An algebra for interaction of cyber-physical components

Lion, B.

Chapter 4

Operational specifications of components

In Chapter 2, we presented a model of components 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 parametrized with an interaction signature. Thus, cyber-physical systems are defined modularly, where the product of two components models the interaction occurring between the two components. The strength, as well as 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 chapter, three operational descriptions of components each at different level of abstraction.

As a first operational specification, we present a state-based description of a component's behavior using labeled transition systems. A *TES transition system* has transitions labeled with observations, and enriches components with states. Different TES transition systems may denote the same component, as a component is oblivious to internal non-determinism of the machinery that manifests its behavior. As for components, we introduce a family of parametrized algebraic products on TES transition systems. The parameter here is a composability relation on observations, and each transition in the product is the result of the composition of a pair of transitions with composable labels. We show that the TES transition system component semantics is compositional with respect to such products, i.e., the component resulting from the product of two TES transition systems is equal to the product of the components resulting from each TES transition system. On components, the composability relation is co-inductively lifted from observations to TESs, and the composition function is set union on observations and interleaving on streams.

Because the composability relation on observations is a step-wise operation, it may lead to deadlock states in the product TES transition system, i.e., states with no outgoing transitions. We call two TES transition systems *compatible* with respect to a composability relation on observations if, for every reachable pair of states, there is at least one pair of transitions whose pair of labels is composable. We give some sufficient conditions for TES transition systems to be compatible, and show that if two TES transition systems are compatible, then their product can be done lazily, i.e., step by step at runtime. Note that, however, a TES transition system may not be executable, e.g., have countably infinite states or transitions.

As a second operational specification, we introduce a finitely representable description of components in a rewriting logic specification. Rewriting logic is a powerful framework to model concurrent systems [69, 68]. Moreover, implementations, such as Maude [29], make system specifications both executable and analyzable. Rewriting logic is suitable for specifying cyber-physical systems, as its underlying equational theory can represent both discrete and continuous changes. We give an operational specification for components as rewriting systems, and show its compositionality under some assumptions.

Our rewriting specification has the following benefits. First, performing lazy composition keeps the representation of an interacting system small. Second, step-wise runtime composition renders our runtime framework modular, where run-time replacement of individual components becomes possible (as long as the update complies with some rules). Finally, the runtime framework more closely matches the architecture of a distributed framework, where entities are physically separated and no party may have access to the whole description of the entire system. A Maude implementation of our framework is provided in Chapter 5, with a series of detailed examples. The rewriting specification of components, however, does not offer mechanisms to resolve non-determinism at the component or the system levels: multiple transitions may be allowed, and one is selected non-deterministically.

As a third operational specification, we introduce an algebra of weighted automata whose transition values model an internal strategy for an agent. The preference structure is compositional, which allows for reasoning about local choices of an agent in a state, or global choices of a system of agents given the product of their respective transition values.

As an example application, we use a subset of the Reo language as a domain specific language to graphically specify preference aware agents in interaction.

4.1 Components as transition systems

In Section 2.1, we give a declarative specification of components, and considers infinite behaviors only. We give, in Section 4.1.1, an operational specification of components using TES transition systems. We relate the parametrized product of TES transition systems with the parametrized product on their corresponding components, and show its correctness. The composition of two TES transition systems may lead to transitions that are not composable, and ultimately to a deadlock, i.e., a state with no outgoing transitions.

Notation Given $\sigma : \mathbb{N} \to \Sigma$, let $\sigma[n] \in \Sigma^n$ be the finite prefix of size n of σ and let $\sim n$ be an equivalence relation on $(N \to \Sigma) \times (N \to \Sigma)$ such that $\sigma \sim_n \tau$ if and only if $\sigma[n] = \tau[n]$. Let $FG(L)$ be the set of left factors of a set $L \subseteq \Sigma^\omega$, defined as $FG(L) = \{\sigma[n] \mid n \in \mathbb{N}, \sigma \in L\}$. We write $\sigma(n)$ for the n-th element of σ.

4.1.1 TES transition systems.

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 32 (TES transition system)**.** *A TES transition system is a triple $(Q, E, \to)$ where Q is a set of state identifiers, E is a set of events, and $\to \subseteq (Q \times \mathbb{N}) \times (\mathcal{P}(E) \times \mathbb{R}_+) \times (Q \times \mathbb{N})$ is a labeled transition relation, where labels on transitions are observations and a state is a pair of a state identifier and a counter value, such that $[q, c] \xrightarrow{(O,t)} [q', c']$ implies that $c' \geq c$.*

Remark 12. *The counter value labeling a state of a TES transition system is related to the number of transitions a TES transition system has taken. The counter value is therefore not related to the time of the observation labeling the transition. However, it is possible for some transitions in the TES transition system to keep the same counter value in the post state. As shown later, we use the counter value to model fairness in the product of two TES transition systems.*

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 6, and we use interchangeably each definition to simplify the proofs of, e.g., Theorem 7.

Example 46 (Strictly progressing TES transition system)**.** *We call a TES transition systems* strictly progressing *if, for all transitions* $[q, c] \xrightarrow{(O,t)} [q', c']$*, we have that* $c' > c$*. An example of a TES transition system that is strictly progressing is one for which the counter label increases by* 1 *for each transition, i.e.,* $[q, c] \xrightarrow{(O,t)} [q', c + 1]$*.*

We use the notation $\theta([q, c])$ to refer to the counter value c labeling the state $[q, c]$.

Semantics 1 (runs). A run of a TES transition system is an infinite sequence of consecutive transitions, such that the sequence of observations labeling the transitions form a TES, and the counter in the state is always eventually strictly increasing. Formally, the set of runs $\mathcal{L}^{\text{inf}}(T, s_0)$ of a TES transition system $T = (Q, E, \rightarrow)$ initially in state s_0 is:

$$\mathcal{L}^{\text{inf}}(T, s_0) = \{\tau \in \mathit{TES}(E) \mid \exists \chi \in (Q \times \mathbb{N})^{\omega}.\chi(0) = s_0 \wedge \forall i.\chi(i) \xrightarrow{\tau(i)} \chi(i+1) \wedge \\ \exists j > 0.\ \theta(\chi(i+j)) > \theta(\chi(i))\}$$

Note that the domain of quantification for $\mathcal{L}^{\text{inf}}(T, s_0)$ ranges over TESs, therefore the time labeling observations is, by definition, strictly increasing and non-Zeno. The component semantics of a TES transition system $T = (Q, E, \rightarrow)$ initially in state q is the component $C = (E, \mathcal{L}^{\text{inf}}(T, q))$.

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 $\mathcal{R} \subseteq \mathit{TES}(E) \times (Q \times \mathbb{N})$. We introduce $\phi_T : \mathcal{P}(\mathit{TES}(E) \times (Q \times \mathbb{N})) \rightarrow \mathcal{P}(\mathit{TES}(E) \times (Q \times \mathbb{N}))$ as the function:

$$\phi_T(\mathcal{R}) = \{(\tau, s) \mid \quad \exists n.\exists p \in (Q \times \mathbb{N}),\ s \xrightarrow{\tau[n]} p\ \wedge \theta(p) > \theta(s) \wedge (\tau^{(n)}, p) \in \mathcal{R}\}$$

where $\tau[n]$ is the prefix of size n of the TES τ.

We can show that ϕ_T is monotonous, and therefore ϕ_T has a greatest post fixed point $\Omega_T = \bigcup\{\mathcal{R} \mid \mathcal{R} \subseteq \phi_T(\mathcal{R})\}$. We write $\Omega_T(q) = \{\tau \mid (\tau, s) \in \Omega_T\}$ for any $s \in Q \times \mathbb{N}$. Note that the two semantics coincide.

Theorem 6 (Equivalence)**.** *For all $s \in Q \times \mathbb{N}$, $\mathcal{L}^{\text{inf}}(T,s) = \{\tau \mid (\tau, s) \in \Omega_T\}$.*

Proof. Let $T = (Q, E \to)$ and $\Omega_T(s) = \{\tau \mid (\tau, s) \in \Omega_T\}$.

$$\begin{aligned}
\tau \in \Omega_T(s_0) \iff & (\tau, s_0) \in \Omega_T \\
\iff & \exists n. \exists s. s_0 \xrightarrow{\tau[n]} s \wedge \theta(s) > \theta(s_0) \wedge (\tau^{(n)}, s) \in \Omega_T \wedge \tau \in \mathit{TES}(E) \\
\iff & \exists n. \exists \chi \in (Q \times \mathbb{N})^\omega.\ \chi(0) = s_0 \wedge \chi(0) \xrightarrow{\tau[n]} \chi(n) \wedge \\
& \theta(\chi(n)) > \theta(\chi(0)) \wedge (\tau^{(n)}, \chi(n)) \in \Omega_T \wedge \tau \in \mathit{TES}(E) \\
\iff & \exists \chi \in (Q \times \mathbb{N})^\omega, \chi(0) = s_0 \wedge \forall n \in \mathbb{N}. \chi(n) \xrightarrow{\tau(n)} \chi(n+1) \wedge \\
& \exists k \in \mathbb{N}.\ \theta(\chi(n+k)) > \theta(\chi(n)) \wedge \tau \in \mathit{TES}(E) \\
\iff & \tau \in \mathcal{L}^{\text{inf}}(T, s_0)
\end{aligned}$$

In the fourth equivalence, we state that the infinite sequence of transitions with $\chi \in (Q \times \mathbb{N})^\omega$ as sequence of states, is labeled by the sequence of observations τ. We prove the step by induction. Let $n \in \mathbb{N}$, and let $\chi \in (Q \times \mathbb{N})^\omega$ be such that, for all $k \leq n$, $\chi(k) \xrightarrow{\tau(k)} \chi(k+1)$ and $(\tau^{(k)}, \chi(k)) \in \Omega_T$. Then, given that $(\tau^{(n)}, \chi(n)) \in \Omega_T$, there exists an $i \in \mathbb{N}$, a sequence of transitions $\chi(j) \xrightarrow{\tau(j+1)} \chi(j+1)$ for $j \leq i$ which proves, by induction, the implication.

The other direction of the equivalence is simpler. If there exists $\chi \in (Q \times \mathbb{N})^\omega$ such that for all $n \in \mathbb{N}$, there exists $k \in \mathbb{N}$ with $\chi(n) \xrightarrow{\tau(n)[k]} \chi(n+k)$, then we have a witness, for every $n \in \mathbb{N}$, that $(\tau^{(n)}, \chi(n))$ is an element of Ω_T. □

The semantics of a TES transition system is defined as the component whose behavior contains all the runs. Operationally, however, the (infinite) step-wise generation of such a sequence does not always return a valid prefix of a run. We introduce finite sequences of a TES transition system, and define a deadlock of a TES transition system as a reachable state without outgoing transition.

Let $T = (Q, E, \to)$ be a TES transition system. We write $q \xrightarrow{u} p$ for the sequence of transitions $q \xrightarrow{u(0)} q_1 \xrightarrow{u(1)} q_2 ... \xrightarrow{u(n-1)} p$, where $u = \langle u(0), ..., u(n-1) \rangle \in (\mathcal{P}(E) \times \mathbb{R}_+)^n$. We write $|u|$ for the size of the sequence u. We use $\mathcal{L}^{\text{fin}}(T, q)$ to denote the set of finite sequences of observables labeling a finite path in T starting from state q, such that

$$\mathcal{L}^{\text{fin}}(T,s) = \{u \mid \exists p. s \xrightarrow{u} p \wedge \forall i < |u| - 1. u(i) = (O_i, t_i) \wedge t_i < t_{i+1}\}$$

Let $FG(L)$ be the set of left factors of a set $L \subseteq \Sigma^\omega$, defined as $FG(L) = \{\sigma[n] \mid n \in \mathbb{N}, \sigma \in L\}$. We write $\sigma(n)$ for the n-th derivative of σ, i.e., the stream such that $\sigma(n)(i) = \sigma(n+i)$ for all $i \in \mathbb{N}$.

