Semantics Foundation for Cyber-Physical Systems Using Higher-Order UTP

XIONG XU, SKLCS, Institute of Software, Chinese Academy of Sciences, China and Inria, France
JEAN-PIERRE TALPIN, Inria, France
SHULING WANG, SKLCS, Institute of Software, Chinese Academy of Sciences, China
BOHUA ZHAN, SKLCS, Institute of Software & UCAS, Chinese Academy of Sciences, China
NAIJUN ZHAN*, State Key Lab. of Computer Science and Science & Technology on Integrated Information Science Lab., Institute of Software & UCAS, Chinese Academy of Sciences, China

Model-based design has become the predominant approach to the design of hybrid and cyber-physical systems (CPSs). It advocates the use of mathematically founded models to capture heterogeneous digital and analog behaviours from domain-specific formalisms, allowing all engineering tasks of verification, code synthesis and validation to be performed within a single semantic body. Guaranteeing the consistency among the different views and heterogeneous models of a system at different levels of abstraction however poses significant challenges. To address these issues, Hoare and He's Unifying Theories of Programming (UTP) proposes a calculus to capture domain-specific programming and modelling paradigms into a unified semantic framework. Our goal is to extend UTP to form a semantic foundation for CPS design. *Higher-Order UTP* (HUTP) is a conservative extension to Hoare and He's theory which supports the specification of discrete, real-time and continuous dynamics, concurrency and communication, and higher-order quantification. Within HUTP, we define a calculus of *normal hybrid designs* to model, analyse, compose, refine and verify heterogeneous hybrid system models. In addition, we define respective formal semantics for HCSP (Hybrid Communicating Sequential Processes) and Simulink using HUTP.

CCS Concepts: • **Theory of computation → Timed and hybrid models**; *Denotational semantics.*

Additional Key Words and Phrases: UTP, CPS, model-based design, semantic model

ACM Reference Format:
Xiong Xu, Jean-Pierre Talpin, Shuling Wang, Bohua Zhan, and Naijun Zhan. 2018. Semantics Foundation for Cyber-Physical Systems Using Higher-Order UTP. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (February 2018), 48 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Hybrid systems, also known under the more architecture-centered denomination of cyber-physical systems (CPSs), exploit networked computing units to monitor and control physical processes via

*Corresponding author.

Authors' addresses: Xiong Xu, SKLCS, Institute of Software, Chinese Academy of Sciences, Beijing, China and Inria, Rennes, France, xux@ios.ac.cn; Jean-Pierre Talpin, Inria, Rennes, France, jean-pierre.talpin@inria.fr; Shuling Wang, SKLCS, Institute of Software, Chinese Academy of Sciences, Beijing, China, wangsl@ios.ac.cn; Bohua Zhan, SKLCS, Institute of Software & UCAS, Chinese Academy of Sciences, Beijing, China, bzhan@ios.ac.cn; Naijun Zhan, State Key Lab. of Computer Science and Science & Technology on Integrated Information Science Lab., Institute of Software & UCAS, Chinese Academy of Sciences, Beijing, China, znj@ios.ac.cn.

1049-331X/2018/2-ART $15.00
https://doi.org/XXXXXXX.XXXXXXX

wired and/or radio communications. CPSs are omnipresent, from high-speed train control systems, power grids, automated factories, to ground, sea, air and space transportations. CPSs can be complex, networked, systems of systems, and are often entrusted with mission- and safety-critical tasks. Therefore, the efficient and verified development of safe and reliable CPSs is a priority mandated by many standards, yet a notoriously difficult and challenging field.

To address design complexity under the necessity of verified CPS development, model-based design (MBD) [18] has become a predominant development approach in the embedded systems industry. The classical development process of MBD can be sketched as: (1) to start from an abstract model, ideally with a precise mathematical semantics; (2) to conduct extensive analysis and verification on this abstract model, so as to locate and correct errors as early as possible; (3) subsequently, to refine that abstract, system-level, model to a semantically concrete specification and, eventually, to executable code, by model transformation.

To assist such a development process, design-by-contract (DbC) is an important tool. It allows to define the abstract model of a composite system from the assembly of simple interfaces declaring the behavioural assumptions and guarantees of its constituents. DbC provides a method to conduct verification and development tasks in a modular and compositional manner. Under the DbC paradigm, developers define contracts with two parts: (1) the required behaviour that a constituent guarantees to implement, and (2) the assumptions the constituent makes about its environment. A composite system can thus be characterised by the contracts of its constituents, the composition of which fulfills the expected system contract. Each system constituent can be shown to fulfill its guaranteed behaviour under given assumptions, the violation of which could lead to an unpredictable behaviour of the entire system.

By allowing an arbitrary level of system decomposition, vertically (domain-specific) or horizontally (cross-domain), design complexity becomes tractable and controllable. Errors can be identified and corrected at the very early stages of system design; correctness and reliability can be guaranteed by simulation, verification and refinement; and developers can reuse existing contracts and components, further improving development efficiency. Expectedly, model- and contract-based design practices are much recommended by many standard bodies for safety-critical system development.

Nonetheless, guaranteeing the consistency of a system based on the models of its components poses significant challenges. To address it, ideally, one would need a uniform semantics domain capable of capturing all component models, at different abstraction levels, across heterogeneous domain concerns, into the same analysis, automated reasoning and verification framework.

Motivated by the Grand Unified Theory in physics, in the literature, Hoare and He's unifying theories of programming (UTP) [28] is the first proposal toward unifying different programming paradigms and models under a common relational calculus suitable for design and verification. In UTP, programs are interpreted as guarded designs consisting of a guard, pre- and post-condition [24, 28]. UTP defines operators and a refinement relation over guarded designs to form a lattice. Hence, in UTP, arbitrarily different programming constructs, paradigms, and models of computation can be logically interpreted and compared using the Galois connection induced by their lattice ordering.

Guaranteeing consistency among different models at different levels of abstraction is a grand challenge in control and software engineering. To address it for the design of hybrid systems, this paper follows the UTP paradigm by extending the classical UTP with higher-order quantification on continuous variables (functions over time) and differential relations to form a higher-order UTP (HUTP), that can be used to give uniform semantics of hybrid systems at different levels of abstraction, from higher-level abstract models like Simulink or Hybrid Communicating Sequential Processes (HCSP), down to the low-level models of computation of their implementation by generated source code in SystemC or ANSI-C, allowing the consistency among models of hybrid

systems at different levels of abstraction/implementation to be proved using theorem proving techniques. Note that HUTP in this paper differs from the higher-order UTP defined in [28, 61, 62]. The latter focuses on higher-order programming, i.e., programs can be treated as higher-order variables in the paradigm. Higher-order quantifications over functions and predicates are also addressed in related works, such as [62]. However, it is still unclear whether the classical UTP can be extended to hybrid systems like the extension to higher-order programs.

Our work starts from [10], which extends UTP by allowing differential relations and higher-order quantification, establishing a theory of *higher-order UTP* (HUTP). Additionally, [10] defines HUTP syntactically, which still leaves the question open of equipping it with an adequate algebraic structure (e.g., complete lattices) to serve as a semantics foundation for CPS design.

During the same period of [10], to address the context of CPS design with UTP, Woodcock et al. introduce time-dependent trajectory variables into the alphabet and define *hybrid relations* [17], based on which the semantics of Modelica [42] with a mixture of differential algebraic equations and event preemption is given. To unify discrete and hybrid theories, they introduce continuous-time traces to the trace model and define a generalised theory of reactive processes [16] in terms of trace algebra. Subsequently in [15], they define reactive design contracts that provide the basis for modelling and verification of reactive systems in the presence of parallelism and communication, yet limited to discrete controllers. Based on the theory of reactive processes, [13] provides an alternative definition of *hybrid relations* to define a denotational semantics of hybrid programs [48]. However, the above bulk of works does not address some important features, including

- *Super-dense computation.* As argued in [38], *super-dense computation* provides a very appropriate abstract computation model for embedded systems, as computer is much faster than other physical devices and computation time of a computer is therefore negligible, although the temporal order of computations is still there. As [17] considers finite timed traces with right continuity, super-dense time computation violates right continuity at time points, where a sequence of discrete events happen, and therefore cannot be modelled, although [16] argues that the theory of finite timed traces is extensible to deal with infinite ones.
- *Parallelism with communication-based synchronisation* is not addressed yet, although widely used in CPS designs.
- *Declaration of local variables and channels.* Variables and channels are interpreted as functions over time, therefore declaration of local variables and channels have to resort to higher-order quantification. [14] uses symmetric lenses to deal with local variables, but only discrete variables are considered.

For the drawbacks of our previous work [10], we propose a new theory of HUTP and investigate the suitable algebraic structure for HUTP by taking all important features of hybrid systems into consideration, to enable its use as a semantics foundation for CPS design. Inspired by the Duration Calculus [66], we introduce real-time variables, denoted as functions over time. Derivatives of real variables are additionally allowed in predicates, to form differential relations. In order to model synchronous communication between processes, we introduce timed traces to record the communication history of processes. Noteworthy contributions of previous works [15, 16, 28] with related aim regarded traces as finite abstract sequences. By contrast, timed traces in this paper can be infinite. Therefore, non-terminating behaviours such as divergence and deadlock can be described explicitly. These definitions ground the elaboration of the concepts of hybrid processes and (normal) hybrid designs, based on which we define formal semantics for HCSP [22, 67] and Simulink [39] using HUTP, and justify the correctness of the translation from Simulink to HCSP in [69] as an application of HUTP. In addition, by abstracting hybrid processes and (normal) hybrid

designs, we propose a skeleton of equipping HUTP with compositional assume-guarantee reasoning by means of contracts.

Our work is similar to [15] in some aspects, as both are extensions of traditional UTP [28]. However, they differ in essence, namely:

- Our HUTP theory separates the concerns of time, state and traces. Especially, we introduce continuous state variables and construct a trace model to address concurrency and communication for hybrid systems. In [15, 16], although continuous states are encapsulated into traces, parallelism based on traces is not made explicit. For example, the interleaving of traces carrying continuous states is confusing, as, in reality, physical evolution never interleaves with discrete models.
- Our approach can deal with infinite behaviour very well, in which therefore deadlock and divergence can be modelled formally. For example, the Zeno's paradox (Example 3.5) can be modelled as a normal non-terminating (and not necessarily unpredictable) behaviour. In contrast, it is unclear how to model the Zeno's paradox and other infinite behaviours using [15], as it seems that only finite traces are involved.
- Since our HUTP theory supports infinite behaviours, auxiliary variables *wait* and *wait'*, denoting reactivity in discrete models [15, 28], are interpreted explicitly with our approach.

In summary, our proposed HUTP is a conservative extension to the classical UTP of [28] and contributes:

- the separation of concerns in hybrid system design into time, state and trace;
- a timed trace model recording execution history and captures communication behaviours;
- real-time variables and their derivatives, which are functions over time, and differential relations over them that are very powerful to express all kinds of continuous dynamics;
- higher-order quantification for specifying locality;
- a calculus of hybrid processes and (normal) hybrid designs to model and analyse hybrid systems with (potentially) infinite behaviours; and
- the HUTP semantics of HCSP and Simulink.

Paper Organisation. The rest of the paper is organised as follows. Section 2 retrospects some preliminary concepts of the Unifying Theories of Programming. Section 3 defines a timed trace model which will play an important role in the HUTP theory. Section 4 introduces hybrid processes which combine time, states and traces to provide a uniform semantics for hybrid systems. To elaborate a sound theory, we extend hybrid processes to (normal) hybrid designs in Section 5. In Section 6, the HUTP is applied to verify the translation from Simulink diagrams to HCSP as an application of HUTP. Section 7 addresses the related work and Section 8 concludes this paper and discusses future works.

2 UNIFYING THEORIES OF PROGRAMMING

UTP [28] is an alphabetised refinement calculus unifying heterogeneous programming paradigms. An alphabetised relation consists of an alphabet $\alpha(P)$, containing its variables x and primes x', and a relational predicate P referring to this vocabulary. The terms x and x' are called observable variables: x is observable at the start of execution and x' is observable at the end of execution. The behaviour of a program is encoded as a relation between the observable variables x and x'. In particular, assignment, sequential composition, conditional statement, non-deterministic choice, and recursion of imperative programs can be specified as alphabetised relations below, where $\mathbf{x}$ and $\mathbf{x}'$ are sequences or vectors of variables, $\mathbf{x}\backslash\{x\}$ ($\mathbf{x}'\backslash\{x'\}$) denotes excluding x (x') from $\mathbf{x}$ ($\mathbf{x}'$).

To start with, the relation calculus comprises all operators of first-order logic.

$$\begin{array}{rcl} x := e & \widehat{=} & x' = e \wedge \mathbf{x}' \backslash \{x'\} = \mathbf{x} \backslash \{x\} \\ P \fatsemi Q & \widehat{=} & \exists \mathbf{x}_* \cdot P[\mathbf{x}_*/\mathbf{x}'] \wedge Q[\mathbf{x}_*/\mathbf{x}] \\ P \lhd b \rhd Q & \widehat{=} & (b \wedge P) \vee (\neg b \wedge Q) \\ P \sqcap Q & \widehat{=} & P \vee Q \\ P \sqcup Q & \widehat{=} & P \wedge Q \end{array}$$

Assignment $x := e$ is defined by observing the update x' of variable x once its value e is evaluated, leaving other variables in the alphabet $\mathbf{x}$ unchanged. Sequence $P \fatsemi Q$ is modelled by locally binding, through $\mathbf{x}_*$, the final state $\mathbf{x}'$ of P and the initial state $\mathbf{x}$ of Q both are instantiated to it. Note that $\fatsemi$ requires that $\alpha_{out}(P) = \alpha'_{in}(Q)$, where $\alpha_{out}(P)$ and $\alpha_{in}(Q)$ denote the sets of output and input variables in $\alpha(P)$ and $\alpha(Q)$, respectively, and $\alpha'_{in}(Q)$ is the primed version by priming all the variables in $\alpha_{in}(Q)$. The conditional $P \lhd b \rhd Q$ evaluates as P if b is true or as Q otherwise. $P \sqcap Q$ non-deterministically chooses P or Q, whereas $P \sqcup Q$ is a conjunction of P and Q.

Let P and Q be two predicates with the same alphabet, say $\{\mathbf{x}, \mathbf{x}'\}$. Then, Q is a *refinement* of P, denoted $P \sqsubseteq Q$, if $\forall \mathbf{x}, \mathbf{x}' \cdot Q \Rightarrow P$. In addition, $P \sqsubseteq Q$ iff $P \sqcap Q = P$ iff $P \sqcup Q = Q$. With respect to the refinement order $\sqsubseteq$, the least (μ) and greatest (ν) fixed points of a program function F can be defined as follows:

$$\begin{array}{rcl} \mu F & \widehat{=} & \bigsqcap \{X \mid F(X) \sqsubseteq X\} \\ \nu F & \widehat{=} & \bigsqcup \{X \mid X \sqsubseteq F(X)\} \end{array}$$

Example 2.1. Given an alphabet $\{x, y, x', y'\}$, the semantics of the assignment $x := 1$ is different from $x' = 1$. The former is equivalent to $x' = 1 \wedge y' = y$, while the latter indicates that y and y' can be arbitrary, i.e., $x' = 1 \sqsubseteq x := 1$. They are equivalent if the alphabet contains only x and x'.

A UTP theory is a well-defined subset of alphabetised relations that satisfies certain conditions, called *healthiness conditions*. If a predicate is a fixed point of the healthiness condition, $P = \mathcal{H}(P)$, then it is said to be $\mathcal{H}$-healthy. In other words, a healthiness condition $\mathcal{H}$ defines an invariant predicate set $\{X \mid \mathcal{H}(X) = X\}$, and is required to be idempotent ($\mathcal{H} \circ \mathcal{H} = \mathcal{H}$), which means that taking the medicine twice leaves you as healthy as taking it once (no overdoses). So, in UTP, the healthy predicates of a theory are the fixed points of idempotent functions. When $\mathcal{H}$ is monotonic with respect to the refinement order $\sqsubseteq$, then according to the Knaster-Tarski theorem [57], the UTP theory satisfying $\mathcal{H}$ forms a complete lattice and, additionally, recursion can be well specified. Multiple healthiness conditions $\mathcal{H}_0, \mathcal{H}_1, \cdots, \mathcal{H}_n$ can also be composed as $\mathcal{H} \mathrel{\widehat{=}} \mathcal{H}_0 \circ \mathcal{H}_1 \circ \cdots \circ \mathcal{H}_n$, provided that $\mathcal{H}_i$ and $\mathcal{H}_j$ commute, for all $0 \leq i, j \leq n$. Healthiness conditions play an important role in UTP, and distinct healthiness conditions can be defined to specify properties of different programming paradigms. In Section 4, we use healthiness conditions to define the concept of hybrid processes.

Example 2.2. Given an alphabet $\{x, x', term, term'\}$, where the Boolean variable $term'$ denote whether the program has terminated. If the preceding program is non-terminated ($\neg term$), the current program P should do nothing but keep the values of all variables unchanged, expressed by the following healthiness condition:

$$\mathcal{H}(P) \quad \widehat{=} \quad P \lhd term \rhd (x' = x \wedge term' = term)$$

The statement $x := 1$ is not $\mathcal{H}$-healthy, but we can make it $\mathcal{H}$-healthy by

$$\mathcal{H}(x := 1) \quad = \quad (x := 1) \lhd term \rhd (x' = x \wedge term' = term)$$

It is easy to check that the above program is indeed a fixed point of $X = \mathcal{H}(X)$. When $term$ is **true**, i.e., the preceding program is terminated, the above program executes $x := 1$. Note that this

assignment is equivalent to $x' = 1 \wedge term' = term$. If non-termination is expected, we can write $\mathcal{H}(x' = 1 \wedge \neg term')$. It is a left-zero of sequential composition, i.e.,

$$\mathcal{H}(x' = 1 \wedge \neg term') \fatsemi P \quad = \quad \mathcal{H}(x' = 1 \wedge \neg term')$$

provided that P is any $\mathcal{H}$-healthy predicate whose alphabet is $\{x, x', term, term'\}$.

Following the convention of UTP, in this paper, we will define a set of healthiness conditions to identify the features of hybrid systems. The operators for classical sequential programs based on first-order relational calculus, such as $\fatsemi$, $\sqcap$ and $\sqcup$, can be easily lifted to the theory for hybrid systems proposed in this paper based on higher-order differential relational calculus, and we will prove that they still hold the desired properties like in classical UTP.

3 TIMED TRACE MODEL

In this section, we construct a timed trace model to capture the communication behaviour of processes. Traces play an important role in our HUTP theory which separates the concerns of time, state and trace. For processes in parallel, we assume that no shared variable is allowed between them and exchange of data between processes is described solely by communications. The synchronous communication between concurrent processes is achieved using input and output data channels. The input action $ch?x$ receives a value along channel ch and assigns it to the variable x, and the output action $ch!e$ sends the value of the expression e along channel ch. A synchronous communication takes place immediately when both the sending and the receiving actions are enabled. If one party is not ready, the other has to wait.

3.1 Trace Blocks

Each timed trace is a sequence of communication, wait and internal blocks defined below:

- A pair $\langle ch*, d\rangle$ is a communication block, where $ch*$, a channel operation, indicates input ($ch?$), output ($ch!$) or synchronised communication (ch), and d is a value along channel ch. It represents that a communication along $ch*$ takes place instantly. For example, $\langle ch?, d\rangle$ describes the communication receiving the value d along the channel ch at current time.
- A pair $\langle \delta, RS\rangle$ is a wait block, where $0 < \delta \leq +\infty$ is the time period and RS is a set of waiting channel operations, called *ready set*. It means the process evolves during the period with the channel operations in RS ready for communication. The dual of RS is defined by

  $$\overline{RS} \quad = \quad \{ch\overline{*} \mid ch* \in RS, * \in \{?, !\}\}$$

  where $\overline{?} = !$ and $\overline{!} = ?$, and we assume that $RS \cap \overline{RS} = \emptyset$.
- The symbol τ is an internal block standing for an internal or invisible action such as a timeless discrete computation.

Example 3.1. For the communication of two HCSP processes: $ch?x \| ch!2$, since $ch?x$ and $ch!2$ are ready simultaneously at the beginning, the communication takes place instantly according to the semantics of HCSP [64]. The timed traces of $ch?x$ and $ch!2$ are $\langle ch?, 2\rangle$ and $\langle ch!, 2\rangle$, respectively.

Example 3.2. Consider the delay communication $ch?x \| (\textbf{wait}\ 1; ch!2)$. Since the right process waits for 1 time unit before $ch!2$ is ready, the left process has to wait for 1 time unit for synchronisation although it has been ready since the beginning. The communication takes place after 1 time unit. Therefore, the timed trace of the left is $\langle 1, \{ch?\}\rangle^\frown\langle ch?, 2\rangle$ and the right is $\langle 1, \emptyset\rangle^\frown\langle ch!, 2\rangle$.

Example 3.3. Consider an ordinary differential equation (ODE) with communication interruption:

$$x := 1; \langle \dot{x} = x \& \textbf{true}\rangle \trianglerighteq [\!] (sensor!x \rightarrow actuator?x) \ \parallel\ \textbf{wait}\ 2; sensor?y; actuator!(y + 1)$$

The assignment $x := 1$ is a timeless discrete action denoting initialisation. The left HCSP process means the continuous evolution depicted by the ODE $\dot{x} = x$ is preempted as soon as the communication $sensor!x$ takes place, followed by $actuator?x$. The right process indicates that the ODE of the left process evolves for 2 time units before $sensor!x$ takes place when the value of x reaches $\exp(2)$. Therefore, the timed trace of the left process is

$$\tau^\frown\langle 2, \{sensor!\}\rangle^\frown\langle sensor!, \exp(2)\rangle^\frown\langle actuator?, \exp(2)+1\rangle$$

and the timed trace of the right process is

$$\langle 2, \emptyset\rangle^\frown\langle sensor?, \exp(2)\rangle^\frown\langle actuator!, \exp(2)+1\rangle$$

3.2 Timed Traces

In the previous works such as [15, 16, 28], traces are finite sequences. In our work, we consider infinite timed traces. A timed trace is a finite or infinite sequence of communication, wait, and internal blocks which records the execution history of a process.

Let Σ be the set of all channels and $\Sigma?$ and $\Sigma!$ the respective sets of receiving and sending events along channels in Σ. The sets of communication and wait blocks are respectively

$$\begin{array}{lcl} \mathbb{C} & \widehat{=} & (\Sigma? \cup \Sigma! \cup \Sigma) \times \mathbb{D} \\ \mathbb{W} & \widehat{=} & \{\langle \delta, RS\rangle \mid 0 < \delta \leq +\infty, RS \subseteq (\Sigma? \cup \Sigma!), RS \cap \overline{RS} = \emptyset\} \end{array}$$

where $\mathbb{D}$ is some domain.

Definition 3.4 (Timed Trace Set). Let $\mathbb{TT} = (\mathbb{C} \cup \mathbb{W} \cup \{\tau\})^* \cup (\mathbb{C} \cup \mathbb{W} \cup \{\tau\})^\omega$ be the timed trace set, ϵ the empty trace and $^\frown$ the concatenation between trace blocks. Then, the algebra $(\mathbb{TT}, \epsilon, ^\frown)$ satisfies the following axioms, where $^\frown$ takes precedence over $=$.

Associativity $\forall \mathtt{tt}_0, \mathtt{tt}_1, \mathtt{tt}_2 \in \mathbb{TT} \cdot (\mathtt{tt}_0{}^\frown\mathtt{tt}_1){}^\frown\mathtt{tt}_2 = \mathtt{tt}_0{}^\frown(\mathtt{tt}_1{}^\frown\mathtt{tt}_2)$
Unit $\forall \mathtt{tt} \in \mathbb{TT} \cdot \epsilon^\frown\mathtt{tt} = \mathtt{tt}^\frown\epsilon = \mathtt{tt}$
Left Zero $\forall \mathtt{tt} \in \mathbb{TT} \cdot \langle +\infty, RS\rangle^\frown\mathtt{tt} = \langle +\infty, RS\rangle$
Additivity $\langle \delta_0, RS\rangle^\frown\langle \delta_1, RS\rangle = \langle \delta_0 + \delta_1, RS\rangle$

The axiom **Left Zero** reflects a simplifying function $lz : \mathbb{TT} \to \mathbb{TT}$ that removes the redundant tail from each trace. For example, $lz(\langle +\infty, RS\rangle^\frown\langle 2, RS\rangle) = \langle +\infty, RS\rangle$. The axiom **Additivity** reflects a simplifying function $add : \mathbb{TT} \to \mathbb{TT}$ that merges all the consecutive wait blocks with the same ready set in each trace. For example, $add(\langle 1, RS\rangle^\frown\langle 2, RS\rangle) = \langle 3, RS\rangle$. Then, the composite function $add \circ lz : \mathbb{TT} \to \mathbb{TT}$ returns the simplest form of each trace. A trace $\mathtt{tt}$ is called *canonical normal form* if $|\mathtt{tt}| = |add \circ lz(\mathtt{tt})|$, where $|\cdot|$ returns the length of traces. Thus, we say $\mathtt{tt}$ is canonical if it cannot be simplified. In what follows, all traces are referred to canonical normal form if not otherwise stated.

We say $\mathtt{tt}_0$ is *equivalent* to $\mathtt{tt}_1$, denoted by $\mathtt{tt}_0 = \mathtt{tt}_1$, if each finite prefix of $\mathtt{tt}_0$ is also a prefix of $\mathtt{tt}_1$, and vice versa. We say $\mathtt{tt}_0 < \mathtt{tt}_1$ if $\mathtt{tt}_0$ is finite and there exists a trace $\mathtt{tt}$ such that $\mathtt{tt}_0{}^\frown\mathtt{tt} = \mathtt{tt}_1$, and in addition we let $\mathtt{tt}_1 - \mathtt{tt}_0 = \mathtt{tt}$. Therefore, $\mathtt{tt}_0 < \mathtt{tt}_1$ means $\mathtt{tt}_0$ is a finite prefix of $\mathtt{tt}_1$, and $\mathtt{tt}_1 - \mathtt{tt}_0$ denotes the suffix after deleting $\mathtt{tt}_0$.

Let $\lim(\mathtt{tt})$ be the sum of the time periods of all the wait blocks in $\mathtt{tt}$. If no wait block appears in $\mathtt{tt}$, then we let $\mathrm{limit}(\mathtt{tt}) = 0$. A trace $\mathtt{tt}$ is *non-terminated* if $\lim(\mathtt{tt}) = +\infty \vee |\mathtt{tt}| = +\infty$ and *divergent* if $\lim(\mathtt{tt}) < +\infty \wedge |\mathtt{tt}| = +\infty$. Let $\mathtt{tt}(n)$ be the n-th (start from 0) block of $\mathtt{tt}$. If $|\mathtt{tt}| < +\infty$, for any $n \geq |\mathtt{tt}|$ we let $\mathtt{tt}(n) = \mathtt{tt}(|\mathtt{tt}| - 1)$, i.e., the last block of $\mathtt{tt}$. We say $\mathtt{tt}$ is *deadlocked* if

$$\lim(\mathtt{tt}) = +\infty \wedge \exists N \in \mathbb{N} \cdot \forall n \geq N \cdot \mathtt{tt}(n) = \langle \delta, RS\rangle \wedge RS \neq \emptyset \tag{1}$$

where $\mathbb{N}$ is the set of natural numbers. (1) means that, from a certain time point, the process will be waiting for communication forever. Clearly, a deadlock is non-terminated because $\lim(\mathtt{tt}) = +\infty$.

This non-termination occurs because some channel operations are never matched. From the view of failures-divergence model [54], the standard denotational semantics for CSP, a ready set is exactly the dual of a refusal set, which contains the events (channel operations) that are not refused. Thus, (1) means that, from a certain time point, the dual channel operations in $\mathtt{tt}$'s ready set are always refused by the environment, which leads to a deadlock. Especially, if $\lim(\mathtt{tt}) = +\infty \wedge |\mathtt{tt}| < +\infty$, the last block of $\mathtt{tt}$ must be a wait block $\langle +\infty, RS\rangle$, in which case $\mathtt{tt}$ will be deadlocked if $RS \neq \emptyset$.

Example 3.5 (Zeno's Paradox). Consider a ball falling from an initial height h with initial velocity 0. The ball falls freely with its dynamics given by the ODE $\dot{v} = -g$, where v is the velocity and g is the gravitational acceleration. After $\sqrt{2h/g}$ time units, it hits the ground ($h = 0$) with $v = -\sqrt{2hg}$. Then, there is a discontinuous update in its velocity. This discrete change can be modelled as a mode switch with the update given by $v := -c \cdot v$. This assumes that the collision is inelastic, and the velocity decreases by a factor c, for some appropriate constant $0 < c < 1$. Next, the ball bounces back with velocity $c\sqrt{2hg}$ and reaches the peak ($v = 0$) after $c\sqrt{2h/g}$ time units. The ball then falls again and this process repeats infinitely. In summary, the timed trace of the bouncing ball is $\langle\sqrt{2h/g}, \emptyset\rangle^\frown\tau^\frown\langle 2c\sqrt{2h/g}, \emptyset\rangle^\frown\tau^\frown\cdots^\frown\langle 2c^{n+1}\sqrt{2h/g}, \emptyset\rangle^\frown\tau^\frown\cdots$ where the internal blocks (τs) denote the mode switch of the velocity ($v := -c \cdot v$). The limit of this timed trace is $\frac{1+c}{1-c} \cdot \sqrt{2h/g}$ but the length of it is infinite, i.e., a divergence.

3.3 Parallel Composition

For each pair of processes in parallel, we assume that they communicate over a set of channels I. Concretely, they can only exchange messages or data synchronously via the channels in I, and they communicate with the environment via channels outside I. In [54], this parallel is called *generalised* or *interface* parallel, and in [64], it is called *alphabetised* parallel and the channels in I are called *common* channels. Therefore, in this paper, the set I is called a *common channel set* between processes (traces). For simplicity, we also assume:

(1) Each input or output channel can only be owned by one sequential process, i.e., a channel operation like $ch?$ cannot occur in two different sequential processes in parallel.
(2) Any pair of symmetric input and output channels cannot be owned by one sequential process, i.e., channel operations like $ch?$ and $ch!$ cannot occur in one sequential process.

Such restrictions are not essential, as the general communication model can always be reduced to this simplified model, please refer to [27] for the details.

