A Rewriting Framework for Cyber-Physical Systems

Benjamin Lion\textsuperscript{a}, Farhad Arbab\textsuperscript{b}, Carolyn Talcott\textsuperscript{c}

\textsuperscript{a}Leiden University, Leiden, The Netherlands
\textsuperscript{b}CWI, Amsterdam, The Netherlands
\textsuperscript{c}SRI International, CA, USA

Abstract
The analysis of cyber-physical systems (CPS) is challenging due to the large state space and the continuous changes occurring in its parts. Design practices favor modularity to help reducing the complexity. In a previous work, we proposed a discrete semantic model for CPS that captures both cyber and physical aspects as streams of discrete observations, which ultimately form the behavior of a component. This semantic model is denotational and compositional, where each composition operator algebraically models the interaction between a pair of components.

In this paper, we propose a specification of some components as rewrite systems. The specification is operational and executable, and we study conditions for its semantics as components to be compositional. We demonstrate our framework on modeling a coordination of robots moving on a shared field. We show that the system of robots can be coordinated by a protocol in order to exhibit emerging behavior. We use an implementation of our framework in Maude to give some practical results.

1. Introduction
Cyber-physical systems are inherently concurrent. From a cyber point of view, the timing at which a decision is taken to sense or act on its physical environment impacts the resulting outcome. Moreover, several cyber entities may share the same physical environment, leading to some race conditions. From a physical point of view, the ordering of events is not always possible, as some events may be independent. Moreover, two observers of the same physical phenomenon may order events differently. For all those reasons, modular designs help in reducing complexity of cyber-physical systems by forcing a designer to specify each component separately, and then specify how components interconnect.

We list some additional benefits inherent to modular design. First, concurrency is explicit as an exogenous operation acting on components, which gives an opportunity to reason about concurrency primitives directly. Then, the representation of system stays small. Often, a modular design offers the choice between composing each component statically to analyse the resulting system,
or dynamically at runtime to keep the state space small while simulating a run. Finally, a component comes with a notion of an interface, that specifies what is visible and what is hidden from other components. This way, both discrete and continuous aspects of components have the same type of interface, namely interfaces containing the set of possible observations over time.

In [10] we present a component model that captures timed-event sequences (TESs) as instances of their behavior. An observation is a set of events with a unique time stamp. A component has an interface that defines which events are observable, and a behavior that denotes all possible sequences of its observations (i.e., a set of TESs). Our component model is equipped with a family of operators, each parametrized with an interaction signature. Thus, cyber-physical systems are defined modularly, where each product of two components models the interaction occurring between the two components. The strength, but also its practical limitation, of our semantic model is its abstraction: there is no fixed machine specification that generates the behavior of a component. We give in this paper an operational description of some components as rewrite systems.

Rewriting logic is a powerful framework to model concurrent systems [16, 11]. Moreover, current research has made the framework both executable and analysable (e.g., using a programming language such as Maude [3]). The framework is suitable for specifying cyber-physical systems, as the underlying equational theory could represent both discrete or continuous changes. We give a semantics for rewriting systems as components, and conjecture its compositionality under some assumptions.

Finally, we apply our work on an example that considers two energy sensitive robots moving on a shared field. The two robots aim at reaching the other robot’s initial position which, by symmetry, eventually leads to a crossing situation. The crossing of the two robots is the source of a livelock behavior which can lead to starving (i.e., no energy left in the battery). We show that, in the case where an exogenous coordination is imposed by a protocol, the two robots coordinate their moves to avoid the livelock situation. We demonstrate the result using our implementation in Maude.

We present the following contributions:

• a specification of components as rewrite systems;
• conjecture some conditions for the rewrite system’s semantics to be compositional;
• an incremental/runtime implementation of composition that avoids exponential blowup in the specification size;
• illustration of how a composed Maude specification can be used to incrementally analyze a system design using a case study involving the behavior of two coordinated robot agents roaming on a field.
2. Cyber-physical system: an example

This section illustrates our approach on an intuitive and simple cyber-physical system consisting of two robots roaming on a shared field. A robot exhibits some cyber aspects, as it takes discrete decisions based on its readings. Every robot interacts, as well, with a shared physical resource as it moves around. The field models the continuous response of each action (e.g., read or move) performed by a robot. A question that will motivate the paper is: given a strategy for both robots (i.e., sequence of moves based on their readings), will both robots, sharing the same physical resource, achieve their goal? If not, can the two robots, without changing their policy, be externally coordinated towards their goal?

In this paper, we specify components in a rewriting framework in order to simulate and analyze their behavior. In this framework, an agent, e.g., a robot or a field, specifies a component as a rewriting theory. A system is a multiset of agents that run concurrently. The equational theory of an agent defines how the agent states are updated, and may exhibit both continuous or discrete transformations. The dynamic is captured by rewriting rules and an equational theory at the system level that describes how agents interact. In our example, for instance, each move of a robot is synchronous with an effect on the field. Each agent therefore specifies how the action affects its state, and the system specifies which composite actions, set of simultaneous actions, may occur. We give hereafter an intuitive example that abstracts from the underlying algebra of each agent.

Agent. A robot and a field are two examples of an agent that specifies a component as a rewriting theory. The dynamic of both agents is captured by a rewrite rule of the form:

\[(s, \emptyset) \Rightarrow (s', O)\]

where \(s\) and \(s'\) are state terms, and \(O\) is a set of alternative actions that the field or the robot proposes as alternatives. Given an action \(o \in O\) from the set of possibilities, a function \(\phi\) updates the state \(s'\) and returns a new state \(\phi(s', o)\). The equational theory that specifies \(\phi\) may capture both discrete and continuous changes. The robot and the field run concurrently in a system, where their actions may interact.

System. A system is a multiset of agents together with a composability constraint \(\kappa\) that restrict their updates. For instance, take a system that consists of a robot \(id\) and a field \(F\). The concurrent execution of the two agents is given by the following system rewrite rule:

\[
\{(s_{id}, O_{id}), (s_{F}, O_{F})\} \Rightarrow S \{ (\phi_{id}(s_{id}, o_{id}), \emptyset), (\phi_{F}(s_{F}, o_{F}), \emptyset) \}
\]

where \(o_{id} \in O_{id}\) and \(o_{F} \in O_{F}\) are two actions related by \(\kappa\).

Each agent is unaware of the other agent decisions. The runtime links each agent and filters choices that do not comply to the composability relation \(\kappa\). As a result, each agent updates its state with the (possibly composite) action chosen
at runtime, from the list of submitted actions. The framework therefore clearly separates the place where agent’s and system’s choices are handled, which is a source of runtime analysis.

Already, at this stage, we can ask the following query on the system: will robot id eventually reach the location \((x, y)\) on the field? Note that the agent alone cannot decide the query, as it depends on the characteristics of the field.

Coordination. Consider now a system with three agents: two robots and a field. Each robot has its own objective (i.e., location to reach) and strategy (i.e., sequence of moves). Since both robots share the same physical field, some exclusion principals apply, e.g., no two robots can be at the same location. It is therefore possible that the system deadlocks if no actions are composable, or livelocks if the robots enter a sequence of repeated moves.

