A modal interface contract theory for guarded input/output automata
with an application in traffic system design

Tung Phan-Minh¹, Steve Guo, Bastian Schürmann, Matthias Althoff, and Richard M. Murray

Abstract—To contribute to efforts of bringing formal design-by-contract methods to hybrid systems, we introduce a variant of modal interface contract theory based on input/output automata with guarded transitions. We present an algebra of operators for interface composition, contract composition, contract conjunction, contract refinement and some theorems demonstrating that our contract object has reasonably universal semantics. As an application, we apply our framework to the design of a networked control systems of traffic.

INTRODUCTION

The growth in scale and complexity of engineering systems has been fueling a practical demand for formal approaches to modular design [1], [6]. Generally speaking, modular design means breaking up a system into more or less standalone modules for a reduction in complexities. To guarantee correct product integration, it is therefore not only a matter of convenience but also of necessity for design choices intended for a module to be made available to others. One way of dealing with this dependency is to divide tasks of designing a module into two parts: specifying an interface and ensuring that the implementation satisfies it. The interface of a module is a relatively simple object that contains all information about the interactions it can offer to other modules. The implementation is a structure that satisfies all the specifications of the interface. The idea is that changes to an implementation of a module should not affect the overall behavior of assembled system as long as the implementation still satisfies the requirements of the interface.

A lightweight automata-theoretic approach to represent interfaces was introduced by de Alfaro and Henzinger [5], in which the temporal behavior of an interface is described by a game-based model in the form of an input/output (I/O) automaton, a formalism that first appeared in Lynch and Tuttle [9]. This was soon followed by modal specifications by Larsen [8], which can state whether an action is optional or obligatory. Later Raclet unified modal specifications and interface automata, paving the way for a preliminary theory of modal interface contracts [11]. More recently, Benveniste et al. subsumed this theory under an elegant, encompassing metatheory, referred to in this work as the metatheory, that aims to unite various formal contract frameworks [2]. In the application domain, however, the semantics of the theory is limited by a lack of clear restrictions on when a transition can trigger and by a peculiar rule for composing actions, namely, requiring that the action obtained from composing an input action with an output action to be an output action, which, while preserving the “interface” semantics, obscures the distinction between open and closed interfaces/systems. These drawbacks make it difficult and sometimes impossible to specify systems whose variables assume a large or infinite set of values.

With an aim to bringing the benefits of the theory of modal interface contract automata to more real-time systems, inspired by symbolic transducers [4], we develop a new theory that includes Boolean guards, a more intuitive definition of how the I/O actions interact, an introduction of a special state that enriches the semantics of the contract object, a simplification in the definitions of interfaces. In addition, we prove that the algebraic operations defined for our contract theory also have metatheoretic properties, implying compatibility with many existing contract frameworks. We then implement a set of tools that carry out the contract algebra in a similar manner to Mica, which implements the modal interface contract in [12]. As an illustrative example application of our theory, we introduce a method for setting up an autonomous traffic system where various interfaces communicate with each other while abiding by the contract protocol. Our concrete case study involves a real-time simulation of a traffic intersection (see Fig. 1) whose components interact with each other in accordance with the contract objects we devise.

Fig. 1. A snapshot of an implementation of a networked control system of traffic presented in [10]

INTERFACE CONTRACT THEORY

Many real-world applications ranging from online payment services to autonomous robots require networking
protocols to control sequential exchanges of information between many subsystems or participating agents. By “sequential” we mean that the interactions must occur in a well-specified, agreed upon temporal order. A good implementation of these protocols presupposes the notion of a set of rules for each subsystem that not only restrict what action the subsystem can perform from a certain state at a certain time but can also be compared to or combined with other sets of rules corresponding to other subsystems. To illustrate these ideas, we will provide definitions for three formal objects, arranged in the hierarchy pyramid in Fig. 2 by their level of abstraction. The higher the object sits, the more abstract it is and the fewer ways there are to implement it. First, we will introduce the interface.