Remark 13 (Deadlock)**.** *Observe that $FG(\mathcal{L}^{\text{inf}}(T,q)) \subseteq \mathcal{L}^{\text{fin}}(T,q)$ which, in the case of strict inclusion, captures the fact that some states may have no outgoing transitions and therefore deadlock.*

Remark 14 (Abstraction)**.** *There may be two different TES transition systems T_1 and T_2 such that $\mathcal{L}^{\text{inf}}(T_1) = \mathcal{L}^{\text{inf}}(T_2)$, i.e., a set of TESs is not uniquely characterized by a TES transition system. In that sense, the TES representation of behaviors is more abstract than TES transition systems.*

We use the transition rule $q \xrightarrow{(O,t)} q'$ where the counter is not written to denote the set of transitions

$$[q,c] \xrightarrow{(O,t)} [q',c']$$

for $c \in \mathbb{N}$ and $c' \in \mathbb{N}$ with $c' \geq c$.

Example 47. *The behavior of a robot introduced earlier is a TES transition system $T_R = (\{q_0\}, E_R, \rightarrow)$ where $q_0 \xrightarrow{(\{e\},t)} q_0$ for abitrary t in $\mathbb{R}_+$ and $e \in E_R$. Similarly, the behavior of a grid is a TES transition system $T_G(I,n,m) = (Q_G, E_G(I,n,m), \rightarrow)$ where:*

- $Q_G \subseteq (I \rightarrow ([0;n] \times [0;m]))$,
- $f \xrightarrow{(O,t)} f'$ *for abitrary t in $\mathbb{R}_+$, such that*
  - $d_R \in O$ *implies $f'(R)$ is updated according to the direction d if the resulting position is within the bounds of the grid;*
  - $(x,y)_R \in O$ *implies $f(R) = (x,y)_R$ and $f'(R) = f(R)$;*
  - $f'(R) = f(R)$, *otherwise.*

The behavior of a swap protocol S(R_i,R_j) with $i < j$ is a TES transition system $T_S(R_1,R_2) = (Q,E,\rightarrow)$ where, for $t_1,t_2,t_3 \in \mathbb{R}_+$ with $t_1 < t_2 < t_3$:

- $Q = \{q_1,q_2,q_3,q_4,q_6\}$;
- $E = E_{R_i} \cup E_{R_j} \cup \{lock(R_i,R_j), unlock(R_i,R_j), start(R_i,R_j), end(R_i,R_j)\}$
- $q_1 \xrightarrow{(\{lock(R_i,R_j)\},t_1)} q_2$;
- $q_2 \xrightarrow{(\{unlock(R_i,R_j)\},t_1)} q_1$;
- $q_1 \xrightarrow{(\{start(R_i,R_j),(x,y)_{R_i},(x+1,y)_{R_j}\},t_1)} q_3$;

- $q_3 \xrightarrow{(\{N_{R_j}\},t_1)} q_4 \xrightarrow{(\{W_{R_j},E_{R_i}\},t_2)} q_5 \xrightarrow{(\{S_{R_j}\},t_3)} q_6$;
- $q_6 \xrightarrow{(\{end(R_i,R_j)\},t_1)} q_1$;

■

The product of two components is parametrized by a composability relation κ on observations and syntactically constructs the product of two TES transition systems.

Definition 33 (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 κ is the TES transition system $T_1 \times_\kappa T_2 = (Q_1 \times Q_2, E_1 \cup E_2, \rightarrow)$ such that:*

$$\frac{[q_1, c_1] \xrightarrow{(O_1,t_1)}_1 [q_1', c_1'] \quad [q_2, c_2] \xrightarrow{(O_2,t_2)}_2 [q_2', c_2'] \quad ((O_1, t_1), (\emptyset, t_1)) \in \kappa(E_1, E_2) \quad t_1 < t_2}{[(q_1, q_2), \min(c_1, c_2)] \xrightarrow{(O_1,t_1)} [(q_1', q_2), \min(c_1', c_2)]}$$

$$\frac{[q_1, c_1] \xrightarrow{(O_1,t_1)}_1 [q_1', c_1'] \quad [q_2, c_2] \xrightarrow{(O_2,t_2)}_2 [q_2', c_2'] \quad ((\emptyset, t_2), (O_2, t_2)) \in \kappa(E_1, E_2) \quad t_2 < t_1}{[(q_1, q_2), \min(c_1, c_2)] \xrightarrow{(O_2,t_2)} [(q_1', q_2), \min(c_1, c_2')]}$$

$$\frac{[q_1, c_1] \xrightarrow{(O_1,t_1)}_1 [q_1', c_1'] \quad [q_2, c_2] \xrightarrow{(O_2,t_2)}_2 [q_2', c_2'] \quad ((O_1, t_1), (O_2, t_2)) \in \kappa(E_1, E_2) \quad t_2 = t_1}{[(q_1, q_2), \min(c_1, c_2)] \xrightarrow{(O_1 \cup O_2,t_1)} [(q_1', q_2'), \min(c_1', c_2')]}$$

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. We write $C_{T_1 \times_\kappa T_2}((s_1, s_2))$ for the component $C_{T_1 \times_\kappa T_2}([(q_1, q_2), \min(c_1, c_2)])$ where $s_1 = [q_1, c_1]$ and $s_2 = [q_2, c_2]$.

Theorem 7 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.

Example 48. *Consider two strictly progressing (as in Example 46) TES transition systems $T_1 = (Q_1, E_1, \rightarrow_1)$ and $T_2 = (Q_2, E_2, \rightarrow_2)$. Then, consider a transition in the product $T_1 \times_\kappa T_2$ such that*

$$[(q_1, q_2), c] \xrightarrow{(O_1,t_1)} [(q_1', q_2), c]$$

we can deduce that T_1 made a step while the counter c labelling the state didn't change. Therefore, T_2 in state q_2 has a counter labelling its state that is higher than the counter labelling the state in q_1. Alternatively, if

$$[(q_1, q_2), c] \xrightarrow{(O_1,t_1)} [(q_1', q_2), c+1]$$

then the counter at q_2 may become lower than the counter at which T_1 performs the next transition, which means that eventually T_2 has to take a transition.

The composability relation κ in the product of two TES transition systems (see Definition 33) accepts an independent step from T_1 (resp. T_2) if the observation labeling the step relates to the simultaneous silent observation from T_2 (resp. T_1). Given two composable TESs σ and τ respectively in the component behavior of T_1 and T_2, the composability relation $[\kappa]$ must relate heads of such TESs co-inductively. As we do not enforce silent observation to be effective from the product rules (1) and (2), we consider composability relations such that:

- if $((O_1, t_1), (\emptyset, t_1)) \in \kappa(E_1, E_2)$ then $((O_1, t_1), (O_2, t_2)) \in \kappa(E_1, E_2)$ for any $O_2 \subseteq \mathcal{P}(E_2)$ and $t_2 > t_1$; and
- if $((\emptyset, t_2), (O_2, t_2)) \in \kappa(E_1, E_2)$ then $((O_1, t_1), (O_2, t_2)) \in \kappa(E_1, E_2)$ for any $O_1 \subseteq \mathcal{P}(E_1)$ and $t_1 > t_2$

The two rules above encode that an observation from T_1 is independent to T_2 (i.e., $((O_1, t_1), (\emptyset, t_1)) \in \kappa(E_1, E_2)$ if and only if T_1 and T_2 can make observations at difference times (i.e., $((O_1, t_1), (O_2, t_2)) \in \kappa(E_1, E_2)$ for arbitrary (O_2, t_2) from T_2 with $t_2 > t_1$.

Theorem 7 (Correctness). *For two TES transition systems T_1 and T_2, and for κ satisfying the constraint above:*

$$C_{T_1 \times_\kappa T_2}(s) = C_{T_1}(s_1) \times_{([\kappa],[\cup])} C_{T_2}(s_2)$$

with $s_1 = [q_1, c_1] \in (Q_1 \times \mathbb{N})$, $s_2 = [q_2, c_2] \in (Q_2 \times \mathbb{N})$, and $s = [(q_1, q_2), \min(c_1, c_2)]$.

Proof. We first show that, for all $\tau \in C_{T_1 \times_\kappa T_2}((s_1, s_2))$, there is always, eventually, an observations of τ that is a label of a transition constructed by rule (1) or (3) (and (2) or (3)) of the product in Definition 33.

We prove by contradiction. Assume that there is an index n for which for all $k > n$, the observation $\tau(k)$ is generated by the rule (2) of the product. Then, let

$\chi \in (Q \times \mathbb{N})^\omega$ be the sequence of states such that $\chi(k) \xrightarrow{\tau(n+k)} \chi(k+1)$ for $k \in \mathbb{N}$, we can find a step $j \geq 0$ such that, for all $m \geq j$, $\theta(\chi(m)) = \theta(\chi(m+1))$. This contradicts the definition of τ, that the counter in the sequence of states must always eventually increase. Thus, we can conclude that we always eventually take a composite transition from rule (1) or (3) (resp. (2) and (3)) to construct a TES $\tau \in C_{T_1 \times_\kappa T_2}(s_1, s_2)$.

We show by induction that we can construct, from a run $\tau \in \mathcal{L}^{\text{inf}}(T, s_0)$, a run $\tau_1 \in \mathcal{L}^{\text{inf}}(T_1, s_1)$. Let $\chi \in (Q \times \mathbb{N})^\omega$ such that $\chi(n) \xrightarrow{\tau(n)} \chi(n+1)$ and let $\tau_1 \in \mathit{TES}(E_1)$ be the sequence of observations occurring in τ generated by the product rules (1) or (3). We show, by induction, that there exists $\chi_1 \in (Q_1 \times \mathbb{N})^\omega$ such that, for all n, $\chi_1(n) \xrightarrow{\tau_1(n)}_1 \chi_1(n+1)$. Indeed, there exists $k \in \mathbb{N}$ such that $\chi(k) \xrightarrow{\tau(k)} \chi(k+1)$ with $\tau(k)$ being generated by rule (1) or (3). We can therefore define states $\chi_1(0)$ and $\chi_1(1)$ from $\chi(k)$ and $\chi(k+1)$, with a corresponding counter value, and $\tau(0)$ from $\tau(k)$.

Assuming that $\chi_1(k) \xrightarrow{\tau_1(k)} \chi_1(k+1)$ for $k \leq n$, we know there exists a $j \geq n$ such that the transition $\chi(j) \xrightarrow{\tau(j)} \chi(j+1)$ is produced by rules (1) or (3). Therefore, we can construct the next element of the run, namely $\chi_1(n) \xrightarrow{\tau_1(n)} \chi_1(n+1)$. By induction, we conclude that $\tau_1 \in \mathcal{L}^{\text{inf}}(T, \chi(0))$.

Thus, given a run $\tau \in C_{T_1 \times_\kappa T_2}((s_1, s_2))$, we have two runs $\tau_1 \in C_{T_1}(q_1)$ and $\tau_2 \in C_{T_2}(q_2)$ applying symmetric arguments to construct τ_2 such that $\tau = \tau_1[\cup]\tau_2$. We prove, by co-induction, that $(\tau_1, \tau_2) \in [\kappa](E_1, E_2)$.