Let $\mathtt{tt}_0 \|_I \mathtt{tt}_1 \rightsquigarrow \mathtt{tt}$ denote that the parallel composition of two timed traces $\mathtt{tt}_0$ and $\mathtt{tt}_1$ over a common channel set I derives a new timed trace $\mathtt{tt}$. Since infinite timed traces are included, we give the following coinductive rules to define the parallel composition of timed traces, where the binding priority among operators is that $^\frown$ prior to $\|_I$ prior to $\rightsquigarrow$, and $\uplus$ is a disjoint union.

$$\frac{}{\epsilon\|_I\epsilon \rightsquigarrow \epsilon}\ [\text{Empty}] \qquad \frac{\overline{RS_0} \cap RS_1 = \emptyset \quad \mathtt{tt}_0\|_I\mathtt{tt}_1 \rightsquigarrow \mathtt{tt}}{\langle\delta, RS_0\rangle^\frown\mathtt{tt}_0\|_I\langle\delta, RS_1\rangle^\frown\mathtt{tt}_1 \rightsquigarrow \langle\delta, RS_0 \uplus RS_1\rangle^\frown\mathtt{tt}}\ [\text{SynWait}]$$

$$\frac{ch \in I \quad \mathtt{tt}_0\|_I\mathtt{tt}_1 \rightsquigarrow \mathtt{tt}}{\langle ch?, d\rangle^\frown\mathtt{tt}_0\|_I\langle ch!, d\rangle^\frown\mathtt{tt}_1 \rightsquigarrow \langle ch, d\rangle^\frown\mathtt{tt}}\ [\text{SynIO}]$$

$$\frac{ch \notin I \quad \mathtt{tt}_0\|_I\mathtt{tt}_1 \rightsquigarrow \mathtt{tt}}{\langle ch*, d\rangle^\frown\mathtt{tt}_0\|_I\mathtt{tt}_1 \rightsquigarrow \langle ch*, d\rangle^\frown\mathtt{tt}}\ [\text{NoSynIO}] \qquad \frac{\mathtt{tt}_0\|_I\mathtt{tt}_1 \rightsquigarrow \mathtt{tt}}{\tau^\frown\mathtt{tt}_0\|_I\mathtt{tt}_1 \rightsquigarrow \tau^\frown\mathtt{tt}}\ [\tau\text{-Act}]$$

Rule [Empty] says that the parallel composition of two empty traces trivially reduces to an empty trace. Rule [SynWait] states that when both sides are waiting for a communication, they can be

synchronised for the same length of period if their ready sets are unmatched, i.e., $\overline{RS_0} \cap RS_1 = \emptyset$, because $\overline{RS_0} \cap RS_1 \neq \emptyset$ indicates that the two sides are waiting for a communication with some channel operations matched, that should be taken immediately, and no time synchronization is allowed. Rule [SynIO] synchronises the input and output communications along some channel $ch \in I$. The synchronization will never be applied to channels outside I or internal actions τ as stated by Rules [NoSynIO] and [τ-Act]. In addition, there are symmetric versions of these rules, which means that $\|_I$ is commutative.

Property 1 (Associativity). *Let I_0, I_1 and I_2 be the respective common channel sets between timed traces $\mathtt{tt}_0$ and $\mathtt{tt}_1$, $\mathtt{tt}_1$ and $\mathtt{tt}_2$, and $\mathtt{tt}_2$ and $\mathtt{tt}_0$, then $(\mathtt{tt}_0 \|_{I_0} \mathtt{tt}_1) \|_{I_1 \uplus I_2} \mathtt{tt}_2 = \mathtt{tt}_0 \|_{I_0 \uplus I_2} (\mathtt{tt}_1 \|_{I_1} \mathtt{tt}_2)$.*

Example 3.6. The timed trace $\mathtt{tt}_0 = \langle 2, \{ch?\}\rangle^\frown\langle ch?, 5\rangle$ indicates a delay for 2 time units, during which the channel operation $ch?$ is waiting, followed by receiving the value 5 along channel ch. The trace $\mathtt{tt}_1 = \langle 2, \emptyset\rangle^\frown\langle ch!, 5\rangle$ indicates a delay for 2 time units, during which no channel operations are waiting, followed by sending the value 5 along channel ch. The parallel composition of $\mathtt{tt}_0$ and $\mathtt{tt}_1$ over the channel ch is

$$\mathtt{tt}_0 \|_{\{ch\}} \mathtt{tt}_1 \quad \rightsquigarrow \quad \langle 2, \{ch?\}\rangle^\frown\langle ch, 5\rangle$$

by [SynWait], [SynIO] and [Empty]. In fact, recording $ch?$ in the ready set of the composed traces is not necessary, as ch is a channel that only connects $\mathtt{tt}_0$ and $\mathtt{tt}_1$, making it impossible for the environment to synchronise with it via ch. However, on the other hand, recording $ch?$ has no side effect, and just indicates that the process itself was waiting for the channel operation $ch!$ during the period, no matter whether ch is a common channel between $\mathtt{tt}_0$ and $\mathtt{tt}_1$ or not. This notion is in accordance to the semantics of HCSP (please refer to [64] for the details).

Example 3.7. The timed trace $\mathtt{tt}_0 = \langle dh?, 3\rangle^\frown\langle 2, \emptyset\rangle$ indicates receiving the value 3 along channel dh at the initial time, followed by a delay of 2 time units during which no channel operations are waiting. Let $\mathtt{tt}_1 = \langle 1, \emptyset\rangle^\frown\langle 1, \{ch?\}\rangle$. Then, we have

$$\mathtt{tt}_0 \|_{\{ch\}} \mathtt{tt}_1 \quad \rightsquigarrow \quad \langle dh?, 3\rangle^\frown\langle 1, \emptyset\rangle^\frown\langle 1, \{ch?\}\rangle$$

by [NoSynIO], [SynWait], [Empty] and **Additivity** of wait blocks (see Definition 3.4). Note that if $\mathtt{tt}_0$ and $\mathtt{tt}_1$ also communicate via dh, then no rules can be applied for $\mathtt{tt}_0 \|_{\{ch,dh\}} \mathtt{tt}_1$ because $dh?$ cannot be synchronised, i.e., they are uncomposable.

Example 3.8. Let $\mathtt{tt}_0 = \langle ch?, 3\rangle^\frown\langle dh?, 4\rangle$ and $\mathtt{tt}_1 = \langle ch!, 3\rangle^\frown\langle dh!, 5\rangle$, then no rules can be applied for $\mathtt{tt}_0 \|_{\{ch,dh\}} \mathtt{tt}_1$ as the values along channel dh are different although the heads of $\mathtt{tt}_0$ and $\mathtt{tt}_1$ are matched by [SynIO], i.e., $\mathtt{tt}_0$ and $\mathtt{tt}_1$ are uncomposable.

Example 3.9. Let $\mathtt{tt}_0 = \langle 1, \{ch?\}\rangle$ and $\mathtt{tt}_1 = \langle 1, \{ch!\}\rangle$, then no rules can be applied for $\mathtt{tt}_0 \|_{\{ch\}} \mathtt{tt}_1$, as it is impossible that the symmetric channel operations $ch?$ and $ch!$ are waiting simultaneously, indicating that $\mathtt{tt}_0$ and $\mathtt{tt}_1$ are uncomposable.

Example 3.10. Consider two infinite timed traces

$$\begin{aligned} \mathtt{tt}_0 &= \langle ch?, 0\rangle^\frown\langle ch?, 1\rangle^\frown \cdots \langle ch?, n\rangle^\frown \cdots \\ \mathtt{tt}_1 &= \tau^\frown\langle ch!, 0\rangle^\frown\tau^\frown\langle ch!, 1\rangle^\frown \cdots^\frown \tau^\frown\langle ch!, n\rangle^\frown \cdots \end{aligned}$$

The timed trace $\mathtt{tt}_0$ indicates that the process is receiving values along channel ch endlessly at current time. The timed trace $\mathtt{tt}_1$ indicates the infinite sending events at current time and before each sending event there is an internal action. Their parallel composition over ch is also infinite:

$$\mathtt{tt}_0 \|_{\{ch\}} \mathtt{tt}_1 \quad \rightsquigarrow \quad \tau^\frown\langle ch, 0\rangle^\frown\tau^\frown\langle ch, 1\rangle^\frown \cdots^\frown \tau^\frown\langle ch, n\rangle^\frown \cdots$$

by [τ-Act] and [SynIO]. It is a divergence as the time never progresses.

Example 3.11. Let $\mathtt{tt}_0 = \tau^\frown\langle 1, \{ch?\}\rangle^\frown\langle 2, \emptyset\rangle$ and $\mathtt{tt}_1 = \langle 2, \emptyset\rangle^\frown\tau^\frown\langle 1, \{ch!\}\rangle$. Then, their parallel composition via channel *ch* is

$$\mathtt{tt}_0 \|_{\{ch\}} \mathtt{tt}_1 \quad \rightsquigarrow \quad \tau^\frown\langle 1, \{ch?\}\rangle^\frown\langle 1, \emptyset\rangle^\frown\tau^\frown\langle 1, \{ch!\}\rangle$$

by [τ-Act], [SynWait] and **Additivity** of wait blocks (see Definition 3.4). Notice that in this example, the symmetric channel operations *ch*? and *ch*! appear in one timed trace. Similar to Example 3.6, *ch* is a common channel between $\mathtt{tt}_0$ and $\mathtt{tt}_1$ and therefore it does not connect to the environment. *ch*? and *ch*! appearing in one timed trace is the result from parallel composition, which indicates this trace denotes a parallel process, not a sequential one. Actually, we can construct the following parallel HCSP processes:

$$\begin{array}{rcl} \mathsf{P} & \widehat{=} & x := 0; \langle \dot{x} = 1 \& x < 1\rangle \trianglerighteq [\!](ch?x \rightarrow \mathbf{skip}); \mathbf{wait}(2) \\ \mathsf{Q} & \widehat{=} & \mathbf{wait}(2); y := 0; \langle \dot{y} = 1 \& y < 1\rangle \trianglerighteq [\!](ch!y \rightarrow \mathbf{skip}) \end{array}$$

The timed trace of P‖Q is exactly $\tau^\frown\langle 1, \{ch?\}\rangle^\frown\langle 1, \emptyset\rangle^\frown\tau^\frown\langle 1, \{ch!\}\rangle$ as shown above.

4 HYBRID PROCESSES

Following Hehner's philosophy "programs as predicates" [25], the first-order relational calculus of UTP [28] can be used to model programs specified with possibly different or heterogeneous programming styles. In this section, we introduce continuous dynamics into rational predicates to model hybrid systems, and use the term *hybrid processes* to denote such extended predicates. Hybrid processes serve as the basis of the HUTP theory in this paper.

To use UTP for modelling hybrid systems' behaviours, one first needs to extend it with an explicit notion of time. This can be done by introducing two observational variables $ti, ti' : \mathbb{R}_{\geq 0} \cup \{+\infty\}$ to specify the start- and end-time of the observed behaviour. Let $tr \in add \circ lz(\mathbb{TT})$, where $add \circ lz(\mathbb{TT})$ denotes the *set of canonical normal* traces, represent the timed trace before the process is started and $tr' \in add \circ lz(\mathbb{TT})$ stand for timed trace up to the moment of observation.

In a real-time setting, process state variables are interpreted as functions over time. To specify its real-time value, we use $\underset{\sim}{s}$ to represent it. In a word, we have three versions for each state variable s:

- the version $s \in \mathbb{D}$ stands for its initial value in the domain $\mathbb{D}$, i.e., the input state variable, where $\mathbb{D}$ could be a Banach space;
- the primed version $s' \in \mathbb{D}$ stands for the final value, i.e., the output state variable; and
- the real time version $\underset{\sim}{s} : [ti, ti') \to \mathbb{D}$ stands for its dynamic trajectory from the start time ti to the end time ti', and $\dot{\underset{\sim}{s}} : (ti, ti') \rightharpoonup \mathbb{D}$ is a partial function denoting the derivative of $\underset{\sim}{s}$.

We use the boldface symbols $\mathbf{s}$, $\mathbf{s}'$, $\underset{\sim}{\mathbf{s}}$ and $\dot{\underset{\sim}{\mathbf{s}}}$ to denote respective vectors of input, output, real-time state variables and the derivatives. The alphabet that our theory depends on is $\{ti, ti', tr, tr', \mathbf{s}, \underset{\sim}{\mathbf{s}}, \dot{\underset{\sim}{\mathbf{s}}}, \mathbf{s}'\}$ by default. Therefore, first-order predicate $P(\mathbf{x}, \mathbf{x}')$ introduced in Section 2 can be extended to high-order differential relation $\underset{\sim}{P}(ti, ti', tr, tr', \mathbf{s}, \underset{\sim}{\mathbf{s}}, \dot{\underset{\sim}{\mathbf{s}}}, \mathbf{s}')$. By introducing some healthiness conditions for high-order differential relations, we can then define hybrid processes.

4.1 Healthiness Conditions

As stated in Section 2, healthiness conditions play an important role in UTP. Therefore, we introduce the following healthiness conditions to identify the features of hybrid processes:

- Time must be irreversible, and trace should be monotonically increasing because processes are not permitted to undo past events, i.e.,

$$\mathcal{H}_0(X) \quad = \quad X \wedge ti \leq ti' \wedge tr \leq tr'$$

This healthiness condition is a variant of **R1** of reactive processes in [28] and [15]. Here, however, **R1** does not deal with time. Similar variants include **HC1** in [23] and **HCT1** in [17], but they do not involve the concept of traces.

- Trace should keep consistent with time:

$$\mathcal{H}_1(X) \quad = \quad X \wedge ti = \lim(tr) \wedge ti' = \lim(tr')$$

- If the preceding process does not terminate, i.e., $ti = +\infty \vee |tr| = +\infty$, the current process should do nothing but keep the time and trace observations unchanged, i.e.,

$$\mathcal{H}_2(X) \quad = \quad (ti = ti' \wedge tr = tr') \lhd (ti = +\infty \vee |tr| = +\infty) \rhd X$$

where $P \lhd b \rhd Q \mathrel{\widehat{=}} (b \wedge P) \vee (\neg b \wedge Q)$. This healthiness condition is similar to **R3** in [28] and $\mathbf{R3}_h$ in [15], except that we specify the auxiliary variable *wait* explicitly as $ti = +\infty \vee |tr| = +\infty$. Hence, unlike in classical UTP, variables *wait* and *wait*′ are not used in our HUTP.

- If the current process does not terminate, i.e., $ti' = +\infty \vee |tr'| = +\infty$, the values of the output state variables are unobservable, i.e.,

$$\mathcal{H}_3(X) \quad = \quad (\exists \mathbf{s}' \cdot X) \lhd (ti' = +\infty \vee |tr'| = +\infty) \rhd X$$

As stated above, $ti' = +\infty \vee |tr'| = +\infty$ specify variable *wait*′ in the traditional definition of reactive processes of the UTP, where *wait*′ being **true** denotes the observation that all primed variables stand for intermediate observations, not final ones. However, in our work, we hide the primed state variables by an existential quantifier when $ti' = +\infty \vee |tr'| = +\infty$, since observing the state values at time infinity makes no sense. The idea of $\mathcal{H}_3$ is similar to **R4** for a healthy process in [9], which expresses the intuition that "program variable state is not visible while waiting for external events".

- If the process evolves for a period of time, i.e., $ti < ti'$, the real-time value $\underset{\sim}{\mathbf{s}}$ should keep right-continuous and semi-differentiable, i.e.,

$$\mathcal{H}_4(X) \quad = \quad X \wedge RC \wedge SD$$

where

$$\begin{array}{lcl} RC & \mathrel{\widehat{=}} & \forall i \cdot \forall t \in [ti, ti') \cdot \underset{\sim}{\mathbf{s}}_i(t) = \underset{\sim}{\mathbf{s}}_i(t^+) \\ SD & \mathrel{\widehat{=}} & \forall i \cdot \forall t \in (ti, ti') \cdot \exists d_0, d_1 \cdot \dot{\underset{\sim}{\mathbf{s}}}_i(t^-) = d_0 \wedge \dot{\underset{\sim}{\mathbf{s}}}_i(t^+) = d_1 \end{array}$$

denote the right-continuity and semi-differentiability, respectively, and $\mathbf{s}_i$ ($\underset{\sim}{\mathbf{s}}_i$ and $\mathbf{s}_i'$) is the i-th variable in $\mathbf{s}$ ($\underset{\sim}{\mathbf{s}}$ and $\mathbf{s}'$). $\mathcal{H}_4$ rules out some ill behaviours, such as the Dirichlet function (returning 1 if t is a rational number and 0 otherwise) and the Weierstrass function (continuous everywhere but differentiable nowhere).

Definition 4.1 (Hybrid Process). A hybrid process HP is a fixed point of $X = \mathcal{H}_{\mathrm{HP}}(X)$, where

$$\mathcal{H}_{\mathrm{HP}} \quad \mathrel{\widehat{=}} \quad \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4$$

and the alphabet of HP is $\alpha(\mathsf{HP}) \mathrel{\widehat{=}} \{ti, ti', tr, tr', \mathbf{s}, \underset{\sim}{\mathbf{s}}, \dot{\underset{\sim}{\mathbf{s}}}, \mathbf{s}'\}$.

Since *wait* and *wait*′, used to denote reactivity in discrete models [15, 28], are interpreted respectively as $ti = +\infty \vee |tr| = +\infty$ and $ti' = +\infty \vee |tr'| = +\infty$ in $\mathcal{H}_2$ and $\mathcal{H}_3$, the hybrid processes defined by Definition 4.1 are essentially reactive.

Theorem 4.2 (Idempotence and Monotonicity). *$\mathcal{H}_{\mathrm{HP}}$ is idempotent and monotonic.*

4.2 Operations

4.2.1 Meet and Join. Let HP_0 and HP_1 be two hybrid processes. Non-determinism is simply modelled by logical disjunction:

$$\mathsf{HP}_0 \sqcap \mathsf{HP}_1 \quad \widehat{=} \quad \mathsf{HP}_0 \vee \mathsf{HP}_1$$

It stands for a process that internally chooses to execute either HP_0 or HP_1, regardless of the environment. The dual of non-determinism choice is conjunction:

$$\mathsf{HP}_0 \sqcup \mathsf{HP}_1 \quad \widehat{=} \quad \mathsf{HP}_0 \wedge \mathsf{HP}_1$$

The alphabets are expanded by $\sqcap$ and $\sqcup$, i.e., $\alpha(\mathsf{HP}_0 \sqcap \mathsf{HP}_1) = \alpha(\mathsf{HP}_0 \sqcup \mathsf{HP}_1) = \alpha(\mathsf{HP}_0) \cup \alpha(\mathsf{HP}_1)$. In the literature, $\sqcap$ is also called *demonic choice* or *meet*, while $\sqcup$ is called *angelic choice* or *join*. Generally, $\sqcap$ means $\wedge$ and $\sqcup$ means $\vee$, but in UTP [15, 28], they always denote $\vee$ and $\wedge$, respectively. Thus, in this paper, we will follow this notational convention.

Property 2. *If* HP_0 *and* HP_1 *are* $\mathcal{H}_{\mathrm{HP}}$*-healthy, then so are* $\mathsf{HP}_0 \sqcap \mathsf{HP}_1$ *and* $\mathsf{HP}_0 \sqcup \mathsf{HP}_1$.

4.2.2 Sequential Composition. The sequential composition in traditional UTP can be adapted to hybrid processes:

$$\mathsf{HP}_0 \fatsemi \mathsf{HP}_1 \quad \widehat{=} \quad \exists ti_*, \mathbf{s}_*, tr_* \cdot \mathsf{HP}_0[ti_*, \mathbf{s}_*, tr_*/ti', \mathbf{s}', tr'] \wedge \mathsf{HP}_1[ti_*, \mathbf{s}_*, tr_*/ti, \mathbf{s}, tr]$$

provided that $\alpha_{out}(\mathsf{HP}_0) = \alpha'_{in}(\mathsf{HP}_1)$, where $\alpha_{out}(\mathsf{HP}_0)$ and $\alpha_{in}(\mathsf{HP}_1)$ denote the sets of output and input variables (containing time and trace variables) in $\alpha(\mathsf{HP}_0)$ and $\alpha(\mathsf{HP}_1)$, respectively, and $\alpha'_{in}(\mathsf{HP}_1)$ is the primed version by priming all the variables in $\alpha_{in}(\mathsf{HP}_1)$. If $\alpha_{out}(\mathsf{HP}_0) \neq \alpha'_{in}(\mathsf{HP}_1)$, then we can extend the alphabets by $\alpha^+_{out}(\mathsf{HP}_0) = \alpha^{+\prime}_{in}(\mathsf{HP}_1) \mathrel{\widehat{=}} \alpha_{out}(\mathsf{HP}_0) \cup \alpha'_{in}(\mathsf{HP}_1)$ to ensure the well-definedness of sequential composition.

Property 3. *If* HP_0 *and* HP_1 *are* $\mathcal{H}_{\mathrm{HP}}$*-healthy, then so is* $\mathsf{HP}_0 \fatsemi \mathsf{HP}_1$.

With the definition of sequential composition, we can prove the following property for the healthiness condition $\mathcal{H}_3$ defined before.

Property 4. *The healthiness condition* $\mathcal{H}_3$ *is equivalent to*

$$X \quad = \quad X \fatsemi \mathsf{skip}$$

where $\mathsf{skip} \mathrel{\widehat{=}} \mathcal{H}_2(ti = ti' \wedge tr = tr' \wedge \mathbf{s} = \mathbf{s}')$.

4.2.3 Complement. The complement of a hybrid process HP is interpreted by

$$\neg_\imath \mathsf{HP} \quad \widehat{=} \quad \mathcal{H}_{\mathrm{HP}}(\neg \mathsf{HP})$$

Note that $\neg\mathsf{HP}$ is not $\mathcal{H}_{\mathrm{HP}}$-healthy, as it may violate constraints such as $tr \leq tr'$. Therefore, we enforce healthiness by $\mathcal{H}_{\mathrm{HP}}$, resulting in $\neg_\imath\mathsf{HP}$. Intuitively, $\neg_\imath\mathsf{HP}$ can be seen as the complement of HP in the context of hybrid processes, i.e., we can prove that $\neg_\imath\mathsf{HP} \sqcap \mathsf{HP} = \bot_{\mathrm{HP}}$ and $\neg_\imath\mathsf{HP} \sqcup \mathsf{HP} = \top_{\mathrm{HP}}$. Then we define implication by

$$\mathsf{HP}_0 \Rightarrow_\imath \mathsf{HP}_1 \quad \widehat{=} \quad \neg_\imath \mathsf{HP}_0 \vee \mathsf{HP}_1$$

These logic operators are similar to $\neg_r$ and $\Rightarrow_r$, used in [15].

4.2.4 Quantification. To support declaration, abstraction and instantiation of variables, as in [28], we introduce quantifications over three versions s, s' and $\underset{\sim}{s}$ of observable state variable s in a hybrid process HP. Existential quantifications of first-order variables s and s' are as usual:

$$\begin{array}{lcl} \exists s \cdot \mathsf{HP} & \widehat{=} & \bigvee \{\mathsf{HP}[d/s] \mid d \in \mathbb{D}\} \\ \exists s' \cdot \mathsf{HP} & \widehat{=} & \bigvee \{\mathsf{HP}[d/s'] \mid d \in \mathbb{D}\} \end{array}$$

The quantification of real-time variable $\underset{\sim}{s}$ is of higher-order and it will meanwhile quantifies the derivative of $\underset{\sim}{s}$, as follows:

$$\exists \underset{\sim}{s} \cdot \mathsf{HP} \quad \widehat{=} \quad \bigvee \left\{ \mathsf{HP}[f, g/\underset{\sim}{s}, \dot{\underset{\sim}{s}}] \;\middle|\; \begin{array}{c} f : [ti, ti') \to \mathbb{D} \qquad g : (ti, ti') \rightharpoonup \mathbb{D} \\ \forall t \in (ti, ti') \cdot \dot{f}(t^-) = g(t^-) \wedge \dot{f}(t^+) = g(t^+) \end{array} \right\}$$

Existential quantification removes the quantified variables from the alphabet of a process, i.e., $\alpha(\exists s \cdot \mathsf{HP}) = \alpha(\mathsf{HP}) \backslash \{s\}$, $\alpha(\exists s' \cdot \mathsf{HP}) = \alpha(\mathsf{HP}) \backslash \{s'\}$ and $\alpha(\exists \underset{\sim}{s} \cdot \mathsf{HP}) = \alpha(\mathsf{HP}) \backslash \{\underset{\sim}{s}, \dot{\underset{\sim}{s}}\}$. Dually, universal quantifications of s, s' and $\underset{\sim}{s}$ can be defined by

$$\begin{array}{lcl} \forall s \cdot \mathsf{HP} & \widehat{=} & \neg_! (\exists s \cdot \neg_! \mathsf{HP}) \\ \forall s' \cdot \mathsf{HP} & \widehat{=} & \neg_! (\exists s' \cdot \neg_! \mathsf{HP}) \\ \forall \underset{\sim}{s} \cdot \mathsf{HP} & \widehat{=} & \neg_! (\exists \underset{\sim}{s} \cdot \neg_! \mathsf{HP}) \end{array}$$

Since real-time variables are functions over time, such quantifications are of higher-order (functional). It can also be proven that quantifiers $\exists$ and $\forall$ are $\mathcal{H}_{\mathrm{HP}}$-preserving. In addition, higher-order $\exists$ and $\forall$ enjoy the following properties:

Property 5 (Quantification).

$$\begin{array}{rcl} \exists \underset{\sim}{x} \cdot (\mathsf{HP}_0 \fatsemi \mathsf{HP}_1) & = & (\exists \underset{\sim}{x} \cdot \mathsf{HP}_0) \fatsemi (\exists \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \forall \underset{\sim}{x} \cdot (\mathsf{HP}_0 \fatsemi \mathsf{HP}_1) & \sqsupseteq & (\forall \underset{\sim}{x} \cdot \mathsf{HP}_0) \fatsemi (\forall \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \exists \underset{\sim}{x} \cdot (\mathsf{HP}_0 \sqcap \mathsf{HP}_1) & = & (\exists \underset{\sim}{x} \cdot \mathsf{HP}_0) \sqcap (\exists \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \forall \underset{\sim}{x} \cdot (\mathsf{HP}_0 \sqcap \mathsf{HP}_1) & \sqsupseteq & (\forall \underset{\sim}{x} \cdot \mathsf{HP}_0) \sqcap (\forall \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \exists \underset{\sim}{x} \cdot (\mathsf{HP}_0 \sqcup \mathsf{HP}_1) & \sqsubseteq & (\exists \underset{\sim}{x} \cdot \mathsf{HP}_0) \sqcup (\exists \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \forall \underset{\sim}{x} \cdot (\mathsf{HP}_0 \sqcup \mathsf{HP}_1) & = & (\forall \underset{\sim}{x} \cdot \mathsf{HP}_0) \sqcup (\forall \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \neg_! \exists \underset{\sim}{x} \cdot \mathsf{HP} & = & \forall \underset{\sim}{x} \cdot \neg_! \mathsf{HP} \\ \neg_! \forall \underset{\sim}{x} \cdot \mathsf{HP} & = & \exists \underset{\sim}{x} \cdot \neg_! \mathsf{HP} \end{array}$$

Note that in the above the refinement order $\sqsubseteq$ for predicates introduced in Section 2 can be lifted to hybrid processes naturally. With the aid of higher-order quantifiers, the lexical locality of a variable s in a hybrid process HP can be specified by

$$\mathbf{local}\ s \cdot \mathsf{HP} \quad \widehat{=} \quad \exists s, \underset{\sim}{s}, s' \cdot \mathsf{HP}$$

Example 4.3. Consider a hybrid process with independent state variables x and y:

$$\mathsf{HP}_0 \quad \widehat{=} \quad \mathcal{H}_{\mathrm{HP}} \begin{pmatrix} ti < ti' \wedge tr' - tr = \langle ti' - ti, \emptyset \rangle \wedge \\ x = \underset{\sim}{x}(ti) \wedge x' = \underset{\sim}{x}(ti'^-) \wedge y = \underset{\sim}{y}(ti) \wedge y' = \underset{\sim}{y}(ti'^-) \\ \wedge \forall t \in (ti, ti') \cdot \dot{\underset{\sim}{x}}(t) = 1 \wedge \dot{\underset{\sim}{y}}(t) = 2 \end{pmatrix}$$

with $\alpha(\mathsf{HP}_0) = \{ti, ti', tr, tr', x, \underset{\sim}{x}, \dot{\underset{\sim}{x}}, x', y, \underset{\sim}{y}, \dot{\underset{\sim}{y}}, y'\}$. A literal abstraction of variable x from HP_0 is

$$\mathbf{local}\ x \cdot \mathsf{HP}_0 \quad = \quad \mathcal{H}_{\mathrm{HP}} \begin{pmatrix} ti < ti' \wedge tr' - tr = \langle ti' - ti, \emptyset \rangle \wedge \\ y = \underset{\sim}{y}(ti) \wedge y' = \underset{\sim}{y}(ti'^-) \wedge \forall t \in (ti, ti') \cdot \dot{\underset{\sim}{y}}(t) = 2 \end{pmatrix}$$

with the alphabet $\{ti, ti', tr, tr', y, \underset{\sim}{y}, \dot{\underset{\sim}{y}}, y'\}$.

Example 4.4. Consider another interesting example, let

$$\mathsf{HP}_1 \;\widehat{=}\; \mathcal{H}_{\mathrm{HP}}\begin{pmatrix} ti < ti' \wedge tr' - tr = \langle ti' - ti, \emptyset\rangle \wedge \\ x = \underset{\sim}{x}(ti) \wedge x' = \underset{\sim}{x}(ti'^{-}) \wedge y = \underset{\sim}{y}(ti) \wedge y' = \underset{\sim}{y}(ti'^{-}) \\ \wedge \forall t \in (ti, ti') \cdot \dot{\underset{\sim}{x}}(t) + \dot{\underset{\sim}{y}}(t) = 0 \end{pmatrix}$$

The abstraction of x from HP_1 is

$$\textbf{local}\ x \cdot \mathsf{HP}_1 \;=\; \mathcal{H}_{\mathrm{HP}}\left(ti < ti' \wedge y = \underset{\sim}{y}(ti) \wedge y' = \underset{\sim}{y}(ti'^{-}) \wedge tr' - tr = \langle ti' - ti, \emptyset\rangle\right)$$

with the alphabet $\{ti, ti', tr, tr', y, \underset{\sim}{y}, \dot{\underset{\sim}{y}}, y'\}$. Note that when x is localised, the ODE $\dot{\underset{\sim}{x}}(t) + \dot{\underset{\sim}{y}}(t) = 0$ is also abstracted away.

4.2.5 Parallel Composition. The parallel composition of two hybrid processes is written as

$$\mathsf{HP}_0 \|_M \mathsf{HP}_1$$

which uses the parallel-by-merge scheme developed in UTP [15, 28]. We assume that the state variables of concurrent processes are disjoint, i.e., $\alpha(\mathsf{HP}_0) \cap \alpha(\mathsf{HP}_1) = \{ti, ti', tr, tr'\}$. Let the notation HP_X make an X-version of HP by adding the shared outputted variables (ti' and tr') in HP with the X-subscript, i.e.,

$$\mathsf{HP}_X \;\widehat{=}\; \mathsf{HP} \fatsemi (ti = ti'_X \wedge tr = tr'_X \wedge \mathbf{s} = \mathbf{s}') \;=\; \mathsf{HP}[ti'_X, tr'_X / ti', tr']$$

Let $\mathbf{s}_0$ and $\mathbf{s}_1$ be state variables ($\mathbf{s}_0 \cap \mathbf{s}_1 = \emptyset$) owned by hybrid processes HP_0 and HP_1, respectively. Let $\mathbf{s} \,\widehat{=}\, \mathbf{s}_0 \uplus \mathbf{s}_1$. Then, the parallel composition of HP_0 and HP_1 can be defined as follows:

$$\mathsf{HP}_0 \|_M \mathsf{HP}_1 \;\widehat{=}\; \mathcal{H}_{\mathrm{HP}}\left((\mathsf{HP}_{0,X} \wedge \mathsf{HP}_{1,Y} \wedge tr = tr') \fatsemi M\right)$$

where M is a merge predicate with $\alpha(M) \,\widehat{=}\, \{ti_X, ti_Y, ti', tr_X, tr_Y, tr, tr', \mathbf{s}, \underset{\sim}{\mathbf{s}}, \dot{\underset{\sim}{\mathbf{s}}}, \mathbf{s}'\}$. Therefore, the alphabet of the parallel composition is $\alpha(\mathsf{HP}_0 \|_M \mathsf{HP}_1) \,\widehat{=}\, \{ti, ti', tr, tr', \mathbf{s}, \underset{\sim}{\mathbf{s}}, \dot{\underset{\sim}{\mathbf{s}}}, \mathbf{s}'\}$. The parallel-by-merge scheme is illustrated pictorially in Fig. 1.