**Interface**

Our formalism rests upon the assumption that each state of the universe fixes the values for a master set of variables \( \mathcal{V} \) whose temporal and algebraic behaviors are governed by mathematical and physical laws. The interface of a component (or interface for short) may be described by a subset of \( \mathcal{V} \) (these are called its reference variables), and can interact with the rest of the universe (i.e., one of its environments) through a set of actions. Any action from this interface may belong to exactly one of the following three classes. An input action of an interface corresponds to the receiving of mass or energy from its environment. An output action, on the other hand, corresponds to sending mass or energy to its environment. An internal or rendezvous action represents an interconnection. Intuitively, an interconnection implies the existence of at least one input and one output action and we can think of it as being internally exchanged within the component. As a syntactic reminder, we will prefix input actions with an exclamation mark (!), output actions with a question mark (?) and rendezvous actions with a hash symbol (#).

An interface is closed if and only if its corresponding set of actions is empty or only consists of rendezvous actions. An interface is called open otherwise. For example, the universe, in its entirety has a closed interface. A wireless router has an open interface because it takes inputs from a modem and emits radio waves.

For any set of variables \( \mathcal{Y} \subseteq \mathcal{V} \), an evaluation \( e \) of \( \mathcal{Y} \), denoted by \( e[\mathcal{Y}] \) is a legal assignment of values to each of the variables in \( \mathcal{Y} \). By legal, we mean that each variable is assigned a value that is in the value set specified by its type (e.g., the reals). In mathematical logic terms, \( e \) is a ground substitution of variables in \( \mathcal{Y} \). The set \( E_{\mathcal{Y}} := \{ e \mid e \text{ is an evaluation of } \mathcal{Y} \} \) contains all possible evaluations of \( \mathcal{Y} \). Since one of our interests is in “connecting” different interfaces it is important to specify the conditions under which this can happen. For this purpose and also to obtain a compact automaton representation of interfaces, we will invoke the notion of guards.

**Definition 1 (Guard):** Let \( \top := \text{True} \) and \( \bot := \text{False} \). A guard \( g \) defined on a set of variables \( \mathcal{Y} \) is a predicate on the variables in this set, namely, \( g : E_{\mathcal{Y}} \rightarrow \{ \top, \bot \} \). The set of all predicates on \( \mathcal{Y} \) is denoted by \( G_{\mathcal{Y}} \).

For example, when \( \mathcal{Y} \) is a set of Boolean variables, then a guard on \( \mathcal{Y} \) is a map from \( 2^\mathcal{Y} \) to \( \{ \top, \bot \} \). Below, we will use the tilde symbol \( \sim \) as a wild card character that acts as a placeholder for one of \( ? \) and \#.

**Definition 2 (Interface):** Each interface \( M \) is defined by a set of reference variables \( \mathcal{Y} \) and a tuple \( \mathcal{S} = (S, s_0, \bot, A, \rightarrow) \) where

(i) \( S \) is a finite set of operational states
(ii) \( s_0 \in S \) is the start state
(iii) \( \bot \notin S \) is the failure state
(iv) \( A \) is a set of actions. We will often write \( A \) as the partition

where \( ?A, !A, #A \) are the smallest sets containing all the input, output, and internal actions of \( A \) respectively.

Below, we will write \( q \stackrel{[g,a]}{\rightarrow} p \) in place of \( q \times [g,a] \times p \in \rightarrow \) as a predicate. The fact that our interfaces are deterministic allows us also to use the shorthand \( q \stackrel{[g,a]}{\rightarrow} p \) to say with little ambiguity that there exists a state \( p \in S \) such that \( q \stackrel{[g,a]}{\rightarrow} p \). Now we are ready to define the interface composition operator, which we will denote by the symbol \( \times \). Intuitively, given two interfaces \( M_1 \) and \( M_2 \) and assuming \( M_1 \) is currently in state \( s_1 \) and \( M_2 \) is currently in state \( s_2 \). If, for example, from \( s_1, M_1 \) has a transition \( [g_1] \mid ?a \) to some state \( s'_1 \) and from \( s_2, M_2 \) has a transition \( [g_2] \mid !a \) to state \( s'_2 \), then if \( M_1 \) and \( M_2 \) were to be composable, it would make sense to require \( M_1 \) and \( M_2 \) to exchange the action \( a \) with one another. This then reduces to checking if \( g_1 \land g_2 \) is satisfiable (there exists an evaluation of the variables in \( g_1 \land g_2 \) that makes it evaluate to \( \text{True} \)). If there is at least one satisfying assignment, then the two interfaces must handshake or “rendezvous” on it, otherwise, they are not composable.

