information in a process algebra that models CPSs.

REFERENCES

[1] Ravi Akella and Bruce M McMillin. 2010. Information flow analysis of energy management in a smart grid. In *International Conference on Computer Safety, Reliability, and Security*. Springer, 263–276.

[2] Ravi Akella, Han Tang, and Bruce M McMillin. 2010. Analysis of information flow security in cyber–physical systems. *International Journal of Critical Infrastructure Protection* 3, 3-4 (2010), 157–173.

[3] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. 1992. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*. Springer, 209–229.

[4] Aslan Askarov and Andrei Sabelfeld. 2007. Gradual Release: Unifying Declassification, Encryption and Key Release Policies. In *IEEE S&P*. 207–221.

[5] Jonathan Bell and Gail Kaiser. 2015. Dynamic taint tracking for java with phosphor. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. 409–413.

[6] Patrick Blackburn and Jerry Seligman. 1995. Hybrid languages. *Journal of Logic, Language and Information* 4, 3 (1995), 251–272.

[7] Brandon Bohrer and André Platzer. 2018. A hybrid, dynamic logic for hybrid-dynamic information flow. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 115–124.

[8] John H Castellanos, Martín Ochoa, and Jianying Zhou. 2018. Finding dependencies between cyber-physical domains for security testing of industrial control systems. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 582–594.

[9] Z Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A Selcuk Uluagac. 2018. Sensitive information tracking in commodity IoT. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1687–1704.

[10] Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. 2017. Software-based gate-level information flow security for IoT systems. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 328–340.

[11] Michael R. Clarkson and Fred B. Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.

[12] Pieter Jan Laurens Cuijpers and Michel A Reniers. 2005. Hybrid process algebra. *The Journal of Logic and Algebraic Programming* 62, 2 (2005), 191–245.

[13] Werner Damm, Alfred Mikschl, Jens Oehlerking, Ernst-Rüdiger Olderog, Jun Pang, André Platzer, Marc Segelken, and Boris Wirtz. 2007. Automating verification of cooperation, control, and design in traffic applications. In *Formal methods and hybrid real-time systems*. Springer, 115–169.

[14] Sriharsha Etigowni, Dave Tian, Grant Hernandez, Saman Zonouz, and Kevin Butler. 2016. CPAC: securing critical infrastructure with cyber-physical access control. In *Proceedings of the 32nd annual conference on computer security applications*. 139–152.

[15] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. 2016. Flowfence: Practical data protection for emerging iot application frameworks. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 531–548.

[16] Pietro Ferrara, Amit Kr Mandal, Agostino Cortesi, and Fausto Spoto. 2020. Static analysis for discovering IoT vulnerabilities. *International Journal on Software Tools for Technology Transfer* (2020), 1–18.

[17] Pietro Ferrara, Luca Olivieri, and Fausto Spoto. 2020. BackFlow: Backward Context-Sensitive Flow Reconstruction of Taint Analysis Results. In *International Conference on Verification, Model Checking, and Abstract Interpretation*. Springer, 23–43.

[18] William Fu, Raymond Lin, and Daniel Inge. 2018. Taintassembly: Taint-based information flow control tracking for webassembly. *arXiv preprint arXiv:1802.01050* (2018).

[19] Vashti Galpin, Luca Bortolussi, and Jane Hillston. 2013. HYPE: Hybrid modelling by composition of flows. *Formal Aspects of Computing* 25, 4 (2013), 503–541.

[20] Thoshitha T Gamage, Bruce M McMillin, and Thomas P Roth. 2010. Enforcing information flow security properties in cyber-physical systems: A generalized framework based on compensation. In *2010 IEEE 34th Annual Computer Software and Applications Conference Workshops*. IEEE, 158–163.

[21] Thomas A Henzinger. 2000. The theory of hybrid automata. In *Verification of digital and hybrid systems*. Springer, 265–292.

[22] Alex Q Huang. 2009. Renewable energy system research and education at the NSF FREEDM systems center. In *2009 IEEE Power & Energy Society General Meeting*. IEEE, 1–6.

[23] Wei Huang, Yao Dong, and Ana Milanova. 2014. Type-based taint analysis for Java web applications. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 140–154.

[24] Neil Klingensmith, Younghyun Kim, and Suman Banerjee. 2019. A Hypervisor-Based Privacy Agent for Mobile and IoT Systems. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. 21–26.

[25] Ivan Lanese, Luca Bedogni, and Marco Di Felice. 2013. Internet of things: a process calculus approach. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. 1339–1346.