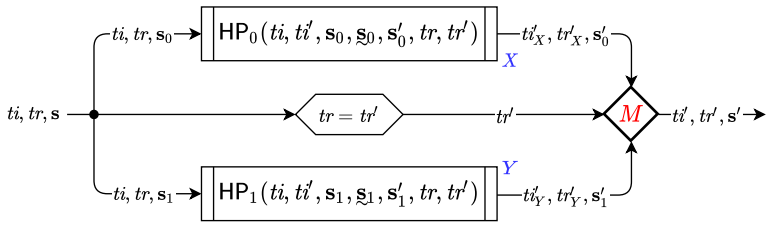

Fig. 1. Parallel-by-merge scheme ($\mathbf{s} = \mathbf{s}_0 \uplus \mathbf{s}_1$)

From the above figure, we can see that by renaming the common dashed variables, two processes $\mathsf{HP}_{0,X}$ and $\mathsf{HP}_{1,Y}$ execute independently and their respective outputs, as well as their common input trace tr, are fed into the merge predicate M. Then, M produces the merged result as the output of the parallel composition. Each merge predicate reflects a corresponding parallel scheme, i.e., the parallel composition is parametric over M. Fig. 1 in this paper looks like Fig. 1 which illustrates the parallel-by-merge dataflow in [15], because both of them are derived from the parallel-by-merge of the classic UTP of [28], but they differ in the following two aspects:

(1) Continuous state variables $\underset{\sim}{\mathbf{s}}_0$ and $\underset{\sim}{\mathbf{s}}_1$ are involved and state variables $\mathbf{s}_0$ and $\mathbf{s}_1$ are disjoint (no sharing state variables) in our setting;
(2) We only transfer the input trace tr to the merge predicate, as it would be necessary in parallel composition (see Definition 4.5 for an example).

A merge predicate M is *well-defined* if it satisfies the following three properties:

(1) $M = M[ti_X, tr_X, \mathbf{s}_0, ti_Y, tr_Y, \mathbf{s}_1 / ti_Y, tr_Y, \mathbf{s}_1, ti_X, tr_X, \mathbf{s}_0]$, indicating $\|_M$ is commutative.

(2) The end time of the parallel composition is infinite iff the end time of one process is infinite:

$$M \quad = \quad M \wedge (ti' = +\infty \lhd (ti_X = +\infty \vee ti_Y = +\infty) \rhd ti' < +\infty)$$

(3) Similarly, the timed trace of the parallel composition is infinite iff the timed trace of one process is infinite:

$$M \quad = \quad M \wedge (|tr'| = +\infty \lhd (|tr_X| = +\infty \vee |tr_Y| = +\infty) \rhd |tr'| < +\infty)$$

The following defines a well-defined merge predicate M_I that specifies parallelism.

Definition 4.5 (M_I). Let $\mathbf{s}_0$ and $\mathbf{s}_1$ be state variables ($\mathbf{s}_0 \cap \mathbf{s}_1 = \emptyset$) owned by HP_0 and HP_1, respectively, and I a set of channels via which HP_0 and HP_1 communicate. Then, we define

$$M_I \quad \widehat{=} \quad \mathsf{syn} \fatsemi \mathsf{mrg}$$

where

$$\begin{array}{lcl}
\mathsf{syn} & \widehat{=} & ti'_X = ti_X \wedge ti'_Y = ti_Y \wedge \mathbf{s}'_0 = \mathbf{s}_0 \wedge \mathbf{s}'_1 = \mathbf{s}_1 \wedge tr' = tr \wedge \\
& & \big(ti_X < ti_Y \Rightarrow \forall t \in [ti_X, ti_Y) \cdot \underset{\sim}{\mathbf{s}}_0(t) = \mathbf{s}_0 \wedge tr'_X = tr_X \frown \langle ti_Y - ti_X, \emptyset \rangle \wedge tr'_Y = tr_Y\big) \wedge \\
& & \big(ti_X > ti_Y \Rightarrow \forall t \in [ti_Y, ti_X) \cdot \underset{\sim}{\mathbf{s}}_1(t) = \mathbf{s}_1 \wedge tr'_X = tr_X \wedge tr'_Y = tr_Y \frown \langle ti_X - ti_Y, \emptyset \rangle\big) \wedge \\
& & \big(ti_X = ti_Y \Rightarrow tr'_X = tr_X \wedge tr'_Y = tr_Y\big) \\
\mathsf{mrg} & \widehat{=} & ti' = \max\{ti_X, ti_Y\} \wedge \mathbf{s}'_0 = \mathbf{s}_0 \wedge \mathbf{s}'_1 = \mathbf{s}_1 \wedge (tr'_X - tr) \|_I (tr'_Y - tr) \rightsquigarrow (tr' - tr)
\end{array}$$

The predicate syn synchronises the termination time ($+\infty$ if non-terminated) of two processes. If $ti_X < ti_Y$, then it delays the termination of the process tagged by X until it reaches t_Y. During the period of delay, the values of the corresponding state variables keep unchanged, and meanwhile the wait block $\langle ti_Y - ti_X, \emptyset \rangle$ should be added to the end of the trace tr_X. Following syn is mrg which merges times, states and traces, respectively. It uses the maximum time as the termination time ($+\infty$ if non-terminated) of the parallel composition ($ti' = \max\{ti_X, ti_Y\}$).

Property 6 (Well-Definedness). *M_I is well-defined.*

4.3 Complete Lattice

Property 7 (Monotonicity). *The operators $\fatsemi$, $\sqcup$, $\sqcap$, $\lhd\rhd$, $\exists$ and $\|_M$ are monotonic and $\neg_!$ is anti-monotone, with respect to the refinement order $\sqsubseteq$.*

A (finite or infinite) set $\mathcal{S}$ of hybrid processes equipped with the refinement order $\sqsubseteq$ forms a partially ordered set, denoted by $(\mathcal{S}, \sqsubseteq)$. Let $\bigsqcap \mathcal{S}$ the disjunction of all hybrid processes in $\mathcal{S}$, denoting a system that behaves as any of hybrid processes in $\mathcal{S}$. Obviously, $\mathsf{HP} \sqsubseteq \bigsqcap \mathcal{S}$ iff $\forall X \in \mathcal{S} \cdot \mathsf{HP} \sqsubseteq X$. Conversely, the conjunction of all hybrid processes in $\mathcal{S}$ is denoted by $\bigsqcup \mathcal{S}$, which denotes a system that only has the behaviour shared by all hybrid processes in $\mathcal{S}$. In lattice theory, $\bigsqcap \mathcal{S}$ and $\bigsqcup \mathcal{S}$ are called the infimum and supremum of a set $\mathcal{S}$, respectively.

Theorem 4.6 (Complete Lattice). *Let $\mathbb{HP}$ be the image of $\mathcal{H}_{\mathrm{HP}}$, i.e., it is the set of all hybrid processes, then it forms a complete lattice with top and bottom:*

$$\begin{array}{lcl}
\top_{\mathrm{HP}} & \widehat{=} & \bigsqcup \mathbb{HP} \quad = \quad \mathcal{H}_{\mathrm{HP}}(\mathbf{false}) \\
\bot_{\mathrm{HP}} & \widehat{=} & \bigsqcap \mathbb{HP} \quad = \quad \mathcal{H}_{\mathrm{HP}}(\mathbf{true})
\end{array}$$

Recursion can be specified once a complete lattice is formed. Theoretically speaking, recursion can be denoted by the fixed points of the equation $X = F(X)$, where F constructs the body of the recursion. If $F(X)$ is monotonic, the fixed points of $X = F(X)$ form a complete lattice according to Knaster-Tarski theorem [57]. The least fixed point is denoted by $\mu X \cdot F(X)$ and the greatest

fixed point by $\nu X \cdot F(X)$. The definitions of the least and greatest fixed points refer to Section 2. In particular, $\mu X \cdot X = \bot_{\mathrm{HP}}$ and $\nu X \cdot X = \top_{\mathrm{HP}}$.

4.4 Non-termination, Deadlock, Divergence and Miracle

Non-termination has a special property, i.e., being a left zero of sequential composition:

Definition 4.7 (Non-termination). A hybrid process $\underline{\mathsf{HP}}$ is non-terminated iff $\underline{\mathsf{HP}} \fatsemi \mathsf{HP} = \underline{\mathsf{HP}}$ for any hybrid process HP.

A hybrid process is *divergent* if its execution time is finite but its trace is infinite; and a *deadlock* if its execution time is infinite with some channel operations waiting forever, specified as follows. It can be demonstrated that both divergence and deadlock are non-terminated by Definition 4.7.

Definition 4.8 (Divergence). A divergence is a fixed point of $X = \mathcal{H}_{\mathrm{HP}} \circ DIV(X)$ where

$$DIV(X) \quad = \quad X \wedge ti' < +\infty \wedge |tr'| = +\infty$$

Definition 4.9 (Deadlock). A deadlock is a fixed point of $X = \mathcal{H}_{\mathrm{HP}} \circ DL(X)$ where

$$DL(X) \quad = \quad X \wedge ti' = +\infty \wedge \exists N \in \mathbb{N} \cdot \forall n \geq N \cdot tr'(n) = \langle \delta, RS \rangle \wedge RS \neq \emptyset$$

A *miracle* is a miraculous process that only occurs after a non-terminating hybrid process, and thus it is impossible to implement a miracle in reality. Formally,

Definition 4.10 (Miracle). A miracle is a fixed point of $X = \mathcal{H}_{\mathrm{HP}} \circ MIR(X)$ where

$$MIR(X) \quad = \quad X \wedge (ti = +\infty \vee |tr| = +\infty)$$

Theorem 4.11 (Miracle). $\top_{\mathrm{HP}}$ *is the only miracle of hybrid processes.*

Miracle refines any hybrid process HP, i.e., $\mathsf{HP} \sqsubseteq \top_{\mathrm{HP}}$, and it enjoys the following two desired properties, where Property 8 indicates miracle is non-terminated by Definition 4.7.

Property 8 (Left Zero of $\fatsemi$). $\top_{\mathrm{HP}} \fatsemi \mathsf{HP} = \top_{\mathrm{HP}}$ *for any hybrid process* HP.

Property 9 (Zero of $\|_M$). $\mathsf{HP} \|_M \top_{\mathrm{HP}} = \top_{\mathrm{HP}} \|_M \mathsf{HP} = \top_{\mathrm{HP}}$ *for any hybrid process* HP.

In contrast to the miracle, $\bot_{\mathrm{HP}}$ is the most unpredictable behaviour, because it allows non-deterministically choosing a behaviour among possibly infinitely many processes; in other words, *chaos*. Generally, chaos is expected to be non-terminated (left-zero of sequential composition) and the parallel composition of any process with chaos should result in chaos (zero of parallel composition). However, the chaos $\bot_{\mathrm{HP}}$ does not enjoy such desired properties as the miracle $\top_{\mathrm{HP}}$ does. This problem will be solved in Section 5.

4.5 Abstract Hybrid Processes

A hybrid process models the state and trace of a system over time to accurately render its concrete behavior. For the purpose of verification, however, such a level of detail is certainly excessive, as instead modular abstraction may be desired. To address this issue, we propose the notion of *abstract hybrid process*, where traces are simply abstracted. Formally,

Definition 4.12 (Abstract Hybrid Process). An abstract hybrid process is a fixed point of

$$X \quad = \quad \mathcal{H}_0^{\mathrm{A}} \circ \mathcal{H}_2^{\mathrm{A}} \circ \mathcal{H}_3^{\mathrm{A}} \circ \mathcal{H}_4(X)$$

where

$$\begin{array}{lcl} \mathcal{H}_0^{\mathrm{A}}(X) & \widehat{=} & X \wedge ti \leq ti' \\ \mathcal{H}_2^{\mathrm{A}}(X) & \widehat{=} & (ti = ti') \lhd ti = +\infty \rhd X \\ \mathcal{H}_3^{\mathrm{A}}(X) & \widehat{=} & (\exists \mathbf{s}' \cdot X) \lhd ti' = +\infty \rhd X \end{array}$$

For brevity, we use $\mathcal{H}_{\mathrm{HP}}^{\mathrm{A}}$ to denote $\mathcal{H}_0^{\mathrm{A}} \circ \mathcal{H}_2^{\mathrm{A}} \circ \mathcal{H}_3^{\mathrm{A}} \circ \mathcal{H}_4$.

Theorem 4.13 (Idempotence and Monotonicity). *$\mathcal{H}^{A}_{HP}$ is idempotent and monotonic.*

Note that $\mathcal{H}^{A}_{0}$, $\mathcal{H}^{A}_{2}$ and $\mathcal{H}^{A}_{3}$ are abstract versions of $\mathcal{H}_0$, $\mathcal{H}_2$ and $\mathcal{H}_3$ in Definition 4.1, respectively. A predicate is an abstract hybrid process iff it is $\mathcal{H}^{A}_{HP}$-healthy. Compared with Definition 4.1 of $\mathcal{H}_{HP}$, abstract hybrid processes only consider time (ti and ti') and state ($\mathbf{s}$, $\underset{\sim}{\mathbf{s}}$, $\dot{\underset{\sim}{\mathbf{s}}}$ and $\mathbf{s}'$), ignoring the timed trace (tr and tr'). Therefore, $\mathcal{H}_1$ is no longer necessary here. By Theorem 4.13, abstract hybrid processes can form a complete lattice with top and bottom

$$\begin{array}{lcll} \top_{AHP} & \widehat{=} & \mathcal{H}^{A}_{HP}(\mathbf{false}) & \equiv \quad ti = ti' = +\infty \\ \bot_{AHP} & \widehat{=} & \mathcal{H}^{A}_{HP}(\mathbf{true}) & \equiv \quad (ti = ti') \lhd ti = +\infty \rhd (ti \le ti' \wedge RC \wedge SD) \end{array}$$

Clearly, $\top_{AHP} \sqsubseteq \top_{HP}$ and $\bot_{AHP} \sqsubseteq \bot_{HP}$ if we ignore the alphabets as $\top_{AHP}$ and $\bot_{AHP}$ do not contain tr or tr'. Abstract hybrid processes also form a Boolean algebra $(\mathbb{AHP}, \sqcap, \sqcup, \neg^{A}_{;}, \top_{AHP}, \bot_{AHP})$, where $\mathbb{AHP}$ is the image of $\mathcal{H}^{A}_{HP}$ and the negation $\neg^{A}_{;}$ can be defined by $\neg^{A}_{;} X \mathrel{\widehat{=}} \mathcal{H}^{A}_{HP}(\neg X)$. Furthermore, *an abstract hybrid condition* is a fixed point of $\neg^{A}_{;} X = \neg^{A}_{;} X \fatsemi \bot_{AHP}$. The equation can be denoted by a healthiness condition $\mathcal{H}^{A}_{HC}$, i.e., a predicate is an abstract hybrid condition iff it is $\mathcal{H}^{A}_{HC}$-healthy. Abstract hybrid conditions are a subset of abstract hybrid processes.

Additionally, by definition of the refinement relation, for a hybrid process HP with explicit timed traces, if $P \sqsubseteq \mathsf{HP}$ (ignoring the alphabets) then HP does satisfy the property P, which is represented by an abstract hybrid process. Hence, thanks to the lattice structure induced by $\sqsubseteq$, we can define an (α, γ) Galois-connection stipulating the concretisation of a property P to be the meet of all hybrid processes that satisfy it, i.e., $\gamma(P) \mathrel{\widehat{=}} \bigsqcap\{\mathsf{HP} \mid P \sqsubseteq \mathsf{HP}\}$ and, dually, the abstraction of a hybrid process HP to be the join of all properties satisfied by it, i.e., $\alpha(\mathsf{HP}) \mathrel{\widehat{=}} \bigsqcup\{P \mid P \sqsubseteq \mathsf{HP}\}$. In particular, $\gamma(\bot_{AHP}) = \bot_{HP}$ and $\alpha(\top_{HP}) = \top_{AHP}$.

5 HYBRID DESIGNS

In this section, we propose the concept of *hybrid design*. Hybrid designs are a subset of hybrid processes but enjoy some desired algebraic properties which are not featured by hybrid processes. Based on hybrid designs, we will then propose *normal hybrid designs*, which are the first-class notions in the HUTP theory.

The notion of design in UTP was first introduced by Hoare and He in [28]. A design consists of a pair of predicates (Pre, $Post$) standing for the pre- and post-condition of a program, augmented with the Boolean variables ok and ok' to express whether the program has started and terminated:

$$Pre \vdash Post \quad \widehat{=} \quad (ok \wedge Pre) \Rightarrow (ok' \wedge Post) \quad \equiv \quad \neg ok \vee \neg Pre \vee (ok' \wedge Post)$$

which states that if the program starts in a state satisfying the pre-condition Pre, it will terminate, and on termination the post-condition $Post$ will be true. The pair (Pre, $Post$) describes a contract between the component and its environment, and therefore supports the decomposition of engineering tasks to resolve system design complexity.

In related works, the meanings of ok and ok' are slightly different. For example, in [15], ok and ok' are used to indicate divergence. Here, however, since divergence can be observed from time and trace ($ti' < +\infty \wedge |tr'| = +\infty$), ok and ok' have different meanings. We extend the notion of design to hybrid systems by interpreting ok and ok' to denote a process's status: $ok' = \mathbf{true}$ indicates that the process executes normally (not necessarily terminating), and hence $ok = \mathbf{false}$ means that the preceding process executed abnormally, causing the behaviour of the current process to be unpredictable. Thus, it is necessary to distinguish normal and abnormal non-termination. Specifically, $\neg ok'$ indicates the abnormal behaviour, which should be *abnormal non-termination*; $ok' \wedge (ti' = +\infty \vee |tr'| = +\infty)$ indicates *normal non-termination*; and $ok' \wedge ti' < +\infty \wedge |tr'| < +\infty$ indicates (normal) termination.

Definition 5.1 (Hybrid Designs). Let HP_A and HP_C be hybrid processes. Then,

$$\mathsf{HP}_A \vdash \mathsf{HP}_C \quad \widehat{=} \quad (ok \wedge \mathsf{HP}_A) \Rightarrow_\imath (ok' \wedge \mathsf{HP}_C)$$

is a *hybrid design* with the alphabet $\alpha(\mathsf{HP}_A) \cup \alpha(\mathsf{HP}_C) \cup \{ok, ok'\}$, where HP_A and HP_C are called *assumption* and *commitment*, respectively.

Hybrid designs are a subset of hybrid processes, as $ok \wedge \mathsf{HP}_A$ and $ok' \wedge \mathsf{HP}_C$ are $\mathcal{H}_{\mathrm{HP}}$-healthy and $\Rightarrow_\imath$ is $\mathcal{H}_{\mathrm{HP}}$-preserving, which means $\mathsf{HP}_A \vdash \mathsf{HP}_C$ is also $\mathcal{H}_{\mathrm{HP}}$-healthy. Intuitively, when a hybrid design starts normally (ok = **true**) and its assumption HP_A holds, then it executes normally and the commitment HP_C holds. The hybrid design $\bot_{\mathrm{HP}} \vdash \mathsf{HP}_C$, where $\bot_{\mathrm{HP}}$ is a variant of **true** (which is a convention in UTP), is abbreviated as $\vdash \mathsf{HP}_C$ meaning not making any assumptions.

5.1 Matrix Representation

Given a hybrid design $\mathsf{HP}_A \vdash \mathsf{HP}_C$, if ok is **false**, i.e., its predecessor is abnormal, then it will behave unpredictably, i.e., chaos $\bot_{\mathrm{HP}}$; if ok is **true** but ok' is **false**, it indicates that the assumption HP_A must be violated, i.e., $\neg_\imath \mathsf{HP}_A$; if both ok and ok' are **true**, the commitment HP_C can be guaranteed as soon as the assumption HP_A holds, i.e., $\mathsf{HP}_A \Rightarrow_\imath \mathsf{HP}_C$. In order to make this semantics of hybrid designs clearer, we use the matrix encoding proposed in [43] to define hybrid designs as 2×2 matrices by instantiating ok and ok' with all four possible combinations:

$$\mathsf{HP}_A \vdash \mathsf{HP}_C \quad \widehat{=} \quad \begin{array}{c} \begin{array}{cc} \neg ok' & ok' \end{array} \\ \begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \neg_\imath \mathsf{HP}_A & \mathsf{HP}_A \Rightarrow_\imath \mathsf{HP}_C \end{pmatrix} \end{array} \begin{array}{c} \neg ok \\ ok \end{array} \tag{2}$$

With the aid of the matrix representation, the proofs of the theorems and properties on hybrid designs become more concise and intuitive. The proofs of this section are largely based on this matrix representation and the calculus on it (see the appendix for the details).

5.2 Normal Hybrid Designs

In general, UTP allows the assumption to refer to both primed and unprimed versions of variables. However, as explained in [21], only *normal designs*, whose assumptions are *conditions* and do not refer to primed (state) variables, are significant. Similarly, only *normal hybrid designs*, whose assumptions are fixed points of the equation below, make sense in hybrid system design.

$$\neg_\imath X \quad = \quad \neg_\imath X \fatsemi \bot_{\mathrm{HP}}$$

Intuitively, the above equation indicates that if an execution violates an assumption, then we expect that any extension of the execution will also violate the assumption. For example, let HP be a hybrid process specifying the expected behaviour, then $\neg_\imath \mathsf{HP}$ represents the unexpected behaviour. It is desirable that all the extensions of $\neg_\imath \mathsf{HP}$ should also be unexpected, i.e., $\neg_\imath \mathsf{HP} \sqsubseteq \neg_\imath \mathsf{HP} \fatsemi \bot_{\mathrm{HP}}$. Meanwhile, we have $\neg_\imath \mathsf{HP} \fatsemi \bot_{\mathrm{HP}} \sqsubseteq \neg_\imath \mathsf{HP}$ according to the monotonicity of $\fatsemi$ (Property 7). In a word, we expect the property $\neg_\imath \mathsf{HP} = \neg_\imath \mathsf{HP} \fatsemi \bot_{\mathrm{HP}}$.

The above equation can be denoted by a healthiness condition

$$\mathcal{H}_{\mathrm{HC}}(X) \quad = \quad \neg_\imath(\neg_\imath X \fatsemi \bot_{\mathrm{HP}})$$

A predicate is called a *hybrid condition* if it is $\mathcal{H}_{\mathrm{HC}}$-healthy in a similar manner as reactive conditions in [15]. Hybrid conditions are a subset of hybrid processes. For example, $\mathcal{H}_{\mathrm{HP}}(s > 0)$ is a hybrid condition while $\mathcal{H}_{\mathrm{HP}}(s' > 0)$ is not. Especially, $\bot_{\mathrm{HP}}$ and $\top_{\mathrm{HP}}$ are hybrid conditions because it is easy to check that $\bot_{\mathrm{HP}} = \neg_\imath(\neg_\imath \bot_{\mathrm{HP}} \fatsemi \bot_{\mathrm{HP}})$ and $\top_{\mathrm{HP}} = \neg_\imath(\neg_\imath \top_{\mathrm{HP}} \fatsemi \bot_{\mathrm{HP}})$.

A hybrid design, say $\mathsf{HC} \vdash \mathsf{HP}$, is called *normal hybrid design*, if HC is a hybrid condition ($\mathcal{H}_{\mathrm{HC}}$-healthy). In what follows, we only consider normal hybrid designs. Operations on normal hybrid designs can be thus reduced to standard matrix operations. In particular, non-deterministic choice (meet) $\sqcap$ and sequential composition $;$ correspond to matrix addition and multiplication, respectively. Moreover, reasoning is completely component-free (with no mention to ok and ok' or variable substitutions), i.e., *compositional*.

Theorem 5.2 (Operations).

(1) $(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \sqcap (\mathsf{HC}_1 \vdash \mathsf{HP}_1) = (\mathsf{HC}_0 \wedge \mathsf{HC}_1) \vdash (\mathsf{HP}_0 \vee \mathsf{HP}_1)$
(2) $(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \sqcup (\mathsf{HC}_1 \vdash \mathsf{HP}_1) = (\mathsf{HC}_0 \vee \mathsf{HC}_1) \vdash ((\mathsf{HC}_0 \Rightarrow_\iota \mathsf{HP}_0) \wedge (\mathsf{HC}_1 \Rightarrow_\iota \mathsf{HP}_1))$
(3) $(\mathsf{HC}_0 \vdash \mathsf{HP}_0) ; (\mathsf{HC}_1 \vdash \mathsf{HP}_1) = (\mathsf{HC}_0 \wedge \neg_\iota(\mathsf{HP}_0 ; \neg_\iota \mathsf{HC}_1)) \vdash (\mathsf{HP}_0 ; \mathsf{HP}_1)$

Parallel-by-merge, as defined for hybrid processes in Section 4.2.5, can naturally be extended to normal hybrid designs, except that the form of merge predicates should be adapted to deal with auxiliary variables ok and ok'. Concretely, the merge predicate for normal hybrid designs can be defined by:

$$\mathrm{NHD}(M) \quad \widehat{=} \quad (ok_X \wedge ok_Y) \Rightarrow_\iota (M \wedge ok')$$

where $\alpha(M) \cap \{ok_X, ok_Y, ok, ok'\} = \emptyset$. It states that only parallel designs execute normally (i.e. $ok_X \wedge ok_Y$) can they merge by M as usual. Otherwise, merge becomes unpredictable.

Theorem 5.3 (Parallel Composition).

$$(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \|_{\mathrm{NHD}(M)} (\mathsf{HC}_1 \vdash \mathsf{HP}_1) \quad = \quad \begin{pmatrix} \neg_\iota(\neg_\iota \mathsf{HC}_0 \|_{\perp_{\mathrm{HP}}} \neg_\iota \mathsf{HC}_1) \\ \wedge \neg_\iota(\neg_\iota \mathsf{HC}_0 \|_{\perp_{\mathrm{HP}}} \mathsf{HP}_1) \\ \wedge \neg_\iota(\neg_\iota \mathsf{HC}_1 \|_{\perp_{\mathrm{HP}}} \mathsf{HP}_0) \end{pmatrix} \vdash \mathsf{HP}_0 \|_M \mathsf{HP}_1$$

Intuitively, the assumption of parallel composition, above, stipulates that unexpected interactions cannot happen. For example, $\neg_\iota \mathsf{HC}_0$ denotes the unexpected behaviour of $\mathsf{HC}_0 \vdash \mathsf{HP}_0$. Hence, its parallel composition with the behaviour of the other side, i.e., $\neg_\iota \mathsf{HC}_0 \|_{\perp_{\mathrm{HP}}} \neg_\iota \mathsf{HC}_1$ and $\neg_\iota \mathsf{HC}_0 \|_{\perp_{\mathrm{HP}}} \mathsf{HP}_1$, should also be unexpected, as negated by the assumption, where the merge predicate $\perp_{\mathrm{HP}}$ denotes the weakest parallelism, i.e., $(\mathsf{P} \|_{\perp_{\mathrm{HP}}} \mathsf{Q}) \sqsubseteq (\mathsf{P} \|_M \mathsf{Q})$ for any merge predicate M.

If state variables play the dominant role in the assumptions and traces are not the concern, such as $\mathcal{H}_{\mathrm{HP}}(s > 0)$, we can give an alternative definition to parallel composition as follows, which is simpler than $\|_{\mathrm{NHD}(M)}$.

Definition 5.4 (Parallel Composition).

$$(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \|_M^{\mathrm{HP}} (\mathsf{HC}_1 \vdash \mathsf{HP}_1) \quad \widehat{=} \quad (\mathsf{HC}_0 \wedge \mathsf{HC}_1) \vdash (\mathsf{HP}_0 \|_M \mathsf{HP}_1)$$

For normal hybrid designs, the assumption for non-chaos is made explicitly. Since parallel execution involves the evaluation of both processes, it is necessary to ensure that both their assumptions are valid to start with. Thus, the assumption for successful execution of the parallel composition is the conjunction of their separate assumptions rather than their disjunction. It states that the parallel composition $\mathsf{HP}_0 \|_M^{\mathrm{HP}} \mathsf{HP}_1$ can only be executed when both of the components successfully start with the assumptions HC_0 and HC_1 holding, otherwise, it is a chaos. Note that $\|_M^{\mathrm{HP}}$ is different from $\|_{\mathrm{NHD}(M)}$, the former composes assumptions and commitments separately while the latter treats hybrid designs as hybrid processes.

Theorem 5.5 (Closure). *Normal hybrid designs are closed on* $\sqcap$, $\sqcup$, $;$, $\|_{\mathrm{NHD}(M)}$ *and* $\|_M^{\mathrm{HP}}$.

Theorem 5.6 (Refinement). $(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \sqsubseteq (\mathsf{HC}_1 \vdash \mathsf{HP}_1)$ *iff*

$$\mathsf{HC}_1 \sqsubseteq \mathsf{HC}_0 \quad \textit{and} \quad \mathsf{HP}_0 \sqsubseteq (\mathsf{HC}_0 \wedge \mathsf{HP}_1)$$

This theorem indicates that the monotonic hybrid refinement relation can be lifted to a contravariant design refinement relation, and hence the monotonicity of modalities for normal hybrid designs can be proved, specified as the following two properties.

Property 10 (Contra-variance). *If* $\mathsf{HC}_0 \sqsupseteq \mathsf{HC}_1$ *and* $\mathsf{HP}_0 \sqsubseteq \mathsf{HP}_1$ *then*

$$(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \sqsubseteq (\mathsf{HC}_1 \vdash \mathsf{HP}_1)$$

Property 11 (Monotonicity). *Operators* $\sqcup$, $\sqcap$, $;$, $\|_{\mathrm{NHD}(M)}$ *and* $\|_M^{HP}$ *for normal hybrid designs are monotonic with respect to* $\sqsubseteq$.

5.3 Complete Lattice

The laws for $\sqcap$ and $\sqcup$ can be respectively generalised to $\bigsqcap$ and $\bigsqcup$, which indicates that normal hybrid designs form a complete lattice, as stated by the following theorem.

Theorem 5.7 (Complete Lattice). *Normal hybrid designs form a complete lattice with top*

$$\top_{\mathrm{NHD}} \;\widehat{=}\; \bot_{\mathrm{HP}} \vdash \top_{\mathrm{HP}}$$

and bottom

$$\bot_{\mathrm{NHD}} \;\widehat{=}\; \top_{\mathrm{HP}} \vdash \bot_{\mathrm{HP}}$$

Property 12 (Chaos). $\bot_{\mathrm{NHD}} = \top_{\mathrm{HP}} \vdash \mathsf{HP}$ *for any hybrid process* HP.