**Definition 3 (Composition of Transitions):** The composition of two transitions \( t_1 = (q_1, g_1) \stackrel{[g_1,a_1]}{\rightarrow} (q'_1, g'_1) \) and \( t_2 = (q_2, g_2) \stackrel{[g_2,a_2]}{\rightarrow} (q'_2, g'_2) \), from automata \( M_1 \) and \( M_2 \) respectively, is possible if \( g_1 \land g_2 \) is satisfiable and \( a_1 = a_2 \). If these conditions are satisfied, the composed transition is given by \( t = (q_1, q_2) \stackrel{[g,a]}{\rightarrow} (q'_1, q'_2) \), where \( g := g_1 \land g_2 \) and \( a := \sim a_1 + \sim a_2 \) where \( + \) is a binary operator acting on \( \sim a_1 \) and \( \sim a_2 \) such that, for \( i \in \{1, 2\} \)

\[
\sim a_1 + \sim a_2 := \begin{cases} 
\#a_1 \sum_{i=1}^{2} (\sim a_1 \in \#A_1 \land \sim a_{3-i} \in \#A_{3-i}) \\
\lor \sim a_1 = \#a_i \\
\#a_1 \sim a_1 \in \#A_1 \land \sim a_2 \in \#A_2 \\
\#a_1 \sim a_1 \in \#A_1 \land \sim a_2 \in \#A_2 
\end{cases}
\]
Any input supplied to an automaton that is an output of the other becomes a rendezvous action of the composed automaton since the composed automata should represent the interconnection of the two automata. Observe also that input (or output) actions that compose with themselves are not converted to rendezvous actions but rather remain unchanged. For convenience, we define $u$ to be a universal “unmasking” function that maps all prefixed actions to pure actions, namely, $u : \sim a \mapsto a$. As an abuse of notation, for a set $A$ of prefixed actions, we write $u(A)$ to mean the set \{(u(a) : a \in A)\}. The composition $M$ of two interfaces $M_1$ and $M_2$ is defined as follows.

**Definition 4 (Interface Composition):** For two interfaces $M_1$ defined by $\gamma_1$ and $(S_1, s_0, \perp_2, A_1, \rightarrow_1)$ and $M_2$ defined by $\gamma_2$ and $(S_2, s_0, \perp_2, A_2, \rightarrow_2)$, their composition $M = M_1 \times M_2$ is defined by $\gamma := \gamma_1 \cup \gamma_2$ and $(S, s_0, \perp, A \rightarrow \gamma)$ such that
\[ S := S_1 \times S_2 \]
\[ s_0 := s_0_1 \times s_0_2 \]
\[ (i) \ A \text{ is partitioned into } A_1 \cup A_2 \cup A_3 \text{ with } \]
\[ \#A := \#A_1 \cup \#A_2 \cup \sum_{i=1}^{2} \#(u(?A_i) \cap u(\bar{A}_3-i))) \]
\[ ?A := ?A_1 \cup ?A_2 \]
\[ !A := !A_1 \cup !A_2 \]

(i) $\rightarrow$ is the Cartesian product of transitions in $\rightarrow_1$ and $\rightarrow_2$ with respect to composition as defined in definition 3. More specifically, $\forall t \in S_1 \times [G_{\gamma_1} \times A_1] \times S_1, \ t_2 \in S_2 \times [G_{\gamma_2} \times A_2] \times S_2$ such that $t_1$ and $t_2$ are composable, $t_1 \times t_2 \in \rightarrow_\gamma$.

(v) associate each (reachable) composite state in $\perp_1 \times S_2 \cup S_1 \times \perp_2$ with the failure state $\perp$.

Observe that due to Definition 4(iv) and the fact that each constituent interface is deterministic, from each composite state there can be at most one transition per unmasked action, so the composition is also a deterministic interface.

**Proposition 1 (Associativity and Commutativity):** The interface composition operator $(\times)$ is associative and commutative. Namely, for all interfaces $M_1, M_2, M_3$, we have
\[ M_1 \times M_2 = M_2 \times M_1 \] (1)
and
\[ (M_1 \times M_2) \times M_3 = M_1 \times (M_2 \times M_3). \] (2)