Let $\Delta_{\tau_1,\tau_2} = \{(\tau_1, \tau_2)^{(n)} \mid n \in \mathbb{N}\}$ be the set of all derivations of (τ_1, τ_2), where, in $(\tau_1, \tau_2)^{(n)}$, the n first observations are dropped from the pair of TESs τ_1 and τ_2. Then, to prove that $\Delta \subseteq [\kappa](E_1, E_2)$, it is sufficient to show that Δ is a post fix point of ϕ_κ, namely that $\Delta \subseteq \phi_\kappa(\Delta)$. We remind that:

$$\phi_\kappa(\Delta) = \{(\sigma, \eta) \mid (\sigma(0), \eta(0)) \in \kappa(E_1, E_2) \land (\sigma, \eta)' \in \Delta\}$$

It is therefore sufficient to prove that $(\sigma, \eta) \in \Delta_{\tau_1,\tau_2}$ implies that $(\sigma(0), \eta(0)) \in \kappa(E_1, E_2)$ in order to conclude that $\Delta_{\tau_1,\tau_2} \subseteq \phi_\kappa(\Delta_{\tau_1,\tau_2})$. This is directly implied by the three rules of the product of T_1 and T_2 constructing τ, and the conditions imposed on κ.

Next, we show the reciprocal: for two κ-composable TESs τ_1 and τ_2, respectively in $C_{T_1}(s_1)$ and $C_{T_2}(s_2)$, $\tau_1[\cup]\tau_2$ is a run of the product $C_{T_1 \times_\kappa T_2}((s_1, s_2))$. We consider the two runs with sequence of states $\chi_1 \in (Q_1 \times \mathbb{N})^\omega$ and $\chi_2 \in (Q_1 \times \mathbb{N})^\omega$, for τ_1 and τ_2 respectively. Then, we show that there exists a sequence of states $\chi \in ((Q_1 \times Q_2) \times \mathbb{N})^\omega$, whose transitions are labeled by τ, and resulting from the composition of the two sequences of states. Moreover, since the counter of χ_1 and χ_2 are always eventually increasing, the counter of χ is also always eventually increasing. Thus, $\tau \in C_{T_1 \times_\kappa T_2}(\chi(0))$. □

Remark 15 (Fairness). *Fairness, in our case is the property that, in a product of two TESs $T_1 \times_\kappa T_2$, then always, eventually, T_1 and T_2 each make progress. The definition of the product of two TES transition systems defines the counter value of the composite state as the minimal counter value from the two compound states. The semantic condition that considers runs with a counter value always eventually increasing is sufficient for having T_1 and T_2 to always eventually take a step, as shown in Theorem 7.*

We give in Example 49 the TES transition systems resulting from the product of the TES transition systems of two robots and a grid. Example 49 defines operationally the components in Section 2.2, i.e., their behavior is generated by a TES transition system.

Example 49. *Let T_{R_1}, T_{R_2} be two TES transition systems for robots R_1 and R_2, and let $T_G(\{1\}, n, m)$ be a grid with robot R_1 alone and $T_G(\{1,2\}, n, m)$ be a grid with robots R_1 and R_2. We use κ^{sync} as defined in Example 11.*

The product of T_{R_1}, T_{R_2}, and $T_G(\{1,2\}, n, m)$ under κ^{sync} is the TES transition system $T_{R_1} \times_{\kappa^{sync}} T_{R_2} \times_{\kappa^{sync}} T_G(\{1,2\}, n, m)$ such that it synchronizes observations of the two robots with the grid, but does not synchronize events of the two robots directly, since the two set of events are disjoint. ■

As a consequence of Theorem 1, letting κ^{sync} be the composability relation used in the product $\bowtie$ and writing $T = T_{R_1} \times_{\kappa^{sync}} T_{R_2} \times_{\kappa^{sync}} T_G$, $C_T(s_1, s_2, s_3)$ is equal to the component $C_{T_{R_1}}(s_1) \bowtie C_{T_{R_2}}(s_2) \bowtie C_{T_G}(s_3)$

Definition 34. *Let T be a TES transition system, and let $C_T(q) = (E, \mathcal{L}^{\inf}(T, s))$ be a component whose behavior is defined by T. Then, C is* deadlock free *if and only if $FG(\mathcal{L}^{\inf}(T, s)) = \mathcal{L}^{\text{fin}}(T, s) \neq \emptyset$. As a consequence, we also say that (T, s) is deadlock free when $C_T(s)$ is deadlock free.*

A class of deadlock free components is that of components that accept arbitrary insertion of $\emptyset$ observables in between two observations. We say that such component is *prefix-closed*, as every sequence of finite observations can be continued by an infinite sequence of empty observables, i.e., C is such that $C = C^*$ (as defined after Definition 33). We say that a TES transition system T is prefix-closed in state s if and only if and only if $C_T(s) = C_T^*(s)$. For instance, if T is such that, for any state s and for any $t \in \mathbb{R}_+$ there is a transition $s \xrightarrow{(\emptyset, t)} s$, then T is prefix-closed.

Lemma 19. *If T_1 and T_2 are prefix-closed in s_1 and s_2 respectively, then $T_1 \times_{\kappa^{sync}} T_2((s_1, s_2))$ is prefix-closed.*

Proof. The proof follows from the fact that $\emptyset$ is independent with any non-empty observable $O \subseteq E_1 \cup E_2$. Then, any pair of silent observation is composable, and therefore the following TES transition system is prefix-closed. □

We search for the condition under which deadlock freedom is preserved under a product. Section 3.3 gives a condition for the product of two deadlock free components to be deadlock free.

4.1.2 Compatibility of TES transition systems

Informally, the condition of κ-compatibility of two TES transition systems T_1 and T_2, respectively in initial state s_{01} and s_{02}, describes the existence of a relation $\mathcal{R}$ on pairs of states of T_1 and T_2 such that $(s_{01}, s_{02}) \in \mathcal{R}$ and for every state $(s_1, s_2) \in \mathcal{R}$, there exists an outgoing transition from T_1 (reciprocally T_2) that composes under κ with an outgoing transition of T_1 (respectively T_2). The pair of outgoing states is in the relation $\mathcal{R}$.

Formally, a TES transition system $T_1 = (Q_1, E_1, \rightarrow_1)$ from state s_{01} is κ-compatible with a TES transition system $T_2 = (Q_2, E_2, \rightarrow_2)$ from state s_{02}, and we say (T_1, s_{01}) is κ-compatible with (T_2, s_{02}) if there exists a relation $\mathcal{R} \subseteq (Q_1 \times \mathbb{N}) \times (Q_2 \times \mathbb{N})$ such that $(s_{01}, s_{02}) \in \mathcal{R}$ and for any $(s_1, s_2) \in \mathcal{R}$,

- there exist $s_1 \xrightarrow{(O_1,t_1)}_1 s_1'$ and $s_2 \xrightarrow{(O_2,t_2)}_2 s_2'$ such that $((O_1, t_1), (O_2, t_2)) \in \kappa(E_1, E_2)$; and
- for all $s_1 \xrightarrow{(O_1,t_1)}_1 s_1'$ and $s_2 \xrightarrow{(O_2,t_2)}_2 s_2'$ if $((O_1, t_1), (O_2, t_2)) \in \kappa(E_1, E_2)$ then $(u_1, u_2) \in \mathcal{R}$, where $u_i = s_i$ if $t_i = \min\{t_1, t_2\}$, and $u_i = s_i'$ otherwise for $i \in \{1, 2\}$.

In other words, if (T_1, s_1) is κ-compatible with (T_2, s_2), then there exists a composable pair of transitions in T_1 and T_2 from each pair of states in $\mathcal{R}$ (first item of the definition), and all pairs of transitions in T_1 composable with a transition in T_2 from a state in $\mathcal{R}$ end in a pair of states related by $\mathcal{R}$. If (T_2, s_2) is κ-compatible to (T_1, s_1) as well, then we say that (T_1, s_1) and (T_2, s_2) are κ-compatible.

Theorem 8 (Deadlock free)**.** *Let (T_1, s_1) and (T_2, s_2) be κ-compatible. Let $C_{T_1}(s_1)$ and $C_{T_2}(s_2)$ be deadlock free, as defined in Definition 34. Then, $C_{T_1}(s_1) \times_{([\kappa],[\cup])} C_{T_2}(s_2)$ is deadlock free.*

Proof. We reason by contradiction. If the product $C_{T_1}(s_1) \times_{([\kappa],[\cup])} C_{T_2}(s_2)$ is not deadlock free, then $\mathcal{L}^{\text{fin}}(T_1 \times_\kappa T_2, (s_1, s_2)) \neq FG(\mathcal{L}^{\text{inf}}(T_1 \times_\kappa T_2, (s_1, s_2)))$. Thus, there exists a state (s_1', s_2'), reachable from (s_1, s_2), such that $\mathcal{L}^{\text{fin}}(T_1 \times_\kappa T_2, (s_1', s_2')) = \emptyset$, i.e., no pairs of compatible transitions from T_1 and T_2 in states s_1' and s_2' respectively. Given the fact that both TES transition systems are deadlock free, and given that s_1' (respectively s_2') is reachable from s_1 (respectively s_2) for T_1 (respectively T_2), then $\mathcal{L}^{\text{fin}}(T_2, s_1) \neq \emptyset$ and $\mathcal{L}^{\text{fin}}(T_1, s_2) \neq \emptyset$.

Since (T_1, s_1) and (T_2, s_2) are κ-compatible, then there exists $\mathcal{R}$ such that for each pair $(s_1', s_2') \in \mathcal{R}$, there exists an outgoing transition in T_1 and T_2 from s_1' and s_2' respectively that is composable under κ. Such property would imply that there is a transition in $T_1 \times_\kappa T_2$ from state (s_1', s_2') and therefore $\mathcal{L}^{\text{fin}}(T_1 \times_\kappa T_2, (s_1', s_2')) \neq \emptyset$. In other words, the property of compatibility contradicts the presence of deadlock in the product $C_{T_1}(s_1) \times_{([\kappa],[\cup])} C_{T_2}(s_2)$. □

In general however, κ-compatibility is not preserved over product, demonstrated by Example 50. For the case of coordinated cyber-physical systems, components are usually not prefix-closed as there might be some timing constraints or some mandatory actions to perform in a bounded time frame.

Example 50. *Suppose three TES transition systems $T_i = (\{q_i\}, \{a, b, c, d\}, \rightarrow_i)$, with $i \in \{1, 2, 3\}$, defined as follow for all $n \in \mathbb{N}$:*

- $q_1 \xrightarrow{(\{a,b\},n)}_1 q_1$ *and* $q_1 \xrightarrow{(\{a,c\},n)}_1 q_1$;
- $q_2 \xrightarrow{(\{a,c\},n)}_2 q_2$ *and* $q_2 \xrightarrow{(\{a,d\},n)}_2 q_2$;
- $q_3 \xrightarrow{(\{a,d\},n)}_3 q_3$ *and* $q_3 \xrightarrow{(\{a,b\},n)}_3 q_3$.