Property 13 (Non-termination). *For any normal hybrid design* NHD,

$$\bot_{\mathrm{NHD}} ; \mathsf{NHD} = \bot_{\mathrm{NHD}} \quad \textit{and} \quad \top_{\mathrm{NHD}} ; \mathsf{NHD} = \top_{\mathrm{NHD}}$$

Property 14 (Parallel Composition). *For any normal hybrid design* NHD,

$$\mathsf{NHD} \|_{\mathrm{NHD}(M)} \top_{\mathrm{NHD}} \;=\; \top_{\mathrm{NHD}} \|_{\mathrm{NHD}(M)} \mathsf{NHD} \;=\; \top_{\mathrm{NHD}}$$

Property 15 (Parallel Composition). *For any normal hybrid design* NHD,

$$\mathsf{NHD} \|_M^{\mathrm{HP}} \bot_{\mathrm{NHD}} \;=\; \bot_{\mathrm{NHD}} \|_M^{\mathrm{HP}} \mathsf{NHD} \;=\; \bot_{\mathrm{NHD}}$$

The above conclusions demonstrate that the chaos $\bot_{\mathrm{NHD}}$ of normal hybrid designs enjoys the desired properties that (1) it is non-terminated (left-zero of sequential composition, see Property 13) and (2) its parallel composition with any normal hybrid design can result in itself (zero of parallel composition $\|_M^{\mathrm{HP}}$, see Property 15), which fixes the hole in the HUTP theory mentioned at the end of Section 4 (where the chaos $\bot_{\mathrm{HP}}$ of hybrid processes does not enjoy such desired properties).

6 REFLECTION OF HCSP AND SIMULINK WITH HUTP

In this section, we give the HUTP semantics of HCSP [64] and Simulink [39], representing two representative imperative and data-flow formalisms for hybrid system design in academia and industry, respectively. Furthermore, we prove the consistency between the operational semantics of HCSP and its HUTP semantics. We illustrate by an example the refinement relation between a Simulink diagram and the corresponding HCSP model within the HUTP framework, demonstrating the expressive capabilities of the UTP as a meta-theory for translation validation.

6.1 Syntactic Sugar

For brevity, we introduce the following syntax sugar for HUTP. We define the notations $[\![\cdot]\!]$ and $\lceil\cdot\rfloor$ which are similar to $[\![\cdot]\!]$ and $\lceil\cdot\rceil$ of Duration Calculus [66]. Let $P(\underset{\sim}{\mathbf{s}}, \dot{\underset{\sim}{\mathbf{s}}})$ denote a predicate relating $\underset{\sim}{\mathbf{s}}$ and its derivative $\dot{\underset{\sim}{\mathbf{s}}}$.

$$[\![P(\underset{\sim}{\mathbf{s}}, \dot{\underset{\sim}{\mathbf{s}}}), RS]\!]_{\phi(ti,ti')} \;\widehat{=}\; \mathcal{H}_{\mathrm{HP}}\begin{pmatrix} ti < ti' < +\infty \wedge \phi(ti, ti') \wedge \mathbf{s} = \underset{\sim}{\mathbf{s}}(ti) \wedge \mathbf{s}' = \underset{\sim}{\mathbf{s}}(ti'^{-}) \wedge \\ \forall t \in (ti, ti') \cdot P(\underset{\sim}{\mathbf{s}}(t), \dot{\underset{\sim}{\mathbf{s}}}(t)) \wedge tr' - tr = \langle ti' - ti, RS \rangle \end{pmatrix}$$

is a finite continuous process over the time interval $[ti, ti')$, and it says that $\underset{\sim}{\mathbf{s}}$ is differentiable with $P(\underset{\sim}{\mathbf{s}}, \underset{\sim}{\dot{\mathbf{s}}})$ holding at every instant t from ti to ti'. During the period, the channel operations in the ready set RS are waiting, recorded by the trace history $tr' - tr$. The predicate $\phi(ti, ti')$ characterises the relation of ti and ti', such as $ti' - ti = 2$. We omit $\phi(ti, ti')$ if $\phi(ti, ti') = (ti < ti')$. For brevity,

$$\begin{array}{lcl} [\![P(\underset{\sim}{\mathbf{s}}, \underset{\sim}{\dot{\mathbf{s}}}), RS]\!]_d & \widehat{=} & [\![P(\underset{\sim}{\mathbf{s}}, \underset{\sim}{\dot{\mathbf{s}}}), RS]\!]_{ti'-ti=d} \\ [\![P(\underset{\sim}{\mathbf{s}}, \underset{\sim}{\dot{\mathbf{s}}}), RS]\!]_{\sim d} & \widehat{=} & [\![P(\underset{\sim}{\mathbf{s}}, \underset{\sim}{\dot{\mathbf{s}}}), RS]\!]_{(ti'-ti)\sim d} \end{array}$$

where $d > 0$ and $\sim \in \{<, \leq, \neq, >, \geq\}$.

A sequence of operations which is assumed to take no time is called *super-dense computation* [38]. Under super-dense computation, the computer processing the sequence of the operations is much faster than the physical devices attached to it, rendering the time to compute the discrete operations negligible. However, the temporal order of computations is still present. Under the assumption of super-dense computation, a discrete process without communication is defined by

$$\lceil P(\mathbf{s}, \mathbf{s}') \rfloor \quad \widehat{=} \quad \mathcal{H}_{\mathrm{HP}}(ti = ti' < +\infty \wedge P(\mathbf{s}, \mathbf{s}') \wedge tr' - tr = \tau)$$

It executes instantly at time $ti = ti' < +\infty$, rather than continuously over a time interval. Especially,

$$\lceil \rfloor \quad \widehat{=} \quad \mathcal{H}_{\mathrm{HP}}(ti = ti' < +\infty \wedge \mathbf{s} = \mathbf{s}' \wedge tr = tr')$$

Note that $\lceil \rfloor \neq \lceil \mathbf{s} = \mathbf{s}' \rfloor$ as their traces are different. We can prove the following property for $\lceil \rfloor$.

Property 16 (Unit). *$\lceil \rfloor$ is a unit of hybrid processes w.r.t. $\fatsemi$.*

Corollary 1 (Unit). *$\vdash \lceil \rfloor$ is a unit of normal hybrid designs w.r.t. $\fatsemi$.*

Instant communications can be defined by discrete processes with communication:

$$\begin{array}{lcl} \lceil ch?, \mathbf{s}_i \rfloor & \widehat{=} & \mathcal{H}_{\mathrm{HP}}(ti = ti' < +\infty \wedge \exists d \cdot \mathbf{s}_i := d \wedge tr' - tr = \langle ch?, d \rangle) \\ \lceil ch!, e(\mathbf{s}) \rfloor & \widehat{=} & \mathcal{H}_{\mathrm{HP}}(ti = ti' < +\infty \wedge \mathbf{s}' = \mathbf{s} \wedge tr' - tr = \langle ch!, e(\mathbf{s}) \rangle) \end{array}$$

where $\mathbf{s}_i$ is the i-th variable in $\mathbf{s}$ and $e(\mathbf{s})$ is an expression containing $\mathbf{s}$.

In addition, we also define infinite continuous processes:

$$[\![P(\underset{\sim}{\mathbf{s}}, \underset{\sim}{\dot{\mathbf{s}}}), RS]\!]_\infty \quad \widehat{=} \quad \mathcal{H}_{\mathrm{HP}}\begin{pmatrix} ti < ti' = +\infty \wedge \mathbf{s} = \underset{\sim}{\mathbf{s}}(ti) \wedge \\ \forall t \in (ti, +\infty) \cdot P(\underset{\sim}{\mathbf{s}}(t), \underset{\sim}{\dot{\mathbf{s}}}(t)) \\ \wedge tr' - tr = \langle +\infty, RS \rangle \end{pmatrix}$$

Since we cannot observe the state variables of the point at infinity, the process will not output any values of state variables, i.e., the output state variables $\mathbf{s}'$ are hidden.

6.2 From HCSP to HUTP

HCSP is a formal language for describing hybrid systems. It extends Communicating Sequential Processes by introducing differential equations for modelling continuous evolutions and interrupts for modelling arbitrary interactions between continuous evolutions and discrete jumps. The syntax of HCSP is given as follows, adapted from [64].

$$\begin{array}{lcll} \mathrm{P} & ::= & \textbf{skip} \mid x := e \mid \textbf{wait}\ d \mid ch?x \mid ch!e \mid X \mid \mathrm{rec}\ X \cdot \mathrm{P} \mid \mathrm{P}; \mathrm{P} \mid \mathrm{P} \sqcap \mathrm{P} \mid B \rightarrow \mathrm{P} \mid & (3) \\ & & \langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B \rangle \mid \langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B \rangle \trianglerighteq_d \mathrm{P} \mid \langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B \rangle \trianglerighteq [\!]_{i \in I}(io_i \rightarrow \mathrm{P}_i) & \\ \mathrm{S} & ::= & \mathrm{P} \mid \mathrm{S} \| \mathrm{S} & \end{array}$$

The HCSP constructs can be defined respectively by normal hybrid designs (Section 5.2) as follows:

- The **skip** statement terminates immediately having no effect on variables, and it is modelled as the relational identity:

  $$[\![\textbf{skip}]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad \vdash \lceil \rfloor$$

  It is a unit of normal hybrid designs w.r.t. $\fatsemi$ by Corollary 1.

- The assignment of the value e to a variable x is modelled as setting x to e and keeping all other variables constant if e can be successfully evaluated [28]. Therefore, the assignment can be interpreted as follows:

$$[\![x := e]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad \mathrm{E}(e) \vdash \lceil x := e \rfloor$$

  where $\mathrm{E}(e)$ is $\mathcal{H}_{\mathrm{HC}}$-healthy and specifies the condition by which e can be evaluated. Note that E is a function supplied for all expressions. For example, $\mathcal{H}_{\mathrm{HP}}(s \neq 0)$ can be such a condition, where s is a state variable in e.
- The **wait** statement will keep idle for d time units keeping variables unchanged:

$$[\![\mathbf{wait}\ d]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad \vdash [\![\dot{\underset{\sim}{\mathbf{s}}} = \mathbf{0}, \emptyset]\!]_d$$

- The sequential composition $\mathsf{P}_0; \mathsf{P}_1$ behaves as P_0 first, and if it terminates, as P_1 afterwards:

$$[\![\mathsf{P}_0; \mathsf{P}_1]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad [\![\mathsf{P}_0]\!]_{\mathrm{HUTP}} \fatsemi [\![\mathsf{P}_1]\!]_{\mathrm{HUTP}}$$

- $\mathsf{P}_0 \sqcap \mathsf{P}_1$ denotes internal choice, which behaves as either P_0 or P_0, and the choice is made by the process. It is interpreted as a demonic choice of two operands in a standard way:

$$[\![\mathsf{P}_0 \sqcap \mathsf{P}_1]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad [\![\mathsf{P}_0]\!]_{\mathrm{HUTP}} \sqcap [\![\mathsf{P}_1]\!]_{\mathrm{HUTP}}$$

- The alternative $B \rightarrow \mathsf{P}$, where B is a Boolean expression, behaves as P if B is true; otherwise it terminates immediately:

$$[\![B \rightarrow \mathsf{P}]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad [\![\mathsf{P}]\!]_{\mathrm{HUTP}} \lhd B \rhd [\![\mathbf{skip}]\!]_{\mathrm{HUTP}}$$

- A process variable X is interpreted as a predicate variable:

$$[\![X]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad X$$

- The recursion $\mathrm{rec}\ X \cdot \mathsf{P}$ means that the execution of P can be repeated by replacing each occurrence of X with $\mathrm{rec}\ X \cdot \mathsf{P}$ itself during executing P, i.e., $\mathrm{rec}\ X \cdot \mathsf{P}$ behaves like $\mathsf{P}[\mathrm{rec}\ X \cdot \mathsf{P}/X]$. The semantics for recursion can be defined as the least or the greatest fixed point by

$$\begin{array}{lcl} [\![\mathrm{rec}\ X \cdot \mathsf{P}]\!]^{\mu}_{\mathrm{HUTP}} & \widehat{=} & \mu X \cdot [\![\mathsf{P}]\!]_{\mathrm{HUTP}} \\ [\![\mathrm{rec}\ X \cdot \mathsf{P}]\!]^{\nu}_{\mathrm{HUTP}} & \widehat{=} & \nu X \cdot [\![\mathsf{P}]\!]_{\mathrm{HUTP}} \end{array}$$

  For example, the semantics of P^* can be defined as

$$[\![\mathsf{P}^*]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad [\![\mathrm{rec}\ X \cdot \mathbf{skip} \sqcap (\mathsf{P}; X)]\!]^{\mu}_{\mathrm{HUTP}} \quad = \quad \exists n \in \mathbb{N} \cdot [\![\mathsf{P}^n]\!]_{\mathrm{HUTP}}$$

  where $\mathbb{N}$ is the set of non-negative integers and $\mathsf{P}^0 \mathrel{\widehat{=}} \mathbf{skip}$.
- A continuous evolution statement $\langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B \rangle$ says that the process keeps waiting, and meanwhile keeps continuously evolving following the differential equations F, until the domain constraint B is violated:

$$[\![\langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B \rangle]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad \mathrm{exit} \sqcap (\mathrm{ode}^{\emptyset} \fatsemi \mathrm{exit})$$

  where

$$\begin{array}{rcl} \mathrm{ode}^{RS} & \widehat{=} & \vdash [\![F(\dot{\underset{\sim}{\mathbf{s}}}, \underset{\sim}{\mathbf{s}}) = 0 \wedge B[\underset{\sim}{\mathbf{s}}/\mathbf{s}], RS]\!] \\ \mathrm{exit} & \widehat{=} & \vdash \lceil \neg B(\mathbf{s}) \wedge \mathbf{s}' = \mathbf{s} \rfloor \end{array}$$

  Note that ode^{RS} and exit take F and B as parameters. For brevity, the parameters are not shown in the following content. The above states that the process can either terminate at the beginning, i.e., exit, or evolve for a finite period before terminating without waiting communications, i.e., $\mathrm{ode}^{\emptyset} \fatsemi \mathrm{exit}$, depending on whether B holds or not.

- $\langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B\rangle \trianglerighteq_d \mathsf{P}$ behaves like $\langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B\rangle$, if the evolution terminates before d time units. Otherwise, after d time units of evolution according to F, it moves on to execute P:

$$[\![\langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B\rangle \trianglerighteq_d \mathsf{P}]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad \mathsf{exit} \sqcap (\mathsf{ode}^{\emptyset}_{<d} \fatsemi \mathsf{exit}) \sqcap (\mathsf{ode}^{\emptyset}_{d} \fatsemi [\![\mathsf{P}]\!]_{\mathrm{HUTP}})$$

  It can either terminate at the beginning, i.e., exit, evolve less than d time units before terminating, i.e., $\mathsf{ode}^{\emptyset}_{<d} \fatsemi \mathsf{exit}$, or evolve for d time units and then continue to execute P, i.e., $\mathsf{ode}^{\emptyset}_{d} \fatsemi [\![\mathsf{P}]\!]_{\mathrm{HUTP}}$.
- $\langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B\rangle \trianglerighteq [\!]_{i\in I}(io_i \rightarrow \mathsf{P}_i)$ behaves like $\langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B\rangle$, except that the continuous evolution is preempted as soon as one of the communications io_i takes place, which is followed by the respective P_i, where io_i stands for a communication event, i.e., either $ch?x$ or $ch!e$. Notice that if the continuous part terminates before a communication among $\{io_i \mid i \in I\}$ occurs, then the process terminates without communicating. Concretely,

$$[\![\langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B\rangle \trianglerighteq [\!]_{i\in I}(io_i \rightarrow \mathsf{P}_i)]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad \mathsf{exit} \sqcap \mathsf{comm} \sqcap (\mathsf{ode}^{RS} \fatsemi (\mathsf{exit} \sqcap \mathsf{comm})) \tag{4}$$

  Without loss of generality, we let $I = I_? \cup I_!$ with $I_? \cap I_! = \emptyset$ such that $io_i = ch_i?x_i$ if $i \in I_?$ and $ch_i!e_i$ otherwise ($i \in I_!$), where e_i is an arithmetic expression. Note that x_i and x_j can denote the same variable even if $i \neq j$. Let $RS = \{ch_i?, ch_j! \mid i \in I_?, j \in I_!\}$ be the set of waiting channel operations. Then, the preemption by communication can be described by

$$\mathsf{comm} \quad \widehat{=} \quad \mathsf{input} \sqcap \mathsf{output}$$

  where

$$\begin{array}{rcl} \mathsf{input} & \widehat{=} & \bigsqcap_{i\in I_?}(\vdash \lceil ch_i?, x_i\rfloor \fatsemi [\![\mathsf{P}_i]\!]_{\mathrm{HUTP}}) \\ \mathsf{output} & \widehat{=} & \bigsqcap_{j\in I_!}(\mathrm{E}(e_j(\mathbf{s})) \vdash \lceil ch_j!, e_j(\mathbf{s})\rfloor \fatsemi [\![\mathsf{P}_j]\!]_{\mathrm{HUTP}}) \end{array}$$

  It states that the process can either terminate or communicate at the beginning: $\mathsf{exit} \sqcap \mathsf{comm}$, evolve for a finite period before terminating or communicating: $\mathsf{ode}^{RS} \fatsemi (\mathsf{exit} \sqcap \mathsf{comm})$.
- $ch?x$ receives a value along channel RS and assigns it to x, and as the dual, $ch!e$ sends the value of e along channel RS:

$$\begin{array}{rcl} [\![ch?x]\!]_{\mathrm{HUTP}} & \widehat{=} & [\![\langle \dot{\mathbf{s}} = \mathbf{0} \& \mathbf{true}\rangle \trianglerighteq (ch?x \rightarrow \mathbf{skip})]\!]_{\mathrm{HUTP}} \\ [\![ch!e]\!]_{\mathrm{HUTP}} & \widehat{=} & [\![\langle \dot{\mathbf{s}} = \mathbf{0} \& \mathbf{true}\rangle \trianglerighteq (ch!e \rightarrow \mathbf{skip})]\!]_{\mathrm{HUTP}} \end{array}$$

  A communication takes place when both the sending and the receiving parties are ready, and may cause one side to wait if the other side is not ready.
- $\mathsf{P}_0 \| \mathsf{P}_1$ behaves as if sequential processes P_0 and P_1 run independently except that all communications along the common channels connecting P_0 and P_1 are to be synchronised. The processes P_0 and P_1 in parallel can neither share variables, nor input or output channels, i.e., a channel operation $ch*$ appears in P_0 iff it does not appear in P_1. The parallel composition of P_0 and P_1 can be translated as the parallel composition defined by Definition 5.4 with the merge predicate M_I defined by Definition 4.5:

$$[\![\mathsf{P}_0 \| \mathsf{P}_1]\!]_{\mathrm{HUTP}} \quad \widehat{=} \quad [\![\mathsf{P}_0]\!]_{\mathrm{HUTP}} \|^{\mathrm{HP}}_{M_I} [\![\mathsf{P}_1]\!]_{\mathrm{HUTP}} \tag{5}$$

  where I is the set of common channels between P_0 and P_1.

In what follows we prove the consistency between the HUTP semantics and the structural operational semantics (SOS) of HCSP given in [64]. The latter consists of a collection of transition rules of the following form[1]:

$$\frac{\text{Pre-condition}}{(\mathsf{P}, (ti, \mathbf{s})) \xrightarrow{\mathrm{tb}} (\mathsf{P}', (ti', \mathbf{s}'), \underset{\sim}{\mathbf{s}})}$$

[1]We here revise the operational semantics in [64] a little bit.

where P and P′ are HCSP processes, $(ti, \mathbf{s})$ and $(ti', \mathbf{s}')$ are process states, described by the state variables with timestamps, tb denote a trace block defined in Section 3, and $\underset{\sim}{\mathbf{s}}$ is a flow depicting the evolution from $\mathbf{s}$ to $\mathbf{s}'$. Note that the flow $\underset{\sim}{\mathbf{s}}$ can be removed if the above transition describes a discrete action. For example, the SOS of an assignment is

$$\frac{e \text{ can be evaluated on } \mathbf{s}}{(x := e, (ti, \mathbf{s})) \xrightarrow{\tau} (\varepsilon, (ti, \mathbf{s}[x \mapsto e(\mathbf{s})]))}$$

where ε denotes termination. A more comprehensive understanding of the operational semantics can be found in [64].

We say the HUTP semantics and the SOS of an HCSP process are consistent iff they keep consistent on time, state and trace. Concretely, the changes of time and state and the generated trace of the SOS of an HCSP process should be reflected by its HUTP semantics, and vice versa. For example, the SOS for the ODE with communication interruption

$$\mathrm{ODE}_{\trianglerighteq} \quad \widehat{=} \quad \langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B \rangle \trianglerighteq \llbracket_{i \in I} (ch_i * _i \rightarrow \mathrm{P}_i)$$

is described by the following rules:

$$\frac{\neg B(\mathbf{s})}{(\mathrm{ODE}_{\trianglerighteq}, (ti, \mathbf{s})) \xrightarrow{\tau} (\varepsilon, (ti, \mathbf{s}))}[\text{Int0}] \quad \frac{\begin{array}{c}\underset{\sim}{\mathbf{s}} \text{ is a solution of } F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \\ \underset{\sim}{\mathbf{s}}(ti) = \mathbf{s} \quad \mathbf{s}' = \underset{\sim}{\mathbf{s}}(ti'^{-}) \quad \forall t \in [ti, ti') \cdot B(\underset{\sim}{\mathbf{s}}(t))\end{array}}{(\mathrm{ODE}_{\trianglerighteq}, (ti, \mathbf{s})) \xrightarrow{\langle ti'-ti, \{ch_i?, ch_j! \mid i \in I_?, j \in I_!\}\rangle} (\mathrm{ODE}_{\trianglerighteq}, (ti', \mathbf{s}'), \underset{\sim}{\mathbf{s}})}[\text{Int1}]$$

$$\frac{i \in I_? \quad ch_i * _i = ch_i ? x_i}{(\mathrm{ODE}_{\trianglerighteq}, (ti, \mathbf{s})) \xrightarrow{\langle ch_i ?, d\rangle} (\mathrm{P}_i, (ti, \mathbf{s}[x_i \mapsto d]))}[\text{Int2}] \quad \frac{\begin{array}{c} j \in I_! \quad ch_j * _j = ch_j ! e_j \\ e_j \text{ can be evaluated on } \mathbf{s}\end{array}}{(\mathrm{ODE}_{\trianglerighteq}, (ti, \mathbf{s})) \xrightarrow{\langle ch_j !, e_j(\mathbf{s})\rangle} (\mathrm{P}_j, (ti, \mathbf{s}))}[\text{Int3}]$$

The rule [Int0] is reflected by exit and the rule [Int1] is reflected by $\mathrm{ode}^{RS} \fatsemi \llbracket \mathrm{ODE}_{\trianglerighteq} \rrbracket_{\mathrm{HUTP}}$, where $RS = \{ch_i?, ch_j! \mid i \in I_?, j \in I_!\}$. The rule [Int2], equivalent to $\vdash \lceil ch_i?, x_i \rfloor$, depicts a receiving event that interrupts the ODE, followed by the process P_i. The HUTP semantics of P_i is denoted by $\llbracket \mathrm{P}_i \rrbracket_{\mathrm{HUTP}}$, and therefore, the HUTP semantics for the receiving event can be described by

$$\vdash \lceil ch_i?, x_i \rfloor \quad \fatsemi \quad \llbracket \mathrm{P}_i \rrbracket_{\mathrm{HUTP}}$$

Similarly, by [Int3], the HUTP semantics of the sending event is denoted by

$$\mathrm{E}(e_j) \vdash \lceil ch_j?, e_j \rfloor \quad \fatsemi \quad \llbracket \mathrm{P}_j \rrbracket_{\mathrm{HUTP}}$$

where $\mathrm{E}(e_j)$ means e_j can be evaluated successfully. The process comm summarises the HUTP semantics of the above communications. Note that $\mathrm{ODE}_{\trianglerighteq}$ terminates after applying these rules except [Int1]. In summary, we can get

$$\mathrm{ODE}_{\trianglerighteq} \quad = \quad \mathsf{exit} \sqcap \mathsf{comm} \sqcap (\mathrm{ode}^{RS} \fatsemi \mathrm{ODE}_{\trianglerighteq})$$

In other words, the SOS of $\mathrm{ODE}_{\trianglerighteq}$ is equivalent to the equation $X = F(X)$, where

$$F(X) \quad \widehat{=} \quad \mathsf{exit} \sqcap \mathsf{comm} \sqcap (\mathrm{ode}^{RS} \fatsemi X)$$

Any solution of $X = F(X)$ can be treated as the HUTP semantics of $\mathrm{ODE}_{\trianglerighteq}$. In this paper, we use the least fixed point:

$$\mu X.F(X) \quad = \quad \mathsf{exit} \sqcap \mathsf{comm} \sqcap \big(\mathrm{ode}^{RS} \fatsemi (\mathsf{exit} \sqcap \mathsf{comm})\big)$$

as shown in (4). It can be checked that the SOS of $\mathrm{ODE}_{\trianglerighteq}$, denoted by the above four transition rules, is reflected by (4), and vice versa.

Remark 1. *Another solution of the equation $X = F(X)$ above is*

$$\mathsf{ode}_{\infty}^{RS} \sqcap \mathsf{exit} \sqcap \mathsf{comm} \sqcap (\mathsf{ode}^{RS} \fatsemi (\mathsf{exit} \sqcap \mathsf{comm}))$$

where $\mathsf{ode}_{\infty}^{RS} \mathrel{\widehat{=}} \vdash \lceil F(\dot{\underset{\sim}{\mathbf{s}}}, \underset{\sim}{\mathbf{s}}) = 0 \wedge B[\underset{\sim}{\mathbf{s}}/\mathbf{s}], RS \rfloor_{\infty}$ *denotes* $\mathsf{ODE}_{\geq}$ *will evolve forever, never terminate. This solution is reasonable, although it is neither the least nor the greatest fixed point.*

Theorem 6.1 (Semantic Consistency). *The HUTP semantics of HCSP is consistent with the structural operational semantics of HCSP.*

6.3 From Simulink to HUTP

Matlab/Simulink [39] is an industrial de-facto standard for modelling embedded systems. A Simulink diagram consists of a set of blocks. Each block has a set of input and output signals. A *discrete* block computes the output signal from the input signals periodically or aperiodically. A *continuous* block computes the output signal continuously by following a differential equation. Blocks can be organised into hierarchical subsystems, such as normal, triggered and enabled subsystems.

In this subsection, we introduce a formal semantics of Simulink by encoding it into HUTP. Overall, a Simulink diagram is split into discrete and continuous sub-diagrams which consist of discrete and continuous blocks, respectively. We define the respective semantics for the discrete and continuous sub-diagrams and then compose them together. The following Simulink diagrams serve as running examples. The left of Fig. 2 is the original diagram and the right is the diagram where the ports between some blocks are added.

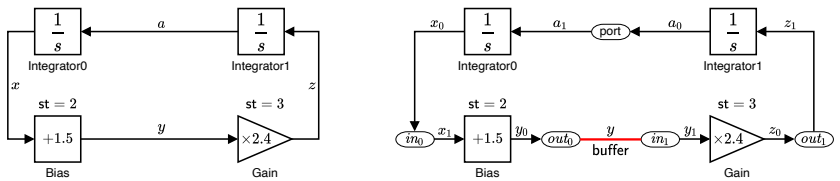

Fig. 2. Examples of plant-control loop

6.3.1 Discrete Block. A discrete block is executed during a finite time interval. The left of Fig. 3 is the typical periodic discrete block of a math operation, such as Bias and Gain in Fig, 2. This kind of blocks are stateless. The block on the right is a periodic Unit Delay block, which is stateful.

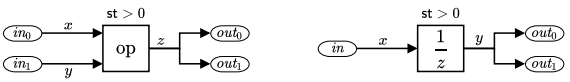

Fig. 3. Discrete blocks

For each discrete block, we use ports to serve as channels for the input and output lines. For a periodic Math Operation block, periodically, one first gets data from the input ports (get), then performs a computation based on the inputs (comp), next puts the computed result to its output ports (put), and finally keeps silent for a period of time (period), as follows:

$$\begin{aligned}
\mathsf{get} &\mathrel{\widehat{=}} \vdash \lceil in_0?, x \rfloor \wedge \mathrm{G}_0(x') \quad \fatsemi \quad \vdash \lceil in_1?, y \rfloor \wedge \mathrm{G}_1(y') \\
\mathsf{comp} &\mathrel{\widehat{=}} \mathrm{A}(x, y) \vdash \lceil z := \mathrm{op}(x, y) \rfloor \\
\mathsf{put} &\mathrel{\widehat{=}} \vdash \lceil out_0!, z \rfloor \quad \fatsemi \quad \vdash \lceil out_1!, z \rfloor \\
\mathsf{period} &\mathrel{\widehat{=}} \vdash \lceil \dot{\underset{\sim}{z}} = 0, \emptyset \rfloor_{\mathsf{st}}
\end{aligned}$$

In the above, get acquires data x and y from channels (ports) in_0 and in_1, respectively. Note that $\lceil in_0?, x \rfloor$ and $\lceil in_1?, y \rfloor$ are instant communications (i.e., fetch data whenever they require). The constraints $G_0(x')$ and $G_1(y')$ mean that only the desired data is received. Receiving does not consume time, so the order of channel operations $in_0?$ and $in_1?$ does not matter. comp denotes a computation, where $A(x, y)$ means that the operands x and y, provided from get, must satisfy some assumption, such as $y \neq 0$. Therefore, the relation $G_0(x) \wedge G_1(y) \Rightarrow A(x, y)$ must hold. If $A(x, y)$ is satisfied, the computation $\lceil z := \mathrm{op}(x, y) \rfloor$ executes. By contrast with get, put just puts the computed result to channels (ports) out_0 and out_1. Note that get, comp and put do not consume time. Then, the process keeps quiescent for some period specified by the sample time st, denoted by period, where $\emptyset$ indicates the $in_0?$, $in_1?$, $out_0!$ and $out_1!$ are not ready to communicate with the context during the period. Finally, it can terminate at any time before the next period arrives (tail). In summary, a periodic Math Operation block can be translated as:

$$\mathsf{MOP} \;\widehat{=}\; \mathsf{get} \fatsemi \mathsf{comp} \fatsemi \mathsf{put} \fatsemi (\mathsf{period} \fatsemi \mathsf{get} \fatsemi \mathsf{comp} \fatsemi \mathsf{put})^* \fatsemi \mathsf{tail} \tag{6}$$

where

$$\mathsf{tail} \;\widehat{=}\; \vdash \lceil\!\lceil \dot{\tilde{z}} = 0, \emptyset \rfloor\!\rfloor_{<\mathrm{st}} \;\sqcap\; \vdash \lceil \rfloor$$

Unlike a Math Operation block, a Unit Delay block is stateful. The right of Fig. 3 is a typical periodic Unit Delay block. At the beginning, its state is initialised (init). Then, periodically, it puts the current state to the output channels (ports) out_0 and out_1, then gets data from the input channel (port) in and updates the state (get), and finally holds the state for some period (period). In summary, a Unit Delay block can be translated as follows:

$$\mathsf{UnitDelay} \;\widehat{=}\; \mathsf{init} \fatsemi \mathsf{put} \fatsemi \mathsf{get} \fatsemi (\mathsf{period} \fatsemi \mathsf{put} \fatsemi \mathsf{get})^* \fatsemi \mathsf{tail}$$

where

$$\begin{aligned}
\mathsf{init} &\;\widehat{=}\; \vdash \lceil y := \mathsf{init_y} \rfloor \\
\mathsf{put} &\;\widehat{=}\; \vdash \lceil out_0!, y \rfloor \;\fatsemi\; \vdash \lceil out_1!, y \rfloor \\
\mathsf{get} &\;\widehat{=}\; \vdash \lceil in?, x \rfloor \;\fatsemi\; \vdash \lceil y := x \rfloor \\
\mathsf{period} &\;\widehat{=}\; \vdash \lceil\!\lceil \dot{\tilde{y}} = 0, \emptyset \rfloor\!\rfloor_{\mathrm{st}} \\
\mathsf{tail} &\;\widehat{=}\; \vdash \lceil\!\lceil \dot{\tilde{y}} = 0, \emptyset \rfloor\!\rfloor_{<\mathrm{st}} \;\sqcap\; \vdash \lceil \rfloor
\end{aligned}$$

6.3.2 Continuous Block. Blocks containing continuous states are called continuous blocks. Computing a continuous state requires one to know the derivative of the state variables. The left of Fig. 4 is an Integrator block and the right is a Math Operation block of which sample time is 0. They are both typical continuous blocks.