**Proof:** (1) easily follows from Definitions 3 and 4. To see why (2) holds, first we note that it is trivial to show that the resulting sets of operational states, the initial states, and the sets of states that eventually become the failure state from both sides of equation 2 are equal. In what follows, we will be referring to the constituents (e.g., action set) of each interface $M_i$ by their standard notations. Now observe from the last two items in Definition 4(iii), that the input and output action sets of the resulting interfaces on both sides of equation (2) are equal. For the internal action set, a direct calculation with an application of the set distributive law shows either side of (2) can be reduced to
\[ \bigcup_{i=1}^{3} \#A_i \bigcup_{i,j=1}^{3} \#(u(?A_i) \cap u(\bar{A}_j)). \] By Definition 4(iv) it remains to be shown that the transition sets are identical, which is to prove
\[ \forall t_1, t_2, t_3, \bigwedge_{i=1}^{3} (t_i \in \rightarrow_i) \implies (t_1 \times t_2) \times t_3 = t_1 \times (t_2 \times t_3) \] (3)

By Definition 3, we can assume that $a_1 = a_2 = a_3 = a$. Let us suppose that the resulting action of $(t_1 \times t_2) \times t_3$ is an internal action (the cases for input and output actions are straightforward). Then either
- at least one of $\sim a_1$ or $\sim a_2$ or $\sim a_3$ is an internal action, in which case the resulting action of $t_1 \times (t_2 \times t_3)$ must also be an internal action since $\#$ “absors” any other type of action.
- or among $\sim a_1, \sim a_2, \sim a_3$ there are exactly two types of actions, namely, input and output, in which case, the resulting action of $t_1 \times (t_2 \times t_3)$ is also an internal action.

Both cases show that (3) holds.

Below, whenever we make a reference to an interface $M_i$, and later a contract, we will be using the same notations used in their respective definitions with the appropriate subscripts to talk about their constituents (set of actions, start states etc.). Given two interfaces, knowledge of whether they are comparable to one another can be very useful. One way to enable this comparison to check whether one interface can “imitate” or simulate the other.

**Definition 5 (Simulation):** Let $M_1$ and $M_2$ be two interface automata. For $i = 1, 2$, let $q_i$ be a state of $M_i$, $q_1$ simulates $q_2$, written $q_1 \preceq q_2$, if for all $a \in A$
\[ q_1 \xrightarrow{[g_i][a]} q_1' \implies \exists q'_2 : q_2 \xrightarrow{[g_2][a]} q'_2 \land (g_1 \implies g_2) \land (q_1' \preceq q'_2) \] (4)

$M_2$ simulates $M_1$, written $M_2 \preceq M_1$, when $s_0 \preceq s_0$. The following is a fact will be useful later when we define contract composition.

**Proposition 2:** If $M_1 \preceq M_2$ and $M_2 \preceq M_3$, then $M_1 \times M_3 \preceq M_2 \times M_4$. 

**Proof:** The start state of $M_1 \times M_3$ can be written as $(q_1, q_3)$ where $q_1$ and $q_3$ are start states of $M_1$ and $M_3$. By Definition 5, the start states $q_2, q_4$ of $M_2$ and $M_4$ satisfy $q_1 \preceq q_2$ and $q_1 \preceq q_4$. We claim $(q_1, q_3) \preceq (q_2, q_4)$. Suppose for some $q'_1, q'_3$, such that $(q_1, q_2) \xrightarrow{[g_1][\alpha]} (q'_1, q'_3)$ where $q_1 \xrightarrow{[g_1][\alpha]} q'_1, q_3 \xrightarrow{[g_1][\alpha]} q'_3$, and $\alpha = \alpha_1 + \alpha_2$. By Definition 5, we have $q_2 \xrightarrow{[g_2][\alpha]} q'_2 \land (g_1 \implies g_2) \land (q_1' \preceq q'_2)$.
and $q_4 \models [g_4 \alpha_2] (q_4' \land (g_3 \Rightarrow g_4) \land q_4 \leq q_4')$. These two yield the transition $(q_2, q_4) \xrightarrow{[g_2 \land g_4] \alpha + \alpha_2} (q_4', q_4')$ in $M_2 \times M_4$ and $g_1 \land g_3 \Rightarrow g_2 \land g_4$. By performing this argument inductively, we have $(q_1', q_3') \leq (q_2', q_4')$ and therefore $q_1 \times q_3 \leq q_2 \times q_4$, from which the claim follows.

**Component**

While the interface is a mathematical object that specifies actions a module can exchange with its environment, a component is any structure (e.g., hardware or software, or human) that satisfies the promises of the interface for that module. To keep the interface representation compact, it is helpful to maintain a small action alphabet. In applications, this may be done by appropriately mapping the numerous actions (due to parametrization or the fact that they stem from different structures) to a small number of classes that represent the actions in the alphabet of the corresponding interface. For a component $K$, letting $\mathcal{O}$ be this action equivalence map, we have the following definition.