We add a protocol agent to the system, which imposes some coordination patterns on the actions performed by robots \(id_1\) and \(id_2\). Typically, a protocol coordinates robots by forcing them to do some specific actions. As a result, given a system configuration \(\{(s_{id_1}, O_{id_1}), (s_{id_2}, O_{id_2}), (s_F, O_F), (s_P, O_P)\}\) the run of robots \(id_1\) and \(id_2\) have to agree with the observations of the protocol, and the sequence of actions for each robot will therefore be conform to the sequence imposed by the protocol.

In the case where the two robots enter a livelock and eventually run out of energy, we show in Section 5 the possibility of using a protocol to remove such behavior.

3. Semantic model: algebra of components

The design of complex systems becomes simpler if such systems can be decomposed into smaller sub-systems that interact with each other. In order to simplify the design of cyber-physical systems, we introduced in [10] a semantic model that abstracts from the internal details of both cyber and physical processes. As first class entities in this model, a component encapsulates a behavior (set of TESs) and an interface (set of events). We recall basic definitions and properties in this section.

3.1. Components

Preliminaries. A timed-event stream, TES, \(\sigma\) over a set of events \(E\) is an infinite sequence of observations, where its \(i^{th}\) observation \(\sigma(i) = (O, t)\), \(i \in \mathbb{N}\), consists of a pair of a subset of events in \(O \subseteq E\), called the observable, and a positive real number \(t \in \mathbb{R}_+\) as time stamp. A timed-event stream (TES) has the additional properties that its consecutive time stamps are monotonically increasing and non-Zeno, i.e., if \(\sigma(i) = (O_i, t_i)\) is the \(i^{th}\) element of TES \(\sigma\), then (1) \(t_i < t_{i+1}\), and (2) for any time \(t \in \mathbb{R}_+\), there exists an element \(\sigma(i) = (O_i, t_i)\) in \(\sigma\) such that \(t < t_i\). We use \(\sigma^{(k)}\) to denote the \(k\)-th derivative of the stream \(\sigma\), such that \(\sigma^{(k)}(i) = \sigma(i + k)\) for all \(i \in \mathbb{N}\). We refer to the stream of observables of \(\sigma\) as its first projection \(pr_1(\sigma) \in \mathcal{P}(E)^\omega\), and the stream of time stamps as its
second projection \( \text{pr}_2(\sigma) \in \mathbb{R}^+ \), such that \( \sigma(i) = (\text{pr}_1(\sigma)(i), \text{pr}_2(\sigma)(i)) \) for all \( i \in \mathbb{N} \). We write \((O, t) \in \sigma\) if there exists \( i \in \mathbb{N} \) such that \( \sigma(i) = (O, t) \). Given a set \( L \) of TESs, we use \( FG(L) \) to denote the set of left factor of \( L \), i.e., the set \( FG(L) = \{ s \mid \exists \sigma. \exists n. \sigma \in L \land s = \sigma[n] \} \).

A component denotes \textit{what} observables are possible, over time, given a fixed set of events. We give three examples of components, which capture some cyber-physical aspects of concurrent systems.

\textbf{Definition 1 (Component).} A \textit{component} \( C = (E, L) \) is a pair of a set of events \( E \), called its \textit{interface}, and a \textit{behavior} \( L \subseteq \text{TES}(E) \).

Given component \( A = (E_A, L_A) \), we write \( \sigma : A \) for a TES \( \sigma \in L_A \).

\textbf{Example 1.} Most of cyber systems, such as embedded systems, run in sync with their clock. The clock therefore dictates how fast a sensor can be read, or how many decisions can be taken in a time interval. The component version of such embedded system has, as timing constraint, that each observable is timestamped according to the clock frequency. As a result, each TES in its behavior has time stamps that are multiple of the period \( T \).

\textbf{Example 2.} Physical parts of systems, such as batteries, heaters, fans, etc. provide a service that must satisfy some qualities. A battery delivers an amount of power that drives, for instance, the wheels of a robot; a heater radiates some energy to warm a room; a fan spins in order to cool down a hot piece. In different contexts, physical parts may behave differently. Capturing some use cases formally help in assessing the quality and the adequacy of a particular system.

For instance, a battery as a component would capture all \textit{significant} (if not \textit{all} possible) sequences of charge/discharge and value readings. While only one of such sequence will be observable for each usage, the analysis of all its behavior is crucial to assess its quality.

\textbf{3.2. Product and division}

Components describe \textit{what} observations occur over time. When run concurrently, observable events from a component may somehow relate to observable events of another component. This relation defines what kind of interaction occurs between the two components, as it may enforce two events to occur within the same observable at the same time (e.g., actuation of a wheel and changes of location of the robot), or it may prevent two events to occur simultaneously (e.g., two robots moving to the same physical location). Interaction constraints are therefore captured in an algebraic operator that acts on components. The result of forming the product of two components is a new component, whose behavior contains the composition of every pair of TESs, one from each product operand, that satisfy the underlying constraints imposed by that specific operator.
Let \( A = (E_A, L_A) \) and \( B = (E_B, L_B) \) be two components. We use the relation \( R(E_A, E_B) \subseteq TES(E_A) \times TES(E_B) \) and the function \( \oplus : TES(E) \times TES(E) \rightarrow TES(E) \) to range over composability relations and composition functions, respectively. We use \( \Sigma \) to range over interaction signatures, i.e., pairs of a composability relation and a composition function.

**Definition 2 (Product).** The product of \( A \) and \( B \) under \( \Sigma = (R, \oplus) \) is the component \( C = A \times \Sigma B = (E_A \cup E_B, L) \) where

\[
L = \{ \sigma \oplus \tau \mid \sigma \in L_A, \tau \in L_B, (\sigma, \tau) \in R(E_A, E_B) \}
\]

For simplicity, we write \( \times \) as a general product when the specific \( \Sigma \) is irrelevant.

**Example 3.** Every move operation performed by a robot discharges its battery, and every read operation of a robot gets the value of the battery energy level. By composing a robot and a battery component, under an appropriate interaction signature, the new component exposes, for every read operation of the robot, the appropriate battery value. Typically, this is formally achieved by defining both robot and battery as components \( R \) and \( B \), for which the suitable interaction signature \( \Sigma \) is given. The resulting product \( R \times \Sigma B \) captures the desired TESs in its behavior.

Consider two components \( B \) and \( C \), and a product \( \times \) capturing some interaction constraints between \( B \) and \( C \). Then, the composite expression \( A = C \times B \) captures, as a component, the concurrent observations of components \( C \) and \( B \) under the interaction modelled by \( \times \). Consider a component \( D \) such that \( C \times B = D \times B \). If \( D \) is different from \( C \), then the equality states that the result of \( D \) interacting with \( B \) is the same as \( C \) interacting with \( B \). Consequently, in this context, component \( C \) can be replaced by component \( D \) while preserving the global behavior of \( A \).

In general, a component \( D \) that can substitute for \( C \) is not unique. The set of alternatives for \( D \) depends, moreover, on the product \( \times \), on the component \( B \), and on the behavior of \( A \). A ‘goodness’ measure may induce an order on this set of components, and eventually give rise to a best substitution. More generally, the problem is to characterize, given two components \( A \) and \( B \) and an interaction product \( \times \), the set of all \( C \) such that \( A = C \times B \).