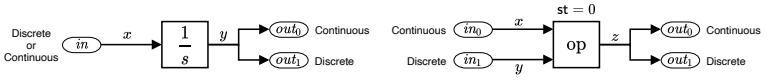

Fig. 4. Continuous blocks

For an Integrator block, the state is first initialised by an initial condition, as follows:

$$\mathsf{init} \;\widehat{=}\; \vdash \lceil y := \mathsf{init\_y} \rfloor$$

Then, its semantics depends on the context it interacts with:

- If the source block is continuous (such as Integrator0 in Fig. 2), then the block will evolve continuously following the ODE $\dot{y} = x$, instead of receiving x periodically or aperiodically from the input port. Meanwhile, it will send the current value of y to the output port out_1

whenever the non-continuous target block requires it (put). In that case, the corresponding HUTP semantics will be given as follows:

$$\mathsf{Integrator} \;\widehat{=}\; \mathsf{init} \fatsemi (\mathsf{put} \sqcap \mathsf{ode}_0)^* \tag{7}$$

where

$$\begin{array}{rcl} \mathsf{put} & \widehat{=} & \vdash \lceil out_1!, y \rfloor \\ \mathsf{ode}_0 & \widehat{=} & \vdash [\![\dot{\underset{\sim}{y}} = \underset{\sim}{x}, \{out_1!\}]\!] \end{array}$$

- On the other hand, if the source block is non-continuous (such as Integrator1 in Fig. 2), then, at each iteration, the block will either get data x from the input channel (get), evolve according to the latest received data (ode_1), or put the current value of y to the output channel (put), as follows:

$$\mathsf{Integrator} \;\widehat{=}\; \mathsf{init} \fatsemi (\mathsf{get} \sqcap \mathsf{put} \sqcap \mathsf{ode}_1)^* \tag{8}$$

where

$$\begin{array}{rcl} \mathsf{get} & \widehat{=} & \vdash \lceil in?, x \rfloor \\ \mathsf{ode}_1 & \widehat{=} & \vdash [\![\dot{\underset{\sim}{y}} = \underset{\sim}{x} \wedge \dot{\underset{\sim}{x}} = 0, \{in?, out_1!\}]\!] \end{array}$$

The semantics of a Math Operation block with sample time 0 in the right of Fig. 4 is a bit more complicated. It requires the conversions between discrete and continuous signals, as illustrated by Fig. 5. Concretely, the conversion of the discrete input signal y_0 to the continuous one (D/A converter) can be specified as follows:

$$\mathsf{converter0} \;\widehat{=}\; (\vdash \lceil in_1?, y_0 \rfloor \wedge \mathrm{G}(y_0')) \fatsemi \vdash [\![\dot{\underset{\sim}{y_0}} = 0, \{in_1?\}]\!])^*$$

At each iteration, converter0 gets the desired data (constrained by $\mathrm{G}(y_0')$) from the input channel in_1 and then keeps the data unchanged for some period. Thus, the input y_0 can be treated as a continuous signal from the view of the continuous block.

On the other hand, the conversion of the continuous output signal z_1 to the discrete one (A/D converter) can be specified as follows:

$$\mathsf{converter1} \;\widehat{=}\; (\vdash \lceil out_1!, z_1 \rfloor \fatsemi \vdash \exists \underset{\sim}{d} \cdot [\![\dot{\underset{\sim}{z_1}} = \underset{\sim}{d}, \{out_1!\}]\!])^*$$

At each iteration, converter1 puts the current value of z_1 to the output channel out_1 and then z_1 continues evolving for some period. Thus, from the view of out_1, the signal z_1 is discrete.

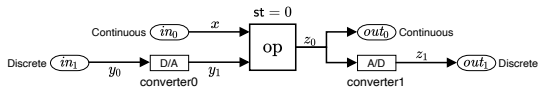

Fig. 5. Continuous Math Operation Block with D/A and A/D converters

Given that all the inputs and outputs are continuous, the computation of the continuous Math Operation block can be translated as follows:

$$\mathsf{comp} \;\widehat{=}\; \mathrm{A}(\underset{\sim}{x}, \underset{\sim}{y_1}) \vdash [\![\underset{\sim}{z_0} = \mathrm{op}(\underset{\sim}{x}, \underset{\sim}{y_1})]\!] \;\sqcap\; \vdash \lceil \rfloor$$

It means that if the continuous inputs $\underset{\sim}{x}$ and $\underset{\sim}{y_1}$ satisfy the expected assumptions during the whole course of action then, the value of $\underset{\sim}{z_0}$ is, at all times, equal to the result of the operation of $\underset{\sim}{x}$ and $\underset{\sim}{y_1}$ in time. $\vdash \lceil \rfloor$ denotes the case that simulation is not started ($ti = ti'$). In summary, the semantics of the continuous Math Operation block of Fig. 5 can be described by the following parallel composition:

$$\mathsf{CMOP} \;\widehat{=}\; \mathcal{H}_{\mathrm{share}}\,(\mathsf{converter0} \|_{\emptyset} \mathsf{comp} \|_{\emptyset} \mathsf{converter1})$$

where (1) $\|_\emptyset$ denotes $\|^{\mathrm{HP}}_{SYN_\emptyset}$ with the merge predicate $SYN_\emptyset \mathrel{\widehat{=}} ti_X = ti_Y \wedge M_\emptyset$. According to Definition 4.5, $M_\emptyset$ allows $ti_X \neq ti_Y$. However, blocks in Simulink are synchronous on time, i.e., $ti_X = ti_Y$; (2) $\mathcal{H}_{\mathrm{share}}$ is the healthiness condition that enforces the consistency between $\underset{\sim}{y}_0$ and $\underset{\sim}{y}_1$ and between $\underset{\sim}{z}_0$ and $\underset{\sim}{z}_1$ when the diagram evolves, specified as follows:

$$\mathcal{H}_{\mathrm{share}}(P \vdash Q) \quad \widehat{=} \quad P \quad \vdash \quad Q \wedge \forall t \in [ti, ti') \cdot \underset{\sim}{y}_0(t) = \underset{\sim}{y}_1(t) \wedge \underset{\sim}{z}_0(t) = \underset{\sim}{z}_1(t)$$

As the parallel-by-merge in this paper (Section 4.2.5) does not support shared state variables, the healthiness condition above is used to enforce the parallel by shared state variables, i.e., it states that the evolutions of y_0 and z_0 should keep consistent with y_1 and z_1, respectively, because y_1 and z_1 are respective aliases of y_0 and z_0.

6.3.3 Lines between discrete Blocks. In order to synchronise communications between discrete blocks, lines between blocks serve as buffers to receive data whenever source blocks provide and target blocks require, as illustrated by the `buffer` in Fig. 2. A typical buffer is shown below:

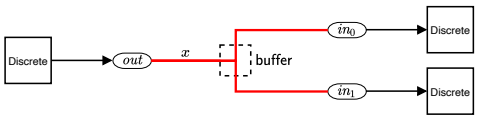

Fig. 6. A two-branch line between discrete blocks

Notice that the buffer must render the causality relation of ports (if the source block provides data and, meanwhile, a target block requires). It should guarantee that the target block can only get the latest data from the source block. In other words, if the channel operations $out?$ and $in_0!$ of the buffer occur simultaneously, $out?$ should be earlier than $in_0!$. Therefore, the buffer should be translated as follows:

$$\mathsf{buffer} \quad \widehat{=} \quad \left(\mathsf{get} \fatsemi (\mathsf{put}_0 \sqcap \mathsf{put}_1 \sqcap \mathsf{ready})^* \fatsemi \mathsf{ready}\right)^* \tag{9}$$

where

$$\begin{array}{rcl} \mathsf{get} & \widehat{=} & \vdash \lceil out?, x \rfloor \\ \mathsf{put}_0 & \widehat{=} & \vdash \lceil in_0!, x \rfloor \\ \mathsf{put}_1 & \widehat{=} & \vdash \lceil in_1!, x \rfloor \\ \mathsf{ready} & \widehat{=} & \vdash [\![\dot{\underset{\sim}{x}} = 0, \{out?, in_0!, in_1!\}]\!] \end{array}$$

This means that, at each iteration, the buffer first gets data from the output port of the source block (get), then puts the received data to the input ports of the target blocks whenever they require it (put_0 and put_1), and finally waits for a period of time, during which channel operations $out?$, $in_0!$ and $in_1!$ are made ready for communication (ready).

6.3.4 Composition. According to the above analysis, before translating the Simulink diagram on the left of Fig. 2, we need to add the necessary ports between some blocks, resulting in the diagram shown on the right of Fig. 2. Before execution, the system must be initialised by setting time to 0 and trace to ϵ, as follows:

$$\mathsf{INIT} \quad \widehat{=} \quad \vdash \mathcal{H}_{\mathrm{HP}}(ti' = 0 \wedge tr' = \epsilon)$$

Then, the composite system, the Diagram on the right of Fig. 2, can be defined in two parts. The Plant part consists of the continuous blocks Integrator0 and Integrator1. The Control part consists

of the discrete blocks Bias and Gain, and a buffer between them. The Diagram can be modelled by the parallel composition of Plant and Control, as follows:

$$\mathsf{Plant} \;\widehat{=}\; \mathcal{H}_{\mathrm{share}}\,(\mathsf{Integrator0} \parallel_{\emptyset} \mathsf{Integrator1}) \tag{10}$$

$$\mathsf{Control} \;\widehat{=}\; \mathsf{Bias} \parallel_{out_0} \mathsf{buffer} \parallel_{in_1} \mathsf{Gain} \tag{11}$$

$$[\![\mathsf{Diagram}]\!]_{\mathrm{HUTP}} \;\widehat{=}\; \mathsf{INIT} \fatsemi (\mathsf{Plant} \parallel_{\{in_0, out_1\}} \mathsf{Control}) \tag{12}$$

where $\parallel_I$ represents $\parallel^{\mathrm{HP}}_{SYN_I}$ with $SYN_I \mathrel{\widehat{=}} ti_X = ti_Y \wedge M_I$, for simplicity, and we use $\parallel_{ch}$ if $I = \{ch\}$. It can be proved that SYN_I is well-defined from Property 6. In the above case,

$$\mathcal{H}_{\mathrm{share}}(P \vdash Q) \;\;\widehat{=}\;\; P \;\;\vdash\;\; Q \wedge \forall t \in [ti, ti') \cdot \underline{a}_0(t) = \underline{a}_1(t) = \underline{a}(t) \tag{13}$$

because the continuous blocks Integrator0 and Integrator1 share a which has two aliases a_0 and a_1.

Example 6.2 (Algebraic Loop). Consider the loop of Bias and Gain in Fig. 7. It forms an algebraic loop. According to (6), the respective prefixes of the communication traces of Bias and Gain are $\langle in_0?, d_0\rangle^\frown\tau^\frown\langle out_0!, d_1\rangle$ and $\langle in_1?, d_2\rangle^\frown\tau^\frown\langle out_1!, d_3\rangle$, where τ denotes timeless computations. According to (9), the prefixes of the communication traces of two buffers between Bias and Gain are $\langle out_0?, d_4\rangle^\frown\tau^\frown\langle in_1!, d_5\rangle$ and $\langle out_1?, d_6\rangle^\frown\tau^\frown\langle in_0!, d_7\rangle$. It can be verified that these four traces cannot be composed by $\parallel_I$, as defined in Section 3.3, which indicates that this unsound system will reduce to $\top_{\mathrm{HD}}$, i.e., it can do nothing.

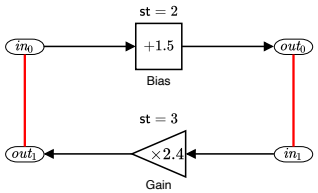

Fig. 7. A loop of discrete blocks

In addition to the above, the semantics of hierarchical diagrams, such as normal, triggered and enabled subsystems, can also be defined using HUTP. Since describing their HUTP semantics clearly needs pages of space, they are not introduced in this paper. Note that compared with the HUTP semantics of Simulink given in [10, 64], the HUTP semantics defined above is much simpler with more solid foundations as HUTP theory is well developed in this paper.

6.4 Applying HUTP to Justify the Translation from Simulink to HCSP

In the work [69], we introduced how to translate Simulink (as well as Stateflow) diagrams to the corresponding HCSP models for the purpose of verifying them using a Hybrid Hoare Logic prover in Isabelle/HOL [65, 68]. The translation is automatic and is implemented as a module `ss2hcsp` in the tool chain MARS[2] [63, 64] which implements the flow from modelling, simulation, verification to code generation for hybrid systems. In this subsection, we justify the correctness of the translation by `ss2hcsp` from the Simulink diagram of Fig. 2 to the corresponding HCSP model as an application of the HUTP proposed in this paper.

Definition 6.3 (Conditional Equivalence). Let $T \mathrel{\widehat{=}} \{t_n\} \subseteq \mathbb{R}_{\geq 0}$ be a chain of increasing time points ($t_n < t_{n+1}$) with $\lim_{n\to+\infty} t_n = +\infty$ and S the set of state variables. Let P and Q be differential relations. Then, we say P is equivalent to Q w.r.t. T and S iff $\mathrm{P} \wedge ti' \in T$ is equivalent to $\mathrm{Q} \wedge ti' \in T$ on the state variables in S, denoted by $\mathrm{P} \equiv_{T,S} \mathrm{Q}$.

[2]https://gitee.com/bhzhan/mars.git

Intuitively, $P \equiv_{T,S} Q$ means that if P and Q terminate at time $ti' \in T$, then their state variables in S are consistent. In this subsection, we wish to justify $[\![\text{Diagram}]\!]_{\text{HUTP}} \equiv_{T,S} [\![[\![\text{Diagram}]\!]_{\text{HCSP}}]\!]_{\text{HUTP}}$ with some T and S, where Diagram is the Simulink diagram of Fig. 2.

For the Simulink Diagram of Fig. 2, ss2hcsp generates the following HCSP process:

$$[\![\text{Diagram}]\!]_{\text{HCSP}} \;\widehat{=}\; \text{Plant}_{\text{HCSP}} \| \text{Control}_{\text{HCSP}}$$

where

$$\begin{aligned}
\text{Plant}_{\text{HCSP}} &\;\widehat{=}\; a := 0; x_0 := 0; \left(\langle \dot{x}_0 = a, \dot{a} = z_1, \dot{z}_1 = 0 \& \textbf{true} \rangle \trianglerighteq [\!] \begin{pmatrix} ch_x!x_0 \to \textbf{skip}, \\ ch_z?z_1 \to \textbf{skip} \end{pmatrix} \right)^* \\
\text{Control}_{\text{HCSP}} &\;\widehat{=}\; ti := 0; ch_x?x_1; y := x_1 + 1.5; z_0 := y \times 2.4; ch_z!z_0; \\
&\quad \begin{pmatrix} \textbf{wait}(1); & ti\%2 = 0 \to (ch_x?x_1; y := x_1 + 1.5); \\ & ti\%3 = 0 \to (z_0 := y \times 2.4; ch_z!z_0) \end{pmatrix}^*
\end{aligned}$$

The period of $\text{Control}_{\text{HCSP}}$ is the greatest common divisor (GCD) of periods of the blocks (Bias and Gain) it contains, i.e., $\text{GCD}(2,3) = 1$, as denoted by $\textbf{wait}(1)$. Since no shared variable is allowed between different HCSP processes in parallel, we let x_0 and x_1 be two aliases of x, and z_0 and z_1 the aliases of z, as shown by the right diagram in Fig. 2.

We first study the conditions under which $\text{Plant}_{\text{HCSP}}$ is consistent with Plant of (10). According to (4), $\text{Plant}_{\text{HCSP}}$ can be translated as follows:

$$[\![\text{Plant}_{\text{HCSP}}]\!]_{\text{HUTP}} \;\widehat{=}\; \text{init}_a \fatsemi \text{init}_{x_0} \fatsemi (\text{put}_x \sqcap \text{get}_z \sqcap (\text{ode} \fatsemi (\text{put}_x \sqcap \text{get}_z)))^*$$

where $\text{init}_a \mathrel{\widehat{=}} \vdash \lceil a := 0 \rfloor$, $\text{init}_{x_0} \mathrel{\widehat{=}} \vdash \lceil x_0 := 0 \rfloor$, $\text{put}_x \mathrel{\widehat{=}} \vdash \lceil ch_x!, x_0 \rfloor$, $\text{get}_z \mathrel{\widehat{=}} \vdash \lceil ch_z?, z_1 \rfloor$ and

$$\text{ode} \;\widehat{=}\; \vdash [\![\dot{\underset{\sim}{x}}_0 = \underset{\sim}{a} \wedge \dot{\underset{\sim}{a}} = \underset{\sim}{z}_1 \wedge \dot{\underset{\sim}{z}}_1 = 0, \{ch_x!, ch_z?\}]\!] \tag{14}$$

According to (7) and (8), Plant of (10) can be unfolded as

$$\text{Plant} \;\widehat{=}\; \mathcal{H}_{\text{share}} \left((\text{init}_{x_0} \fatsemi (\text{put}_x \sqcap \text{ode}_0)^*) \|_{\emptyset} (\text{init}_{a_0} \fatsemi (\text{get}_z \sqcap \text{ode}_1)^*) \right) \tag{15}$$

where $\text{init}_{a_0} \mathrel{\widehat{=}} \vdash \lceil a_0 := 0 \rfloor$, $\text{put}_x \mathrel{\widehat{=}} \vdash \lceil in_0!, x_0 \rfloor$, $\text{get}_z \mathrel{\widehat{=}} \vdash \lceil out_1?, z_1 \rfloor$ and

$$\begin{aligned}
\text{ode}_0 &\;\widehat{=}\; \vdash [\![\dot{\underset{\sim}{x}}_0 = \underset{\sim}{a}_1, \{in_0!\}]\!] \\
\text{ode}_1 &\;\widehat{=}\; \vdash [\![\dot{\underset{\sim}{a}}_0 = \underset{\sim}{z}_1 \wedge \dot{\underset{\sim}{z}}_1 = 0, \{out_1?\}]\!]
\end{aligned}$$

Note that in_0 and out_1 are the aliases of ch_x and ch_z, respectively, so we do not distinguish put_x and get_z in $[\![\text{Plant}_{\text{HCSP}}]\!]_{\text{HUTP}}$ and Plant. Now we demonstrate Plant of (15) is equivalent to

$$\text{init}_a \fatsemi \text{init}_{x_0} \fatsemi (\text{get}_z \sqcap \text{put}_x \sqcap \text{ode})^* \tag{16}$$

if τs are removed from traces, where ode is defined in (14). The traces of $\text{init}_{x_0} \fatsemi (\text{put}_x \sqcap \text{ode}_0)^*$ and $\text{init}_{a_0} \fatsemi (\text{get}_z \sqcap \text{ode}_1)^*$ can be modelled by the finite state machines (FSMs) in Fig. 8, where τ_x and τ_a denote the actions of init_{x_0} and init_{a_0}, respectively.

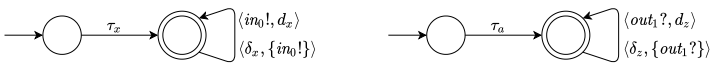

Fig. 8. FSMs of traces of $\text{init}_{x_0} \fatsemi (\text{put}_x \sqcap \text{ode}_0)^*$ (left) and $\text{init}_{a_0} \fatsemi (\text{get}_z \sqcap \text{ode}_1)^*$ (right)

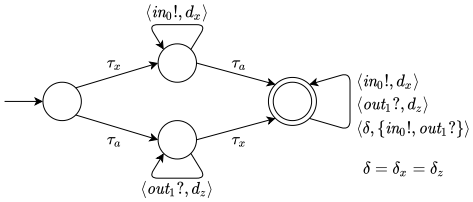

Fig. 9. The composition of FSMs in Fig. 8 by $\|_\emptyset$

According to the definition of $\|_\emptyset$, the two FSMs in Fig. 8 only synchronise on wait blocks $\langle \delta_x, \{in_0!\}\rangle$ and $\langle \delta_z, \{out_1?\}\rangle$, which indicates $\delta_x = \delta_z$. Therefore, their composition is illustrated in Fig. 9, which reflects the following result:

$$\big((\mathsf{init}_{x_0} \fatsemi \mathsf{put}_x^* \fatsemi \mathsf{init}_{a_0}) \sqcap (\mathsf{init}_{a_0} \fatsemi \mathsf{get}_z^* \fatsemi \mathsf{init}_{x_0})\big) \fatsemi (\mathsf{put}_x \sqcap \mathsf{get}_z \sqcap \mathsf{ode}_{01})^* \tag{17}$$

where

$$\mathsf{ode}_{01} \;\widehat{=}\; \vdash \llbracket \dot{\underset{\sim}{x}}_0 = \underset{\sim}{a}_1 \wedge \dot{\underset{\sim}{a}}_0 = \underset{\sim}{z}_1 \wedge \dot{\underset{\sim}{z}}_1 = 0, \{in_0!, out_1?\} \rrbracket$$

If we ignore τs in traces, then $(\mathsf{init}_{x_0} \fatsemi \mathsf{put}_x^* \fatsemi \mathsf{init}_{a_0})$ and $(\mathsf{init}_{a_0} \fatsemi \mathsf{get}_z^* \fatsemi \mathsf{init}_{x_0})$ are equivalent to $(\mathsf{init}_{a_0} \fatsemi \mathsf{init}_{x_0} \fatsemi \mathsf{put}_x^*)$ and $(\mathsf{init}_{a_0} \fatsemi \mathsf{init}_{x_0} \fatsemi \mathsf{get}_z^*)$, respectively. Then, (17) can be simplified:

$$\begin{aligned} & \big((\mathsf{init}_{a_0} \fatsemi \mathsf{init}_{x_0} \fatsemi \mathsf{put}_x^*) \sqcap (\mathsf{init}_{a_0} \fatsemi \mathsf{init}_{x_0} \fatsemi \mathsf{get}_z^*)\big) \fatsemi (\mathsf{put}_x \sqcap \mathsf{get}_z \sqcap \mathsf{ode}_{01})^* \\ = \;& \mathsf{init}_{a_0} \fatsemi \mathsf{init}_{x_0} \fatsemi (\mathsf{put}_x^* \sqcap \mathsf{get}_z^*) \fatsemi (\mathsf{put}_x \sqcap \mathsf{get}_z \sqcap \mathsf{ode}_{01})^* \\ = \;& \mathsf{init}_{a_0} \fatsemi \mathsf{init}_{x_0} \fatsemi (\mathsf{put}_x \sqcap \mathsf{get}_z \sqcap \mathsf{ode}_{01})^* \end{aligned} \tag{18}$$

Note that the above result should be constrained by the healthiness condition of (13) which states a_0 and a_1 are aliases of a. Therefore, the above result is equivalent to (16), which is equivalent to Plant of (15) (if τs are removed from traces).

However, (16) is not equivalent to $[\![\mathsf{Plant}_{\mathrm{HCSP}}]\!]_{\mathrm{HUTP}}$, because the latter always terminates with put_x or get_z. Thus, the premise of the equivalence should be (1) ϕ_0: ignoring τs in traces and (2) ϕ_1: Plant terminates with put_x or get_z. In summary,

$$\mathsf{Plant} \;\equiv_{\phi_0 \wedge \phi_1}\; [\![\mathsf{Plant}_{\mathrm{HCSP}}]\!]_{\mathrm{HUTP}} \tag{19}$$

On the other hand, we study the conditions under which $\mathsf{Control}_{\mathrm{HCSP}}$ is consistent with Control of (11) under some conditions. $\mathsf{Control}_{\mathrm{HCSP}}$ can be translated as follows:

$$\begin{aligned} [\![\mathsf{Control}_{\mathrm{HCSP}}]\!]_{\mathrm{HUTP}} \;\widehat{=}\; & \vdash \lceil ti := 0 \rfloor \fatsemi \mathsf{get}_x \fatsemi \mathsf{comp}_y \fatsemi \mathsf{comp}_z \fatsemi \mathsf{put}_z \fatsemi \\ & \begin{pmatrix} \vdash \llbracket \dot{\underset{\sim}{x}}_1 = \dot{\underset{\sim}{y}} = \dot{\underset{\sim}{z}}_0 = 0, \emptyset \rrbracket_1 \fatsemi \\ (\mathsf{get}_x \fatsemi \mathsf{comp}_y) \lhd ti\%2 = 0 \rhd [\![\mathbf{skip}]\!]_{\mathrm{HUTP}} \fatsemi \\ (\mathsf{comp}_z \fatsemi \mathsf{put}_z) \lhd ti\%3 = 0 \rhd [\![\mathbf{skip}]\!]_{\mathrm{HUTP}} \end{pmatrix}^* \end{aligned}$$

as illustrated by Fig. 10, where $\mathsf{get}_x \;\widehat{=}\; \vdash \lceil ch_x?, x_1 \rfloor$, $\mathsf{put}_z \;\widehat{=}\; \vdash \lceil ch_z!, z_0 \rfloor$, $\mathsf{comp}_y \;\widehat{=}\; \vdash \lceil y := x_1 + 1.5 \rfloor$ and $\mathsf{comp}_z \;\widehat{=}\; \vdash \lceil z_0 := y \times 2.4 \rfloor$. During the time intervals, the values of x_1, y and z_0 keep unchanged, denoted by the side condition on the left of the time line.

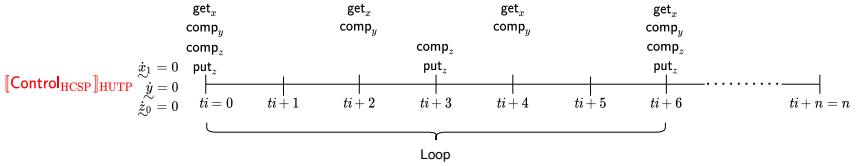

Fig. 10. Visual representation of HUTP semantics of $\mathsf{Control}_{\mathrm{HCSP}}$

Similarly, the behavior of Control in (11) can be illustrated by Fig. 11, where $\text{put}_y^0 \mathrel{\widehat{=}} \vdash \lceil out_0!, y_0 \rfloor$, $\text{get}_y^0 \mathrel{\widehat{=}} \vdash \lceil out_0?, y \rfloor$, $\text{put}_y^1 \mathrel{\widehat{=}} \vdash \lceil in_1!, y \rfloor$ and $\text{get}_y^1 \mathrel{\widehat{=}} \vdash \lceil in_1?, y_1 \rfloor$. Compared to Fig. 10, the behaviour of Control is consistent with $[\![\text{Control}_{\text{HCSP}}]\!]_{\text{HUTP}}$, except that the latter does not contain syn_y^0 (synchronous communication between put_y^0 and get_y^0) and syn_y^1 (synchronous communication between put_y^1 and get_y^1), because $[\![\text{Control}_{\text{HCSP}}]\!]_{\text{HUTP}}$ is a sequential process (and does not involve synchronous communication). This will lead to the inconsistency of the traces of $[\![\text{Control}_{\text{HCSP}}]\!]_{\text{HUTP}}$ and Control. In addition, the end time ti' of Control is not necessarily equal to the end time $ti{+}n = n$ of $[\![\text{Control}_{\text{HCSP}}]\!]_{\text{HUTP}}$. Thus, the premise of the consistency should be (1) φ_0: ignoring trace variables and (2) φ_1: $ti'\%1 = 0$. In summary,

$$\text{Control} \quad \equiv_{\varphi_0 \wedge \varphi_1} \quad [\![\text{Control}_{\text{HCSP}}]\!]_{\text{HUTP}} \tag{20}$$

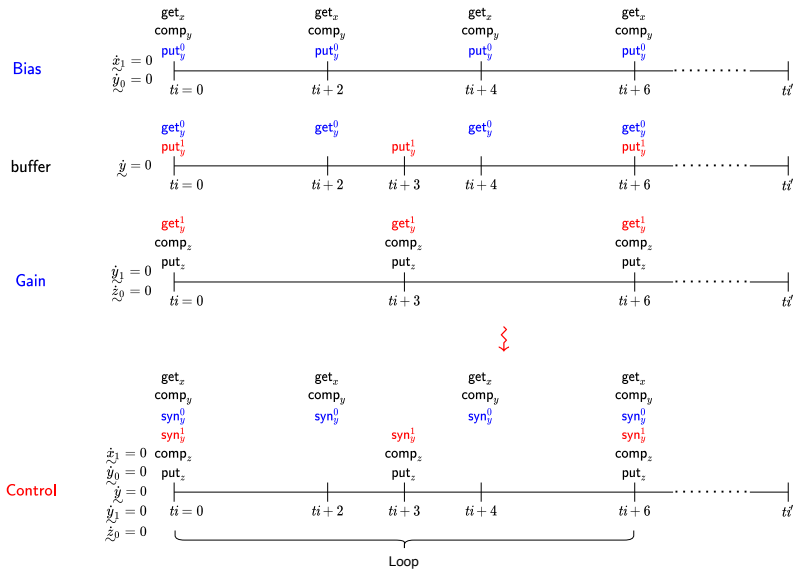

Fig. 11. Visual representation HUTP semantics of Control

Finally, according to (5), the HUTP semantics of $[\![\text{Diagram}]\!]_{\text{HCSP}}$ is

$$\begin{aligned} [\![[\![\text{Diagram}]\!]_{\text{HCSP}}]\!]_{\text{HUTP}} &= [\![\text{Plant}_{\text{HCSP}} \| \text{Control}_{\text{HCSP}}]\!]_{\text{HUTP}} \\ &= [\![\text{Plant}_{\text{HCSP}}]\!]_{\text{HUTP}} \|^{\text{HP}}_{M_{\{ch_x, ch_z\}}} [\![\text{Control}_{\text{HCSP}}]\!]_{\text{HUTP}} \end{aligned}$$

Note that the merge predicate M_I (Definition 4.5) allows the different termination time of parallel processes and it will delay the shorter one for the final time synchronisation. For example, $[\![\text{Plant}_{\text{HCSP}}]\!]_{\text{HUTP}}$ always terminate with put_x or get_z, while $[\![\text{Control}_{\text{HCSP}}]\!]_{\text{HUTP}}$ can terminate with $\text{put}_z ; \vdash \lceil \dot{x}_1 = \underset{\sim}{\dot{y}} = \dot{z}_0 = 0, \emptyset \rfloor_1 ; [\![\textbf{skip}]\!]_{\text{HUTP}} ; [\![\textbf{skip}]\!]_{\text{HUTP}}$. For time synchronisation, $M_{\{ch_x, ch_z\}}$ will delay $[\![\text{Plant}_{\text{HCSP}}]\!]_{\text{HUTP}}$ for 1 time unit. In fact, the termination time difference raises when $[\![\text{Control}_{\text{HCSP}}]\!]_{\text{HUTP}}$ terminates with $ti'\%2 \neq 0 \wedge ti'\%3 \neq 0$. Thus, if $ti'\%2 = 0 \vee ti'\%3 = 0$, $\|^{\text{HP}}_{M_{\{ch_x, ch_z\}}}$ will degrade into $\|_{\{ch_x, ch_z\}}$. Consider the condition $\gamma : ti'\%2 = 0 \vee ti'\%3 = 0$, it implies φ_1, and furthermore, it implies ϕ_1 because Control terminates with get_x or put_z under γ, which indicates Plant will terminate with put_x or get_z. In a word, $\gamma \Rightarrow \phi_1 \wedge \varphi_1$. Additionally, it is known $\varphi_0 \Rightarrow \phi_0$.