**Definition 6 (Component):** We say a component $K$ models an interface $M$ under the action equivalence map $\mathcal{O}$ and write $K \models \mathcal{O} M$ if there exists a state machine representation $\hat{K}$ of $K$ modulo $\mathcal{O}$ such that $\hat{K} \leq M$.

Immediately from the definition, we have for all interfaces $M_1$ and $M_2$, $M_1 \leq M_2 \implies M_1 \models \mathcal{O} M_2$, where $\mathcal{O}$ is the identity map.

**Contract**

At the top of the contract design hierarchy is the contract object, which is defined as follows.

**Definition 7 (Guarded Modal Interface Contracts):** A guarded modal interface contract $\mathcal{C}$ consists of a set of reference variables $\Psi$ and a tuple of the form $\mathcal{A} = (S, S_0, \bot, A, \rightarrow, \rightarrow)$, where $S, S_0, \bot, A$ are defined as in the interface automaton object, $\rightarrow$ and $\rightarrow$ are two transition relations called must and may respectively. Intuitively, a may transition with guard $g$ and action $\alpha$ in the interface contract specifies that any interface implementing the contract is allowed but not required to perform $\alpha$ as long as the guard is satisfied. On the other hand, a must transition in the interface contract specifies a transition that any interface implementing it is required to include. Clearly, this implies that any must transition must also be a may transition, namely for $q \in S, g_1, g_2 \in G_{\Psi}$, and $\alpha \in A$, we have

$$(q \models [g_1] \alpha) \implies (q \models [g_2] \alpha) \land (g_1 \implies g_2) \tag{5}$$

(5) says that the existence of a must transition implies the existence of a may transition with a weaker guard. A modal interface contract $\mathcal{C}$ naturally induces two interface automata $M_{\text{must}}$ and $M_{\text{may}}$ with only $\rightarrow$ and $\rightarrow$ as transition relations respectively and fixes a set of environments of the contract, denoted by $E_{\mathcal{C}}$. An environment $E \in E_{\mathcal{C}}$ is an interface automaton such that $E \times M_{\text{may}}$ is closed (by (5), $E \times M_{\text{must}}$ is also closed) and for each reachable state $(q_E, q_M) \in E \times M_{\text{may}}$ and any (unprefix) action $a$

$$q_E \xrightarrow{[g_E] a} q_M \xrightarrow{[g_M] ?, \# a} (g_E \implies g_M) \tag{6}$$

$V \models [g_E] a \implies q_M \models [g_M] ?, \# a \land (g_E \implies g_M) \tag{7}$

Here $\#$ indicates that $\alpha$ can be either input or internal. Together, these mean that any time the environment is only willing to emit an output or request an input if $M_{\text{may}}$ can accept it. $\mathcal{C}$ also fixes a set of interfaces $M_{\text{E}}$ that implement $C$, such that $M \in M_{\text{E}}$ if

$$M_{\text{must}} \leq M \leq M_{\text{may}} \tag{8}$$

which is essentially stating that all reachable must transitions must be included in $M$, and $M$ can only use may transitions of the contract. Below, we will use as a shorthand $\leq_{\text{may}(\text{must})}$ as the simulation relation with respect to the may(must) transitions only in the contract object. Contract refinement, conjunction, and composition are defined as follows.

**Definition 8 (Modal Refinement):** Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two guarded modal interface contracts. Then $\mathcal{C}_2$ refines $\mathcal{C}_1$, written $\mathcal{C}_2 \preceq \mathcal{C}_1$ if and only if

$$M_{\text{2\_may}} \leq M_{\text{1\_may}} \tag{9}$$

$$M_{\text{1\_must}} \leq M_{\text{2\_must}} \tag{10}$$

**Proposition 3:** A more refined contract allows for more environments, namely

$$\mathcal{C}_2 \preceq \mathcal{C}_1 \implies E_{\mathcal{C}_2} \supseteq E_{\mathcal{C}_1} \tag{11}$$