Naturally, the definition of product comes with a dual decomposition operation. A quotient is a component that, when product (under the same interaction signature) with the divisor, yields the dividend. As there might be several possible decomposition, we define the set of such possible quotients. For simplicity, we assume \( \times \) to be commutative. Right and left quotients can be similarly defined when \( \times \) is not commutative.

**Definition 3 (Quotients).** The quotients of \( A \) by \( B \) under the interaction signature \( \Sigma \), written \( A/\Sigma B \), is the set \( \{ C \mid B \times \Sigma C = A \} \).

We say that \( A \) is divisible by \( B \) (or \( B \) divides \( A \)) under \( \Sigma \) if the set of quotients is not empty. We define division operators that pick, given a choice function, the best element from their respective sets of quotients as their quotients.
Example 4. Consider a robot that performs 5 moves, and then stops. Each move consumes some energy, and the robot therefore requires sufficient amount of energy to achieve its moves. The product of a robot $C$ with its battery $B$ under the interaction signature $\Sigma$ is given by the expression $A = C \times_\Sigma B$, where $\Sigma$ synchronizes a move of robot $C$ with battery $B$. Note that different batteries behave differently. The set of batteries that would lead to the same behavior is given by the quotients of $A$ by $B$.

Definition 4 (Division). The division of $A$ by $B$, under the interaction signature $\Sigma$ and the choice function $\chi$ over the quotients, is the element $\chi(A/\Sigma B)$. We write $A/\Sigma B$ to represent the division.

Example 5. It is usual (e.g., [17]) to consider the greatest common divisor when forming the product of cyber-physical components, so that no observation is missed. Our operation of division, however, gives an alternative perspective. Let $C(H)$ be a component whose observations have time stamps multiple of $H \in \mathbb{R}_+$. Then, let $A = C(H_1) \times \Sigma C(H_2)$. The set of components $\{C(H) \mid A = C(H_1) \times \Sigma C(H), H \in \mathbb{R}_+\}$ contains all the quotients of $A$ divisible by $C(H_1)$. The selection of the component with the lowest period $H$ would be one choice function for the division of $A$ by $C(H)$ under $\Sigma$.

Lemma 1. Let $\times_\Sigma$ be commutative. Given $A$ divisible by $B$ under $\Sigma$ and $\chi$ a choice function on the set of quotients of $A$ divisible by $B$, then $B \in A/\Sigma(A/\Sigma B)$.

Proof. If $A$ is divisible by $B$ under $\Sigma$ and if $\chi$ selects one quotient over the set, then $C = A/\Sigma B$ is such that $A = B \times_\Sigma C$. By commutativity of $\times_\Sigma$, $A = C \times_\Sigma B$ and $B \in A/\Sigma C$.

Lemma 2. Let $\times_\Sigma$ be associative. If $A$ is divisible by $B$ under $\Sigma$ and $B$ is divisible by $C$ under $\Sigma$, then $A$ is divisible by $C$ under $\Sigma$.

Proof. If $A$ is divisible by $B$ under $\Sigma$, then there exists $D$ such that $A = B \times_\Sigma D$. If $B$ is divisible by $C$ under $\Sigma$, then there exists $E$ such that $B = C \times_\Sigma E$. By substitution, we have $A = (C \times_\Sigma E) \times_\Sigma D$. Using associativity of $\times_\Sigma$, we get $A = C \times_\Sigma (E \times_\Sigma D)$ which proves that $A$ is divisible by $C$ under $\times_\Sigma$.

4. System of agents and compositional semantics

Components, as introduced in Section 3, can express both cyber and physical aspects in their behavior. Moreover, the family of parametrized operators acting on components enables the construction of complex and interacting components. In this section, we fix an interaction signature $\Sigma = (|\kappa|, \cup)$, where $|\kappa|$ is co-inductively defined [11] using the relation on observations $\kappa$, such that $\times_\Sigma$ is associative and commutative. As a result, the product $\times_\Sigma$ applies on multiset of components, and we write $\times_\Sigma\{C_1, ..., C_n\}$ for the product $C_1 \times_\Sigma ... \times_\Sigma C_n$.

In order to run some simulations and analysis on our model, we look for a finite
specification of some components that also captures the algebraic nature of our semantics. Rewriting logic is an appropriate specification framework for some components. First, modularity is a first class concept. Indeed, a module defines its own equational theory, which can model both discrete or continuous changes. This way, various aspects of components can be specified within a module. Also, the framework is concurrent as term matching may lead to some race conditions, and rewriting may lead to some non-determinisms at runtime. Thus, having an executable and yet concurrent specification language enables some analysis on the resulting specification to explore concurrent executions.

We introduce basic notions of multi-sorted algebra. Let $S$ be a set of sorts. An $S$-sorted set $X$ is a family of sets $(X_s)_{s \in S}$, indexed by sorts in $S$. A signature $\Sigma$ is a tuple $(S, \Omega)$ where $S$ is a set of sorts, and $\Omega$ is a set of function symbols $((S^r \times S)$-sorted set). A $\Sigma$-algebra $A$ consists of an $S$-sorted set $|A|$ of carrier sets; and for each $f : s_1 \times \ldots \times s_n \rightarrow s$ in $\Omega$, a function $f_A : |A_{s_1}| \times \ldots \times |A_{s_n}| \rightarrow |A_s|$. For any algebra $A$, we use $|A|$ to denote the carrier of the algebra, i.e., the set of its terms.

We fix a signature $\Sigma = (S, \Omega)$. We use $T_\Sigma$ to denote the initial $\Sigma$-algebra. Let $V$ be an $S$-sorted set of variables disjoint from elements in $\Omega$. We use $T_\Sigma(V)$ to denote the free $\Sigma$-algebra on generators $V$. Given a term $t \in |T_\Sigma(V)|$, we write $FV(t) \subseteq V$ for the set of free variables occurring in $t$.

Given a function $h : V \rightarrow |A|$ that maps variables in $V$ into the carrier of a $\Sigma$-algebra $A$, we note $\overline{h} : T_\Sigma(V) \rightarrow A$ the unique $\Sigma$-homomorphism that extends $h$ in the sense that $\overline{h}(x) = h(x)$ for all $x \in V$.

A $\Sigma$-equation is of the form $t_1 = t_2$ where $t_1, t_2 \in |T_\Sigma(V)|$ and $t_1$ and $t_2$ are of the same sort. Given a function $\nu : V \rightarrow |A|$, we say that $\nu$ satisfies $t_1 = t_2$ if $\overline{\nu}(t_1) = \overline{\nu}(t_2)$. We then write $\nu \models t_1 = t_2$. Given a set of $\Sigma$-equations $E$, we say that $\nu$ satisfies $E$ if and only if $\nu$ satisfies $e$ for all $e \in E$. Given a $\Sigma$-equation $t_1 = t_2$, we write $FV(e) = FV(t_1) \cup FV(t_2) \subseteq V$ for the set of free variables occurring in $e$, and $FV(E) = \{FV(e) \mid e \in E\}$ for a set of $\Sigma$-equations $E$.

### 4.1. Action, agent, and system

The specification of a multiset of interacting components as a rewrite system depends on the algebraic specification of observations occurring in a component, the specification of a component behavior, and the specification of composition of components. We define in the next paragraphs action, agent, and system, whose respective semantics is a set of observations, a component, and a multiset of components parametrized by a composability relation on observations.