[26] Ruggero Lanotte and Massimo Merro. 2017. A calculus of cyber-physical systems. In *International Conference on Language and Automata Theory and Applications*. Springer, 115–127.

[27] Ruggero Lanotte and Massimo Merro. 2018. A semantic theory of the Internet of Things. *Information and Computation* 259 (2018), 72–101.

[28] Ruggero Lanotte, Massimo Merro, Riccardo Muradore, and Luca Viganò. 2017. A formal approach to cyber-physical attacks. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 436–450.

[29] Jed Liu, Joe Corbett-Davies, Andrew Ferraiuolo, Alexander Ivanov, Mulong Luo, G Edward Suh, Andrew C Myers, and Mark Campbell. 2018. Secure autonomous cyber-physical systems through verifiable information flow control. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*. 48–59.

[30] Amit Mandal, Pietro Ferrara, Yuliy Khlyebnikov, Agostino Cortesi, and Fausto Spoto. 2020. Cross-program taint analysis for IoT systems. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. 1944–1952.

[31] Eric Rothstein Morris, Carlos G Murguia, and Martín Ochoa. 2017. Design-time quantification of integrity in cyber-physical systems. In *Proceedings of the 2017 Workshop on Programming Languages and Analysis for Security*. 63–74.

[32] André Platzer. 2007. Differential dynamic logic for verifying parametric hybrid systems. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*. Springer, 216–232.

[33] André Platzer. 2008. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning* 41, 2 (2008), 143–189.

[34] André Platzer. 2012. The complete proof theory of hybrid systems. In *2012 27th Annual IEEE Symposium on Logic in Computer Science*. IEEE, 541–550.

[35] André Platzer. 2018. *Logical foundations of cyber-physical systems*. Vol. 662. Springer.

[36] Andrei Sabelfeld and Andrew C Myers. 2003. Language-based Information-flow Security. *IEEE Journal on selected areas in communications* 21, 1 (2003), 5–19.

[37] Fred B Schneider. 2000. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)* 3, 1 (2000), 30–50.

[38] Daniel Schoepe, Musard Balliu, Benjamin C. Pierce, and Andrei Sabelfeld. 2016. Explicit Secrecy: A Policy for Taint Tracking. In *IEEE EuroS&P*. 15–30.

[39] Philipp Dominik Schubert, Ben Hermann, and Eric Bodden. 2019. PhASAR: An inter-procedural static analysis framework for C/C++. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 393–410.

[40] Julian Schütte and Gerd Stefan Brost. 2018. LUCON: Data flow control for message-based IoT systems. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 289–299.

[41] Koushik Sen, Swaroop Kalasapur, Tasneem Brutch, and Simon Gibbs. 2013. Jalangi: a selective record-replay and dynamic analysis framework for JavaScript. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 488–498.

[42] Christian Skalka, Sepehr Amir-Mohammadian, and Samuel Clark. 2020. Maybe tainted data: Theory and a case study. *Journal of Computer Security* Preprint (2020), 1–41.

[43] Hao Sun, Xiangyu Zhang, Chao Su, and Qingkai Zeng. 2015. Efficient dynamic tracking technique for detecting integer-overflow-to-buffer-overflow vulnerability. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. 483–494.
[44] Pengfei Sun, Luis Garcia, and Saman Zonouz. 2019. Tell Me More Than Just Assembly! Reversing Cyber-Physical Execution Semantics of Embedded IoT Controller Software Binaries. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 349–361.
[45] Aron Szanto, Timothy Tamm, and Artidoro Pagnoni. 2018. Taint tracking for webassembly. *arXiv preprint arXiv:1807.08349* (2018).
[46] Dennis M. Volpano. 1999. Safety versus Secrecy. In *SAS*. 303–311.
[47] Jingming Wang and Huiqun Yu. 2014. Analysis of the composition of non-deducibility in cyber-physical systems. *Applied Mathematics & Information Sciences* 8, 6 (2014), 3137.
[48] Ran Wang, Guangquan Xu, Xianjiao Zeng, Xiaohong Li, and Zhiyong Feng. 2018. TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting. *J. Parallel and Distrib. Comput.* 118 (2018), 100–106.

A PROOFS

Lemma A.1 (Tainting faithfulness). *Let* $(\omega, \tau, \omega', \tau', \sigma) \in [\![\alpha]\!]_T$. *This implies that* $(\omega, \omega', \sigma) \in [\![\alpha]\!]_U$.

Proof. By induction on the structure of α. □

Lemma A.1 implies that we can use state transformers for HPs with the dynamic taint analysis introduced in Section **??**. This notion is implicit the proofs of the following lemmas and theorems.