It is easy to show that $T_1(q_1)$, $T_2(q_2)$, and $T_3(q_3)$ are pairwise κ^{sync}-compatible. However, $T_1(q_1)$ is not κ^{sync}-compatible with $T_2(q_2) \times_{\kappa^{sync}} T_3(q_3)$. ■

Lemma 20. *Let $\times_\kappa$ be commutative and associative, and for arbitrary E_1, E_2, and $t \in \mathbb{R}_+$, then $((\emptyset, t), (\emptyset, t)) \in \kappa(E_1, E_2)$. Let S be a set of TES transition systems, such that for $T \in S$ and every state $[q, n]$ in T, then $[q, n] \xrightarrow{(\emptyset,t)} [q, n]$. For $S = S_1 \uplus S_2$ a partition of S, $\times_\kappa\{T\}_{T \in S_1}$ and $\times_\kappa\{T\}_{T \in S_2}$ are κ-compatible and the component $C_{\times_\kappa\{T\}_{T\in S}}$ is deadlock free.*

Proof. □

The consequence of two TES transition systems T_1 and T_2 to be κ-compatible on (s_1, s_2) and deadlock free, is that they can be run *step-by-step* from (s_1, s_2) and ensure that we would not generate a sequence of observations that is not a prefix on an infinite run. However, there is still an obligation for the *step-by-step* execution to produce a run that is in the behavior of the product, i.e., to perform a step-by-step product at runtime. Indeed, the resulting sequence of states must always increase the counter value, which means that the selection of a step must be *fair* (as introduced in Remark 15). We show in Example 51 an example for which an infinite sequence of transitions in the product (e.g., produced by a step-by-step implementation of the product) would not give a run, due to fairness violation.

Example 51. *Let* $T_1 = (\{q_1\}, \{a\}, \rightarrow_1)$ *and* $T_2 = (\{q_2\}, \{b\}, \rightarrow_2)$ *be two TES transition systems such that:* $[q_1, c] \xrightarrow{(\{a\},t)}_1 [q_1, c+1]$ *and* $[q_2, c] \xrightarrow{(\{b\},t)}_2 [q_2, c+1]$*for all* $t \in \mathbb{R}_+$ *and all* $c \in \mathbb{N}$*. Let* κ *be such that* $((\{a\},t),(\emptyset,t)) \in \kappa(\{a\},\{b\})$ *and* $((\emptyset,t),(\{b\},t)) \in \kappa(\{a\},\{b\})$*. Then, the product* $T_1 \times_\kappa T_2$ *has the composite transitions* $[(q_1,q_2),c] \xrightarrow{(\{a\},t)} [(q_1,q_2),c]$ *and* $[(q_1,q_2),c] \xrightarrow{(\{b\},t)} [(q_1,q_2'),c]$ *for all* $c \in \mathbb{N}$ *and* $t \in \mathbb{R}_+$*.*
The product, therefore has runs of the kind $[(q_1,q_2),c] \xrightarrow{(\{a\},t_i)} [(q_1',q_2),c]$ *where for all* $i \in \mathbb{N}$*,* $c_i + 1 = c_{i+1}$ *and* $t_i < t_{i+1}$ *(increasing) and there exists* $j \in \mathbb{N}$ *with* $i < t_j$ *(non-Zeno). Thus, this run does only transitions from* T_1 *and none from* T_2*: there is a step for which the counter c does not increase anymore. One reason is that rule* (1) *of the product is always chosen. Instead, by imposing that we always eventually take rule* (3)*, we ensure that the step-by-step product is fair.*

We consider a class of TES transition systems for which a *step-by-step* implementation of their product is fair, i.e., always eventually the counter of the composite state increases. More particularly, we consider TES transition systems that always eventually require synchronization. Therefore, the product always eventually performs rule (3), and the runs are consequently fair. Such property is a composite property, that can be obtained compositionally by imposing a trace property on a TES transition system, such as: for every trace, there is always eventually a state for which all outgoing transitions must synchronize with an observation from the other TES transition system.

Remark 16. *In the actor model, fairness is usually defined as an individual property: always eventually an action that is enabled (such as reading a message in a queue) will be performed. This notion of fairness differs from the one we introduced for TES transition systems. Here, fairness models a collective property, namely that each component always eventually makes an observation.*

Definition 35 (k-synchronizing). *Two TES transition systems T_1 and T_2 are k-synchronizing under κ if all sequences of k transitions in the product $T_1 \times_\kappa T_2$ contain at least one transition constructed from rule* (3) *of the product in Definition 33.*

Lemma 21. *Let T_1 and T_2 be two k-synchronizing TES transition systems. Then, a step-by-step execution of the product $T_1 \times_\kappa T_2$ is fair, namely, all finite sequences of transitions are prefix of infinite runs in the product behavior, i.e., $FG(\mathcal{L}^{\text{inf}}(T_1 \times_\kappa T_2, q)) = \mathcal{L}^{\text{fin}}(T_1 \times_\kappa T_2, q)$.*

Proof. The inclusion $FG(\mathcal{L}^{\text{inf}}(T_1 \times_\kappa T_2, q)) \subseteq \mathcal{L}^{\text{fin}}(T_1 \times_\kappa T_2, q)$ is straightforward, as a finite prefix of a TES labeling an infinite run is, by definition, a finite sequence of labels of a finite sequence of transitions in $\mathcal{L}^{\text{fin}}(T_1 \times_\kappa T_2, q)$.
The inclusion $\mathcal{L}^{\text{fin}}(T_1 \times_\kappa T_2, q) \subseteq FG(\mathcal{L}^{\text{inf}}(T_1 \times_\kappa T_2, q))$ comes from the assumption that T_1 and T_2 are k-synchronizing under κ. Then, for any finite sequence in $\mathcal{L}^{\text{fin}}(T_1 \times_\kappa T_2, q)$, the post state on which the sequence ends has, due to the assumption, a continuation as a run in $\mathcal{L}^{\text{inf}}(T_1 \times_\kappa T_2, q)$, which proves the inclusion. $\square$

Remark 17. *The step-by-step implementation of the product is sound if TES transition systems always eventually synchronize on a transition. Definition 35 and Lemma 21 show that if two TES transition systems are k-synchronizing, then their product can be formed lazily, step-by-step, at runtime.*

4.2 Components as rewrite systems

We start by giving an illustration of 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 actions 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 section 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 goals? If not, can the two robots, without changing their policy, be externally coordinated towards their goals?

In this section, 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 set of agents that run concurrently. The equational theory of an agent defines how the agent states are updated, and may exhibit both continuous and discrete transformations. The dynamics 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 (i.e., 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 dynamics of both agents is captured by a rewrite rule of the form:

$$(s, \emptyset) \Rightarrow (s', \mathit{acts})$$

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

Example 52 (Battery)**.** *A battery is characterized by a set of internal physical laws that describe the evolution of its energy profile over time under external stimulations. We consider three external stimuli for the battery as three events: a charge, a discharge, and a read event. Each of those events may change the profile of the battery, and we assume that in between two events, the battery energy follows some fixed internal laws. Formally, we model the energy profile of a battery as a function $f : \mathbb{R}_+ \to [0, 100\%]$ where $f(t) = 50\%$ means that the charge of the battery at time t is of 50%. In general, the co-domain of f may be arbitrarily complex, and captures the response of event occurrences (e.g., charge, discharge, read) and passage of time coherently with the underlying laws (e.g., differential equation). For instance, a charge (or discharge) event at a time t coincides with a change of slope in the function f after time t and before the next event occurrence.*
For simplicity, we consider a battery for which f is piecewise linear in between any two events. The slope changes according to some internal laws at points where the battery is used for charge or discharge.
In our model, a battery interacts with its environment only at discrete time points. Therefore, we model the observables of a battery as a function $l : \mathbb{N} \to [0, 100\%]$ that intuitively samples the state of the battery at some monotonically increasing and

non-Zeno sequence of timestamp values. We capture, in Definition 1, the continuous profile of a battery as a component whose behavior contains all of such increasing and non-Zeno sampling sequences for all continuous functions f.

Example 53 (Robot)**.** *A robot's state contains the previously read values of its sensors. Based on its state, a robot decides to move in some specific direction or read its sensors.*
Similarly to the battery, we assume that a robot acts periodically at some discrete points in time, such as the sequence $move(E)$ (i.e., moving East) at time 0, $read((x, y), l)$ (i.e., reading the position (x, y) and the battery level l) at time T, $move(W)$ (i.e., moving West) at time $3T$ while doing nothing at time $2T$, etc. The action may have as effect to change the robot's state: typically, the action $read((x, y), l)$ updates the state of the robot with the coordinate (x, y) and the battery value l.

System A system is a set of agents together with a composability constraint κ that restricts 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:

$$\{(s_{\text{id}}, acts_{\text{id}}), (s_F, acts_F)\} \Rightarrow_S \{(\phi_{\text{id}}(s_{\text{id}}, a_{\text{id}}), \emptyset), (\phi_F(s_F, a_F), \emptyset)\}$$

where $a_{\text{id}} \in acts_{\text{id}}$ and $a_F \in acts_F$ are two actions related by κ.

Each agent is unaware of the other agent's decisions. The system rewrite $\Rightarrow_S$ filters actions that do not comply with the composability relation κ. As a result, each agent updates its state with the (possibly composite) action chosen at runtime, from the list of its 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 question on the system: will robot id eventually reach the location (x, y) on the field? Note that the agent alone cannot answer the query, as the answer depends on the characteristics of the field.

Example 54 (Battery-Robot)**.** *Typically, a move of the robot* synchronizes *with a change of state in the battery, and a read of the robot occurs at the same time as a sampling of the battery value.*
The system behavior therefore consists of sequences of simultaneous events occurring between the battery and the robot. By composition, the battery exposes the subset of its behavior that conforms to the specific frequency of read and move actions of the

robot. The openness of the battery therefore is reflected by its capacity to adapt to any observation frequency.

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 on the field at the same time. It is therefore possible that the system deadlocks if no actions are composable, or livelocks if the robots enter an infinite sequence of repeated moves.

We add a protocol agent to the system, which imposes some coordination constraints 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 $\{(s_{\mathrm{id}_1}, acts_{\mathrm{id}_1}), (s_{\mathrm{id}_2}, acts_{\mathrm{id}_2}), (s_F, acts_F), (s_P, acts_P)\}$ the run of robots id_1 and id_2 has to agree with the observations of the protocol, and the sequence of actions for each robot will therefore be conform to a permissible sequence under the protocol.

In the case where the two robots enter a livelock and eventually run out of energy, we show in Section 5.4.1 the possibility of using a protocol to remove such behavior.

Example 55 (Safety property)**.** *A safety property is typically a set of traces for which nothing bad happens. In our framework, we consider only observable behaviors, and a safety property therefore declares that nothing bad* is observable*. However, it is not sufficient for a system to satisfy a safety property to conclude that it is safe: an observation that would make a sequence violate the safety property may be absent, not because it did not actually happen, but merely because the system missed to detect it. For example, consider a product of a battery component and a robot with a sampling period T, as introduced in Example 54. Consider the safety property:* the battery energy is between the energy thresholds e_1 and e_2*. The resulting system may exhibit observations with energy readings between the two thresholds only, and therefore satisfy the property. However, had the robot used a smaller sampling period $T' = T/2$, which adds a reading observation of its battery between every two observations, we may have been able to detect that the system is not safe because it produces sequences at this finer granularity sampling rate that violate the safety property. We show how to algebraically capture the safety of a system constituted of a battery-robot.*

4.2.1 System of agents and compositional semantics

Components in Chapter 2 are declarative. Their behavior consists of a set of TESs. The abstraction of internal states in components makes the specification of observables and their interaction easier. The downside of such declarative specification lies in the difficulty of generating an element from the behavior, and ultimately verifying properties on a product expression.