**Action.** The Action sort has terms of the form $a @ d$ where $a$ is of sort ActionName and $d$ is of sort DueTime. In some cases, $a$ could be time sensitive and take $d$ as parameter as well. The action name could be composite, and we use the binary operator $\cdot$ on ActionName to define composite action names. For instance, $\text{move}(R_1, dr) \cdot \text{read}(R_2, l)$ is an action name that models the move of $R_1$ in the direction $dr$ and a read from $R_2$ of its location $l$. We use the constant $\star$ for the idling action name, which semantically captures the empty observable.
The Action sort semantically maps to the algebra of observations $A_{\text{Action}}$, which are pairs of an observable and a time stamp.

**Remark 1.** The due time $d$ in an action $a@d$ models the future relative time for which the action will be executed. We do not consider action with zero delay. As shown later, we disallow Zeno sequences of delays, i.e., the cumulative sum of $d$ always eventually progresses and goes to infinity. We also assume a supersort Time of DueTime that, semantically, maps to the same algebra $A_{\text{Time}}$. We fix $A_{\text{Time}} = (\mathbb{R}_+, +)$ to be the positive reals equipped with addition.

**Agent.** An agent is a rewrite theory $(\Lambda, \Omega, E, \Rightarrow)$ where $\Lambda$ is a set of sorts containing State and Action, the sorts for state and action terms respectively. A pair of a state and a set of actions is called a configuration.

The set of function symbols $\Omega$ contains $\phi : \text{State} \times \text{Action} \to \text{State}$, that takes a pair of a state and an action term to produce a new state. As shown later, the update function is called when the agent needs to update its state after an action.

The $(\Lambda, \Omega)$-equational theory $E$ more particularly specifies the update function $\phi$. The set of equations that specify the function $\phi$ could both make $\phi$ a continuous or discrete function.

The rewrite rule $\Rightarrow$ updates a configuration with an empty set to a new configuration, i.e., $(s, \emptyset) \Rightarrow (s', O)$ with $O$ a non-empty set of action terms, and $s'$ a new state. We call an agent *productive* if, for any state $s : \text{State}$, there exists a state $s'$ with $(s, \emptyset) \Rightarrow (s', O)$ and $O$ non empty set. A productive agent may eventually do the idling action $\star@d$ for some due time $d$. We call an agent *idle enabled* if, for any state $s : \text{State}$, there exists a state $s'$ with $(s, \emptyset) \Rightarrow (s', O)$ and $\star@d \in O$.

**Remark 2.** We use $\star(id)@d$ to refer to the idling action specific to agent with identifier $id$. We distinguish between the singleton action set $\{\star(id)@d\}$ and the empty set of action $\emptyset$. Semantically, the action $\star(id)@d$ gives an empty observable, while the empty set of action does not give any observation. Practically, it models the difference between “looking and seeing nothing” and “not looking at all”. Moreover, the equational specification $E$ for $\phi$ may be that $\phi(s, \emptyset) =_{E} s$ while $\phi(s, \star(id)@t)$ could return a different state.

Given an agent $\mathcal{A} = (\Lambda, \Omega, E, \Rightarrow)$, we use $A$ to denote the free algebra semantics of the theory $(\Lambda, \Omega, E)$, i.e., the $(\Lambda, \Omega)$-sorted algebra such that $A \models E$. We recall the notation $|A_s|$ for the carrier of the algebra $A_s$ for sort $s$.

We first define the transition system of an agent $\mathcal{A}$ as a TES transition system, and then give a finite and infinite semantics of an agent as a component. We point to [9] and [Appendix A] for more results on TES transition systems. Given an agent $\mathcal{A}$, its TES transition system $\mathcal{T}_\mathcal{A} = (Q, E, \rightarrow)$ is such that $Q = |A_{\text{State}}| \times |A_{\text{Time}}|$ is the set of pairs of states with a time stamp (we use the notation $[s, t]$ for states in $Q$), and $E$ the union of all observables labeling the transition relation $\rightarrow \subseteq Q \times (\mathcal{P}(E) \times \mathbb{R}_+) \times Q$ defined as the smallest set such
that:

\[
(s, \emptyset) \Rightarrow (s', O) \quad o \oplus d \in O \quad \nu \models \phi(s', o \oplus d)) = \varepsilon' s''
\]

for some \( \nu : T_{(\Lambda, \Omega)}(X) \rightarrow |A| \) and \( t \in |ATime| \) with \( X \) a \( \Lambda \)-sorted set of variables. Moreover, we write \( q \xrightarrow{s} p \) for the sequence of transitions \( q \xrightarrow{s(0)} q_1 \xrightarrow{s(1)} q_2 \cdots \xrightarrow{s(n-1)} p \), where \( s \in (P(E) \times \mathbb{R}_+)^n \) is a sequence of \( n \) labels. We write \( |s| \) for the size of the sequence \( s \). We use \( \rightarrow^* \) and \( \rightarrow^\omega \) to denote, respectively, the set of finite and infinite sequences of consecutive transitions in \( \rightarrow \).

We use \( L^{\text{fin}}(T, q) \) to denote the set of finite sequences of observables labeling a finite path in \( T \) starting from state \( q \), such that

\[
L^{\text{fin}}(T, q) = \{ s \mid q \xrightarrow{s} q', \forall i < |s| - 1. s(i) = (O_i, t_i) \land t_i < t_{i+1} \}
\]

Additionally, the set \( L^{\text{fin}}^*(T, q) \) is the set of sequences from \( L^{\text{fin}}(T, q) \) postfixed with empty observations, i.e., the set

\[
L^{\text{fin}}^*(T, q) = \{ s\tau \in TES(E) \mid s \in L^{\text{fin}}(T, q) \text{ and } \tau \in TES(\emptyset) \}
\]

We use \( L^{\text{inf}}(T, q) \) to denote the set of TESs labeling infinite paths in \( T \) starting from state \( q \), such that

\[
L^{\text{inf}}(T, q) = \{ \sigma \in TES(E) \mid \forall n. \sigma[n] \in L^{\text{fin}}(T, q) \}
\]

where, as introduced above, \( \sigma[n] \) is the prefix of size \( n \) of \( \sigma \).

Let \( X \subseteq TES(E) \), we use \( cl(X) \) to denote the operation that builds, the continuation with empty observations of any prefix of element in \( X \), i.e., \( cl(X) = \{ s\tau \in TES(E) \mid \tau \in TES(\emptyset) \text{ and } \exists \sigma. \exists i. \sigma \in X \land \sigma[i] = s \} \). Given a component \( C = (E, L) \), we write \( cl(C) \) for the new component \( (E, cl(L)) \).

Let \( A = (\Lambda, \Omega, \mathcal{E}, \Rightarrow) \) be an agent initially in state \( s_0 \in S \) at time \( t_0 \). The finite, respectively infinite, component semantics is the component \([A([s_0, t_0])]^* = (E, L^{\text{fin}}(T_A, [s_0, t_0])), \) respectively \([A([s_0, t_0])] = (E, L^{\text{inf}}(T_A, [s_0, t_0])) \), with \( E = \{ o \mid (o, t) \in |A|_{\text{Action}} \} \).

**Lemma 3 (Closure).** Let \( A \) be a productive agent initially in state \([s_0, t_0] \). Then \([A([s_0, t_0])]^* = cl([A([s_0, t_0])]^*) \).