Definition A.2 (Taint equivalent states). Two states ω and ω' are taint equivalent, $\omega =_\tau \omega'$, iff for any x, if $\tau[\![x]\!] = \mathrm{L}$ then $\omega(x) = \omega'(x)$.

Lemma A.3. *Let* $\tau[\![e]\!] = \mathrm{L}$ *and* $\omega =_\tau \omega'$. *Then,* $\omega[\![e]\!] = \omega'[\![e]\!]$.

Proof. By induction on the structure of e. □

Lemma A.4. *Let* $\hat{\omega}$ *and* $\hat{\omega}'$ *be two low-equivalent states in* $\mathcal{S}_{init}(\alpha)$. *If* $(\omega_0, \sigma) \in f_{\alpha,\omega}(\hat{\omega})$, *then there exists some* ω_0' *such that* $(\omega_0', \sigma) \in f_{\alpha,\omega}(\hat{\omega}')$, *and* $\omega_0 =_{\mathrm{L}} \omega_0'$.

Proof. By induction on the structure of α, and applying Lemma A.3. □

Proof of Theorem 5.2. By induction on the structure of α.

- Let α be $x := e$. From $(\omega, \tau_{init}, \omega', \tau, \sigma) \in [\![x := e]\!]_T$ we infer that $\tau_{init}[\![e]\!] \preceq \tau_{init}[\![x]\!]$. Moreover, if $lev(x) = \mathrm{L}$ then $\sigma = [\omega[\![e]\!]]$, otherwise $\sigma = \varepsilon$.
  - Let $lev(x) \neq \mathrm{L}$. Then the property holds since for any two states $\hat{\omega}$ and $\hat{\omega}'$, we have $\pi_2(f_{x:=e,\omega}(\hat{\omega})) = \pi_2(f_{x:=e,\omega}(\hat{\omega}')) = \{\varepsilon\}$.
  - Let $lev(x) = \mathrm{L}$. Then, $\tau_{init}[\![e]\!] = \mathrm{L}$ according to $\tau_{init}[\![e]\!] \preceq \tau_{init}[\![x]\!]$. Let $\hat{\omega} =_{\tau_{init}} \hat{\omega}'$. According to Lemma A.3, $\hat{\omega}[\![e]\!] = \hat{\omega}'[\![e]\!]$. This entails that the observables $\pi_2(f_{x:=e,\omega}(\hat{\omega})) = [\hat{\omega}[\![e]\!]]$ and $\pi_2(f_{x:=e,\omega}(\hat{\omega}')) = [\hat{\omega}'[\![e]\!]]$ are equal.
- Let α be $x := *$. From $(\omega, \tau_{init}, \omega', \tau, \sigma) \in [\![x := *]\!]_T$ we infer that if $lev(x) = \mathrm{L}$ then $\sigma = [r]$ where $r \in \mathbb{R}$, otherwise $\sigma = \varepsilon$.
  - Let $lev(x) \neq \mathrm{L}$. Then the property holds since for any two states $\hat{\omega}$ and $\hat{\omega}'$, we have $\pi_2(f_{x:=e,\omega}(\hat{\omega})) = \pi_2(f_{x:=e,\omega}(\hat{\omega}')) = \{\varepsilon\}$.
  - Let $lev(x) = \mathrm{L}$. Let $\hat{\omega} =_{\tau_{init}} \hat{\omega}'$. We have $\pi_2(f_{x:=*,\omega}(\hat{\omega})) = \pi_2(f_{x:=*,\omega}(\hat{\omega}')) = \{[r] \mid r \in \mathbb{R}\}$.
- Let α be $?P$. Since $\omega \in \mathcal{S}_{init}(?P)$, $\omega \models P$. Let $\hat{\omega} =_{\tau_{init}} \hat{\omega}'$. Then, we have $\pi_2(f_{?P,\omega}(\hat{\omega})) = \pi_2(f_{?P,\omega}(\hat{\omega}')) = \{\varepsilon\}$.
- Let α be $x' = e \;\&\; P$. From $(\omega, \tau_{init}, \omega', \tau, \sigma) \in [\![x' = e \;\&\; P]\!]_T$ we infer that there exists some solution $\varphi : [0, r] \to \mathcal{S}$ for ODE $x' = e$, for any $t \in [0, r]$ we have $\varphi(t) \vDash P$, $\omega = \varphi(0)$ and $\omega' = \varphi(r)$. Moreover, $\tau_{init}[\![\varphi(t)(x)]\!] \preceq \tau_{init}[\![x]\!]$.
  - Let $lev(x) \neq \mathrm{L}$. Then the property holds since for any two states $\hat{\omega}$ and $\hat{\omega}'$, we have the same trivial observable: $\pi_2(f_{x'=e \;\&\; P,\omega}(\hat{\omega})) = \pi_2(f_{x'=e \;\&\; P,\omega}(\hat{\omega}')) = \{\varepsilon\}$.
  - Let $lev(x) = \mathrm{L}$. Then for any $t \in [0, r]$, $\tau_{init}[\![\varphi(t)(x)]\!] = \mathrm{L}$ according to $\tau_{init}[\![\varphi(t)(x)]\!] \preceq \tau_{init}[\![x]\!]$. Let $\hat{\omega} =_{\tau_{init}} \hat{\omega}'$. According to Lemma A.3, $\hat{\omega}[\![\varphi(t)(x)]\!] = \hat{\omega}'[\![\varphi(t)(x)]\!]$. This entails that observables $\pi_2(f_{x'=e \;\&\; P,\omega}(\hat{\omega})) = [\hat{\omega}[\![\varphi(r)(x)]\!]]$ and $\pi_2(f_{x'=e \;\&\; P,\omega}(\hat{\omega}')) = [\hat{\omega}'[\![\varphi(r)(x)]\!]]$ are equal.
- Let the HP be $\alpha \cup \beta$. The induction hypotheses assume that the property holds for α and β. Let's call these hypotheses as $H(\alpha)$ and $H(\beta)$, resp. From $(\omega, \tau_{init}, \omega', \tau, \sigma) \in [\![\alpha \cup \beta]\!]_T$ we infer that $(\omega, \tau_{init}, \omega', \tau, \sigma) \in [\![\alpha]\!]_T \cup [\![\beta]\!]_T$.
  - If $(\omega, \tau_{init}, \omega', \tau, \sigma) \in [\![\alpha]\!]_T$, then according to $H(\alpha)$, we have $ES \vDash \alpha, \omega$. This entails that for any two low-equivalent states $\hat{\omega}, \hat{\omega}' \in \mathcal{S}_{init}(\alpha)$, we have the same observables $\pi_2(f_{\alpha,\omega}(\hat{\omega})) = \pi_2(f_{\alpha,\omega}(\hat{\omega}'))$.
  - Similarly, if $(\omega, \tau_{init}, \omega', \tau, \sigma) \in [\![\beta]\!]_T$, then according to $H(\beta)$, we have $ES \vDash \beta, \omega$. This entails that for any two low-equivalent states $\hat{\omega}, \hat{\omega}' \in \mathcal{S}_{init}(\beta)$, we have $\pi_2(f_{\beta,\omega}(\hat{\omega})) = \pi_2(f_{\beta,\omega}(\hat{\omega}'))$.