**Proof:** Let $E \in E_{\mathcal{C}_1}$. By Definitions 5 and (9), we have $M_{\text{2\_may}} \leq_{\text{may}} M_{\text{1\_may}}$ and for any reachable state $(q_E, q_2)$ of $E \times M_{\text{2\_may}}$, there exists a reachable state $(q_E, q_1)$ in $E \times M_{\text{1\_may}}$ such that $q_2 \leq q_1$. So for any outgoing transition of $(q_E, q_2)$ in $E \times M_{\text{2\_may}}$ doing an action $\alpha$, there is a corresponding transition from $(q_E, q_1)$ in $E \times M_{\text{1\_may}}$ that also does $\alpha$. Since $E \times M_{\text{1\_may}}$ is closed, $\alpha$ must be an internal action. Therefore $E \times M_{\text{2\_may}}$ is also closed. Furthermore,

$$q_E \xrightarrow{[g_E] a} q_1 \xrightarrow{[g_1], ?, \# a} \tag{9} q_2 \xrightarrow{[g_2], ?, \# a}$$

and

$$E \xrightarrow{E_{\mathcal{C}_2}} g_1 \xrightarrow{g_2}$$

satisfying (6). Similarly, (7) also holds, implying $E \in E_{\mathcal{C}_2}$. $\blacksquare$

**Proposition 4:** A contract is more refined than another if and only if its implementations are also the other’s implementations.

$$\mathcal{C}_2 \preceq \mathcal{C}_1 \iff M_{\mathcal{C}_2} \subseteq M_{\mathcal{C}_1} \tag{12}$$

**Proof:** ($\Rightarrow$): Let $M \in M_{\mathcal{C}_2}$, by (9) we have $M \leq M_{\text{2\_may}} \leq M_{\text{1\_may}}$. By (10), we have $M_{\text{1\_must}} \leq M_{\text{2\_must}}$ and therefore $M_{\text{1\_must}} \leq M$. This proves $M$ has property (8).

($\Leftarrow$): First, we have $M_{\text{2\_may}} \in M_{\mathcal{C}_2} \subseteq M_{\mathcal{C}_1} \implies M_{\text{2\_may}} \leq M_{\text{1\_may}}$. On the other hand, $M_{\text{2\_must}} \subseteq M_{\mathcal{C}_2} \subseteq M_{\mathcal{C}_1}$ and hence $M_{\text{1\_must}} \subseteq M_{\text{2\_must}}$. This shows that $\mathcal{C}_2 \preceq \mathcal{C}_1$.

Propositions (3) and (4) immediately yield

**Corollary 1:** Modal refinement and metatheoretic refinement are equivalent. Contract refinement allows us to compare levels of abstractions of contracts; for instance, a contract that involves details on how to perform local control actions may refine a contract for a car driving safely into a traffic intersection.
The conjunction of two contracts $C_1$ and $C_2$ is defined as the greatest common lower bound (GCLB) of $C_1$ and $C_2$, or in other words, the most abstract contract $C$ that refines both $C_1$ and $C_2$.

**Definition 9 (Contract Conjunction):** Conjunction is defined for two modal interface contracts $C_1$ and $C_2$ if $A_1$ and $A_2$ are equal and have the same decomposition. Then the pre-conjunction $C_1 \land C_2$ has states $S = S_1 \times S_2$, start state $s_{0,12} = s_{0,1} \times s_{0,2}$ and the same alphabet as $C_1$ and $C_2$, with transitions defined by the following relations, assuming for any subscript $i$ if a transition from $q_i$ to $q'_i$ doesn’t exist then we add it in with a False guard

$$
(q_1, q_2) \xrightarrow{[g_1 \land g_2] \alpha} (q'_1, q'_2) \iff q_1 \xrightarrow{[g_1] \alpha} q'_1 \land q_2 \xrightarrow{[g_2] \alpha} q'_2
$$

(13)

A state $(q_1, q_2)$ of $C_1 \land C_2$ is illegal if it is inconsistent, that is, the “must implies may” condition in (5) does not hold. Specifically, if there exists $\alpha \in A$ such that $(q_1, q_2) \xrightarrow{[g_1] \alpha} \land (q_1, q_2) \xrightarrow{[g_2] \alpha} q_1 \Rightarrow q_2$, then we prune it by deleting all may transitions leading to $(q_1, q_2)$, if $(q_1, q_2)$ a start state, it will simply get removed. Repeating this procedure and deleting all non-may reachable states yields the conjunction $C_1 \land C_2$ (note that the deletion must terminate because the number of states is finite).