**Proof.** See appendix \textbf{Appendix B}.

**System.** As introduced in Section 3, a product on components is parametrized by an interaction signature. Here, we consider products that are commutative and associative, defined co-inductively \[10, 9, 8\] given a relation on observations. In our specification, \( \kappa \) relates terms of sort \( \text{Action} \). More precisely, we use \( \kappa \) to range over composability relations on actions, and we consider valid updates of agents as pairs of actions that are related by \( \kappa \).
Remark 3. The due time of an observation in the rewriting framework is relative to the agent’s time and expresses the delta until execution of the action. The due time is merged with a shared time at the semantic level. It is, however, still possible to use the value of the due time to reason about synchrony and exclusion of action in \( \kappa \).

A system is a quadruple \(( \mathcal{A}, \Lambda, \Omega, \mathcal{E}, \Rightarrow \)) where \( \mathcal{A} \) is a set of agents. We use \(( \mathcal{A}_i, \Omega_i, \mathcal{E}_i, \Rightarrow_i )\) to refer to agent \( \mathcal{A}_i \in \mathcal{A} \).

The set of sorts \( \Lambda \) contains a sort Action \( \in \Lambda \) which is a super sort of each sort Action, for \( \mathcal{A}_i \in \mathcal{A} \), and a sort StateSet \( \in \Lambda \) that contains sets of terms of sort State\(_i\) for \( \mathcal{A}_i \in \mathcal{A} \).

The set \( \Omega \) contains the relation symbol \( \kappa \) of signature Action \( \times \) Action, that relates pairs of action terms. If \( \kappa(o_1, o_2) \) is true, then the two actions \( o_1 \) and \( o_2 \) are said to be composable. Semantically, a composability relation is a set of pairs of observations, i.e., a subset of \(|A_{\text{Action}}| \times |A_{\text{Action}}|\).

The set of equations \( E \) specifies the composability relation \( \kappa \). More precisely, we assume that the composability relation \( \kappa \) relates simultaneous actions only, i.e., for actions \( o_1 \circ d_1 \) and \( o_2 \circ d_2 \), \( \kappa(o_1 \circ d_1, o_2 \circ d_2) \) implies \( d_1 = d_2 \). Moreover, \( \kappa \) must be symmetric and transitive, i.e., for all actions \( a_1, a_2, a_3 : \text{Action} \), \( \kappa(a_1, a_2) \iff \kappa(a_2, a_1) \) and \( \kappa(a_1, a_2) \land \kappa(a_2, a_3) \implies \kappa(a_1, a_3) \). Given \( O \) a set of action terms, we use the notation \( \kappa(O) \) for the conjunction \( \forall o_1, o_2 \in O. o_1 \neq o_2 \implies \kappa(o_1, o_2) \). We call \( O \) a clique when \( \kappa(O) \) holds.

Remark 4. Commutativity and associativity of \( \times \Sigma \) respectively imply symmetry and transitivity of \( \kappa \). As a result, \( \kappa \) can be graphically modelled as an undirected graph relating observations, where cliques are strongly connected components that can occur simultaneously. In the sequel, we look for sequences of cliques as emergent behavior of the system of agents.

The rewrite rule \( \Rightarrow_S \) picks a composable action from each agent and applies the update accordingly, as for \( \{k_1, ..., k_j\} \subseteq \{1, ..., n\} \):

\[
\{(s_{k_1}, O_{k_1}), ..., (s_{k_j}, O_{k_j})\} \Rightarrow_S \{(\phi_{k_1}(s_{k_1}, o_{k_1} \circ d), \emptyset), ..., (\phi_{k_j}(s_{k_j}, o_{k_j} \circ d), \emptyset)\}
\]

if \( \kappa(\bigcup_{i \in [1, j]} \{o_{k_i} \circ d\}) \cup \bigcup_{i \in [1, n] \setminus [k_1, k_j]} \{\star(id_i) \circ d\} \) where \( o_{k_i} \circ d \in O_{k_i} \) for \( i \in [1, j] \).

Remark 5. Note that a system does not update all agents in lock steps. It is entirely possible that an agent is in the configuration \((s, \emptyset)\) while some other agents make a joint observation.

Remark 6. There is non-determinism at the system level when multiple cliques are possible. Then, different strategies can be implemented as, for instance, to chose the clique with the lowest time stamp or to chose the largest clique.

Remark 7. The rule \( \Rightarrow_S \) is conditional on finding a clique \( \bigcup_{i \in [1, j]} \{o_{k_i} \circ d\} \cup \bigcup_{i \in [1, n] \setminus [k_1, k_j]} \{\star(id_i) \circ d\} \). Note that at most one action per set of proposed actions is allowed, and the set is completed with the idle action for the other agents.
Example 6 (Composability relation). Consider action name constructed as \( an(id, P, r) \) where \( an \) is the name of the constructor, \( id \) is the identifier of the agent doing the action, \( P \) is a set of parameters of the action, and \( r \) is an agent identifier used as a resource. Composite action names are such that two action names cannot match, i.e., they have either a different source or a different target identifier used as a resource. Composite action names are such that two action names cannot match, i.e., they have either a different source or a different target identifier used as a resource. As an example of specification of a composability relation \( \kappa \), we add two equations

1. \( \forall id, r, d, a_1, a_2. \kappa(an(id, P, r) \cdot a_1 @ d, an(r, P', id) \cdot a_2 @ d) \iff P = P' \);
2. \( \forall id, id', r, d, a. \kappa(an(id, P, r) \cdot a @ d, \cdot (id' @ d)) \implies id' \neq r. \)

Such composability relation, as specified with equation 1, forces every resource to simultaneously generate an action with some parameters, which can then be used to extract some value (e.g., reading sensors, move action). For instance, the pair \( (read(R, l, F), read(F, (x, y), R)) \), where \( l \) is a variable and \( (x, y) \) is a constant location, is composable when \( l = (x, y) \), which eventually updates the robot agent \( R \) with the value given by the field \( F \). The second equation allows for other agents to do nothing if the agent is not used as a resource in at least one of its actions.

We define the transition system for \( S = (A, \Lambda, \Omega, E, \Rightarrow_S) \) as the TES transition system \( T_S = (Q, E, \rightarrow) \) with \( Q = \exp|A_{StateSet}| \times |A_{Time}| \) the set of states, \( E \) the union of all observables labeling the transition relation \( \rightarrow \subseteq Q \times (P(E) \times \mathbb{R}_+) \times Q \) being the smallest transition relation such that:

\[
[(s_{ki}, O_{ki})]_{i \in [1,j]} \Rightarrow_s \{ (\phi_{ki}(s_{ki}, o_{ki} @ d), \emptyset) \}_{i \in [1,j]} \quad \nu \models \bigwedge_{i \in [1,j]} \phi_{ki}(s_{ki}, o_{ki} @ d) =_{\varepsilon, s_{ki}} \]

\[
[(\nu(s_i))_{i \in [1,n]}], t \xrightarrow{((\nu(o_{ki}))_{i \in [1,j]}, t + v(d))} [(\nu(s_1), ..., \nu(s'_{k_j}), ..., \nu(s''_{k_j}), \nu(s_n)), t + \nu(d)]
\]

for some \( \nu : T_{\Lambda \Omega}(X) \rightarrow |A| \) and \( t \in |A_{Time}| \) with \( X \) a set of \( \Lambda \)-sorted variables and where we use the notation \( \{x_i\}_{i \in [1,n]} \) for the set \( \{x_1, ..., x_n\} \).