Combining (19), (20) and the monotonicity of $\|_{\{ch_x, ch_y\}}$ (Property 11),

$$\mathsf{Plant} \|_{\{in_0, out_1\}} \mathsf{Control} \quad \equiv_{\varphi_0 \wedge \gamma} \quad [\![\mathsf{Plant}_{\mathrm{HCSP}}]\!]_{\mathrm{HUTP}} \|_{\{ch_x, ch_z\}} [\![\mathsf{Control}_{\mathrm{HCSP}}]\!]_{\mathrm{HUTP}}$$

given that in_0 and out_1 are aliases of ch_x and ch_z, respectively. In summary, we can conclude

$$[\![\mathsf{Diagram}]\!]_{\mathrm{HUTP}} \quad \equiv_{T,S} \quad [\![[\![\mathsf{Diagram}]\!]_{\mathrm{HCSP}}]\!]_{\mathrm{HUTP}}$$

where $T = \{t \geq 0 \mid t\%2 = 0 \vee t\%3 = 0\}$ and $S = \{x_0, x_1, y, z_0, z_1, a_0, a_1\}$.

7 RELATED WORK

Hoare and He's unifying theories of programming (UTP) [28] is the first proposal towards unifying different programming paradigms and models under a common relational calculus suitable for design and verification. Another notable contribution in the same spirit is Milner's theory of bigraphs [40].

The concept of higher-order UTP dates back to [28], where higher-order variables are introduced to represent predicates of methods and procedures. Based on [28], Zeyda et al. propose a concept of higher-order UTP to set up a UTP theory for object-orientation [61, 62]. [61] builds a UTP theory for object-orientation by addressing four problems: consistency of the program model, redefinition of methods in subclasses, recursion and mutual recursion, and simplicity. [62] supports recursion, dynamic binding, and compositional method definitions all at the same time. It shows how higher-order programming can be used to reason about object-oriented programs in a compositional manner and exemplifies its use by creating an object-oriented variant of a refinement language for real-time systems. In addition, it introduces the higher-order quantification over predicates. Unlike these higher-order extensions to UTP [28, 61, 62], that presented in this paper focuses on higher-order quantification and the semantics of hybrid systems in the UTP theory.

Besides the de facto standards Simulink and Modelica, many more programming paradigms and domain-specific formalisms have been explored in the aim of modeling hybrid systems.

- Functional programming variants of data-flow concepts comprise Yampa [29], an extension of Haskell's functional reactive programming model (FRP, [46]), or Zelus [8], a functional synchronous data-flow language to model ordinary differential equations in a so-called non-standard time model. Like Modelica and Simulink, both lines of formalisms are devoted to simulation supporting type-based analysis of basic safety properties, such as causality.
- Another line of work regards earlier attempts to extend rich process algebras like the π-calculus with continuous real variables [7, 11, 30]. These extensions lead to the corresponding formulations of bi-simulations as a foundation to verification.

Our approach is to define such an algebraically expressive formalism, yet devoted to the primary purpose of verifying hybrid system models, by embedding its definition, verification and proof capabilities in higher-order logic (Isabelle/HOL).

Hybrid automata [3] are a popular formalism to model hybrid systems, but composition/decomposition of automata results in an exponential blow up which is intractable to analyze in practice. *I/O hybrid automata* [35] is an extension of hybrid automata with explicit inputs and outputs. Assume-guarantee reasoning [26] on such automata tackles composability to prevent state-space explosion. However, their applicability is in practice restricted to linear hybrid automata.

Alternatively, proof-theoretic approaches to hybrid systems verification, such as differential dynamic logic (dL, [47, 50]), action systems [4], hybrid Hoare logic together with hybrid communicating sequential processes (HHL/HCSP, [22, 31, 64, 67]) and hybrid Event-B [2, 12], have gained much attention by offering powerful abstraction, refinement and proof mechanisms to scale hybrid system verification to practical use cases. Also note an early model of CPSs in the Coq proof assistant [37], which introduces a library VeriDrone, a foundational framework for reasoning

about CPSs at all levels from high-level models to C code that implements the system. Empowered by the extension of SAT modulo-theory of real variables, implementing the seminal works on δ-decidability of [19, 20] in tools like dReal and dReach, provers allow for the tactics-guided mechanized verification of logic models of hybrid systems. [41] introduces a verification-driven engineering toolset Sφnx, which uses KeYmaera as hybrid verification tool, for modelling hybrid systems, exchanging and comparing models and proofs, and managing verification tasks.

Differential dynamic logic (dL) [47, 48] extends dynamic logic [51] to hybrid systems by allowing modalities over hybrid programs that extend classical sequential programs with ordinary differential equations to model the continuous evolution of physical artifacts. In order to deal with more complex behaviour of hybrid systems, some variants of dL were established, e.g., stochastic differential dynamic logic [48] and differential game logic [49].

In [32] Loos and Platzer propose a differential refinement logic to cope with refinement relations among different levels of abstraction for a given simple hybrid system. It remains an open problem whether the approach can apply to more complex hybrid system. In [52] the authors investigate how to apply dL to define architecture of CPSs. However, parallelism and communication are not considered in dL and its variants. Recent component-based verification methodologies developed in dL [44, 45, 55] propose composition operators in dL to split verification of systems into more manageable pieces. [33, 34] extend them with parallel composition (true concurrency) using design patterns defined using the primitive constructs of dL, which requires the corresponding mechanization effort as tactics in KeYmaeraX.

HCSP is a hybrid extension of Hoare's Communicating Sequential Processes [27]. It features a native parallel composition operator and communicating primitives in addition to standard constructs for hybrid systems (sequences, loops, ODEs) and a proof calculus called *hybrid Hoare logic* (HHL) [31], as well as several extensions for dealing with more complex behavious, e.g., fault-tolerance [58] and noise [59]. Also, the notion of *approximate bisimulation* was proposed in [60], with which a refinement relation between continuous models and between continuous models and discrete models can be well coped with. However, it is difficult to use HHL/HCSP to play a semantics foundation for the design of hybrid systems using MBD and DbC, as it is not easy to unify different models and views at different level of abstraction in HHL/HCSP.

Röonkkö and Ravn, and Back *et al.* extended action system [5] to hybrid systems, called *hybrid action system* [53] and *continuous action system* [4], respectively. As conservative extensions of action systems, theoretically speaking, both hybrid action system and continuous action system could support MBD and DbC. But both of them are very weak on dealing with continuous behaviours. In particular, they lack refinement relations, different continuous models and/or discrete models, which are essential for the MBD and DbC of hybrid systems.

8 CONCLUSIONS AND FUTURE WORKS

This paper defines a conservative extension to Hoare and He's UTP theory with higher-order quantification and provides a formal semantics for modeling and verifying hybrid systems, mixing discrete real-time processes and continuous dynamics. To this end, HUTP introduces real-time variables, denoted as functions over time, and allows the derivative of a variable to occur in a predicate, to define differential relations. This introduction allows to draw the concepts of hybrid processes and (normal) hybrid designs.

Our goal is to use the HUTP to define a formal semantics for engineering and formally verifying models such as Matlab's Simulink/Stateflow and/or the SAE's AADL, but also HCSP, and support the workflow of these environment with formal verification and certification functionalities inherited from its implementation in higher-order logic (Isabelle/HOL). We intend to additionally use HUTP to

justify the correctness of the translation between these models and HCSP by translation validation as an appliaction of HUTP.

Future Works. Starting from the definition of normal hybrid design contracts presented in this paper, we are seeking principled abstractions to support modular analysis and compositional verification of hybrid system models. Our initial intuition was to start from the definition of abstract hybrid processes in Section 4.5 and define a theory of contracts using Abadi and Lamport's "composing specifications" [1] or Benveniste's meta-theory of contracts [6]. Indeed, consider a normal hybrid design $\mathsf{HC} \vdash \mathsf{HP}$ and $\mathcal{H}^{\mathrm{A}}_{\mathrm{HP}}$-healthy abstract hybrid processes C and P such that $\mathsf{HC} \sqsubseteq C$ and $P \sqsubseteq \mathsf{HP}$. The abstract design $C \vdash P$ naturally forms a contractual property of $\mathsf{HC} \vdash \mathsf{HP}$, in the sense of Abadi and Lamport or Benveniste.

However, related works in the UTP [15] show that the algebraic elaboration of such a contract theory requires non-constructive definitions such as weakest liberal preconditions and weakest rely, which prevent the structural decomposition and abstraction of parallel composition and communications from the abstracted hybrid designs (unless one uses, e.g., Lamport's encoding of hand-shake communications using shared variables).

As a result, our aim will instead be to elaborate a dependently-typed session calculus [36] in the UTP and equip it with abstract model checking capabilities [56].

ACKNOWLEDGMENTS

This research is partly supported by NSFC under grant No. 62192732, 61625206, 62032024, 61972385 and 61732001, and is also partly funded by Inria's joint research project CONVEX. The authors would like to thank the editors and anonymous reviewers, whose criticisms and suggestions did improve the presentation of our work very much.

REFERENCES

[1] M. Abadi and L. Lamport. 1993. Composing Specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 15, 1 (January 1993), 73–132.

[2] J.-R. Abrial, W. Su, and H. Zhu. 2012. Formalizing Hybrid Systems with Event-B. In *3rd International Conference on Abstract State Machines, Alloy, B, VDM, and Z (ABZ) (Lecture Notes in Computer Science, Vol. 7316)*. Springer, Pisa, Italy, 178–193.

[3] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. 1993. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*. Springer, 209–229.

[4] R.-J. Back, L. Petre, and I. Porres. 2000. Generalizing Action Systems to Hybrid Systems. In *6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT) (Lecture Notes in Computer Science, Vol. 1926)*. Pune, India, 202–213.

[5] R.-J. Back and J. v. Wright. 1998. *Refinement Calculus–A Systematic Introduction.* Springer.

[6] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A.-L. Sangiovanni-Vincentelli, W. Damm, T.-A. Henzinger, and K. G. Larsen. 2018. Contracts for System Design. *Foundations and Trends in Electronic Design Automation* 12, 2–3 (2018), 124–400.

[7] J. A. Bergstra and C. A. Middelburg. 2005. Process Algebra for Hybrid Systems. *Theoretical Computer Science* 335, 2–3 (May 2005), 215–280.

[8] T. Bourke and M. Pouzet. 2013. Zélus: A Synchronous Language with ODEs. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC)*. ACM, Philadelphia, PA, USA, 113–118.

[9] A. Butterfield, P. Gancarski, and J. Woodcock. 2009. State Visibility and Communication in Unifying Theories of Programming. In *3rd IEEE International Symposim on Theoretical Aspects of Software Engineering*. IEEE Computer Society, Tianjin, China, 47–54.

[10] M. Chen, A. P. Ravn, S. Wang, M. Yang, and N. Zhan. 2016. A Two-Way Path between Formal and Informal Design of Embedded Systems. In *6th International Symposium on Unifying Theories of Programming (UTP) (Lecture Notes in Computer Science, Vol. 10134)*. Springer, Reykjavik, Iceland, 65–92.

[11] P. J. L. Cuijpers and M. A. Reniers. 2005. Hybrid Process Algebra. *Journal of Logic and Algebraic Programming* 62, 2 (February 2005), 191–245.

[12] G. Dupont, Y. Ait-Ameur, N. K. Singh, and M. Pantel. 2021. Event-B Hybridation: A Proof and Refinement-based Framework for Modelling Hybrid Systems. *ACM Transactions on Embedded Computing Systems* 20, 4 (June 2021), 35:1–35:37.

[13] S. Foster. 2019. Hybrid Relations in Isabelle/UTP. In *7th International Symposium on Unifying Theories of Programming (UTP), Dedicated to Tony Hoare on the Occasion of His 85th Birthday (Lecture Notes in Computer Science, Vol. 11885)*. Springer, Porto, Portugal, 130–153.

[14] S. Foster and J. Baxter. 2020. Automated Algebraic Reasoning for Collections and Local Variables with Lenses. In *18th International Conference on Relational and Algebraic Methods in Computer Science (RAMiCS) (Lecture Notes in Computer Science, Vol. 12062)*. Springer, Palaiseau, France, 100–116.

[15] S. Foster, A. Cavalcanti, S. Canham, J. Woodcock, and F. Zeyda. 2020. Unifying Theories of Reactive Design Contracts. *Theoretical Computer Science* 802 (January 2020), 105–140.

[16] S. Foster, A. Cavalcanti, J. Woodcock, and F. Zeyda. 2018. Unifying Theories of Time with Generalised Reactive Processes. *Inform. Process. Lett.* 135 (July 2018), 47–52.

[17] S. Foster, B. Thiele, A. Cavalcanti, and J. Woodcock. 2016. Towards a UTP Semantics for Modelica. In *6th International Symposium on Unifying Theories of Programming (UTP) (Lecture Notes in Computer Science, Vol. 10134)*. Springer, Reykjavik, Iceland, 44–64.

[18] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner. 2009. *Embedded System Design: Modeling, Synthesis, Verification*. Springer-Verlag.

[19] S. Gao, J. Avigad, and E. M. Clarke. 2012. Delta-Decidability over the Reals. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, Dubrovnik, Croatia, 305–314.

[20] S. Gao, S. Kong, and E. M. Clarke. 2013. dReal: An SMT Solver for Nonlinear Theories over the Reals. In *24th International Conference on Automated Deduction on Automated Deduction (CADE) (Lecture Notes in Computer Science, Vol. 7898)*. Springer, Lake Placid, NY, USA, 208–214.

[21] W. Guttmann and B. Möller. 2010. Normal Design Algebra. *Journal of Logic and Algebraic Programming* 79, 2 (February 2010), 144–173.

[22] J. He. 1994. From CSP to Hybrid Systems. In *A Classical Mind: Essays in Honour of C. A. R. Hoare*. Prentice Hall International (UK) Ltd., 171–189.

[23] J. He and Q. Li. 2017. A Hybrid Relational Modelling Language. In *Concurrency, Security, and Puzzles - Essays Dedicated to Andrew William Roscoe on the Occasion of His 60th Birthday*. Number 10160 in Lecture Notes in Computer Science. Springer, 124–143.

[24] J. He, X. Li, and Z. Liu. 2006. A Theory of Reactive Components. *Electronic Notes in Theoretical Computer Science* 160 (August 2006), 173–195.

[25] E. C. R. Hehner. 1993. *A Practical Theory of Programming*. Springer.

[26] T. A. Henzinger, M. Minea, and V. Prabhu. 2001. Assume-Guarantee Reasoning for Hierarchical Hybrid Systems. In *4th International Workshop on Hybrid Systems: Computation and Control (HSCC) (Lecture Notes in Computer Science, Vol. 2034)*. Springer, Rome, Italy, 275–290.

[27] C. A. R. Hoare. 1978. Communicating Sequential Processes. *Communications of the ACM (CACM)* 21, 8 (August 1978), 666–677.

[28] C. A. R. Hoare and J. He. 1998. *Unifying Theories of Programming*. Prentice Hall, Englewood Cliffs.

[29] P. Hudak, A. Courtney, H. Nilsson, and J. Peterson. 2002. Arrows, Robots, and Functional Reactive Programming. In *4th International School on Advanced Functional Programming (AFP) (Lecture Notes in Computer Science, Vol. 2638)*. Springer, Oxford, UK, 159–187.

[30] R. Lanotte and M. Merro. 2017. A Calculus of Cyber-Physical Systems. In *11th International Conference on Language and Automata Theory and Applications (LATA) (Lecture Notes in Computer Science, Vol. 10168)*. Springer International Publishing, Umeå, Sweden, 115–127.

[31] J. Liu, J. Lv, Z. Quan, N. Zhan, H. Zhao, C. Zhou, and L. Zou. 2010. A Calculus for Hybrid CSP. In *8th Asian Symposium on Programming Languages and Systems (APLAS) (Lecture Notes in Computer Science, Vol. 6461)*. Springer, Shanghai, China, 1–15.

[32] S. M. Loos and A. Platzer. 2016. Differential Refinement Logic. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. ACM, New York, NY, USA, 505–514.

[33] S. Lunel, B Boyer, and J.-P. Talpin. 2017. Compositional Proofs in Differential Dynamic Logic dL. In *17th International Conference on Application of Concurrency to System Design (ACSD)*. IEEE Computer Society, Zaragoza, Spain, 19–28.

[34] S. Lunel, S. Mitsch, B Boyer, and J.-P. Talpin. 2019. Parallel Composition and Modular Verification of Computer Controlled Systems in Differential Dynamic Logic. In *3rd World Congress on Formal Methods (FM) (Lecture Notes in Computer Science, Vol. 11800)*. Springer, Porto, Portugal, 354–370.

[35] N. A. Lynch, R. Segala, and F. W. Vaandrager. 2003. Hybrid I/O Automata. *Information and Computation* 185, 1 (August 2003), 105–157.

[36] R. Majumdar, N. Yoshida, and D. Zufferey. 2020. Multiparty Motion Coordination: From Choreographies to Robotics Programs. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (November 2020), 134:1–134:30.
[37] G. Malecha, D. Ricketts, M. M. Alvarez, and S. Lerner. 2016. Towards Foundational Verification of Cyber-Physical Systems. In *Science of Security for Cyber-Physical Systems Workshop*. IEEE Computer Society, Vienna, Austria, 1–5.
[38] O. Maler, Z. Manna, and A. Pnueli. 1991. From Timed to Hybrid Systems. In *REX Workshop on Real-Time: Theory in Practice (Lecture Notes in Computer Science, Vol. 600)*. Springer, Mook, The Netherlands, 447–484.
[39] MathWorks. 2013. *Simulink® User's Guide.* http://www.mathworks.com/help/pdf_doc/simulink/sl_using.pdf.
[40] R. Milner. 2006. Pure Bigraphs: Structure and Dynamics. *Information and Computation* 204, 1 (January 2006), 60–122.
[41] S. Mitsch. 2013. *Modeling and Analyzing Hybrid Systems with Sphinx.* Carnegie Mellon University / Johannes Kepler University.
[42] Modelica. 2014. *Modelica® –A Unified Object-Oriented Language for Systems Modeling (Language Specification)* (version 3.3 revision 1 ed.). Modelica Association.
[43] B. Möller. 2006. The Linear Algebra of UTP. In *Proceedings of the 8th International Conference on Mathematics of Program Construction (MPC) (Lecture Notes in Computer Science, Vol. 4014)*. Springer, Kuressaare, Estonia, 338–358.
[44] A Müller, S Mitsch, W Retschitzegger, W Schwinger, and A Platzer. 2016. A Component-Based Approach to Hybrid Systems Safety Verification. In *12th International Conference on Integrated Formal Methods (IFM) (Lecture Notes in Computer Science, Vol. 9681)*. Springer, Reykjavik, Iceland, 441–456.
[45] A. Müller, S. Mitsch, W. Retschitzegger, W. Schwinger, and A. Platzer. 2018. Tactical Contract Composition for Hybrid System Component Verification. *International Journal on Software Tools for Technology Transfer (STTT)* 20, 6 (November 2018), 615–643.
[46] J. Peterson, Z. Wan, P. Hudak, and H. Nilsson. 2001. *Yale FRP User's Manual.* Department of Computer Science, Yale University. http://www.haskell.org/frp/manual.html.
[47] A. Platzer. 2008. Differential Dynamic Logic for Hybrid Systems. *Journal of Automated Reasoning* 41, 2 (August 2008), 143–189.
[48] A. Platzer. 2010. *Logical Analysis of Hybrid Systems.* Springer.
[49] A. Platzer. 2015. Differential Game Logic. *ACM Transactions on Computational Logic* 17, 1 (December 2015), 1:1–1:51.
[50] A. Platzer. 2018. *Logical Foundations of Cyber-Physical Systems.* Springer.
[51] V. R. Pratt. 1976. Semantical Considerations on Floyd-Hoare Logic. In *17th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, Houston, Texas, USA, 109–121.
[52] A. Rajhans, A. Bhave, I Ruchkin, B. H. Krogh, D Garlan, A. Platzer, and B. R. Schmerl. 2014. Supporting Heterogeneity in Cyber-Physical Systems Architectures. *IEEE Trans. Automat. Control* 59, 12 (December 2014), 3178–3193.
[53] M. Rönkkö and A. P. Ravn. 1997. Action Systems with Continuous Behaviour. In *Proceedings of the 5th International Workshop on Hybrid Systems (Hybrid Systems V) (Lecture Notes in Computer Science, Vol. 1567)*. Springer, Notre Dame, IN, USA, 304–323.
[54] A. W. Roscoe. 1997. *The Theory and Practice of Concurrency.* Prentice Hall.
[55] I. Ruchkin, B. Schmerl, and D. Garlan. 2015. Architecture Abstractions for Hybrid Programs. In *18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*. ACM, Montreal, QC, Canada, 65–74.
[56] A. Scalas, N. Yoshida, and E. Benussi. 2019. Verifying Message-Passing Programs with Dependent Behavioural Types. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. Phoenix, AZ, USA, 502—516.
[57] A. Tarski. 1955. A Lattice-Theoretical Fixpoint Theorem and Its Applications. *Pacific J. Math.* 5, 2 (1955), 285–309.
[58] S. Wang, F. Nielson, H. R. Nielson, and N. Zhan. 2017. Modelling and Verifying Communication Failure of Hybrid Systems in HCSP. *Comput. J.* 60, 8 (August 2017), 1111–1130.
[59] S. Wang, N. Zhan, and L. Zhang. 2017. A Compositional Modelling and Verification Framework for Stochastic Hybrid Systems. *Formal Aspects of Computing* 29, 4 (July 2017), 751–775.
[60] G. Yan, L. Jiao, S. Wang, L. Wang, and N. Zhan. 2020. Automatically Generating SystemC Code from HCSP Formal Models. *ACM Transactions on Software Engineering and Methodology* 29, 1 (January 2020), 4:1–4:39.
[61] F. Zeyda and A. Cavalcanti. 2012. Higher-Order UTP for a Theory of Methods. In *4th International Symposium on Unifying Theories of Programming (UTP) (Lecture Notes in Computer Science, Vol. 7681)*. Springer, Paris, France, 204–223.
[62] F. Zeyda, T. Santos, A. Cavalcanti, and A. Sampaio. 2014. A Modular Theory of Object Orientation in Higher-Order UTP. In *19th International Symposium on Formal Methods (FM) (Lecture Notes in Computer Science, Vol. 8442)*. Springer, Singapore, 627–642.
[63] Bohua Zhan, Bin Gu, Xiong Xu, Xiangyu Jin, Shuling Wang, Bai Xue, Xiaofeng Li, Yao Chen, Mengfei Yang, and Naijun Zhan. 2021. Brief Industry Paper: Modeling and Verification of Descent Guidance Control of Mars Lander. In *RTAS 2021*. 457–460.
[64] N. Zhan, S. Wang, and H. Zhao. 2017. *Formal Verification of Simulink/Stateflow Diagrams–A Deductive Approach.* Springer.

[65] H. Zhao, M. Yang, N. Zhan, B. Gu, L. Zou, and Y. Chen. 2014. Formal Verification of a Descent Guidance Control Program of a Lunar Lander. In *19th International Symposium on Formal Methods (FM) (Lecture Notes in Computer Science, Vol. 8442)*. Springer, Singapore, 733–748.

[66] C. Zhou, C. A. R. Hoare, and A. P. Ravn. 1991. A Calculus of Durations. *Inform. Process. Lett.* 40, 5 (December 13 1991), 269–276.

[67] C. Zhou, J. Wang, and A. P. Ravn. 1995. A Formal Description of Hybrid Systems. In *Proceedings of the DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, Hybrid Systems III: Verification and Control (Lecture Notes in Computer Science, Vol. 1066)*. Springer, Ruttgers University, New Brunswick, NJ, USA, 511–530.

[68] L. Zou, N. Zhan, S. Wang, and M. Fränzle. 2015. Formal Verification of Simulink/Stateflow Diagrams. In *13th International Symposium on Automated Technology for Verification and Analysis (ATVA) (Lecture Notes in Computer Science, Vol. 9364)*. Springer, Shanghai, China, 464–481.

[69] L. Zou, N. Zhan, S. Wang, M. Fränzle, and S. Qin. 2013. Verifying Simulink Diagrams Via A Hybrid Hoare Logic Prover. In *International Conference on Embedded Software (EMSOFT)*. IEEE, Montreal, QC, Canada, 9:1–9:10.

A PROOFS

A.1 Proofs for Timed Trace Model

Property 1 (Associativity). *Let I_0, I_1 and I_2 be the respective common channel sets between timed traces $\mathtt{tt}_0$ and $\mathtt{tt}_1$, $\mathtt{tt}_1$ and $\mathtt{tt}_2$, and $\mathtt{tt}_2$ and $\mathtt{tt}_0$, then $(\mathtt{tt}_0 \|_{I_0} \mathtt{tt}_1) \|_{I_1 \uplus I_2} \mathtt{tt}_2 = \mathtt{tt}_0 \|_{I_0 \uplus I_2} (\mathtt{tt}_1 \|_{I_1} \mathtt{tt}_2)$.*

Proof. We prove by induction on the length of traces. Since the traces we consider could be infinite, the proof is coinductive. We start from the cases that $|\mathtt{tt}_0|, |\mathtt{tt}_1|, |\mathtt{tt}_2| \leq 1$, such as $\mathtt{tt}_0 = \mathtt{tt}_1 = \mathtt{tt}_2 = \epsilon$, then by the fact that I_0, I_1 and I_2 are disjoint, we can get

$$(\mathtt{tt}_0 \|_{I_0} \mathtt{tt}_1) \|_{I_1 \uplus I_2} \mathtt{tt}_2 \quad = \quad \mathtt{tt}_0 \|_{I_0 \uplus I_2} (\mathtt{tt}_1 \|_{I_1} \mathtt{tt}_2)$$

which indicates the associativity of the parallel composition of timed traces. □

A.2 Proofs for Hybrid Processes

Theorem 4.2 (Idempotence and Monotonicity). *$\mathcal{H}_{\mathrm{HP}}$ is idempotent and monotonic.*

Proof. First, $\mathcal{H}_{\mathrm{HP}}$ is monotonic because it is constructed by monotonic operators. Then, we prove it is also idempotent. It can be checked that $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ and $\mathcal{H}_4$ themselves are idempotent and $\mathcal{H}_0$ and $\mathcal{H}_4$ are commutative with their partners. Therefore,

$$\begin{aligned} \mathcal{H}_{\mathrm{HP}} \circ \mathcal{H}_{\mathrm{HP}} &= (\mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4) \circ (\mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4) \\ &= (\mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3) \circ (\mathcal{H}_0 \circ \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4 \circ \mathcal{H}_4) \\ &= (\mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3) \circ (\mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4) \end{aligned} \tag{21}$$

It is not difficult to check that $\mathcal{H}_3$ is commutative with $\mathcal{H}_0$ and $\mathcal{H}_1$, and we prove it is also commutative with $\mathcal{H}_2$. For short, let $\mathsf{inf} \mathrel{\widehat{=}} (ti = +\infty \vee |tr| = +\infty)$ and $\mathsf{inf}' \mathrel{\widehat{=}} (ti' = +\infty \vee |tr'| = +\infty)$ correspondingly. Let $\natural \mathrel{\widehat{=}} (ti = ti' \wedge tr = tr')$. Then,

$$\mathcal{H}_2 \circ \mathcal{H}_3(X) \quad = \quad \natural \lhd \mathsf{inf} \rhd ((\exists \mathbf{s}' \cdot X) \lhd \mathsf{inf}' \rhd X)$$

and

$$\begin{aligned} \mathcal{H}_3 \circ \mathcal{H}_2(X) &= (\exists \mathbf{s}' \cdot (\natural \lhd \mathsf{inf} \rhd X)) \lhd \mathsf{inf}' \rhd (\natural \lhd \mathsf{inf} \rhd X) \\ &= (\natural \lhd \mathsf{inf} \rhd \exists \mathbf{s}' \cdot X) \lhd \mathsf{inf}' \rhd (\natural \lhd \mathsf{inf} \rhd X) \\ &= \natural \lhd \mathsf{inf} \rhd (\exists \mathbf{s}' \cdot X \lhd \mathsf{inf}' \rhd X) \end{aligned}$$

Therefore, $\mathcal{H}_2 \circ \mathcal{H}_3 = \mathcal{H}_3 \circ \mathcal{H}_2$, and we can continue the equation (21):

$$\begin{aligned} \mathcal{H}_{\mathrm{HP}} \circ \mathcal{H}_{\mathrm{HP}} &= (\mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3) \circ (\mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4) \\ &= \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_3 \circ \mathcal{H}_4 \\ &= \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4 \end{aligned}$$

Lastly, we prove the composition $\mathcal{H}_1 \circ \mathcal{H}_2$ is idempotent. Concretely,

$$\mathcal{H}_1 \circ \mathcal{H}_2(X) \quad = \quad \mathcal{H}_1(\natural) \lhd \mathsf{inf} \rhd \mathcal{H}_1(X)$$

Then,

$$\begin{array}{rcl} \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_1 \circ \mathcal{H}_2(X) & = & \mathcal{H}_1(\natural) \lhd \mathsf{inf} \rhd \mathcal{H}_1\left(\mathcal{H}_1(\natural) \lhd \mathsf{inf} \rhd \mathcal{H}_1(X)\right) \\ & = & \mathcal{H}_1(\natural) \lhd \mathsf{inf} \rhd \left(\mathcal{H}_1(\natural) \lhd \mathsf{inf} \rhd \mathcal{H}_1(X)\right) \\ & = & \mathcal{H}_1(\natural) \lhd \mathsf{inf} \rhd \mathcal{H}_1(X) \quad = \quad \mathcal{H}_1 \circ \mathcal{H}_2(X) \end{array}$$

In summary, $\mathcal{H}_{\mathrm{HP}} \circ \mathcal{H}_{\mathrm{HP}} = \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4 = \mathcal{H}_{\mathrm{HP}}$, i.e., it is idempotent. □

Property 2. *If* HP_0 *and* HP_1 *are* $\mathcal{H}_{\mathrm{HP}}$*-healthy,* $\mathsf{HP}_0 \sqcap \mathsf{HP}_1$ *and* $\mathsf{HP}_0 \sqcup \mathsf{HP}_1$ *are* $\mathcal{H}_{\mathrm{HP}}$*-healthy.*

Proof. It can be checked that $\mathcal{H}_4$ is commutative with its partners, thus

$$\begin{array}{rcll} \mathcal{H}_{\mathrm{HP}}(X) & = & \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4(X) & \\ & = & \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_4 \circ \mathcal{H}_2 \circ \mathcal{H}_3(X) & \\ & = & \mathcal{H}_{014}\left(\natural \lhd \mathsf{inf} \rhd \left((\exists \mathbf{s}' \cdot X) \lhd \mathsf{inf}' \rhd X\right)\right) & (22) \end{array}$$

where $\mathcal{H}_{014} \mathrel{\widehat{=}} \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_4$. Then, $\mathcal{H}_{\mathrm{HP}}(X) \sqcap \mathcal{H}_{\mathrm{HP}}(Y)$ and $\mathcal{H}_{\mathrm{HP}}(X) \sqcup \mathcal{H}_{\mathrm{HP}}(Y)$ are

$$\begin{array}{rcl} \mathcal{H}_{014}\left(\natural \lhd \mathsf{inf} \rhd \left((\exists \mathbf{s}' \cdot X \vee Y) \lhd \mathsf{inf}' \rhd (X \vee Y)\right)\right) & = & \mathcal{H}_{\mathrm{HP}}(X \vee Y) \\ \mathcal{H}_{014}\left(\natural \lhd \mathsf{inf} \rhd \left((\exists \mathbf{s}' \cdot X \wedge Y) \lhd \mathsf{inf}' \rhd (X \wedge Y)\right)\right) & = & \mathcal{H}_{\mathrm{HP}}(X \wedge Y) \end{array}$$

Therefore, $\sqcap$ and $\sqcup$ are $\mathcal{H}_{\mathrm{HP}}$-preserving. □

Lemma A.1. *$\mathcal{H}_{014}(X) \fatsemi \mathcal{H}_{014}(Y) = \mathcal{H}_{014}(X \fatsemi Y)$, where $\mathcal{H}_{014} \mathrel{\widehat{=}} \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_4$.*

Proof. The computation is performed as follows:

$$\begin{array}{rcl} \mathcal{H}_{014}(X) \fatsemi \mathcal{H}_{014}(Y) & = & X \wedge \lim(tr) = ti \leq ti' = \lim(tr') \wedge tr \leq tr' \wedge RC \wedge SD \quad \fatsemi \\ & & Y \wedge \lim(tr) = ti \leq ti' = \lim(tr') \wedge tr \leq tr' \wedge RC \wedge SD \\ & = & \exists ti_*, \mathbf{s}_*, tr_* \cdot X[ti_*, \mathbf{s}_*, tr_*/ti', \mathbf{s}', tr'] \wedge Y[ti_*, \mathbf{s}_*, tr_*/ti, \mathbf{s}, tr] \\ & & \wedge \lim(tr) = ti \leq ti_* = \lim(tr_*) \leq ti' = \lim(tr') \wedge tr \leq tr_* \leq tr' \\ & & \wedge RC[ti_*/ti'] \wedge RC[ti_*/ti] \wedge SD[ti_*/ti'] \wedge SD[ti_*/ti] \\ & = & (X \fatsemi Y) \wedge \lim(tr) = ti \leq ti' = \lim(tr') \wedge tr \leq tr' \wedge RC \wedge SD \\ & = & \mathcal{H}_{014}(X \fatsemi Y) \end{array}$$

□

Property 3. *If* HP_0 *and* HP_1 *are* $\mathcal{H}_{\mathrm{HP}}$*-healthy, then so is* $\mathsf{HP}_0 \fatsemi \mathsf{HP}_1$.