  We have $f_{\alpha\cup\beta,\omega}(\hat{\omega}) = f_{\alpha,\omega}(\hat{\omega}) \cup f_{\beta,\omega}(\hat{\omega})$ by definition. This entails that $\pi_2(f_{\alpha\cup\beta,\omega}(\hat{\omega})) = \pi_2(f_{\alpha,\omega}(\hat{\omega})) \cup \pi_2(f_{\beta,\omega}(\hat{\omega}))$. Using the results from the two above cases, we have then have $\pi_2(f_{\alpha\cup\beta,\omega}(\hat{\omega})) = \pi_2(f_{\alpha,\omega}(\hat{\omega}')) \cup \pi_2(f_{\beta,\omega}(\hat{\omega}'))$, which implies $\pi_2(f_{\alpha\cup\beta,\omega}(\hat{\omega})) = \pi_2(f_{\alpha\cup\beta,\omega}(\hat{\omega}'))$.
- Let the HP be $\alpha; \beta$. The induction hypotheses assume that the property holds for α and β. Let's call these hypotheses as $H(\alpha)$ and $H(\beta)$, resp. From $(\omega, \tau_{init}, \omega', \tau, \sigma) \in [\![\alpha; \beta]\!]_T$ we infer that there exist ω_0, τ_0, σ_0, and σ_1 such that the following hold: $(\omega, \tau_{init}, \omega_0, \tau_0, \sigma_0) \in [\![\alpha]\!]_T$, $(\omega_0, \tau_0, \omega', \tau, \sigma_1) \in [\![\beta]\!]_T$, and $\sigma = \sigma_0\sigma_1$. From $(\omega, \tau_{init}, \omega_0, \tau_0, \sigma_0) \in [\![\alpha]\!]_T$ and $H(\alpha)$ we infer that $ES \vDash \alpha, \omega$, i.e., for any two low equivalent states $\hat{\omega}, \hat{\omega}' \in \mathcal{S}_{init}(\alpha)$, we have $\pi_2(f_{\alpha,\omega}(\hat{\omega})) = \pi_2(f_{\alpha,\omega}(\hat{\omega}'))$. From $(\omega_0, \tau_0, \omega', \tau, \sigma_1) \in [\![\beta]\!]_T$ and $H(\beta)$ we infer that $ES \vDash \beta, \omega_0$, i.e., for any two low equivalent states $\hat{\omega}, \hat{\omega}' \in \mathcal{S}_{init}(\beta)$, we have $\pi_2(f_{\beta,\omega_0}(\hat{\omega})) = \pi_2(f_{\beta,\omega_0}(\hat{\omega}'))$. Then we have,