Remark 8. The top left part of the rule is a rewrite transition at the system level. As defined earlier, the condition for such rewrite to apply is the formation of a clique by all of the actions in the update. The states and labels of the TES transition system (below the rule) are sets of states and sets of labels from the TES transition systems of each agent in the system.

Let \( A = \{A_1, ..., A_n\} \) be a set of agents, and let \( S = (A, \Lambda, \Omega, E, \Rightarrow_S) \) be a system initially in state \( \{ (s_{0i}, \emptyset) \}_{i \in [1,n]} \) at time \( t_0 \) such that, for all \( i \in [1,n] \), \( A_i \) is initially in state \( s_{0i} \) at time \( t_0 \). The finite, respectively infinite, semantics of initialized system \( S([s_{0}, t_0]) \), is the component \( [S([s_{0}, t_0])]^* = (E, \mathcal{L}^{fin}(T_S, [s_{0}, t_0])) \), respectively \( [S([s_{0}, t_0])] = (E, \mathcal{L}^{inf}(T_S, [s_{0}, t_0])) \), where \( E = \bigcup_{i \in [1,n]} \{ o \mid (o, d) \in |A_{Action, i}| \} \).

We fix constructors and equations for the composability relation as defined in Example 6. We assume a closed system \( S = (A, \Lambda, \Omega, E, \Rightarrow_S) \), i.e., such that, given \( |A_{Action, i}| \) the sort for actions of agent \( A_id \in A \), all actions \( an(id, P, id') : |A_{Action, i}| \) have a matching resource in the system \( A_id' \in A \).
Conjecture 1 (Compositional semantics). Let $S = (\mathcal{A}, \Lambda, \Omega, \mathcal{E}, \Rightarrow_S)$ be a closed system of $n$ agents with $\{(s_{01}, \ldots, s_{0n}), t_0\}$ as initial state and actions and composability relation specified as in Example 6. Then, $\llbracket S([s_0, t_0]) \rrbracket = \times \Sigma \{ \llbracket A_i([s_{0i}, t_0]) \rrbracket \}_{i \in [1, n]}$ and $\llbracket S([s_0, t_0]) \rrbracket^* = \times \Sigma \{ \llbracket A_i([s_{0i}, t_0]) \rrbracket^* \}_{i \in [1, n]}$ with $\Sigma = ([\kappa_A], \cup)$ the interaction signature specified by the relation $\kappa$ of the system $S$.

Proof (Sketch). See appendix Appendix B.

5. Application

An implementation of the agent framework in Maude can be found at [8]. Currently, only idle-enabled and time insensitive agents are supported.

We model two robots roaming on a shared field, with energy sensitive behavior: each move of the robot consumes some energy from its battery. We use the same structure for actions as the one introduced in Example 6. For instance, a move of robot $R_1$ on the field $F$ is the action $\text{move}(R_1, \text{dir}, F)@d$ where $\text{dir}$ is a constant that indicates the direction. A robot reads its position from the field with the action $\text{read}(R_1, l, F)@d$ where $l$ is a variable ranging over coordinate locations.

We fix a composability relation on actions, which is similar to that introduced in Example 6. Namely, a clique should contain the both actions, one from each direction, i.e., if $\text{move}(R_1, \text{dir}, F)@d \in O$, then $\text{move}(F, v, R_1)@d$ should also be an action in $O$. As a consequence, every assignment, for that clique, should map the variable $v$ to the direction $\text{dir}$, and update the field accordingly. Alternatively, the action $\text{read}(R_1, l, F)@d$ from the agent occurs in the same clique as the action $\text{read}(F, (x, y), R_1)@d$ with $(x, y)$ a constant location.

We use the structure $[\text{state}; \text{flag}; \text{actionSet}]$ to denote the configuration of an agent, where the state contains the identifier of the agent (e.g., $\text{bat}(1)$, field, or $\text{id}(1)$ for, respectively, the identifier of a battery, a field, or a robot), the flag notifies that the agent is ready, and the actionSet is a set of actions that the agent submits.

System. We consider a multiset of agents, containing two robot agents with identifiers $\text{id}(0)$ and $\text{id}(1)$, paired with two batteries $\text{bat}(0)$ and $\text{bat}(1)$, and sharing a field $\text{field}$. The state of each agent is described as a map of the form $k(\text{"key"}) \mapsto \text{value}$. The goal for each agent is to reach the initial location of the other agent. As a matter of fact, both agents need to cross, eventually. The crossing can lead to a livelock, where agents move symmetrically until the energy of the battery runs out.

The initial system term is given by:

```
eq init = \llbracket [\text{id}(0) : \text{Troll} \mid k(\text{"goal"}) \mapsto (5 \ ; \ 5) \ ; \ false \ ; \ null ] \\
[\text{bat}(0) : \text{Battery} \mid k(\text{"bat"}) \mapsto \text{nd(capacity)} \ ; \ false \ ; \ null ] \\
[\text{id}(1) : \text{Troll} \mid k(\text{"goal"}) \mapsto (0 \ ; \ 5) \ ; \ false \ ; \ null ] \\
[\text{bat}(1) : \text{Battery} \mid k(\text{"bat"}) \mapsto \text{nd(capacity)} \ ; \ false \ ; \ null ] \\
[\text{field} : \text{Field} \mid (k((0 \ ; \ 5)) \mapsto d(\text{id}(0)) , k((5 \ ; \ 5)) \mapsto d(\text{id}(1))) \ ; \ false \ ; \ null] \rrbracket .
```
**Analysis in Maude.** We analyze in Maude two scenarios. In one, each robot has as strategy to take the shortest path to reach its goal. As a consequence, a robot reads its position, computes the shortest path, and submits a set of optimal actions. A robot can sense an obstacle on its direct next location, which then allows for sub-optimal lateral moves (e.g., if the obstacle is in the direct next position on the West direction, the robot may go either North or South). In the other scenario, we add a protocol that swaps the two robots if robot \(id(0)\) is on the direct next location at the west of robot \(id(1)\). The swapping is a sequence of moves that ends in an exchange of position for both robots.

In the two scenarios, we analyze the behavior of the resulting system with two queries. The first query asks if the battery can reach an energy level of 0, which means that the two robots can no longer move:

```plaintext
search [1] init =>* [ sys :: Sys
   [ bat(1) : Battery | k(level) |-> 0 ; true ; null],
   [ bat(2) : Battery | k(level) |-> 0 ; true ; null] ] .
```

The second query asks if the two robots successfully reached their goal, and ended in the expected location:

```plaintext
search [1] init =>* [ sys :: Sys
   [ field : Field | k(( 5 ; 5 )) |-> d(id(0)),
     k(( 0 ; 5 )) |-> d(id(1)) ; true ; null] ] .
```

As a result, when the protocol is absent, the two robots can enter in a livelock behavior and eventually starve from energy with an empty battery:

**Solution 1 (state 80)**
states: 81 rewrites: 223566 in 73 ms cpu (74 ms real) (3053554 rewrites/second)

Alternatively, when the protocol is used, the livelock is removed by use of the exogenous coordination. The two robots therefore successfully reach their end location, and stop before running out of battery:

**No solution.**
states: 102 rewrites: 720235 in 146 ms cpu (145 ms real) (4920041 rewrites/second)

In both cases, the second query succeeds, as there exists a path for both scenarios where the two robots reach their end goal location. The results can be reproduced by downloading the archive at [8].

