Chapter 6

Conclusion

Modeling and analysis of cyber-physical systems are still challenging [57]. One reason is that cyber-physical systems involve many different parts (cyber or physical), of different nature (discrete or continuous), and in constant interaction via sensing and actuating. This thesis proposes a semantic framework in which both the behavior of cyber-physical systems and their interaction can be expressed. Below, we give a short summary of the contribution for each chapter, and provide a list of points for future work.

The component based model introduced in Chapter 2 proposes to explicitly model interactions in cyber-physical systems with parametrized algebraic operators, particularly for composition and decomposition. In this model, components are terms and interactions are captured by user defined operations on components. Properties of operators, such as associativity, commutativity, and idempotency, are expressed in terms of the properties of their interaction signatures. For instance, an interaction constraint that is symmetric leads to a commutative product on component. To ease the specification of interaction constraints, a co-inductive construction of composability relations is proposed to specify constraints on Timed-Event Streams given a constraint on observations. Similar algebraic properties for a co-inductively defined operator on components are deduced given properties of the underlying constraints on observations. The model not only provides ways to compose components, but also allows, under certain conditions, for decomposition of components. An operator of division is introduced that selects, out of a set of candidates, a component that in product with the divisor give back the dividend. Several such division operators are possible as each involves a way of choosing a component within a set of candidates.

In the case where a metric is used to order components, the division operator can be defined as taking the best (defined by the metric) of the candidate components. The cost of coordination is discussed as such a possible metric.

We instantiate our component model in Chapter 3 on a set of cyber components whose behaviors are independent of time. We give an algebraic semantics of the Reo coordination language, where ports are components and circuits are product of ports under some interaction signatures. The result adds some strength to existing work on Reo, by providing a suitable algebraic framework to show equivalence of connector behaviors in Reo. Within the class of order sensitive components, we consider two meaningful classes: transactional components that have observations with more than one event, and linear components that have observations with at most one event. The former class of components is adequate to represent behavior at a design/specification level where events within the same set are declared to be atomic; the latter class of components represents sequential machines that can generate only one event at a time. We study translation of (product of) transactional components into a (product of) linear components, which leads to the definition of correctness criteria for implementation of high-level transactional specifications of concurrent systems into their behaviorally equivalent implementations on (sets of interacting) sequential machines. Two instances of correct and compositional translations are defined.

Finally, we study temporal properties of some order sensitive components. Particularly, we consider temporal properties of Reo connectors, specifically to verify data flow properties of composite connectors. For that purpose, we give a translation to generate an executable in Promela from a logical specification of Reo connectors, using which the model checker Spin verifies properties written in Linear Temporal Logic (LTL). To ease the specification of LTL properties, we introduce a domain specific language for data flow properties, e.g., the temporal property that specifies the simultaneous or exclusive firing of a port.

In order to make our component model executable, in Chapter 4 we present several steps that lead to an operational yet compositional specification of components. First, we operationally define the specification of a component behavior, i.e., a set of Timed-Event Stream, as the semantics of a labeled transition system called a TES transition system. We introduce several operators on TES transition systems, and show them to semantically correspond to their component products: the behavior of a syntactic product of TES transition systems is the behavior of the semantic product of their corresponding two components.

While a TES transition system is not necessarily finitely representable (its sets of states and transitions may be infinite), it serves as a first step towards a finite executable model, and gives some results as to when the product of two TES transition systems can be done lazily at runtime without deadlock.

In order to have a finite executable specification of components, we introduce agents as rewriting theories for component behaviors. An agent specifies finitely, with a set of equations and a rewrite rule, a TES transition system, which ultimately denotes a component. A system of agents runs each agent concurrently, while ensuring that each agent performs composable actions. As a result, the behavior of a system of agents is shown to coincide with the product (under the composability relation imposed by the system) of the behaviors of each agent.

Finally, to reduce the large state space involved in cyber-physical systems, we introduce a framework to extend agents with preferences. The use of preferences allows for ordering actions that an agent can perform. Thus, actions that are possible but less preferred can be discarded and therefore reduce the state space. Notions of compromise emerge out of the composition of agents with different preferences for their actions.

To demonstrate the utility of our component model, in Chapter 5 we give an implementation of the rewriting framework in Maude. We define agent as a module that implements a set of operations for interacting with other agents. A system module runs all agents concurrently, and ensures composability of their actions. To demonstrate the usefulness of this framework, we present three main applications. First, we specify ports and connectors in Reo as agents that run concurrently. As a result, a Reo circuit can be composed at runtime, and several search queries can be performed to extract properties of protocols. Next, we consider a system consisting of a controller, a valve, and N water reservoirs. The controller needs to move the valve in order to keep the water level in the reservoirs within some thresholds. We contrast the safety property of the controller not seeing any invalid states with the possibility for the controller to miss some states. In the latter case, the safety property may still be valid from the controller's perspective, while it effectively violates the property as the water level of the reservoirs may unobservably go out of bound. Finally, we analyse a system consisting of a set of robots, each equipped with a battery, running on a shared field. More specifically, we consider liveness and safety properties, such as the possibility for a robot to patrol through a set of points without running out of battery, or to coordinate with another robot to swap position and get in a sorted position.