$$\begin{aligned}\pi_2(f_{\alpha;\beta,\omega}(\hat{\omega})) &= \pi_2\Big(\bigcup_{\omega_0:(\omega,\omega_0,\_)\in[\![\alpha]\!]_U} (f_{\beta,\omega_0} \odot f_{\alpha,\omega})(\hat{\omega})\Big)\\ &= \pi_2\big(\{(\hat{\omega}_1, \hat{\sigma}_0\hat{\sigma}_1) \mid \exists\hat{\omega}_0, \hat{\sigma}_0, \hat{\sigma}_1.(\hat{\omega}_0, \hat{\sigma}_0) \in f_{\alpha,\omega}(\hat{\omega}),\\ &\qquad (\hat{\omega}_1, \hat{\sigma}_1) \in f_{\beta,\omega_0}(\hat{\omega}_0)\}\big)\\ &= \{\hat{\sigma}_0\hat{\sigma}_1 \mid \exists\hat{\omega}_0.(\hat{\omega}_0, \hat{\sigma}_0) \in f_{\alpha,\omega}(\hat{\omega}),\\ &\qquad \hat{\sigma}_1 \in \pi_2(f_{\beta,\omega_0}(\hat{\omega}_0))\}.\end{aligned}$$

Similarly,

$$\pi_2(f_{\alpha;\beta,\omega}(\hat{\omega}')) = \{\hat{\sigma}_0'\hat{\sigma}_1' \mid \exists\hat{\omega}_0'.(\hat{\omega}_0', \hat{\sigma}_0') \in f_{\alpha,\omega}(\hat{\omega}'), \hat{\sigma}_1' \in \pi_2(f_{\beta,\omega_0}(\hat{\omega}_0'))\}.$$

$\hat{\sigma}_0$ and $\hat{\sigma}_0'$ range over the same set, since $\pi_2(f_{\alpha,\omega}(\hat{\omega})) = \pi_2(f_{\alpha,\omega}(\hat{\omega}'))$. Using Lemma A.4, we infer that $\hat{\omega}_0 =_{\mathrm{L}} \hat{\omega}_0'$. Then, since $ES \vDash \beta, \omega_0$, we have $\pi_2(f_{\beta,\omega_0}(\hat{\omega}_0)) = \pi_2(f_{\beta,\omega_0}(\hat{\omega}_0'))$.

Therefore, $\hat{\sigma}_1$ and $\hat{\sigma}_1'$ range over another set as well, and thus conclude that $\pi_2(f_{\alpha;\beta,\omega}(\hat{\omega})) = \pi_2(f_{\alpha;\beta,\omega}(\hat{\omega}'))$.

- Let the HP be α^*. The induction hypothesis assume that the property holds for α. Let's call these hypothesis as $H(\alpha)$. Let $\hat{\omega}$ and $\hat{\omega}'$ be low-equivalent. We first show that for any $n \in \mathbb{N}$, the property holds for α^n. This is done by induction on n.
  - Let $n = 0$. Then the property holds trivially, as $\pi_2(f_{\alpha^0,\omega}(\hat{\omega})) = \pi_2(f_{\alpha^0,\omega}(\hat{\omega}')) = \{\varepsilon\}$.
  - Let the property hold for α^k, i.e., $H(\alpha^k)$. Since $\alpha^{k+1} = \alpha^k;\alpha$, we use the sequence case (previous case), and assumptions regarding the property on α^k and α (i.e., $H(\alpha^k)$ and $H(\alpha)$), and conclude that the property holds for α^{k+1}.

  Therefore, for any $n \in \mathbb{N}$, the property holds for α^n. Since $f_{\alpha^*,\omega}(\omega_0) = \bigcup_{n\in\mathbb{N}} f_{\alpha^n,\omega}(\omega_0)$, the property holds for α^*.