An operational specification of a component provides a mechanism to construct elements in its behavior. An *agent* is the operational specification that produces finite sequences of observations that, in the limit, determine the behavior of a component. An agent is stateful, and has transitions between states, each labeled by an observation, i.e., a set of events with a time-stamp. We consider a finite specification of an agent as a rewrite theory, where finite applications of the agent's rewrite rules generate a sequence of observables that form a prefix of some elements in the behavior of its corresponding component. We restrict the current work to integer time labeled observations. While in the cyber-physical world, time is a real quantity, we consider in our fragment a countable infinite domain for time, i.e., natural numbers. The time interval between two tics is therefore the same for all agents, and may be interpreted as, e.g., seconds, milliseconds, femtoseconds, etc. We show how an agent may synchronize with a local clock that forbids actions at some time values, thus modeling different execution speeds.

An operational specification of a composite component provides a mechanism to construct elements in the behavior of a product expression. The product on components is parametrized by an interaction signature that tells *which* TESs can compose, and *how* they compose to a new TES. We consider, in the operational fragment of this section, interaction signatures each of whose composability relation is co-inductively defined from a relation on observations κ. Intuitively, such restriction enables a step-by-step operation to check that the head of each sequence is valid, i.e., extends the sequence to be a prefix of some elements in the composite component. Moreover, we require κ to be such that the product on component $\times_{([\kappa],\cup)}$ is commutative and associative (see [62]). By *system* we mean a set of agents that compose under some interaction signature $\Sigma = ([\kappa], \cup)$. A system is stateful, where each state is formed from the states of its component agents, and has transitions between states, each labeled by an observation, formed from the component agent observations. We consider a finite specification of a system as the composition of a set of rewriting theories (one for each agent), and a system rewrite rule that produces a composite observation complying with the relation κ. We prove compositionality: the system component is equal to the product under the interaction signature $\Sigma = ([\kappa], \cup)$ of every one of its constituent agent components.

We give the operational counterparts of an observation, a component, and a product of components as, respectively, an action, an agent, and a system of agents.

Action Actions are terms of sort Action. An action has a name of sort AName and some parameters. We distinguish two typical actions, the idle action $\star$ and the ending action end. A term of sort Action corresponds to an observable, i.e., a set of events. The idle action $\star$ and the ending action end both map to the empty set of events. An example of an action is $\mathsf{move(R1,d)}$ or $\mathsf{read(R1,\ position,\ l)}$ that, respectively, moves agent $\mathsf{R1}$ in direction d or reads the value l from the position sensor of $\mathsf{R1}$. The semantics of action $\mathsf{move(R1,\ d)}$ consists of all singleton events of the form $\{\mathsf{move(R1,\ d)}\}$ with d a constant direction value. We use the associative, commutative, and idempotent operation $\cdot : \mathsf{Action}\ \mathsf{Action} \to \mathsf{Action}$ to construct a composite action $\mathsf{a1} \cdot \mathsf{a2}$ out of two actions $\mathsf{a1}$ and $\mathsf{a2}$.

Agent An agent operationally specifies a component in rewriting logic. We give the specification of an agent as a rewrite theory, and provide the semantics of an agent as a component. An agent is a four tuple $(\Lambda, \Omega, \mathcal{E}, \Rightarrow)$, each of whose elements we introduce as follow.

The set of sorts Λ contains the State sort and the Action sort, respectively for state and action terms. A pair of a state and a set of actions is called a configuration. The set of function symbols Ω contains $\phi : \mathsf{State} \times \mathsf{Action} \to \mathsf{State}$, that takes a pair of a state and an action term to produce a new state. The (Λ, Ω)-equational theory $\mathcal{E}$ specifies the update function ϕ. The set of equations that specify the function ϕ can make ϕ either a continuous or discrete function.

The rule pattern in (4.1) updates a configuration with an empty set to a new configuration, i.e.,

$$(s, \emptyset) \Rightarrow (s', acts) \tag{4.1}$$

with $acts$ a non-empty set of action terms, and s' a new state. We call an agent *productive* if, for any state $s : \mathsf{State}$, there exists a state s' with $(s, \emptyset) \Rightarrow (s', acts)$ and $acts$ non empty set.

We give a semantics of an agent as a component by considering the limit application of the agent rewrite rules. We construct a TES transition system $\mathcal{T}_{\mathcal{A}} = (Q, E, \to)$ as an intermediate representation for agent $\mathcal{A} = (\Lambda, \Omega, \mathcal{E}, \Rightarrow)$. The set of states $Q = \mathsf{State} \times \mathbb{N}$

is the set of pairs of a state of $\mathcal{A}$ and a time-stamp natural number. We use the notation $[s,t]$ for states in Q where $t \in \mathbb{N}$. The set of events E is the union of all observables labeling the transition relation $\rightarrow \subseteq Q \times (\mathcal{P}(E) \times \mathbb{N}) \times Q$, defined as the smallest set such that, for $t \in \mathbb{N}$ and $n \in \mathbb{N}$:

$$\frac{(s,\emptyset) \Rightarrow (s', acts) \qquad a \in acts \qquad \phi(s',a) =_{\mathcal{E}} s'' \qquad d \in \mathbb{N}}{[s,n] \xrightarrow{(a,t)} [s'', n+1]} \tag{4.2}$$

$$\frac{}{[s,n] \xrightarrow{(\emptyset,t)} [s, n+1]}\ [owise] \tag{4.3}$$

An agent that performs a rewrite moves the global time from an arbitrary but finite amount of time units. Note that we can safely consider $d \neq 0$, as the case of two consecuitive observations with the same time stamp is ruled out in the TES behavior of an agent (see below). All agents share the same time semantically, and we show some mechanisms at the system level to artificially run some agents *faster* than others.

Let $\mathcal{A} = (\Lambda, \Omega, \mathcal{E}, \Rightarrow)$ be an agent initially in state $s_0 \in S$ at time $t_0 \in \mathbb{N}$. The component semantics of $\mathcal{A}$ is the component $[\![\mathcal{A}([s_0,t_0])]\!] = C_{\mathcal{T}_\mathcal{A}}([s_0,t_0])$, with $C_{\mathcal{T}_\mathcal{A}}([s_0,t_0])$ defined in Section 4.1.1.

Remark 18. *An agent $\mathcal{A}$ initially in state s_0 at t_0 denotes a component $[\![\mathcal{A}([s_0,t_0])]\!]$. Note that, a strategy for agent $\mathcal{A}$ would be any mechanism that, given a state for agent $\mathcal{A}$, filters a subset of possible actions. For instance, an agent may decide to discard actions that bring it further from its goal. We give in Section 4.3 a Domain Specific Language to describe agents equipped with a strategy.*

System A system gives an operational specification of a product of a set of components under $\Sigma = ([\kappa], \cup)$. The composability relation κ is fixed to be symmetric, so that the product $\times_\Sigma$ is commutative. We define $[\kappa]$ co-inductively, as in [62, 61]. Formally, a system consists of a set of agents with additional sorts, operations, and rewrite rules. A system is a tuple $(\mathcal{A}, \Lambda, \Omega, \mathcal{E}, \Rightarrow_S)$ where $\mathcal{A}$ is a set of agents. We use $(\Lambda_i, \Omega_i, \mathcal{E}_i, \Rightarrow_i)$ to refer to agent $\mathcal{A}_i \in \mathcal{A}$.

The set of sorts Λ contains a sort $\mathsf{Action} \in \Lambda$ which is a super sort of each sort Action_i for $\mathcal{A}_i \in \mathcal{A}$. The set Ω contains the function symbol $\mathsf{comp} : \mathcal{P}(\mathsf{Action}) \rightarrow \mathsf{Bool}$, which says which set of actions are *composable.* We call a set `actions` of actions for which `comp(actions)` holds, a *clique.* The conditions for a set of actions to form a clique models the fact that each action in the clique is independent from agent $\mathcal{A}_i$ with no action in that clique (see Chapter 5 for an instance of comp). The relation comp can be graphically modelled as an undirected graph relating actions, where a clique is a connected component.

The rewrite rule pattern in (4.4) selects a set of actions, at most one from each agent, checks that the set of actions forms a clique with respect to comp, and applies the update accordingly. For $\{k_1, ..., k_j\} \subseteq \{1, ..., n\}$:

$$\{(s_{k_1}, acts_{k_1}), ..., (s_{k_j}, acts_{k_j})\} \Rightarrow_S \{(\phi_{k_1}(s_{k_1}, a_{k_1}), \emptyset), ..., (\phi_{k_j}(s_{k_j}, a_{k_j}), \emptyset)\} \tag{4.4}$$

if $\mathsf{comp}(\bigcup_{i\in[1,j]}\{a_{k_i}\})$ and $a_{k_i} \in acts_{k_i}$. As we show later, a system does not necessarily update all agents in lock steps, and an agent not doing an action may stay in the configuration $(s, \emptyset)$. As multiple cliques may be possible, there is non-determinism at the system level. Different strategies may therefore choose different cliques as, for instance, taking the largest clique.

We define the transition system for $\mathcal{S} = (\mathcal{A}, \Lambda, \Omega, \mathcal{E}, \Rightarrow_S)$ as the TES transition system $\mathcal{T}_\mathcal{S} = (Q, E, \rightarrow)$ with $Q = \mathsf{StateSet} \times \mathbb{N}$ the set of states, E the union of all observables labeling the transition relation $\rightarrow \subseteq Q \times (\mathcal{P}(E) \times \mathbb{N}) \times Q$, which is the smallest transition relation such that, for $\{k_1, ..., k_j\} \subseteq \{1, ..., n\}$:

$$\frac{\{(s_{k_i}, acts_{k_i})\}_{i\in[1,j]} \Rightarrow_S \{(\phi_{k_i}(s_{k_i}, a_{k_i}), \emptyset)\}_{i\in[1,j]} \quad \bigwedge_{i\in[1,j]} \phi_{k_i}(s_{k_i}, a_{k_i}) =_{\mathcal{E}_i} s''_{k_i}}{[\{s_i\}_{i\in[1,n]}, n] \xrightarrow{(\bigcup_{i\in[1,j]} a_{k_i}, t)} [\{s_1, ..., s''_{k_1}, ..., s''_{k_j}, ..., s_n\}, n+1]} \tag{4.5}$$

$$\frac{}{[s, n] \xrightarrow{(\emptyset, t)} [s, n+1]}\ [owise] \tag{4.6}$$

for $t \in \mathbb{N}$ and where we use the notation $\{x_i\}_{i\in[1,n]}$ for the set $\{x_1, ..., x_n\}$.

Remark 19. *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 (bottom of the rule) are sets of states and sets of labels from the TES transition system of every agent in the system.*

Let $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ be a set of agents, and let $\mathcal{S} = (\mathcal{A}, \Lambda, \Omega, \mathcal{E}, \Rightarrow_S)$ be a system initially in state $\{(s_{0i}, \emptyset)\}_{i\in[1,n]}$ at time t_0 such that, for all $i \in [1, n]$, $\mathcal{A}_i$ is initially in state s_{0i} at time t_0. The infinite semantics of initialized system $\mathcal{S}([s_0, t_0])$ is the component $[\![\mathcal{S}([s_0, t_0])]\!] = C_{\mathcal{T}_\mathcal{S}}([s_0, t_0])$, with $C_{\mathcal{T}_\mathcal{S}}([s_0, t_0])$ defined as in Section 4.1.1.