6. Related work

**Rewriting logic.** In [15], the modeling of cyber-physical systems from an actor perspective is discussed. The notion of event comes as a central concept to model interaction between agents. Rewriting logic is an expressive framework to model concurrent systems [16, 11]. Programming languages such as Maude [3] makes the framework executable and analysable.

In [3] a multiset rewriting model of time sensitive distributed systems such as cyber-physical agent, is introduced. Two verification problems are defined
relative to a given property $P$: realizability (is there a trace that satisfies $P$), and
survivability (do all traces satisfy $P$) and their complexity is analyzed. In [6]
the theory is extended with two further properties that concern the ability to
avoid reaching a bad state.

Algebra, co-algebra. The algebra of components described in this paper is an
extension of [10]. Algebra of communicating processes [4] (ACP) achieves similar
objectives as decoupling processes from their interaction. For instance, the
encapsulation operator in process algebra is a unary operator that restricts
which action to occur, i.e., $\delta_H(t || s)$ prevent $t$ and $s$ to perform actions in $H$.
Moreover, composition of actions is expressed using communication functions,
i.e., $\gamma(a, b) = c$ means that actions $a$ and $b$, if performed together, form the
new action $c$. Different types of coordination over communicating processes are
studied in [2]. In [1], the authors present an extension of ACP to include time
sensitive processes.

Discrete Event Systems. Our work represents both cyber and physical aspects
of systems with a unified model of discrete event systems. In [7], the author lists
the current challenges in modelling cyber-physical in such a way. The author
points to the problem of modular control, where even though two modules run
without problems in isolation, the same two modules may block when they
are used in conjunction. In [14], the authors present procedures to synthesize
supervisors that control a set of interacting processes and, in the case of failure,
report a diagnosis. An application for large scale controller synthesis is given
in [12].

Coordination. In [13], the author describes infinite behaviors of process and
their synchronization. Notably, the problem of non-blockingness is stated: if
two processes eventually interact on some actions, how to make sure that both
processes will not block each others. The concept of centrality of a process is
introduced.

7. Conclusion

We approach the challenge of designing cyber-physical systems using alge-
braic methods. Components denote sequences of observations over time, and
operations on components capture the interaction that arises from the behaviors
of those components.

We define agents that specifies components as rewriting theories and systems
that concurrently run agents under a composability constraint. We conjecture
conditions for our component semantics to be compositional.

We apply our framework on an example consisting of two robots roaming on
a shared field. We analyze the behavior of the resulting system before and after
coordination with a protocol, and show how the protocol can prevent livelock
behavior.
Acknowledgement. Talcott was partially supported by the U. S. Office of Naval Research under award numbers N00014-15-1-2202 and N00014-20-1-2644, and NRL grant N0017317-1-G002. Arbab was partially supported by the U. S. Office of Naval Research under award number N00014-20-1-2644.

References

[1] Jos C. M. Baeten and Cornelis A. Middelburg. Real time process algebra with time-dependent conditions. *J. Log. Algebraic Methods Program.*, 48(1-2):1–38, 2001.

[2] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1):109–137, 1984.

[3] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[4] Wan J. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2000.

[5] Max I. Kanovich, Tajana Ban Kirigin, Vivek Nigam, Andre Scedrov, and Carolyn L. Talcott. Timed multiset rewriting and the verification of time-sensitive distributed systems. *CoRR*, abs/1606.07886, 2016.

[6] Max I. Kanovich, Tajana Ban Kirigin, Vivek Nigam, Andre Scedrov, and Carolyn L. Talcott. On the complexity of verification of time-sensitive distributed systems: Technical report. *CoRR*, abs/2105.03531, 2021.

[7] Stéphane Lafortune. Discrete event systems: Modeling, observation, and control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):141–159, 2019.

[8] Benjamin Lion. Cyber-physical framework in maude, May 2022.

[9] Benjamin Lion, Farhad Arbab, and Carolyn Talcott. Runtime composition of systems of interacting cyber-physical components, 2022.

[10] Benjamin Lion, Farhad Arbab, and Carolyn L. Talcott. A semantic model for interacting cyber-physical systems. In Julien Lange, Anastasia Mavridou, Larisa Safina, and Alceste Scalas, editors, *Proceedings 14th Interaction and Concurrency Experience, ICE 2021, Online, 18th June 2021*, volume 347 of *EPTCS*, pages 77–95, 2021.

[11] José Meseguer. Membership algebra as a logical framework for equational specification. In Francesco Parisi Presicce, editor, *Recent Trends in Algebraic Development Techniques*, pages 18–61, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.
[12] L. Moormann, J.M. van de Mortel-Fronczak, W.J. Fokkink, P. Maessen, and J.E. Rooda. Supervisory control synthesis for large-scale systems with isomorphisms. Control Engineering Practice, 115:104902, 2021.

[13] Maurice Nivat. Behaviors of Processes and Synchronized Systems of Processes, pages 473–551. Springer Netherlands, Dordrecht, 1982.

[14] Meera Sampath, Stéphane Lafortune, and Demosthenis Teneketzis. Active diagnosis of discrete-event systems. IEEE Trans. Autom. Control, 43(7):908–929, 1998.

[15] Carolyn L. Talcott. Cyber-physical systems and events. In Martin Wirsing, Jean-Pierre Banâtre, Matthias M. Hölzl, and Axel Rauschmayer, editors, Software-Intensive Systems and New Computing Paradigms - Challenges and Visions, volume 5380 of Lecture Notes in Computer Science, pages 101–115. Springer, 2008.

[16] Patrick Viry. Rewriting: An effective model of concurrency. In Costas Halatsis, Dimitrios Maritsas, George Philokyprou, and Sergios Theodoridis, editors, PARLE’94 Parallel Architectures and Languages Europe, pages 648–660, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[17] Xiong Xu, Jean-Pierre Talpin, Shuling Wang, Bohua Zhan, and Naijun Zhan. Semantics foundation for cyber-physical systems using higher-order utp. ACM Trans. Softw. Eng. Methodol., feb 2022. Just Accepted.
Appendix A. TES transition system

In [9], we prove that the behavior of a component as in Definition 1 is a set of TESs. We give an operational definition of such set using a labelled transition system.

Definition 5 (TES transition system). A TES transition system is a triple $(Q, E, \rightarrow)$ where $Q$ is a set of states, $E$ is a set of events, and $\rightarrow \subseteq Q \times (\mathcal{P}(E) \times \mathbb{R}_+ \times Q)$ is a labeled transition relation, where labels are observations. \(\triangle\)

We present two different ways to give a semantics to a TES transition system: inductive and co-inductive. Both definitions give the same behavior, as shown in Theorem 1 in [9].