Proof. As stated in the proof of Property 2 (see (22)), a hybrid process $\mathcal{H}_{\mathrm{HP}}(X)$ can be represented by $\mathcal{H}_{014}\left(\natural \lhd \mathsf{inf} \rhd (\exists \mathbf{s}' \cdot X \lhd \mathsf{inf}' \rhd X)\right)$. This form can be transformed into a matrix:

$$\begin{array}{c} \begin{array}{cc} \neg\mathsf{inf}' & \mathsf{inf}' \end{array} \\ \begin{pmatrix} \mathcal{H}_{014}(X \wedge \neg\mathsf{inf} \wedge \neg\mathsf{inf}') & \mathcal{H}_{014}(\exists \mathbf{s}' \cdot X \wedge \neg\mathsf{inf} \wedge \mathsf{inf}') \\ \mathbf{false} & \mathcal{H}_{014}(\natural \wedge \mathsf{inf} \wedge \mathsf{inf}') \end{pmatrix} \begin{array}{c} \neg\mathsf{inf} \\ \mathsf{inf} \end{array} \end{array}$$

Imagine $\fatsemi$ and $\vee$ as matrix multiplication and addition, respectively, then $\mathcal{H}_{\mathrm{HP}}(X) \fatsemi \mathcal{H}_{\mathrm{HP}}(Y)$ is

$$\begin{pmatrix} \mathcal{H}_{014}(X \wedge \neg\mathsf{inf} \wedge \neg\mathsf{inf}') \fatsemi & \mathcal{H}_{014}(X \wedge \neg\mathsf{inf} \wedge \neg\mathsf{inf}') \fatsemi \mathcal{H}_{014}(\exists \mathbf{s}' \cdot Y \wedge \neg\mathsf{inf} \wedge \mathsf{inf}') \\ \mathcal{H}_{014}(Y \wedge \neg\mathsf{inf} \wedge \neg\mathsf{inf}') & \vee \mathcal{H}_{014}(\exists \mathbf{s}' \cdot X \wedge \neg\mathsf{inf} \wedge \mathsf{inf}') \fatsemi \mathcal{H}_{014}(\natural \wedge \mathsf{inf} \wedge \mathsf{inf}') \\ \\ \mathbf{false} & \mathcal{H}_{014}(\natural \wedge \mathsf{inf} \wedge \mathsf{inf}') \end{pmatrix}$$

By Lemma A.1, it can be simplified to

$$\begin{pmatrix} \mathcal{H}_{014}\left((X \fatsemi Y) \wedge \neg \mathsf{inf} \wedge \neg \mathsf{inf}'\right) & \mathcal{H}_{014}\left(\exists \mathbf{s}' \cdot (X \fatsemi Y \ \vee \ X) \wedge \neg \mathsf{inf} \wedge \mathsf{inf}'\right) \\ \mathbf{false} & \mathcal{H}_{014}(\natural \wedge \mathsf{inf} \wedge \mathsf{inf}') \end{pmatrix}$$

which is equivalent to

$$\mathcal{H}_{014}\left(\natural \lhd \mathsf{inf} \rhd (\exists \mathbf{s}' \cdot (X \fatsemi Y \ \vee \ X) \lhd \mathsf{inf}' \rhd (X \fatsemi Y))\right) \tag{23}$$

where $\fatsemi$ takes precedence over $\vee$. The predicate $\natural \lhd \mathsf{inf} \rhd (\exists \mathbf{s}' \cdot (X \fatsemi Y \vee X) \lhd \mathsf{inf}' \rhd (X \fatsemi Y))$ is $\mathcal{H}_2$-healthy, and we prove it is also $\mathcal{H}_3$-healthy.

$$\begin{array}{l} \mathcal{H}_3\left(\natural \lhd \mathsf{inf} \rhd (\exists \mathbf{s}' \cdot (X \fatsemi Y \vee X) \lhd \mathsf{inf}' \rhd (X \fatsemi Y))\right) = \\ \quad (\natural \lhd \mathsf{inf} \rhd (\exists \mathbf{s}' \cdot (X \fatsemi Y \vee X) \lhd \mathsf{inf}' \rhd \exists \mathbf{s}' \cdot (X \fatsemi Y))) \\ \quad \lhd \mathsf{inf}' \rhd \quad (\natural \lhd \mathsf{inf} \rhd (\exists \mathbf{s}' \cdot (X \fatsemi Y \vee X) \lhd \mathsf{inf}' \rhd (X \fatsemi Y))) \end{array}$$

which can be simplified to $(\natural \lhd \mathsf{inf} \rhd (\exists \mathbf{s}' \cdot (X \fatsemi Y \vee X))) \lhd \mathsf{inf}' \rhd (\natural \lhd \mathsf{inf} \rhd (X \fatsemi Y))$, equivalent to $\natural \lhd \mathsf{inf} \rhd (\exists \mathbf{s}' \cdot (X \fatsemi Y \vee X) \lhd \mathsf{inf}' \rhd (X \fatsemi Y))$ itself. Therefore, (23) is $\mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_4 \circ \mathcal{H}_2 \circ \mathcal{H}_3$-healthy, i.e., $\mathcal{H}_{\mathrm{HP}}(X) \fatsemi \mathcal{H}_{\mathrm{HP}}(Y)$ is $\mathcal{H}_{\mathrm{HP}}$-healthy. □

Property 4. *The healthiness condition $\mathcal{H}_3$ is equivalent to*

$$X \quad = \quad X \fatsemi \mathsf{skip}$$

where $\mathsf{skip} \mathrel{\widehat{=}} \mathcal{H}_2(ti = ti' \wedge tr = tr' \wedge \mathbf{s} = \mathbf{s}')$.

Proof. With the aid of matrix representation, skip can be transformed into a matrix:

$$\begin{array}{c} \begin{array}{cc} \neg \mathsf{inf}' & \mathsf{inf}' \end{array} \\ \begin{pmatrix} \natural \wedge \mathbf{s} = \mathbf{s}' \wedge \neg \mathsf{inf} \wedge \neg \mathsf{inf}' & \mathbf{false} \\ \mathbf{false} & \natural \wedge \mathsf{inf} \wedge \mathsf{inf}' \end{pmatrix} \end{array} \begin{array}{c} \neg \mathsf{inf} \\ \mathsf{inf} \end{array}$$

Then, $X \fatsemi \mathsf{skip}$ can be computed as

$$\begin{pmatrix} X \wedge \neg \mathsf{inf} \wedge \neg \mathsf{inf}' & X \wedge \neg \mathsf{inf} \wedge \mathsf{inf}' \\ X \wedge \mathsf{inf} \wedge \neg \mathsf{inf}' & X \wedge \mathsf{inf} \wedge \mathsf{inf}' \end{pmatrix} \fatsemi \begin{pmatrix} \natural \wedge \mathbf{s} = \mathbf{s}' \wedge \neg \mathsf{inf} \wedge \neg \mathsf{inf}' & \mathbf{false} \\ \mathbf{false} & \natural \wedge \mathsf{inf} \wedge \mathsf{inf}' \end{pmatrix}$$

which can be simplified to

$$\begin{pmatrix} X \wedge \neg \mathsf{inf} \wedge \neg \mathsf{inf}' & \exists \mathbf{s}' \cdot X \wedge \neg \mathsf{inf} \wedge \mathsf{inf}' \\ X \wedge \neg \mathsf{inf} \wedge \neg \mathsf{inf}' & \exists \mathbf{s}' \cdot X \wedge \neg \mathsf{inf} \wedge \mathsf{inf}' \end{pmatrix}$$

The above is equivalent to $(\exists \mathbf{s}' \cdot X) \lhd \mathsf{inf}' \rhd X$, which denotes $\mathcal{H}_3$. □

Property 5 (Quantification).

$$\begin{array}{rcl} \exists \underset{\sim}{x} \cdot (\mathsf{HP}_0 \fatsemi \mathsf{HP}_1) & = & (\exists \underset{\sim}{x} \cdot \mathsf{HP}_0) \fatsemi (\exists \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \forall \underset{\sim}{x} \cdot (\mathsf{HP}_0 \fatsemi \mathsf{HP}_1) & \sqsupseteq & (\forall \underset{\sim}{x} \cdot \mathsf{HP}_0) \fatsemi (\forall \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \exists \underset{\sim}{x} \cdot (\mathsf{HP}_0 \sqcap \mathsf{HP}_1) & = & (\exists \underset{\sim}{x} \cdot \mathsf{HP}_0) \sqcap (\exists \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \forall \underset{\sim}{x} \cdot (\mathsf{HP}_0 \sqcap \mathsf{HP}_1) & \sqsupseteq & (\forall \underset{\sim}{x} \cdot \mathsf{HP}_0) \sqcap (\forall \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \exists \underset{\sim}{x} \cdot (\mathsf{HP}_0 \sqcup \mathsf{HP}_1) & \sqsubseteq & (\exists \underset{\sim}{x} \cdot \mathsf{HP}_0) \sqcup (\exists \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \forall \underset{\sim}{x} \cdot (\mathsf{HP}_0 \sqcup \mathsf{HP}_1) & = & (\forall \underset{\sim}{x} \cdot \mathsf{HP}_0) \sqcup (\forall \underset{\sim}{x} \cdot \mathsf{HP}_1) \\ \neg_{\imath} \exists \underset{\sim}{x} \cdot \mathsf{HP} & = & \forall \underset{\sim}{x} \cdot \neg_{\imath} \mathsf{HP} \\ \neg_{\imath} \forall \underset{\sim}{x} \cdot \mathsf{HP} & = & \exists \underset{\sim}{x} \cdot \neg_{\imath} \mathsf{HP} \end{array}$$

Proof. We only prove the second item, and others can be proved similarly.

$$
\begin{aligned}
&(\forall \underset{\sim}{x} \cdot \mathsf{HP}_0) \fatsemi (\forall \underset{\sim}{x} \cdot \mathsf{HP}_1) \\
&= \bigwedge\{\mathsf{HP}_0[f_0, g_0/\underset{\sim}{x}, \dot{\underset{\sim}{x}}] \mid f_0 : [ti, ti') \to \mathbb{D} \wedge g_0 : (ti, ti') \rightharpoonup \mathbb{D} \wedge \mathrm{CON}(f_0, g_0, ti, ti')\} \fatsemi \\
&\quad \bigwedge\{\mathsf{HP}_1[f_1, g_1/\underset{\sim}{x}, \dot{\underset{\sim}{x}}] \mid f_1 : [ti, ti') \to \mathbb{D} \wedge g_1 : (ti, ti') \rightharpoonup \mathbb{D} \wedge \mathrm{CON}(f_1, g_1, ti, ti')\} \\
&= \exists ti_*, \mathbf{s}_*, tr_* \cdot \bigwedge \left\{ \mathsf{HP}_0[ti_*, \mathbf{s}_*, tr_*, f_0, g_0/ti', \mathbf{s}', tr', \underset{\sim}{x}, \dot{\underset{\sim}{x}}] \middle| \begin{array}{c} f_0 : [ti, ti') \to \mathbb{D} \wedge g_0 : (ti, ti') \rightharpoonup \mathbb{D} \\ \wedge\ \mathrm{CON}(f_0, g_0, ti, ti') \end{array} \right\} \\
&\qquad \wedge \bigwedge \left\{ \mathsf{HP}_1[ti_*, \mathbf{s}_*, tr_*, f_1, g_1/ti', \mathbf{s}', tr', \underset{\sim}{x}, \dot{\underset{\sim}{x}}] \middle| \begin{array}{c} f_1 : [ti, ti') \to \mathbb{D} \wedge g_1 : (ti, ti') \rightharpoonup \mathbb{D} \\ \wedge\ \mathrm{CON}(f_1, g_1, ti, ti') \end{array} \right\} \\
&= \exists ti_*, \mathbf{s}_*, tr_* \cdot \bigwedge \left\{ \begin{array}{l} \mathsf{HP}_0[ti_*, \mathbf{s}_*, tr_*, f_0, g_0/ti', \mathbf{s}', tr', \underset{\sim}{x}, \dot{\underset{\sim}{x}}] \\ \wedge \mathsf{HP}_1[ti_*, \mathbf{s}_*, tr_*, f_1, g_1/ti, \mathbf{s}, tr, \underset{\sim}{x}, \dot{\underset{\sim}{x}}] \end{array} \middle| \begin{array}{l} f_0 : [ti, ti_*) \to \mathbb{D} \wedge g_0 : (ti, ti_*) \rightharpoonup \mathbb{D} \wedge \\ f_1 : [ti_*, ti') \to \mathbb{D} \wedge g_1 : (ti, ti_*) \rightharpoonup \mathbb{D} \wedge \\ \mathrm{CON}(f_0, g_0, ti, ti_*) \wedge \mathrm{CON}(f_1, g_1, ti_*, ti') \end{array} \right\} \\
&= \exists ti_*, \mathbf{s}_*, tr_* \cdot \bigwedge \left\{ \begin{pmatrix} \mathsf{HP}_0[ti_*, \mathbf{s}_*, tr_*, f_0, g_0/ti', \mathbf{s}', tr', \underset{\sim}{x}, \dot{\underset{\sim}{x}}] \\ \wedge \mathsf{HP}_1[ti_*, \mathbf{s}_*, tr_*, f_1, g_1/ti, \mathbf{s}, tr, \underset{\sim}{x}, \dot{\underset{\sim}{x}}] \end{pmatrix} [f, g/\underset{\sim}{x}, \dot{\underset{\sim}{x}}] \middle| \begin{array}{l} f : [ti, ti') \to \mathbb{D} \wedge \\ g : (ti, ti') \rightharpoonup \mathbb{D} \wedge \\ \mathrm{CON}(f, g, ti, ti') \end{array} \right\} \\
&\sqsubseteq \bigwedge \left\{ \exists ti_*, \mathbf{s}_*, tr_* \cdot \begin{pmatrix} \mathsf{HP}_0[ti_*, \mathbf{s}_*, tr_*, f_0, g_0/ti', \mathbf{s}', tr', \underset{\sim}{x}, \dot{\underset{\sim}{x}}] \\ \wedge \mathsf{HP}_1[ti_*, \mathbf{s}_*, tr_*, f_1, g_1/ti, \mathbf{s}, tr, \underset{\sim}{x}, \dot{\underset{\sim}{x}}] \end{pmatrix} [f, g/\underset{\sim}{x}, \dot{\underset{\sim}{x}}] \middle| \begin{array}{l} f : [ti, ti') \to \mathbb{D} \wedge \\ g : (ti, ti') \rightharpoonup \mathbb{D} \wedge \\ \mathrm{CON}(f, g, ti, ti') \end{array} \right\} \\
&= \bigwedge\{(\mathsf{HP}_0 \fatsemi \mathsf{HP}_1)[f, g/\underset{\sim}{x}, \dot{\underset{\sim}{x}}] \mid f : [ti, ti') \to \mathbb{D} \wedge g : (ti, ti') \rightharpoonup \mathbb{D} \wedge \mathrm{CON}(f, g, ti, ti')\} \\
&= \forall \underset{\sim}{x} \cdot (\mathsf{HP}_0 \fatsemi \mathsf{HP}_1)
\end{aligned}
$$

where f is the concatenation of f_0 and f_1, g is the concatenation of g_0 and g_1, and

$$\mathrm{CON}(f, g, ti, ti') \quad \widehat{=} \quad \forall t \in (ti, ti') \cdot \dot{f}(t^-) = g(t^-) \wedge \dot{f}(t^+) = g(t^+)$$

□

Property 6 (Well-Definedness). *M_I is well-defined.*

Proof. The conditions (1) and (2) of well-definedness of M_I hold by its definition. We prove the condition (3) also holds for M_I. First, if $|tr_X| < +\infty$ and $|tr_Y| < +\infty$, by the definition of $\|_I$, $|tr'| < +\infty$. Then, if $|tr_X| = +\infty$ or $|tr_Y| = +\infty$, say $|tr_X| = +\infty$, there are two cases: (a) $|tr| = +\infty$, since $tr \le tr'$ by $\mathcal{H}_0$, $|tr'| = +\infty$; (b) $|tr| < +\infty$, meaning $|tr_X - tr| = +\infty$. According to the definition of $\|_I$, the composed trace is not shorter than the individual traces, i.e., $|tr' - tr| \ge |tr_X - tr| = +\infty$, implying $|tr'| = +\infty$. Therefore, the condition (3) holds, indicating M_I is well-defined. □

Property 7 (Monotonicity). *The operators $\fatsemi$, $\sqcup$, $\sqcap$, $\lhd\rhd$, $\exists$ and $\|_M$ are monotonic and $\neg_\wr$ is anti-monotone, with respect to the refinement order $\sqsubseteq$.*

Proof. These operators except $\neg_\wr$ are constructed by monotonic atom operators. □

Theorem 4.6 (Complete Lattice). *Let $\mathbb{HP}$ be the image of $\mathcal{H}_{\mathrm{HP}}$, i.e., it is the set of all hybrid processes, then it forms a complete lattice with top and bottom:*

$$\begin{array}{rcccl} \top_{\mathrm{HP}} & \widehat{=} & \bigsqcup \mathbb{HP} & = & \mathcal{H}_{\mathrm{HP}}(\mathbf{false}) \\ \bot_{\mathrm{HP}} & \widehat{=} & \bigsqcap \mathbb{HP} & = & \mathcal{H}_{\mathrm{HP}}(\mathbf{true}) \end{array}$$

Proof. Since $\mathcal{H}_{\mathrm{HP}}$ is idempotent and monotonic by Theorem 4.2, the complete lattice can be proved directly by the properties of $\bigsqcap$ and $\bigsqcup$. □

Theorem 4.11 (Miracle). *$\top_{\mathrm{HP}}$ is the only miracle of hybrid processes.*

Proof. As mentioned in the proof of Property 2 (see (22)), a hybrid process $\mathcal{H}_{\mathrm{HP}}(X)$ can be unfolded as $\mathcal{H}_{014}\,(\natural \lhd \mathsf{inf} \rhd ((\exists \mathbf{s}' \cdot X) \lhd \mathsf{inf}' \rhd X))$, where $\mathcal{H}_{014} \mathrel{\widehat{=}} \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_4$. Then,

$$\begin{array}{rcl} \mathcal{H}_{\mathrm{HP}} \circ \mathit{MIR}(X) & = & \mathcal{H}_{014}\,(\natural \lhd \mathsf{inf} \rhd ((\exists \mathbf{s}' \cdot X \wedge \mathsf{inf}) \lhd \mathsf{inf}' \rhd (X \wedge \mathsf{inf}))) \\ & = & \mathcal{H}_{014}\,(\natural \lhd \mathsf{inf} \rhd (\mathbf{false} \lhd \mathsf{inf}' \rhd \mathbf{false})) \\ & = & \mathcal{H}_{014} \circ \mathcal{H}_2 \circ \mathcal{H}_3(\mathbf{false}) \quad = \quad \mathcal{H}_{\mathrm{HP}}(\mathbf{false}) \end{array}$$

which is equivalent to $\top_{\mathrm{HP}}$. □

Property 8 (Left Zero of $;$). $\top_{\mathrm{HP}} ; \mathsf{HP} = \top_{\mathrm{HP}}$ *for any hybrid process* HP.

Proof. As mentioned in the proof (see (23)) of Property 3, $\top_{\mathrm{HP}} ; \mathcal{H}_{\mathrm{HP}}(X)$ is equivalent to

$$\mathcal{H}_{\mathrm{HP}}(\mathbf{false}) ; \mathcal{H}_{\mathrm{HP}}(X) \quad = \quad \mathcal{H}_{014}\,(\natural \lhd \mathsf{inf} \rhd (\exists \mathbf{s}' \cdot (\mathbf{false} ; X \vee \mathbf{false}) \lhd \mathsf{inf}' \rhd (\mathbf{false} ; X)))$$

which can be simplified to $\mathcal{H}_{\mathrm{HP}}(\mathbf{false}) = \top_{\mathrm{HP}}$. □

Property 9 (Zero of $\|_M$). $\mathsf{HP} \|_M \top_{\mathrm{HP}} = \top_{\mathrm{HP}} \|_M \mathsf{HP} = \top_{\mathrm{HP}}$ *for any hybrid process* HP.

Proof. As mentioned in the proof (see (22)) of Property 2, $\top_{\mathrm{HP}}$ can be unfolded as

$$\mathcal{H}_{014}(\natural \lhd \mathsf{inf} \rhd \mathbf{false}) \quad = \quad \mathsf{inf} \wedge \mathcal{H}_{014}(\natural)$$

Then, $\top_{\mathrm{HP},X} = \mathsf{inf} \wedge \mathcal{H}_{014}(\natural)_X$ (note that inf has no primed variables). Therefore,

$$\begin{array}{rcl} \top_{\mathrm{HP}} \|_M \mathsf{HP} & = & \mathcal{H}_{\mathrm{HP}}\left(\top_{\mathrm{HP},X} \wedge \mathsf{HP}_Y \wedge \mathbb{I} \quad ; \quad M\right) \\ & = & \mathcal{H}_{\mathrm{HP}}\,(\mathsf{inf} \wedge (\mathcal{H}_{014}(\natural)_X \wedge \mathsf{HP}_Y \wedge \mathbb{I} \quad ; \quad M)) \\ & = & \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_4 \circ \mathcal{H}_3 \circ \mathcal{H}_2\,(\mathsf{inf} \wedge (\mathcal{H}_{014}(\natural)_X \wedge \mathsf{HP}_Y \wedge \mathbb{I} \quad ; \quad M)) \end{array}$$

where $\mathbb{I} \mathrel{\widehat{=}} ti = ti' \wedge \mathbf{s} = \mathbf{s}' \wedge tr = tr'$. Notice that $\mathcal{H}_2\,(\mathsf{inf} \wedge (\mathcal{H}_{014}(\natural)_X \wedge \mathsf{HP}_Y \wedge \mathbb{I} ; M))$ is

$$\natural \lhd \mathsf{inf} \rhd (\mathsf{inf} \wedge (\mathcal{H}_{014}(\natural)_X \wedge \mathsf{HP}_Y \wedge \mathbb{I} ; M)) \quad = \quad \natural \lhd \mathsf{inf} \rhd \mathbf{false} \quad = \quad \mathcal{H}_2(\mathbf{false})$$

which means

$$\top_{\mathrm{HP}} \|_M \mathsf{HP} \quad = \quad \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_4 \circ \mathcal{H}_3 \circ \mathcal{H}_2(\mathbf{false}) \quad = \quad \top_{\mathrm{HP}}.$$

Similarly, we can prove $\mathsf{HP} \|_M \top_{\mathrm{HP}} = \top_{\mathrm{HP}}$. □

Theorem 4.13 (Idempotence and Monotonicity). $\mathcal{H}^A_{\mathrm{HP}}$ *is idempotent and monotonic.*

Proof. First, $\mathcal{H}^{\mathrm{A}}_{\mathrm{HP}}$ is monotonic as it is constructed by monotonic operators. Then, it can be proved that $\mathcal{H}^{\mathrm{A}}_0$, $\mathcal{H}^{\mathrm{A}}_2$, $\mathcal{H}^{\mathrm{A}}_3$ and $\mathcal{H}_4$ are idempotent and commutative with each other, thus $\mathcal{H}^{\mathrm{A}}_{\mathrm{HP}} = \mathcal{H}^{\mathrm{A}}_0 \circ \mathcal{H}^{\mathrm{A}}_2 \circ \mathcal{H}^{\mathrm{A}}_3 \circ \mathcal{H}_4$ is idempotent. □

A.3 Proofs for Hybrid Designs

Theorem 5.2 (Operations).