The composability relation comp is an n-ary relation on sets of actions, while the interaction signature $\Sigma = ([\kappa_{\mathsf{comp}}], \cup)$ contains a binary composability relation κ_{comp} on pair of actions. We define κ_{comp} from $\mathtt{comp}$ to be such that, for $\mathtt{O}$ a set of actions if $\mathtt{comp}(\mathtt{O})$, then, for all $n \in \mathbb{N}$, we have:

1. $((O \cap E_1, n), (O \cap E_2, n)) \in \kappa_{\mathsf{comp}}(E_1, E_2)$ with E_1 and E_2 interfaces of different agents, i.e., two composable (sets of) actions occur at the same time;

2. κ_{comp} satisfies the axiom for associativity:

$$\begin{aligned} &((O_1, n), (O_2, n)) \in \kappa(I_1, I_2) \wedge ((O_1 \cup O_2, n), (O_3, n)) \in \kappa(I_1 \cup I_2, I_3) \\ \iff &((O_2, n), (O_3, n)) \in \kappa(I_2, I_3) \wedge ((O_1, n), (O_2 \cup O_3, n)) \in \kappa(I_1, I_2 \cup I_3) \end{aligned}$$

   for arbitrary I_1, I_2, I_3

Note that the cases where $E_1 \cap O = \emptyset$ (or $E_2 \cap O = \emptyset$) model independent progress from agents with interface in E_1 (or E_2). Then, if κ_{comp} relates empty observations, the composability relation allows independent progress when used in the product of TES transition systems (see rules (1) and (2) in Section 4.1.1).

Lemma 22 (Composability). *If* $\mathtt{Action_i} \cap \mathtt{Action_j} = \emptyset$ *for all disjoint agents i and j, then the product* $\times_{([\kappa_{\mathsf{comp}}], \cup)}$ *is commutative and associative.*

Proof. By definition of $\mathtt{comp}$ and item (1) in definition of κ_{comp}, we have that κ_{comp} is symmetric, and therefore $\times_{([\kappa_{\mathsf{comp}}], \cup)}$ is commutative. By item (2) in definition of κ_{comp}, the asumptions of Lemma 10 are satisfied and $\times_{([\kappa_{\mathsf{comp}}], \cup)}$ is associative. $\square$

Theorem 9 (Compositional semantics). *Let* $\mathcal{S} = (\mathcal{A}, \Lambda, \Omega, \mathcal{E}, \Rightarrow_S)$ *be a system of n agents with disjoint actions and* $[\{s_{01}, ..., s_{0n}\}, t_0]$ *as initial state. We fix* $\Sigma = ([\kappa_{\mathsf{comp}}], \cup)$. *Then,* $[\![\mathcal{S}([s_0, t_0])]\!] = \times_\Sigma \{[\![\mathcal{A}_i([s_{0i}, t_0])]\!]\}_{i \in [1,n]}$.

Proof. The proof uses the result of Lemma 22 that $\times_{([\kappa_{\mathsf{comp}}], \cup)}$ is associative and commutative. Then, we give an inductive proof that $[\![\mathcal{S}([s_0, t_0])]\!] = \times_\Sigma \{[\![\mathcal{A}_i([s_{0i}, t_0])]\!]\}_{i \in [1,n]}$. We fix $\mathcal{S} = (\{\mathcal{A}_1, ..., \mathcal{A}_n\}, \Lambda, \Omega, \mathcal{E}, \Rightarrow_S)$ and $\mathcal{A}_{n+1} = (\Lambda_{n+1}, \Omega_{n+1}, \mathcal{E}_{n+1}, \Rightarrow_{n+1})$, such that $\mathtt{comp}$ in Ω relates actions of agents in $\{\mathcal{A}_1, ..., \mathcal{A}_{n+1}\}$.
Let $\mathcal{S}' = (\{\mathcal{A}_1, ..., \mathcal{A}_n, \mathcal{A}_{n+1}\}, \Lambda, \Omega, \Rightarrow_S)$. We show that $\mathcal{T}_{\mathcal{S}} \times_\kappa \mathcal{T}_{\mathcal{A}_{n+1}} = (Q, E, \rightarrow)$ and $\mathcal{T}_{\mathcal{S}'} = (Q', E', \rightarrow')$ have the same component semantics, namely that a run is in $\mathcal{T}_{\mathcal{S}} \times_\kappa \mathcal{T}_{\mathcal{A}_{n+1}}$ if and only if it is in $\mathcal{T}_{\mathcal{S}'}$.

By construction of the system $\mathcal{S}'$, every run in $\mathcal{T}_{\mathcal{S}'}$ is also a run in $\mathcal{T}_{\mathcal{S}} \times_\kappa \mathcal{T}_{\mathcal{A}_{n+1}}$. Indeed, for a set of composable actions O with $\mathsf{comp}(\mathsf{O})$ that the system performs, we know that all agents that have an action in O will take a transition labeled with that action, and all agents that have no action in O will do a silent transition. Thus, there is a run in $\mathcal{T}_{\mathcal{S}} \times_{\kappa} \mathcal{T}_{\mathcal{A}_{n+1}}$ that has the same sequence of observations as runs in $\mathcal{S}'$.

Alternatively, every run in $\mathcal{T}_{\mathcal{S}} \times_{\kappa} \mathcal{T}_{\mathcal{A}_{n+1}}$ also corresponds to a run in $\mathcal{T}_{\mathcal{S}'}$, with the counter increasing at each step.

As a result, by associativity and commutativity of $\times_{\Sigma}$, we conclude that $[\![\mathcal{S}([s_0, t_0])]\!] = \times_{\Sigma}\{[\![\mathcal{A}_i([s_{0i}, t_0])]\!]\}_{i\in[1,n]}$. $\square$

Remark 20. *A sufficient criteria for a sound step-by-step implementation of a system of interacting agents is to prove that the agents always eventually synchronize within a bounded number of transitions. This way, using the result of Lemma 21, we can find k as the largest of such steps, and prove that the agents are k-synchronizing. In this case, we can show that the agents are 1-synchronizing, as every agent either takes a silent transition in its TES transition system, or performs an action.*

4.3 DSL for agents with preferences

Agents introduced in Section 4.2 specifiy sequence of possible actions. Due to the interaction taking place among agents, an agent's action may discard some of other agent's actions. For instance, an action may need to synchronize with another agent's action, and is therefore disabled if the required action is unavailable. As a result, an agent may add to its set of actions an order that reflects its internal preference. After composition with other agents, the set of actions is ranked to select an action with the highest preference. An agent can therefore adapt, at runtime, to an open set of agents.

In [50], we propose an automata-based paradigm based on soft constraint automata [9, 49], called soft component automata (SCAs). An SCA is a state-transition system where transitions are labeled with actions and preferences. Higher-preference transitions typically contribute more towards the goal of the component; if a component is in a state where it wants the system to move north, a transition with action north has a higher preference than a transition with action south. At run-time, preferences provide a natural fallback mechanism for an agent: in ideal circumstances, the agent would perform only actions with the highest preferences, but if the most-preferred actions fail, the agent may be permitted to choose a transition of lower preference. At design-time, preferences can be used to reason about the behavior of the SCA in suboptimal conditions, by allowing all actions whose preference is bounded from below by a threshold. In particular, this is useful if the designer wants to determine the circumstances where a property is no longer verified by the system.

The algebraic structure for preferences is called a *constraint semiring* and was proposed in [22]. A *c-semiring* is a tuple $(A, +, \times, 0, 1)$ such that

1. A is a carrier set that contains two element $0, 1 \in A$;

2. $+$ is a commutative associative idempotent binary operator, with unit element 0 and absorbing element 1;

3. $\times$ is a commutative associative binary operator that distributes over $+$, with unit element 1, and absorbing element 0.

Well-known instances of c-semirings are the

- *boolean* c-semiring $\mathbb{B} = (\{0, 1\}, \min, \max, 0, 1)$;
- *fuzzy* c-semiring $\mathbb{F} = ([0, 1], \min, \max, 0, 1)$;
- *bottleneck* c-semiring $\mathbb{K} = (\mathbb{R}_{\geq} \cup \{\infty\}, \max, \min, 0, \infty)$;
- *probabilistic* or *Viterbi* c-semiring $\mathbb{V} = ([0, 1], \max, \times, 0, 1)$;
- *weighted* c-semiring $\mathbb{W} = (\mathbb{R}_{\geq} \cup \{\infty\}, \min, +, \infty, 0)$.

Every c-semiring admits an order $\leq$ defined by $r \leq s$ iff $r + s = s$. It is shown in [22] that $\leq$ satisfies the following properties:

1. $\leq$ is a partial order, with minimum 0 and maximum 1;

2. $x + y$ is the least upper bound of x and y;

3. $x \times y$ is a lower bound of x and y;

4. $(S, \leq)$ is a complete lattice (i.e., the greatest lower bound exists);

5. $+$ and $\times$ are monotone on $\leq$.

6. if $\times$ is idempotent, then $+$ distributes over $\times$, $x \times y$ is the greatest lower bound of x and y, and $(S, \leq)$ is a distributive lattice.

The composability of actions and their resulting composition is defined in [50] with a Component Action System (CAS). A CAS can be lifted to an interaction signature on TESs by using, for instance, the synchronous composability relation defined in Chapter 2.

Definition 36 (Component action system)**.** *A* component action system (CAS) *is a tuple* $\langle \Sigma, \odot, \boxdot \rangle$*, such that* Σ *is a finite set of* actions*,* $\odot \subseteq \Sigma \times \Sigma$ *is a reflexive and symmetric relation and* $\boxdot : \odot \to \Sigma$ *is an idempotent, commutative and associative* $\odot$*-operator on* Σ*. We call* $\odot$ *the* composability relation*, and* $\boxdot$ *the* composition operator*.*

A Soft Component Automaton (SCA) is a finite characterization of an agent behavior, equipped with a strategy. The csemiring value labeling each transition induces a partial order, for each state, on the set of outgoing transitions. An agent may therefore filter its behavior by allowing only the k best actions from the partially ordered set of outgoing transitions.

Definition 37 (Soft component automaton)**.** *A* soft component automaton (SCA) *is a tuple* $\langle Q, \Sigma, \mathbb{E}, \to, q^0, t \rangle$ *where* Q *is a finite set of* states*, with* $q^0 \in Q$ *the* initial state*,* Σ *is a CAS, and* $\mathbb{E}$ *is a c-semiring, with* $t \in \mathbb{E}$ *the* threshold*. Lastly,* $\to \subseteq Q \times \Sigma \times \mathbb{E} \times Q$ *is a finite relation called the* transition relation*. We write* $q \xrightarrow{a,\, e} q'$ *when* $\langle q, a, e, q' \rangle \in \to$*.*

The threshold determines which actions have sufficient preference for inclusion in the behavior. Intuitively, the threshold is an indication of the amount of flexibility allowed. In the context of composition, lowering the threshold of a component is a form of compromise: the component potentially gains behavior available for composition. Setting a lower threshold makes a component more permissive, but may also make it harder (or impossible) to achieve its goal.

Definition 38 (Behavior of an SCA)**.** *Let* $A = \langle Q, \Sigma, \mathbb{E}, \to, q^0, t \rangle$ *be an SCA. We say that a stream* $\sigma \in \Sigma^\omega$ *is a* behavior *of* A *if there exist streams* $\mu \in Q^\omega$ *and* $\nu \in \mathbb{E}^\omega$ *such that* $\mu(0) = q^0$*, and for all* $n \in \mathbb{N}$*, we have* $t \leq \nu(n)$ *and* $\mu(n) \xrightarrow{\sigma(n),\ \nu(n)} \mu(n+1)$*. The set of behaviors of* A*, denoted by* $L(A)$*, is called the* language *of* A*.*

To discuss an example of SCA, we introduce the SCA A_s in Figure 4.1, which models the crop surveillance drone's objective to take a snapshot of every location before moving to the next. The CAS of A_s includes the pairwise incomposable actions *pass*, *move* and *snapshot*, and its c-semiring is the weighted c-semiring $\mathbb{W}$. We leave the threshold value t_s undefined for now. The purpose of A_s is reflected in its states: q_Y (resp. q_N) represents that a snapshot of the current location was (resp. was not) taken since moving there. If the drone moves to a new location, the component moves to q_N, while q_Y is reached by taking a snapshot. If the drone has not yet taken a snapshot, it prefers to do so over moving to the next spot (missing the opportunity).