Semantics 1 (runs). Let $T = (Q, E, \rightarrow)$ be a TES transition system. Given $s \in (\mathcal{P}(E) \times \mathbb{R}_+)^n$, we write $q \xrightarrow{s} p$ for the sequence of transitions $q \xrightarrow{s(0)} q_1 \xrightarrow{s(1)} q_2 \ldots \xrightarrow{s(n)} p$. We use $\rightarrow^*$ and $\rightarrow^\omega$ to denote, respectively, the set of finite and infinite sequences of consecutive transitions in $\rightarrow$. Then, finite sequences of observables form the set $\mathcal{L}^{\text{fin}}(T, q) = \{\sigma \in \mathcal{T}(E) \mid q \xrightarrow{\sigma} q', \exists n.s = \sigma[n] \land \sigma' \in \mathcal{TES}(\emptyset)\}$ and infinite ones, the set $\mathcal{L}^{\text{inf}}(T, q) = \{\sigma \in \mathcal{T}(E) \mid \forall n.\sigma[n] \in \mathcal{L}^{\text{fin}}(T, q)\}$ where, as introduced above, $\sigma[n]$ is the prefix of size $n$ of $\sigma$. The semantics of such a TES transition system $T = (Q, E, \rightarrow)$, starting in a state $q \in Q$, is the component $C_T(q) = (E, \mathcal{L}^{\text{inf}}(T, q))$.

Semantics 2 (greatest post fixed point). Alternatively, the semantics of a TES transition system is the greatest post fixed point of a function over sets of TESs paired with a state. For a TES transition system $T = (Q, E, \rightarrow)$, let $\mathcal{R} \subseteq \mathcal{T}(E) \times Q$. We introduce $\phi_T : \mathcal{P}(\mathcal{T}(E) \times Q) \rightarrow \mathcal{P}(\mathcal{T}(E) \times Q)$ as the function:

$$
\phi_T(\mathcal{R}) = \{((\tau, q) \mid \exists p \in Q, q \xrightarrow{\tau} p \land (\tau', p) \in \mathcal{R}\}
$$

The product of two components is parametrized by a composability relation and a composition function and syntactically constructs the product of two TES transition systems.

Definition 6 (Product). The product of two TES transition systems $T_1 = (Q_1, E_1, \rightarrow_1)$ and $T_2 = (Q_2, E_2, \rightarrow_2)$ under the constraint $\kappa$ is the TES transition system $T_1 \times \kappa T_2 = (Q_1 \times Q_2, E_1 \cup E_2, \rightarrow)$ such that:

$$
\begin{align*}
q_1 \xrightarrow{(O_1, t_1)} q'_1 &\quad q_2 \xrightarrow{(O_2, t_2)} q'_2 &\quad ((O_1, t_1), (O_2, t_2)) \in \kappa(E_1, E_2) &\quad t_1 < t_2 &\quad (q_1, q_2) \xrightarrow{(O_1, t_1)} (q'_1, q_2) \\
q_1 \xrightarrow{(O_1, t_1)} q'_1 &\quad q_2 \xrightarrow{(O_2, t_2)} q'_2 &\quad ((O_1, t_1), (O_2, t_2)) \in \kappa(E_1, E_2) &\quad t_2 < t_1 &\quad (q_1, q_2) \xrightarrow{(O_2, t_2)} (q_1, q'_2)
\end{align*}
$$
\[
\frac{q_1 \xrightarrow{(O_1,t_1)} q_1'} {q_2 \xrightarrow{(O_2,t_2)} q_2'} (q_1, q_2) \xrightarrow{(O_1 \cup O_2, t_1)} (q_1', q_2')
\]

Observe that the product is defined on pairs of transitions, which implies that if \( T_1 \) or \( T_2 \) has a state without outgoing transition, then the product has no outgoing transitions from that state. The reciprocal is, however, not true in general.

Theorem 1 in [9] states that the product of TES transition systems denotes (given a state) the set of TESs that corresponds to the product of the corresponding components (in their respective states). Then, the product that we define on TES transition systems does not add nor remove behaviors with respect to the product on their respective components.

**Appendix B. Proofs**

**Proof (Sketch - Conjecture [1]).** We fix \( S = (\{A_1, ..., A_n\}, \Lambda, \Omega, E, \Rightarrow_S) \) and \( A_{n+1} = (\Lambda_{n+1}, \Omega_{n+1}, E_{n+1}, \Rightarrow_{n+1}) \). Let \( S' = (\{A_1, ..., A_n, A_{n+1}\}, \kappa, \Rightarrow_S) \).

We show that \( T_S \times \kappa T_{A_{n+1}} = (Q, E, \rightarrow) \) and \( T_S = (Q', E', \rightarrow') \) are bisimilar, which consists in the existence of a relation \( R \subseteq Q \times Q' \) such that, for all \( (q, r) \in R \):

1. \( \forall q' \in Q \) with \( q \xrightarrow{(O,t)} q' \), there exists \( r' \in Q' \) with \( r \xrightarrow{(O,t)} r' \); and
2. \( \forall r' \in Q' \) with \( r \xrightarrow{(O,t)} r' \), there exists \( q' \in Q \) with \( q \xrightarrow{(O,t)} q' \).

First, we define an equivalence relation \( \sim \) on states in \( Q \) as \( ([s_S, t], [s_A, t']) \sim ([s_S, \max(t, t')], [s_A, \max(t, t')]) \). Then, we define the set of states \( Q_\sim \) such that \( ([s_S, \max(t, t')], [s_A, \max(t, t')]) \in Q_\sim \) if and only if \( ([s_S, t], [s_A, t]) \in Q \) or \( ([s_S, t'], [s_A, t]) \in Q \). We show that the TES transition system \( T_S \times \kappa T_{A_{n+1}} \), projected to states in \( Q_\sim \), is bisimilar to \( T_S \times \kappa T_{A_{n+1}} \). The reason is that the transition rules in \( T_S \) and \( T_{A_{n+1}} \) universally quantify over time \( t \in |A_{Time}| \), which allows arbitrary positive translation in time. As a consequence, states in \( Q' \) can be embedded in states in \( Q \).

We now prove 1 and 2 by showing that \( ([s_1', t], [s_2', t']) \xrightarrow{(O,t')} ([s_1', t'], [s_2', t']) \) if and only if \( ([s_1, t], [q_1, t]) \xrightarrow{(O,t')} ([s_2, \max(t, t')], [q_2, \max(t, t')]) \) where \( s_1 \) and \( s_2 \) are states in \( T_S \), \( q_1 \) and \( q_2 \) are states in \( T_{A_{n+1}} \) and \( s_1' \) and \( s_2' \) are states in \( T_S \). We split cases on whether the observation comes from \( S \), from \( A \), or is a joint observation.

We use the equational theory of the system to prove the result. \( \triangle \)

**Lemma 4 (Closure).** Let \( A \) be a productive agent initially in state \( [s_0, t_0] \). Then \( [A([s_0, t_0])]^* = cl([A([s_0, t_0])]) \).
The other direction comes from the assumption that $\mathcal{A}$ is productive. Then, every reachable state in $\mathcal{T}_A$ has an outgoing transition and therefore every finite sequence of transition is a prefix of an infinite sequence. Thus, $L^{\text{fin}*}(\mathcal{T}_A, [s_0, t_0]) \subseteq \text{cl}(L^{\text{inf}}(\mathcal{T}_A, [s_0, t_0]))$. \qed