A set of challenges emerge from the work presented in the chapters of this thesis that we consider as relevant future work. Hereafter is a list of few points, organized per chapters, that are considered relevant for future investigation.

1.1 Metrics for division.

Our algebra of components introduces a family of operators to compose components into larger systems. Dually, the operation of decomposition is equally important to extract some structure from a composite component, and eventually update some of its parts. For that matter, the operation of division is defined to return one of the possible quotients. As the division operator assumes a choice function over a set of components, several metrics can be used to partially order components, and therefore act as the choice function for the operator of division. One of such metric, discussed in Section 2.2, is the *cost of coordination* that orders components with respect to the amount of interaction required during composition. As future work, other measures may be considered, as well as their corresponding implication on system's decomposition.

1.2 Laws of distribution.

The algebraic properties of operators in the algebra of components in Section 2.1 are mostly studied independently (i.e., commutativity, associativity, idempotency). However, the law of distribution of one product over another product would allow for additional reasoning on component expression by showing equivalence between different types of interaction. More precisely, being able to factorize or distribute an operation may practically reflect the distinction between the cases where a group of components needs global as opposed to local interaction. Considering for instance the operation of division, the distribution of division over multiplication may, in some cases, allow for modular decomposition. Moreover, on the level of components, distribution of a component over a product could capture the fact that such component is independent of the interaction constraints imposed by that product. As a result, such distributive law can reveal mechanisms to split a protocol component into subparts, that get distributed onto the smallest set of components that needs their coordination.

2.1 Extension of Reo with cyber-physical connectors.

In Reo, the main primitive of interaction is a port, which serves as the junction through which data flow between a component and its environment. By definition, a port in Reo observes the transport of a finite amount of data within a time interval, which precludes the possibility of continuous physical data transfer. Instead, a physical phenomenon must be encapsulated into a component that samples the physics at a fix rate. Only then can a port transport the value, at such rate, to other Reo components. Note that the choice of the sampling rate for the component encapsulating a physical process will influence the overall protocol, as other components may not be able to see some relevant data, or take action accordingly. Alternatively, introducing a cyber-physical variant of a port, with sampling as a parameter, leads to the possibility of having cyber and physical components in Reo. Instead of the standard semantics where the firing of a port occurs if and only if a data of a port is present, a cyber-physical port is equipped with a sampler that can only observe a datum within a time period. Similarly, the composition operator can be extended such that ports tune their sampling rates for transmission. As a result, such cyber-physical extension of Reo may enable compositional construction to find the highest frequency (i.e., smallest amount of observations) for samplers while not missing any important event.

2.2 Temporal properties as Reo connectors.

The definition of components allows both executable specifications and properties to be denoted as such. As a result, the structure of our algebra can be exploited to specify temporal properties compositionally. More precisely, as already hinted in Section 3.3, the use of graphical and compositional primitives of Reo can serve as a starting point for languages to construct temporal properties. As a result of such modular specification, ways of distributing the property over components, or decomposing the property into a set of simpler properties may facilitate modular verification.

3.1 Real time extension of agents.

The current specification of agents uses rewriting logic without real time. Agents run in steps, and may interact with other agents within each step. The time that lapses between two steps is a constant that is fixed at the beginning of the execution. Alternatively, our framework can be extended with real time variables, that account for the variable time that lapses between two steps. As well, making the time of a step explicit while having a modular specification of each agent may facilitate certain meta level reasoning, such as finding the largest sampling rate such that resulting system satisfies, for instance, a safety property.

3.2 Fusion of agents.

Each composition operator of our algebra captures a constraint of interaction between two components. Practically, such an interaction constraint may represent actual physical constraints (e.g., spatial, time, hardware) that enforce the practical observations to be related. For instance, two components tied with each other by a synchronous product may be realised by two processes running on the same machine. Following the result of Section 4.2, agents denote components, and the product of agents denotes the product of their corresponding components. In some cases, it can be useful to define a new agent out of the product of two agents, such that the newly formed agent denotes the same behavior as the product component. For instance, forming such new agent may reduce the search space, or improve some runtime measures, while preserving the system behavior. Practically, such transformation acts on agent theories, and needs to generate a new agent specification out of two agent specifications.

3.3 Symbolic execution under a class of environments.

The rewriting framework developed in Section 4.2 currently assumes that the collection of agents forming a system is *closed*. A closed system is such that every agent's action is matched with that of its corresponding recipient agent in the set. While this condition holds for many practical examples, allowing for *open* system analysis is still desirable. In an open system, one can analyse the behavior of an agent under a class of environments, and conclude which of a set of potential interacting agents complies or conflicts with some system properties.

3.4 Implementability of division.

Our algebra of components introduces several operations on behaviors that are shown to be useful for design and modular analysis. One such operation is the operation of division, which enables reasoning about alternative compositions that preserve the same behavior. The operation of division does not yet enjoy the same operational specification as the operation of composition. Studying how to implement the operation of division within our agent framework may enable various important reasoning schemes for updates and fault diagnosis. Practically, the operator of division may reveal some independences within an agent, and factor such agents into two independent parts. The two specifications may be run in parallel, speeding up the overall execution time.