Another example of an SCA is A_e, drawn in Figure 4.2; its CAS contains the incomposable actions *charge*, *discharge*$_1$ and *discharge*$_2$, and its c-semiring is the weighted

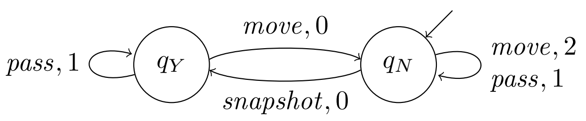

Figure 4.1: A component modeling the desire to take a snapshot at every location, A_s.

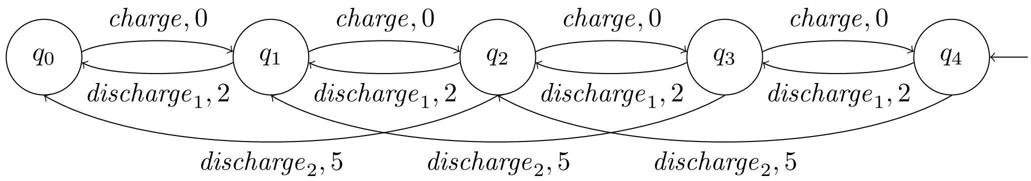

Figure 4.2: A component modeling energy management, A_e.

c-semiring $\mathbb{W}$. This particular SCA can model the component of the crop surveillance drone responsible for keeping track of the amount of remaining energy in the system; in state q_n (for $n \in \{0, 1, \ldots, 4\}$), the drone has n units of energy left, meaning that in states q_1 to q_4, the component can spend one unit of energy through $discharge_1$, and in states q_2 to q_4, the drone can consume two units of energy through $discharge_2$. In states q_0 to q_3, the drone can try to recharge through *charge*. Recall that, in $\mathbb{W}$, higher values reflect a lower preference (a higher *weight* or *cost*); thus, *charge* is preferred over $discharge_1$.

Here, A_e is meant to describe the possible behavior of the energy management component only. Availability of the actions within the *total model* of the drone (i.e., the composition of all components) is subject to how actions compose with those of other components; for example, the availability of *charge* may depend on the state of the component modeling position. Similarly, preferences attached to actions concern energy management only. In states q_0 to q_3, the component prefers to top up its energy level through *charge*, but the preferences of this component under composition with some other component may cause the composed preferences of actions composed with *charge* to be different. For instance, the total model may prefer executing an action that captures $discharge_2$ over one that captures *charge* when the former entails movement and the latter does not, especially when survival necessitates movement.

Nevertheless, the preferences of A_e affect the total behavior. For instance, the weight of spending one unit of energy (through $discharge_1$) is lower than the weight of spending two units (through $discharge_2$). This means that the energy component prefers to spend a small amount of energy in a single step. This reflects a level of care:

by preferring small steps, the component hopes to avoid situations where too little energy is left to avoid disaster.

Reo as a DSL We define a domain specification language for finite state preference aware agents as a subset of Reo. One reason for using Reo is its graphical syntax, which gives an intuitive encoding of soft component automata in terms of graphical connectors and interaction primitives. Moreover, Reo reflects the modular and compositional aspects that make SCAs suitable for specifying complex behaviors: connectors compose into more complex connectors, just like how SCAs compose into more complex SCAs. We take advantage of this feature and, after defining an encoding of SCAs into Reo connectors, we represent the composition of SCAs as the composition of their corresponding connectors. Another reason is the existence of a compilation chain that makes it possible to compile the same Reo model to an execution language (such as Java or C) or to a language that supports verification (such as the rewriting logic language Maude [13]). Effective optimizations implemented in the current Reo compiler help to keep the size of resulting composed models manageable, yielding similarly manageable models in Maude, Java, etc.

Some existing research has considered the question of synthesizing Reo circuits for constraint automata [16]. In our work, similar channels are used for encoding the structure of the automaton (syncfifo, xrouter, and merger), but a new channel, the bfilter, is introduced to encode the soft part of the action labeling transitions of SCAs. Moreover, we provide, along with the description, the representation of the Reo connector in a textual language, used as input for the compiler developed in [64].

We propose a general approach to represent SCAs and their composition as Reo circuits. Recall that, by Definition 37, an SCA is formally defined as a tuple $\langle Q, \Sigma, \mathbb{E}, \rightarrow, q^0, t\rangle$ where Q is the set of states, Σ a component action system, $\mathbb{E}$ a c-semiring, $\rightarrow$ a transition relation, $q^0 \in Q$ the initial state, and $t \in \mathbb{E}$ is the threshold value of the SCA. In the sequel, we give a procedure to write an SCA as a Reo circuit. The set of connectors defined hereafter constitutes a domain specific fragment of Reo for building SCA. We conclude this section with an example of composition of two SCAs obtained through composition of their respective Reo representations.

Actions and c-semirings Given Σ a CAS of an SCA, we map each action $a \in \Sigma$ into a Reo port with the same name. We consider the SCA "doing action a" equivalent to "firing of port a". Given the threshold $t \in \mathbb{E}$, we associate each c-semiring value $e \in \mathbb{E}$ with a predicate $P_t(e)$ whose semantics reflects the truth value of $t \leq e$ in the

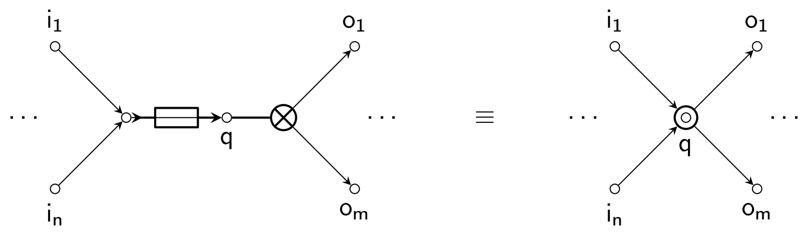

Figure 4.3: Graphical abbreviation for a state.

c-semiring. In order to mirror the semantics defined previously for composition of SCA, the c-semiring value and the threshold value of a predicate may change during composition. We consider the c-semiring to be fixed and shared by all SCAs.

States We define a state of an SCA as a Reo circuit, which we then graphically abbreviate as a user-defined node. Essentially, a state is mapped into a syncfifo channel, the empty/full status of whose buffer reflects whether or not the SCA is currently in that state. As depicted in the circuit below, we identify the source end of the syncfifo with the name of the state. Thus, to be in state q of the SCA corresponds to the syncfifo whose source end is q being full. The initial state q^0 starts with a full syncfifo buffer; the syncfifo buffers of all other states start empty. Intuitively, all incoming $(\mathsf{i}_1, \dots, \mathsf{i}_n)$ transitions into a state q, merge at the source end of the syncfifo, and all outgoing transitions $(\mathsf{o}_1, \dots, \mathsf{o}_m)$ out of q synchronize via mutual exclusion with one another on the sink end of the syncfifo. The reason for using the syncfifo instead of the standard fifo primitive is that an outgoing transition can also be an incoming transition into the same state, i.e., allow get and put operations on its ends to synchronously empty and fill its buffer. We use an n-ary *exclusive router* to express that only one outgoing transition is taken from a state with n outgoing transitions. The n-ary xrouter can be constructed out of a ternary xrouter.

We call our constructed circuit a *state*, and use ◎ to represent a state of an SCA as a graphical abbreviation and present it as a user-defined node in Reo with n inputs and m outputs. We use ⦿ as graphical abbreviation for the Reo circuit that corresponds to the initial (and current) state of an SCA.

Besides the graphical construct for a state, we introduce a State connector in the textual language of Reo shown in Listing 4.1. We adopt a convention, and prefix the input and output ports of a state with the name of the state. For instance, the component State(q0i[1..n], q0o[1..m]) represents the state q^0 with n incoming transitions and

```
State(qi[1..n],qo[1..m]) {
  { sync(qi[k],x)  |  k:<1..n> }
  syncfifo1<"0">(x,y)
  xrouter(y,qo[1..m])
}
```

Listing 4.1: Component defining a state q in textual Reo.

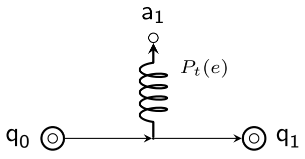

Figure 4.4: Reo circuit for a transition of a soft component automaton.

m outgoing transitions. We refer to the k-th incoming, resp. k-th outgoing, transition to state q0 with the port q0i[k], respectively q0o[k].

Listing 4.1 shows an example of a component defined using conditional set notation. The number of input ports in the interface of the State component influences how its body is instantiated. The variable k ranges over the list $[1, .., n]$, and thus creates a set of sync channels.

Transitions A transition in an SCA involves an action, a c-semiring value, a pre-state and a post-state. When the transition is enabled (i.e., its c-semiring value is above the threshold), the transition synchronously fires the action port, and moves the SCA from its pre-state to its post-state. We model this behavior in Reo as the circuit in Figure 4.4, which represents the conditional activation of a transition using a blocking-filter channel that compares the c-semiring value of the transition with the threshold of the SCA. Given a c-semiring value e, the predicate $P_t(e)$ of the blocking-filter channel is true if and only if the c-semiring value e is greater than or equal to the threshold value t.

The circuit in Figure 4.4 moves the token from node q_0 to node q_1 and fires port a_1, only if $P_t(e)$ is true. If $P_t(e)$ is not satisfied, the circuit in Figure 4.4 blocks the transfer of the token from q_0 to q_1, mirroring the fact that its corresponding SCA transition cannot be taken.

The transition primitive in textual Reo is written in Listing 4.2. The transition component takes three ports in its interface, q_0 and q_1, being respectively the pre-state and post-state, and a_1 being the action. Two values are provided as parameters to a

```
Transition<e,t>(q0,q1,a1) {
  sync(q0,x)
  bfilter<e,t>(x,a)
  sync(x,q1)
}
```

Listing 4.2: Component defining a transition in textual Reo.

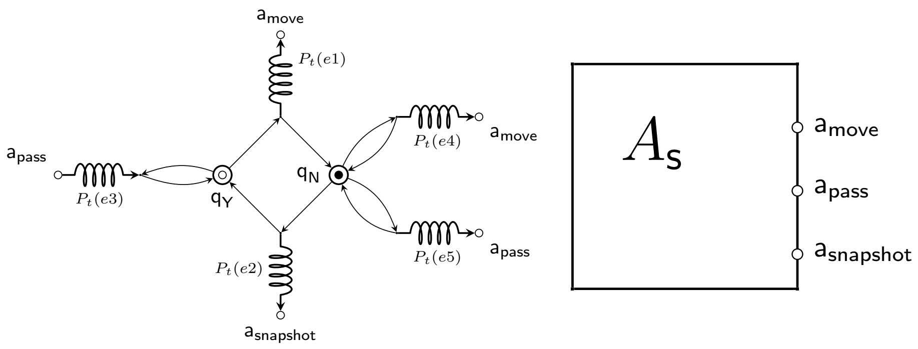
$\mathsf{a}_{\mathsf{move}}$
$P_t(e1)$
$P_t(e4)$
$\mathsf{a}_{\mathsf{move}}$
$\mathsf{a}_{\mathsf{pass}}$
q_{N}
$P_t(e3)$
q_{Y}
$P_t(e5)$
$\mathsf{a}_{\mathsf{pass}}$
$P_t(e2)$
$\mathsf{a}_{\mathsf{snapshot}}$
A_{s}
$\mathsf{a}_{\mathsf{move}}$
$\mathsf{a}_{\mathsf{pass}}$
$\mathsf{a}_{\mathsf{snapshot}}$

Figure 4.5: Reo circuit for the Snapshot SCA.

transition component: the c-semiring value e, and the threshold value t. Internally, the transition component connects the pre-state to the post-state through synchronization with the bfilter. The bfilter takes a c-semiring value of a given type as parameter, and performs internal comparison with the threshold value.