(1) $(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \sqcap (\mathsf{HC}_1 \vdash \mathsf{HP}_1) = (\mathsf{HC}_0 \wedge \mathsf{HC}_1) \vdash (\mathsf{HP}_0 \vee \mathsf{HP}_1)$
(2) $(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \sqcup (\mathsf{HC}_1 \vdash \mathsf{HP}_1) = (\mathsf{HC}_0 \vee \mathsf{HC}_1) \vdash ((\mathsf{HC}_0 \Rightarrow_\imath \mathsf{HP}_0) \wedge (\mathsf{HC}_1 \Rightarrow_\imath \mathsf{HP}_1))$
(3) $(\mathsf{HC}_0 \vdash \mathsf{HP}_0) ; (\mathsf{HC}_1 \vdash \mathsf{HP}_1) = (\mathsf{HC}_0 \wedge \neg_\imath(\mathsf{HP}_0 ; \neg_\imath \mathsf{HC}_1)) \vdash (\mathsf{HP}_0 ; \mathsf{HP}_1)$

Proof. The matrix representations of $\mathsf{HC}_0 \vdash \mathsf{HP}_0$ and $\mathsf{HC}_1 \vdash \mathsf{HP}_1$ are respectively

$$\begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \neg_\imath \mathsf{HC}_0 & \mathsf{HC}_0 \Rightarrow_\imath \mathsf{HP}_0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \neg_\imath \mathsf{HC}_1 & \mathsf{HC}_1 \Rightarrow_\imath \mathsf{HP}_1 \end{pmatrix}$$

Their meet ($\sqcap$, $\vee$) and join ($\sqcup$, $\wedge$) can be computed component-wisely as

$$\begin{pmatrix} \bot_{HP} \vee \bot_{HP} & \bot_{HP} \vee \bot_{HP} \\ \neg_\imath HC_0 \vee \neg_\imath HC_1 & \begin{matrix}(HC_0 \Rightarrow_\imath HP_0) \vee \\ (HC_1 \Rightarrow_\imath HP_1)\end{matrix} \end{pmatrix} = \begin{pmatrix} \bot_{HP} & \bot_{HP} \\ \neg_\imath (HC_0 \wedge HC_1) & \begin{matrix}(HC_0 \wedge HC_1) \Rightarrow_\imath \\ (HP_0 \vee HP_1)\end{matrix} \end{pmatrix}$$
$$\begin{pmatrix} \bot_{HP} \wedge \bot_{HP} & \bot_{HP} \wedge \bot_{HP} \\ \neg_\imath HC_0 \wedge \neg_\imath HC_1 & \begin{matrix}(HC_0 \Rightarrow_\imath HP_0) \wedge \\ (HC_1 \Rightarrow_\imath HP_1)\end{matrix} \end{pmatrix} = \begin{pmatrix} \bot_{HP} & \bot_{HP} \\ \neg_\imath (HC_0 \vee HC_1) & \begin{matrix}(HC_0 \Rightarrow_\imath HP_0) \wedge \\ (HC_1 \Rightarrow_\imath HP_1)\end{matrix} \end{pmatrix}$$

respectively, which are respectively equivalent to

$$(HC_0 \wedge HC_1) \vdash (HP_0 \vee HP_1) \text{ and } (HC_0 \vee HC_1) \vdash ((HC_0 \Rightarrow_\imath HP_0) \wedge (HC_1 \Rightarrow_\imath HP_1))$$

according to (2). Imagine $\fatsemi$ and $\vee$ as matrix multiplication and addition, respectively, then

$$\begin{aligned}
&\begin{pmatrix} \bot_{HP} & \bot_{HP} \\ \neg_\imath HC_0 & HC_0 \Rightarrow_\imath HP_0 \end{pmatrix} \fatsemi \begin{pmatrix} \bot_{HP} & \bot_{HP} \\ \neg_\imath HC_1 & HC_1 \Rightarrow_\imath HP_1 \end{pmatrix} \\
&= \begin{pmatrix} (\bot_{HP} \fatsemi \bot_{HP}) \vee (\bot_{HP} \fatsemi \neg_\imath HC_1) & (\bot_{HP} \fatsemi \bot_{HP}) \vee (\bot_{HP} \fatsemi (HC_1 \Rightarrow_\imath HP_1)) \\ (\neg_\imath HC_0 \fatsemi \bot_{HP}) \vee ((HC_0 \Rightarrow_\imath HP_0) \fatsemi \neg_\imath HC_1) & (\neg_\imath HC_0 \fatsemi \bot_{HP}) \vee ((HC_0 \Rightarrow_\imath HP_0) \fatsemi (HC_1 \Rightarrow_\imath HP_1)) \end{pmatrix} \\
&= \begin{pmatrix} \bot_{HP} & \bot_{HP} \\ \neg_\imath HC_0 \vee (HP_0 \fatsemi \neg_\imath HC_1) & \neg_\imath HC_0 \vee (HP_0 \fatsemi \neg_\imath HC_1) \vee (HP_0 \fatsemi HP_1) \end{pmatrix} \\
&= \begin{pmatrix} \bot_{HP} & \bot_{HP} \\ \neg_\imath (HC_0 \wedge \neg_\imath (HP_0 \fatsemi \neg_\imath HC_1)) & (HC_0 \wedge \neg_\imath (HP_0 \fatsemi \neg_\imath HC_1)) \Rightarrow_\imath (HP_0 \fatsemi HP_1) \end{pmatrix}
\end{aligned}$$

which is equivalent to $(HC_0 \wedge \neg_\imath (HP_0 \fatsemi \neg_\imath HC_1)) \vdash (HP_0 \fatsemi HP_1)$ by (2). □

Theorem 5.3 (Parallel Composition).

$$(HC_0 \vdash HP_0) \|_{NHD(M)} (HC_1 \vdash HP_1) = \begin{pmatrix} \neg_\imath (\neg_\imath HC_0 \|_{\bot_{HP}} \neg_\imath HC_1) \\ \wedge \neg_\imath (\neg_\imath HC_0 \|_{\bot_{HP}} HP_1) \\ \wedge \neg_\imath (\neg_\imath HC_1 \|_{\bot_{HP}} HP_0) \end{pmatrix} \vdash HP_0 \|_M HP_1$$

Proof. According to Section 4.2.5, $(HC_0 \vdash HP_0) \|_{NHD(M)} (HC_1 \vdash HP_1)$ can be unfolded as

$$\mathcal{H}_{HP} ((HC_0 \vdash HP_0)_X \wedge (HC_1 \vdash HP_1)_Y \wedge \mathbb{I} \quad \fatsemi \quad NHD(M)) \tag{24}$$

where $\mathbb{I} \mathrel{\widehat{=}} (ti = ti' \wedge \mathbf{s} = \mathbf{s}' \wedge tr = tr')$. With the aid of the matrix representation, the predicate $(HC_0 \vdash HP_0)_X \wedge (HC_1 \vdash HP_1)_Y \wedge \mathbb{I}$ can be rewritten as

$$\begin{array}{c} \begin{matrix} \neg ok'_X \wedge \neg ok'_Y & ok'_X \wedge \neg ok'_Y & \neg ok'_X \wedge ok'_Y & ok'_X \wedge ok'_Y \end{matrix} \\ \begin{pmatrix} \bot_{HP}^X \wedge \bot_{HP}^Y \wedge \mathbb{I} & \bot_{HP}^X \wedge \bot_{HP}^Y \wedge \mathbb{I} & \bot_{HP}^X \wedge \bot_{HP}^Y \wedge \mathbb{I} & \bot_{HP}^X \wedge \bot_{HP}^Y \wedge \mathbb{I} \\ (\neg_\imath HC_0)_X \wedge (\neg_\imath HC_1)_Y \wedge \mathbb{I} & (\neg_\imath HC_0)_X \wedge \mathbb{I} \wedge (HC_1 \Rightarrow_\imath HP_1)_Y & (\neg_\imath HC_1)_Y \wedge \mathbb{I} \wedge (HC_0 \Rightarrow_\imath HP_0)_X & (HC_0 \Rightarrow_\imath HP_0)_X \wedge \mathbb{I} \wedge (HC_1 \Rightarrow_\imath HP_1)_Y \end{pmatrix} \begin{matrix} \neg ok \\ ok \end{matrix} \end{array}$$

Similarly, NHD(M) can also be represented by the transpose of the following matrix:

$$NHD(M)^\intercal = \begin{array}{c} \begin{matrix} \neg ok_X \wedge \neg ok_Y & ok_X \wedge \neg ok_Y & \neg ok_X \wedge ok_Y & ok_X \wedge ok_Y \end{matrix} \\ \begin{pmatrix} \bot_{HP} & \bot_{HP} & \bot_{HP} & \top_{HP} \\ \bot_{HP} & \bot_{HP} & \bot_{HP} & M \end{pmatrix} \begin{matrix} \neg ok' \\ ok' \end{matrix} \end{array}$$

Then, $(\mathrm{HC}_0 \vdash \mathrm{HP}_0)_X \wedge (\mathrm{HC}_1 \vdash \mathrm{HP}_1)_Y \wedge \mathbb{I} \fatsemi \mathrm{NHD}(M)$, where $\wedge$ takes precedence over $\fatsemi$, is equivalent to the following matrix, by treating $\fatsemi$ and $\vee$ as matrix multiplication and addition, respectively.

$$\begin{array}{c} \begin{array}{cc} \neg ok' & ok' \\ \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \end{array} \\ \begin{pmatrix} (\neg_\imath \mathrm{HC}_0)_X \wedge (\neg_\imath \mathrm{HC}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \vee \\ (\neg_\imath \mathrm{HC}_0)_X \wedge (\mathrm{HC}_1 \Rightarrow_\imath \mathrm{HP}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \\ \vee (\mathrm{HC}_0 \Rightarrow \mathrm{HP}_0)_X \wedge (\neg_\imath \mathrm{HC}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \end{array} & \begin{array}{c} (\neg_\imath \mathrm{HC}_0)_X \wedge (\neg_\imath \mathrm{HC}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \vee \\ (\neg_\imath \mathrm{HC}_0)_X \wedge (\mathrm{HC}_1 \Rightarrow_\imath \mathrm{HP}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \vee \\ (\mathrm{HC}_0 \Rightarrow \mathrm{HP}_0)_X \wedge (\neg_\imath \mathrm{HC}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \vee \\ (\mathrm{HC}_0 \Rightarrow \mathrm{HP}_0)_X \wedge (\neg_\imath \mathrm{HC}_1 \Rightarrow_\imath \mathrm{HP}_1)_Y \wedge \mathbb{I} \fatsemi M \end{array} \end{array} \begin{array}{c} \neg ok \\ ok \end{array}$$

This matrix can be simplified as follows:

$$\begin{array}{c} \begin{array}{cc} \neg ok' & ok' \\ \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \end{array} \\ \begin{pmatrix} (\neg_\imath \mathrm{HC}_0)_X \wedge (\neg_\imath \mathrm{HC}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \\ \vee (\neg_\imath \mathrm{HC}_0)_X \wedge (\mathrm{HP}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \\ \vee (\mathrm{HP}_0)_X \wedge (\neg_\imath \mathrm{HC}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \end{array} & \begin{array}{c} (\neg_\imath \mathrm{HC}_0)_X \wedge (\neg_\imath \mathrm{HC}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \\ \vee (\neg_\imath \mathrm{HC}_0)_X \wedge (\mathrm{HP}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \\ \vee (\mathrm{HP}_0)_X \wedge (\neg_\imath \mathrm{HC}_1)_Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \\ \vee (\mathrm{HP}_0)_X \wedge (\mathrm{HP}_1)_Y \wedge \mathbb{I} \fatsemi M \end{array} \end{array} \begin{array}{c} \neg ok \\ ok \end{array}$$

Therefore, the hybrid process of (24) is equivalent to the following matrix:

$$\begin{array}{c} \begin{array}{cc} \neg ok' & ok' \\ \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \end{array} \\ \begin{pmatrix} (\neg_\imath \mathrm{HC}_0 \|_{\bot_{\mathrm{HP}}} \mathrm{HP}_1) \vee (\mathrm{HP}_0 \|_{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1) \\ \vee (\neg_\imath \mathrm{HC}_0 \|_{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1) \end{array} & \begin{array}{c} (\neg_\imath \mathrm{HC}_0 \|_{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1) \vee (\neg_\imath \mathrm{HC}_0 \|_{\bot_{\mathrm{HP}}} \mathrm{HP}_1) \\ \vee (\mathrm{HP}_0 \|_{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1) \vee (\mathrm{HP}_0 \|_M \mathrm{HP}_1) \end{array} \end{array} \begin{array}{c} \neg ok \\ ok \end{array}$$

which is equivalent to

$$\neg_\imath \begin{pmatrix} (\neg_\imath \mathrm{HC}_0 \|_{\bot_{\mathrm{HP}}} \mathrm{HP}_1) \vee (\mathrm{HP}_0 \|_{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1) \\ \vee (\neg_\imath \mathrm{HC}_0 \|_{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1) \end{pmatrix} \vdash (\mathrm{HP}_0 \|_M \mathrm{HP}_1)$$

by the matrix representation of hybrid designs (see (2)). □

Lemma A.2. $\mathcal{H}_{\mathrm{HP}}(X) \fatsemi \bot_{\mathrm{HP}} = \mathcal{H}_{\mathrm{HP}}(X \fatsemi \bot_{\mathrm{HP}})$.

Proof. As mentioned in the proof (see (23)) of Property 3, $\mathcal{H}_{\mathrm{HP}}(X) \fatsemi \bot_{\mathrm{HP}}$ is equivalent to

$$\mathcal{H}_{\mathrm{HP}}(X) \fatsemi \mathcal{H}_{\mathrm{HP}}(\bot_{\mathrm{HP}}) \quad = \quad \mathcal{H}_{014}\,(\natural \lhd \mathsf{inf} \rhd (\exists \mathsf{s}' \cdot (X \fatsemi \bot_{\mathrm{HP}} \vee X) \lhd \mathsf{inf}' \rhd X \fatsemi \bot_{\mathrm{HP}}))$$

which can be simplified to $\mathcal{H}_{\mathrm{HP}}(X \fatsemi \bot_{\mathrm{HP}})$ according to (22). □

Theorem 5.5 (Closure). *Normal hybrid designs are closed on* $\sqcap$, $\sqcup$, $\fatsemi$, $\|_{\mathrm{NHD}(M)}$ *and* $\|_M^{\mathrm{HP}}$.

Proof. We only prove the closure of normal hybrid designs on $\|_{\mathrm{NHD}(M)}$. The key is to prove

$$\mathsf{Assumption} \quad \widehat{=} \quad \neg_\imath(\neg_\imath \mathrm{HC}_0 \|_{\bot_{\mathrm{HP}}} \mathrm{HP}_1) \wedge \neg_\imath(\mathrm{HP}_0 \|_{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1) \wedge \neg_\imath(\neg_\imath \mathrm{HC}_0 \|_{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1)$$

is $\mathcal{H}_{\mathrm{HC}}$-healthy, i.e., $\neg_\imath \mathsf{Assumption} \fatsemi \bot_{\mathrm{HP}} = \neg_\imath \mathsf{Assumption}$. Concretely,

$$\neg_\imath \mathsf{Assumption} \fatsemi \bot_{\mathrm{HP}} \quad = \quad \big((\neg_\imath \mathrm{HC}_0 \|_{\bot_{\mathrm{HP}}} \mathrm{HP}_1) \vee (\mathrm{HP}_0 \|_{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1) \vee (\neg_\imath \mathrm{HC}_0 \|_{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1)\big) \fatsemi \bot_{\mathrm{HP}}$$

By Lemma A.2, $(\neg_\imath \mathrm{HC}_0 \| _{\bot_{\mathrm{HP}}} \mathrm{HP}_1) \fatsemi \bot_{\mathrm{HP}}$ is equivalent to

$$\begin{array}{rcl} \mathcal{H}_{\mathrm{HP}}(\neg_\imath \mathrm{HC}_0^X \wedge \mathrm{HP}_1^Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}}) \fatsemi \bot_{\mathrm{HP}} & = & \mathcal{H}_{\mathrm{HP}}(\neg_\imath \mathrm{HC}_0^X \wedge \mathrm{HP}_1^Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}} \fatsemi \bot_{\mathrm{HP}}) \\ & = & \mathcal{H}_{\mathrm{HP}}(\neg_\imath \mathrm{HC}_0^X \wedge \mathrm{HP}_1^Y \wedge \mathbb{I} \fatsemi \bot_{\mathrm{HP}}) \\ & = & \neg_\imath \mathrm{HC}_0 \| _{\bot_{\mathrm{HP}}} \mathrm{HP}_1 \end{array}$$

Similarly, we can prove that $(\mathrm{HP}_0 \| _{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1) \fatsemi \bot_{\mathrm{HP}}$ and $(\neg_\imath \mathrm{HC}_0 \| _{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1) \fatsemi \bot_{\mathrm{HP}}$ are equivalent to $\mathrm{HP}_0 \| _{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1$ and $\neg_\imath \mathrm{HC}_0 \| _{\bot_{\mathrm{HP}}} \neg_\imath \mathrm{HC}_1$, respectively. Thus, $\neg_\imath \mathrm{Assumption} \fatsemi \bot_{\mathrm{HP}} = \neg_\imath \mathrm{Assumption}$, i.e., Assumption is $\mathcal{H}_{\mathrm{HC}}$-healthy. □

Lemma A.3. $\mathrm{HP}_0 \sqsubseteq \mathrm{HP}_1$ *iff* $\mathrm{HP}_0 \sqcap \neg_\imath \mathrm{HP}_1 = \bot_{\mathrm{HP}}$ *for* HP_0 *and* HP_1 *are hybrid processes.*

Proof. $\mathrm{HP}_0 \sqsubseteq \mathrm{HP}_1$ iff $\neg_\imath \mathrm{HP}_1 \sqsubseteq \neg_\imath \mathrm{HP}_0$ iff

$$\bot_{\mathrm{HP}} \quad \sqsubseteq \quad (\mathrm{HP}_0 \sqcap \neg_\imath \mathrm{HP}_1) \quad \sqsubseteq \quad (\mathrm{HP}_0 \sqcap \neg_\imath \mathrm{HP}_0) \quad = \quad \bot_{\mathrm{HP}}$$

according to the monotonicity of $\sqcap$ (Property 7). □

Theorem 5.6 (Refinement). $(\mathrm{HC}_0 \vdash \mathrm{HP}_0) \sqsubseteq (\mathrm{HC}_1 \vdash \mathrm{HP}_1)$ *iff*

$$\mathrm{HC}_1 \sqsubseteq \mathrm{HC}_0 \quad \textit{and} \quad \mathrm{HP}_0 \sqsubseteq (\mathrm{HC}_0 \wedge \mathrm{HP}_1)$$

Proof. According to the matrix representation, $(\mathrm{HC}_0 \vdash \mathrm{HP}_0) \sqsubseteq (\mathrm{HC}_1 \vdash \mathrm{HP}_1)$ iff

$$\begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \neg_\imath \mathrm{HC}_0 & \mathrm{HC}_0 \Rightarrow_\imath \mathrm{HP}_0 \end{pmatrix} \quad \sqsubseteq \quad \begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \neg_\imath \mathrm{HC}_1 & \mathrm{HC}_1 \Rightarrow_\imath \mathrm{HP}_1 \end{pmatrix}$$

iff $\neg_\imath \mathrm{HC}_0 \sqsubseteq \neg_\imath \mathrm{HC}_1$ and $(\mathrm{HC}_0 \Rightarrow_\imath \mathrm{HP}_0) \sqsubseteq (\mathrm{HC}_1 \Rightarrow_\imath \mathrm{HP}_1)$. Then,

$$\begin{array}{rcll} \bot_{\mathrm{HP}} & = & (\mathrm{HC}_0 \Rightarrow_\imath \mathrm{HP}_0) \sqcap \neg_\imath (\mathrm{HC}_1 \Rightarrow_\imath \mathrm{HP}_1) & \text{[Lemma A.3]} \\ & = & \neg_\imath \mathrm{HC}_0 \sqcap \mathrm{HP}_0 \sqcap (\mathrm{HC}_1 \sqcup \neg_\imath \mathrm{HP}_1) & \\ & = & \neg_\imath \mathrm{HC}_0 \sqcap \mathrm{HP}_0 \sqcap \neg_\imath \mathrm{HP}_1 & [\neg_\imath \mathrm{HC}_0 \sqsubseteq \neg_\imath \mathrm{HC}_1 \text{ and Lemma A.3}] \end{array}$$

Applying Lemma A.3 again, the above result is equivalent to

$$\mathrm{HP}_0 \quad \sqsubseteq \quad \neg_\imath (\neg_\imath \mathrm{HC}_0 \sqcap \neg_\imath \mathrm{HP}_1) \quad \equiv \quad (\mathrm{HC}_0 \wedge \mathrm{HP}_1)$$

□

Property 10 (Contra-variance). *If* $\mathrm{HC}_0 \sqsupseteq \mathrm{HC}_1$ *and* $\mathrm{HP}_0 \sqsubseteq \mathrm{HP}_1$ *then*

$$(\mathrm{HC}_0 \vdash \mathrm{HP}_0) \sqsubseteq (\mathrm{HC}_1 \vdash \mathrm{HP}_1)$$

Proof. It can be proved directly from Theorem 5.6. □

Property 11 (Monotonicity). *Operators* $\sqcup, \sqcap, \fatsemi, \|_{\mathrm{NHD}(M)}$ *and* $\|_M^{\mathrm{HP}}$ *for normal hybrid designs are monotonic with respect to* $\sqsubseteq$.

Proof. These operators are constructed from monotonic atomic operators. □

Theorem 5.7 (Complete Lattice). *Normal hybrid designs from a complete lattice with top*

$$\top_{\mathrm{NHD}} \quad \widehat{=} \quad \bot_{\mathrm{HP}} \vdash \top_{\mathrm{HP}}$$

and bottom

$$\bot_{\mathrm{NHD}} \quad \widehat{=} \quad \top_{\mathrm{HP}} \vdash \bot_{\mathrm{HP}}$$

Proof. According to Theorem 5.2, $\sqcap$ and $\sqcup$ can be generalised to

$$\begin{array}{lcl} \bigsqcap_i(\mathsf{HC}_i \vdash \mathsf{HP}_i) & = & (\bigwedge_i \mathsf{HC}_i) \vdash (\bigvee_i \mathsf{HP}_i) \\ \bigsqcup_i(\mathsf{HC}_i \vdash \mathsf{HP}_i) & = & (\bigvee_i \mathsf{HC}_i) \vdash (\bigwedge_i(\mathsf{HC}_i \Rrightarrow_\wr \mathsf{HP}_i)) \end{array}$$

Therefore, normal hybrid designs form a complete lattice under $\sqsubseteq$ with

$$\begin{array}{lcl} \top_{\mathrm{NHD}} & \widehat{=} & \bot_{\mathrm{HP}} \vdash \top_{\mathrm{HP}} \\ \bot_{\mathrm{NHD}} & \widehat{=} & \top_{\mathrm{HP}} \vdash \bot_{\mathrm{HP}} \end{array}$$

□

Property 12 (Chaos). $\bot_{\mathrm{NHD}} = \top_{\mathrm{HP}} \vdash \mathsf{HP}$ *for any hybrid process* HP.

Proof. By the matrix representation,

$$\top_{\mathrm{HP}} \vdash \mathsf{HP} \quad = \quad \begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \neg_\wr \top_{\mathrm{HP}} & \top_{\mathrm{HP}} \Rrightarrow_\wr \mathsf{HP} \end{pmatrix} \quad = \quad \begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \end{pmatrix} \quad = \quad \bot_{\mathrm{NHD}}$$

□

Property 13 (Non-termination). *For any normal hybrid design* NHD,

$$\bot_{\mathrm{NHD}} \fatsemi \mathsf{NHD} = \bot_{\mathrm{NHD}} \quad \textit{and} \quad \top_{\mathrm{NHD}} \fatsemi \mathsf{NHD} = \top_{\mathrm{NHD}}$$

Proof. Let $\mathsf{NHD} \mathrel{\widehat{=}} \mathsf{HC} \vdash \mathsf{HP}$. By matrix representations,

$$\begin{array}{lcl} \bot_{\mathrm{NHD}} \fatsemi \mathsf{NHD} & = & \begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \end{pmatrix} \fatsemi \begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \neg_\wr \mathsf{HC} & \mathsf{HC} \Rrightarrow_\wr \mathsf{HP} \end{pmatrix} = \begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \end{pmatrix} = \bot_{\mathrm{NHD}} \\ \top_{\mathrm{NHD}} \fatsemi \mathsf{NHD} & = & \begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \top_{\mathrm{HP}} & \top_{\mathrm{HP}} \end{pmatrix} \fatsemi \begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \neg_\wr \mathsf{HC} & \mathsf{HC} \Rrightarrow_\wr \mathsf{HP} \end{pmatrix} = \begin{pmatrix} \bot_{\mathrm{HP}} & \bot_{\mathrm{HP}} \\ \top_{\mathrm{HP}} & \top_{\mathrm{HP}} \end{pmatrix} = \top_{\mathrm{NHD}} \end{array}$$

□

Property 14 (Parallel Composition). *For any normal hybrid design* NHD,

$$\mathsf{NHD} \|_{\mathrm{NHD}(M)} \top_{\mathrm{NHD}} \quad = \quad \top_{\mathrm{NHD}} \|_{\mathrm{NHD}(M)} \mathsf{NHD} \quad = \quad \top_{\mathrm{NHD}}$$

Proof. It can be proved by Property 9 and Theorem 5.3. □

Property 15 (Parallel Composition). *For any normal hybrid design* NHD,

$$\mathsf{NHD} \|^{\mathrm{HP}}_M \bot_{\mathrm{NHD}} \quad = \quad \bot_{\mathrm{NHD}} \|^{\mathrm{HP}}_M \mathsf{NHD} \quad = \quad \bot_{\mathrm{NHD}}$$

Proof. Let $\mathsf{NHD} \mathrel{\widehat{=}} \mathsf{HC} \vdash \mathsf{HP}$, then by Definition 5.4 and Property 12 we can get

$$\bot_{\mathrm{NHD}} \|_M \mathsf{NHD} \quad = \quad (\top_{\mathrm{HP}} \vdash \bot_{\mathrm{HP}}) \|_M (\mathsf{HC} \vdash \mathsf{HP}) \quad = \quad \top_{\mathrm{HP}} \vdash (\bot_{\mathrm{HP}} \|_M \mathsf{HP}) \quad = \quad \bot_{\mathrm{NHD}}$$

Similarly, we can prove $\mathsf{NHD} \|_M \bot_{\mathrm{NHD}} = \bot_{\mathrm{NHD}}$. □

A.4 Proofs for Reflection of HCSP and Simulink with HUTP

Property 16 (Unit). $\lceil\rfloor$ *is a unit of hybrid processes w.r.t.* $;$.

Proof. According to the intermediate result (23) from Property 3,

$$\begin{array}{rcl}\mathcal{H}_{\mathrm{HP}}(X) ; \lceil\rfloor & = & \mathcal{H}_{\mathrm{HP}}(X) ; \mathcal{H}_{\mathrm{HP}}(ti = ti' < +\infty \wedge \mathbf{s} = \mathbf{s}' \wedge tr = tr') \\ & = & \mathcal{H}_{014}(\natural \lhd \mathsf{inf} \rhd ((\exists \mathbf{s}' \cdot X) \lhd \mathsf{inf}' \rhd X)) \quad = \quad \mathcal{H}_{\mathrm{HP}}(X)\end{array}$$

by the intermediate result (22) from Property 2. Similarly, we can prove $\lceil\rfloor ; \mathcal{H}_{\mathrm{HP}}(X) = \mathcal{H}_{\mathrm{HP}}(X)$. □

Corollary 1 (Unit). $\vdash \lceil\rfloor$ *is a unit of hybrid processes w.r.t.* $;$.

Proof. According to Property 16 and Theorem 5.2(3), for any normal hybrid design $\mathrm{HC} \vdash \mathrm{HP}$,

$$\begin{array}{rcl}(\vdash \lceil\rfloor) ; (\mathrm{HC} \vdash \mathrm{HP}) & = & (\bot_{\mathrm{HP}} \wedge \neg_{\imath}(\lceil\rfloor ; \neg_{\imath}\mathrm{HC})) \vdash (\lceil\rfloor ; \mathrm{HP}) \quad = \quad \mathrm{HC} \vdash \mathrm{HP} \\ (\mathrm{HC} \vdash \mathrm{HP}) ; (\vdash \lceil\rfloor) & = & (\mathrm{HC} \wedge \neg_{\imath}(\mathrm{HP} ; \neg_{\imath}\bot_{\mathrm{HP}})) \vdash (\mathrm{HP} ; \lceil\rfloor) \quad = \quad \mathrm{HC} \vdash \mathrm{HP}\end{array}$$

□

Theorem 6.1 (Semantics Consistency). *The HUTP semantics of HCSP is consistent with the operational semantics of HCSP.*

Proof. We say the HUTP semantics and the structural operational semantics (SOS) of an HCSP process are consistent iff they keep consistent on time, state and trace. The proof for the consistency is divided into two stages: first, we prove the consistency between the SOS and the HUTP semantics of the syntactic entities in (3); then, we prove the semantics consistency for the operators, such as sequential and parallel compositions, which connect the syntactic entities. For brevity, we just show the key points of the proof in the following content.

The semantics consistency for ODE with communication interruption ($\mathrm{ODE}_{\trianglerighteq}$) has been proved in the end of Section 6.2. $\mathrm{ODE}_{\trianglerighteq}$ is representative because it covers the characteristic features of HCSP, such as continuous evolution, communication and interruption. The semantics consistency for other syntactic entities of HCSP like $\langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B\rangle$ and $ch?x$ can be proved similarly. So, we only focus on proving the semantics consistency for the connector of parallel composition as it plays an important role in composite systems.

The SOS for parallel composition of HCSP processes is described by the following rules, where I is the common channel set between parallel operands.

$$\frac{}{(\varepsilon \| \varepsilon, (ti, \mathbf{s}_P \uplus \mathbf{s}_Q)) \longrightarrow (\varepsilon, (ti, \mathbf{s}_P \uplus \mathbf{s}_Q))}[\mathrm{Par0}] \quad \frac{(\mathrm{P}, (ti, \mathbf{s}_P)) \xrightarrow{\tau} (\mathrm{P}', (ti, \mathbf{s}'_P))}{(\mathrm{P} \| \mathrm{Q}, (ti, \mathbf{s}_P \uplus \mathbf{s}_Q)) \xrightarrow{\tau} (\mathrm{P}' \| \mathrm{Q}, (ti, \mathbf{s}'_P \uplus \mathbf{s}_Q))}[\mathrm{Par1}]$$

$$\frac{ch \in I \quad (\mathrm{P}, \mathbf{s}_P) \xrightarrow{\langle ch?, d\rangle} (\mathrm{P}', \mathbf{s}'_P) \quad (\mathrm{Q}, \mathbf{s}_Q) \xrightarrow{\langle ch!, d\rangle} (\mathrm{Q}', \mathbf{s}'_Q)}{(\mathrm{P} \| \mathrm{Q}, \mathbf{s}_P \uplus \mathbf{s}_Q) \xrightarrow{\langle ch, d\rangle} (\mathrm{P}' \| \mathrm{Q}', \mathbf{s}'_P \uplus \mathbf{s}'_Q)}[\mathrm{Par2}] \quad \frac{ch \notin I \quad (\mathrm{P}, (ti, \mathbf{s}_P)) \xrightarrow{\langle ch*, d\rangle} (\mathrm{P}', (ti, \mathbf{s}'_P))}{(\mathrm{P} \| \mathrm{Q}, \mathbf{s}_P \uplus \mathbf{s}_Q) \xrightarrow{\langle ch*, d\rangle} (\mathrm{P}' \| \mathrm{Q}, \mathbf{s}'_P \uplus \mathbf{s}_Q)}[\mathrm{Par3}]$$

$$\frac{(\mathrm{P}, (ti, \mathbf{s}_P)) \xrightarrow{\langle ti'-ti, RS_P\rangle} (\mathrm{P}', (ti', \mathbf{s}'_P), \underset{\sim}{\mathbf{s}}_P) \quad (\mathrm{Q}, (ti, \mathbf{s}_Q)) \xrightarrow{\langle ti'-ti, RS_Q\rangle} (\mathrm{Q}', (ti', \mathbf{s}'_Q), \underset{\sim}{\mathbf{s}}_Q) \quad \overline{RS_P} \cap RS_Q = \emptyset}{(\mathrm{P} \| \mathrm{Q}, (ti, \mathbf{s}_P \uplus \mathbf{s}_Q)) \xrightarrow{\langle ti'-ti, RS_P \uplus RS_Q\rangle} (\mathrm{P}' \| \mathrm{Q}', (ti', \mathbf{s}'_P \uplus \mathbf{s}'_Q), \underset{\sim}{\mathbf{s}}_P \uplus \underset{\sim}{\mathbf{s}}_Q)}[\mathrm{Par4}]$$

Since parallel HCSP processes do not share variables, the HUTP semantics and the SOS of the parallel connector are consistent on state variables naturally. Then, we focus on the parallel

composition of timed traces, i.e., prove the consistency on timed traces. The above five rules correspond to the respective five composition rules of timed traces in Section 3.3. Concretely, [Par0], [Par1], [Par2], [Par3] and [Par4] relate to [Empty], [τ-Act], [SynIO], [NoSynIO] and [SynWait], respectively.

Consider two HCSP processes P and Q with respective states $\mathbf{s}_P$ and $\mathbf{s}_Q$. Their execution histories are recorded by timed traces $\mathtt{tt}_P$ and $\mathtt{tt}_Q$, respectively, denoted by

$$(\mathrm{P}, (ti, \mathbf{s}_P)) \xrightarrow[*]{\mathtt{tt}_P} (\varepsilon, (ti', \mathbf{s}'_P), \underset{\sim}{\mathbf{s}}_P) \quad \text{and} \quad (\mathrm{Q}, (ti, \mathbf{s}_Q)) \xrightarrow[*]{\mathtt{tt}_Q} (\varepsilon, (ti', \mathbf{s}'_Q), \underset{\sim}{\mathbf{s}}_Q)$$

where $\underset{\sim}{\mathbf{s}}_P$ and $\underset{\sim}{\mathbf{s}}_Q$ can be removed if there is no flow generated. The proof is by induction because all the traces involved are finite in the semantics of HCSP. So, this proceeds by induction on the length of traces and start from cases that $|\mathtt{tt}_P|, |\mathtt{tt}_Q| \leq 1$, such as $\mathtt{tt}_P = \mathtt{tt}_Q = \epsilon$ or $\mathtt{tt}_P$ and $\mathtt{tt}_Q$ are just trace blocks. Then, by induction on the derivation of $\mathtt{tt}_P \|_I \mathtt{tt}_Q \rightsquigarrow \mathtt{tt}$, we can prove that the parallel composition $\|_I$ of timed traces in HUTP can be interpreted by the structural operational semantics of the parallel composition of HCSP. On the other side, we can prove that the SOS of the parallel composition of HCSP can also be interpreted by $\|_I$ of timed traces in HUTP by induction on the derivation of

$$(\mathrm{P} \| \mathrm{Q}, (ti, \mathbf{s}_P \uplus \mathbf{s}_Q)) \xrightarrow[*]{\mathtt{tt}} (\varepsilon, (ti', \mathbf{s}'_P \uplus \mathbf{s}'_Q))$$

from the cases that $|\mathtt{tt}| \leq 1$. In summary, the SOS and the HUTP semantics of the parallel composition of HCSP are consistent on time, state and trace. The semantics consistency for other operators can be proved similarly. □