Soft component automata Given the constructs for states and transitions, we can build a Reo circuit for every SCA. For instance, the circuit for the automaton in Figure 4.1 is shown in Figure 4.5. The two states q_{Y} and q_{N} are represented as two state-nodes, with q_{N} initially full (designating it as the initial state).

To avoid visual clutter, we repeat the names of ports in the circuit (e.g., $\mathsf{a}_{\mathsf{move}}$ appears twice in Figure 6), but all occurrences of the same port name correspond to a single, unique port. Each of the five transitions of A_{s} is an instance of the transition component in Reo. For example, the *move* transition from q_Y to q_N is represented by the transition connector with input from the state q_Y, output from the state q_N, blocking filter with predicate $P_t(e_1)$, and action port $\mathsf{a}_{\mathsf{move}}$. The corresponding component view of the automaton is represented by a box that abstracts away the details of its Reo circuit, exposing as its interface the boundary ports on which other components can synchronize.

```
As<t>(move,pass,snap) {
  Transition<1,t>(qYo[1],qYi[1],pass)
  Transition<0,t>(qYo[2],qNi[1],move)
  Transition<0,t>(qNo[1],qYi[2],snap)
  Transition<2,t>(qNo[2],qNi[2],move)
  Transition<1,t>(qNo[3],qNi[3],pass)

  State(qYi[1..2],qYo[1..2])     //State qY
  State(qNi[1..3],qNo[1..3])       //State qN
}
```

Listing 4.3: Component defining the snapshot SCA in textual Reo.

The snapshot SCA A_{e} is built out of the State and Transition connectors in Reo defined in Listings 4.1 and 4.2. We show the instance of the Snapshot SCA A_{s} in Listing 4.3, and adopt the convention defined previously to denote ports of incoming and outgoing transitions.

Component action system The composition of two SCAs can also be written as a Reo circuit, by encoding the composed SCA. However, such an approach uses the SCA composition and disregards the compositional nature of Reo. Instead, we propose to encode each individual SCA as a Reo circuit, and then compose those encodings on the level of Reo, to obtain a Reo circuit equivalent to their composed automaton. This approach allows for a transparent and incremental translation.

Since composition on the level of SCAs is mediated by their (common) CAS, composition at the level of Reo should also take the CAS into account. To do this, we encode the CAS as a Reo circuit of its own; composition of two automata at the level of Reo is then given by the (Reo) composition of their individual encodings, together with the circuit obtained from their CAS. Furthermore, we hide all ports that are not output ports of the CAS after the composition, so that the only actions observable in the resulting Reo circuit are the actions that are brokered between the operand circuits by the CAS.

There are three "sides" (collections of ports) to a CAS component: one for each of the two operands in the composition, respectively called the *left* and the *right (operand) side*, and a *composite side* for the result of the composition. For each action α, we add three ports to the circuit, one in each side, labeled α_ℓ, α_r and α_c for the left, right and composite sides respectively. The ports on the operand sides are input ports, and the ports on the composite side are output ports.

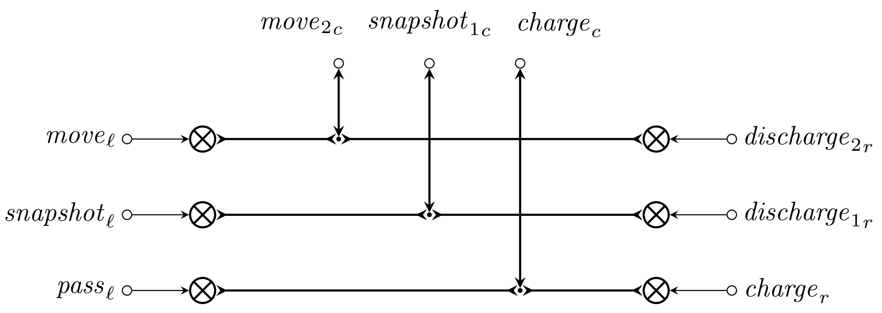

Figure 4.6: Partial encoding of a CAS.

The intention of the circuit structure is as follows. If the operand circuits are ready to perform actions α and β respectively, then ports α_ℓ and β_r will be enabled for writing. If $\alpha \odot \beta$, then the CAS circuit brokers their composition, by allowing α_ℓ and β_r to fire simultaneously, synchronously firing the port that represents their composition in the composite side, i.e., $(\alpha \boxdot \beta)_c$, as well. Moreover, the circuit ensures that firing two ports in the left and right sides (when permitted) gives rise to exactly one port firing in the composite side.

More formally, the circuit is built as follows. On the operand sides, each port α_o (where $o \in \{\ell, r\}$) is connected to an exclusive router labeled α_o^R. For each pair of actions in the left and right operand sides that are compatible, i.e., all $\alpha, \beta \in \Sigma$ such that $\alpha \odot \beta$, we draw a synchronous drain from α_ℓ^R and β_r^R to an internal node labeled $\alpha\beta$. Each of these nodes is then connected through a syncspout channel to the composite side node labeled $(\alpha \boxdot \beta)_c$.

The CAS defined for the SCAs A_e and A_s is depicted in Figure 4.6. In this example, the exclusive router has a single output, and is not strictly necessary. In general, the CAS could define multiple composite actions out of the same side action. For instance, suppose that the drone in our example is equipped with solar panels, and that the net result of charging using the solar panels while moving is that the energy level does not change. As a result, the energy component's action pass is compatible with the action move, and their composition is the action solar, which means "move with energy from the solar panels". Note how in this scenario, the firing of move_ℓ can occur only in conjunction with firing $\mathsf{discharge}_{2r}$ or pass_r, but not both; in the first case, the composite interface port move_{2c} fires, while in the second case the port solar_c fires.

```
cas(move,pass,snap,dchge1,dchge2,
    chge,move2,charge,snapshot1){
  syncdrain(move,x)  syncspout(x,move2)
  syncdrain(dchge2,x)
  syncdrain(pass,y) syncspout(y,charge)
  syncdrain(chge,y)
  syncdrain(snap,z) syncspout(z,snapshot1)
  syncdrain(dchge1,z)
}
```

Listing 4.4: Component defining the CAS for the composition of A_e and A_s in textual Reo

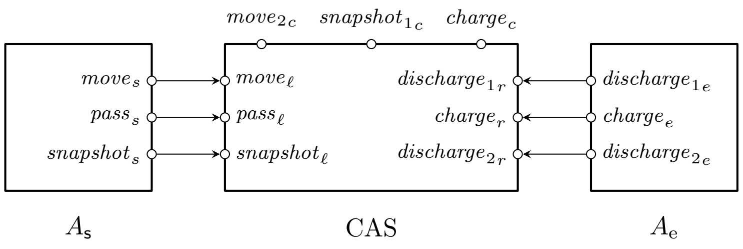

Figure 4.7: Composition of two component automata with their component action system.

We give in Listing 4.4 the corresponding Reo component for the CAS described in Figure 4.6 for the composition of the snapshot SCA and the energy SCA. We omitted the exclusive routers, since, in this case, they are not necessary.

Composition The Reo circuit corresponding to a composition of two soft component automata can now be defined as the composition of the Reo circuits for the individual soft component automata, together with the Reo circuit for the relevant component action system. Following the method above, we translate each of A_s and A_e, respectively representing the snapshot component and the energy management component, into its respective Reo connector.

Based on the steps described above, it is now possible to define a Reo circuit for both A_e and A_s, that we name respectively `Ae` and `As` in textual Reo. The resulting composition, shown in Listing 4.5, consists of a set containing the connector for each SCA together with the connector for the component action system. The two thresholds values are provided as parameter.

```
composite(move2,charge,snapshot1) {
  Ae<t1>(move,pass,snap)
  cas(move,pass,snap,dchge1,dchge2,
      chge,move2,charge,snapshot1)
  As<t2>(dchge1,dchge2,chge)
|
  t1 = 5,
  t2 = 3
}
```

Listing 4.5: Component in textual Reo defining the composition between A_e and A_s.

4.4 Related work and future work

Real-time Maude Real-Time Maude is implemented in Maude as an extension of Full Maude [74], and is used in applications such as in [58]. There are two ways to interpret a real-time rewrite theory, called the pointwise semantics and the continuous semantics. Our approach to model time is similar to the pointwise semantics for real-time Maude, as we fix a global time stamp interval before execution. The addition of a composability relation, that may discard actions to occur within the same rewrite step, differs from the real-time Maude framework.

Models based on rewriting logic In [91], 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. Softagents [84] is a framework for specifying and analyzing adaptive cyber-physical systems implemented in Maude. It has been used to analyze systems such as vehicle platooning [30] and drone surveillance [66]. In Softagents agents interact by sharing knowledge and resources implemented as part of the system timestep rule.

Softagents only considers compatibility in the sense of reachability of desired or undesired states. Our approach provides more structure enabling static analysis. Our framework allows, for instance, to consider compatibility of a robot with a battery (i.e., changing the battery specification without altering other agents in the system), and coordination of two robots with an exogenous protocol, itself specified as an agent.

Hybrid programs Other models for cyber-physical systems exist, such as hybrid systems (e.g., Hybrid Programs [75], Hybrid automata [40, 65, 79]), and our semantic model differs and complements existing work in, at least two main points. First, we model interaction externally, as constraints that apply on the behavior of each component. Interaction is not limited to input/outputs as in most hybrid descriptions, and the difference between cyber and physical aspects is abstracted in the general concept of a component. The generality of the semantic model enables to give a *specification* of a component as a hybrid program, or as an I/O hybrid automata, and define suitable composition operators in the algebra to compositionally define cyber-physical systems. Second, we choose to model the interaction occurring between components in a discrete way, as sequences of observations: we choose to model the continuity of physical systems within their description as a set of discrete sequences of observations. This description closely represents runtime observable behaviors of cyber-physical systems, and highlights new challenges such as proving that a cyber-physical system is safe when considering safety of runtime observables only.

Timed Automaton Several operations on Timed Automata have been defined to model different aspects of concurrency. The UPPAAL modeling language allows such concurrent operations, and the UPPAAL tool computes the product automaton on the fly during verification.

It is shown that reachability is decidable, and it is proven that the infinite state-space of timed automata can be finitely partitioned into symbolic states using clock constraints known as zones.

UPPAAL is a tool in which network of timed automata are considered. Similarly to the case of a single timed automata, two types of transitions are considered: delay transitions, and action transitions. The difference is that action transitions decline into two kinds: single action transitions, and synchronous action transitions.

UPPAAL makes use of CTL formulas, that are dynamically verified on the tree unfolding of the transition system, making use of the zone optimization. UPPAAL is well-suited for timed automata but has some limitations in the support of hybrid automata, e.g. restricting their continuous parts to simple dynamics or applying the Euler integration method.