**Proposition 5:** $C_1 \land C_2$ has a start state if and only if $C_1$ and $C_2$ have a common lower bound.

**Proof:** ($\Rightarrow$): We prove $C_1 \land C_2 \preceq C_1, C_2$, by showing (9) and (10). Letting $(q_1, q_2)$ be the start state of $C_1 \land C_2$, we have for $i \in \{1, 2\}$ and any $(q'_1, q'_2)$ in $C_1 \land C_2$ such that $(q_1, q_2) \xrightarrow{[g_1] \alpha} (q'_1, q'_2)$, we have by (13) that $q_i \xrightarrow{[g_1] \alpha} q'_i$ and continuing inductively, we conclude $(q_1, q_2) \preceq_{\text{may}} q_i$. Fixing $i$, for any $q_i'$, $\beta$ such that $q_1 \xrightarrow{[g_1] \beta} q'_i$, by (14), $(q_1, q_2) \xrightarrow{[g_1 \lor g_2] \beta} (q'_1, q'_2)$ in $C_1 \land C_2$. Since $(q_1, q_2)$ is not illegal, so is $(q'_1, q'_2)$, because otherwise, the may transition that performs $\beta$ from $(q_1, q_2)$ to $(q'_1, q'_2)$ would have been deleted during pruning, violating (5) for $(q_1, q_2)$. This shows that $q_i \preceq_{\text{must}} (q_1, q_2)$.

($\Leftarrow$): Suppose $C \preceq C_1, C_2$. Instead of showing the start state of $C_1 \land C_2$ is not pruned, we will prove a stronger result, namely that $C \preceq C_1 \land C_2$. Indeed, if $q$ is the start state of $C$, then by definition, the start state $q_i$ of $C_i$ for $i = 1, 2$ satisfies for $\alpha \in A$, $(q_i \xrightarrow{[g_i] \alpha} q'_i) \Rightarrow (q \xrightarrow{[g_1] \alpha} q'_1) \land (q \xrightarrow{[g_2] \alpha} q'_2) \land (q \preceq_{\text{may}} q'_1)$ and $g' \Rightarrow g'_1 \land g'_2$ and $q \preceq_{\text{may}} q'_i$. By (9), we have $q \xrightarrow{[g_1] \alpha} q' \Rightarrow (q \xrightarrow{[g_1 \lor g_2] \beta} (q'_1, q'_2))$. Since $q$ is not illegal, there is an $\alpha \in A$ such that $\beta = \alpha$ and also $g'' = g'_1$, then by determinism $q' = q''$ and $q'_i = q''_i$. Clearly, $q'_1 \lor q'_2 \Rightarrow g'' \Rightarrow g'_1 \land g'_2$ so that the must transition from $(q_1, q_2)$ to $(q'_1, q'_2)$ doing $\beta$ is also legal. Finally, continuing this chain of inductive reasoning, we obtain (9) and (10) for $C$ and $C_1 \land C_2$, proving the claim.

**Proposition 5** and the stronger result shown in the reverse direction of its proof imply

**Proposition 6:** Modal conjunction and metatheoretic conjunction are equivalent, that is, the modal conjunction of two contracts is their GCLB.

**Definition 10 (Contract Composition):** Contract composition is denoted by the operator $\otimes$. The pre-composition $C_1 \otimes C_2$ of two contracts $C_1$ and $C_2$ is given by

$$M_{1 \otimes 2, \text{must}} = M_{1, \text{must}} \times M_{2, \text{must}}$$

$$M_{1 \otimes 2, \text{may}} = M_{1, \text{may}} \times M_{2, \text{may}}$$

A state $(q_1, q_2)$ of $C_1 \otimes C_2$ is illegal if one automaton attempts to supply an input but the other refuses it. Furthermore, state $(q_1, q_2)$ is illegal if it is impossible for the guards to match in input/output matching, resulting in the input being rejected. Define $\text{SAT}_\forall$ to be the set of satisfiable predicates over $\forall$. For $i \in \{1, 2\}$, assuming $g_i$ is satisfiable, then $(q_i, q_{3-i})$ is illegal if either there exists $\alpha \in A_i$ such that

$$q_i \xrightarrow{[g_i] \alpha} q' \land (q_{3-i} \xrightarrow{[g_{3-i}] \alpha} \land (g_i \land g_{3-i} \in \text{SAT}_\forall))$$

Pruning of states is done as in contract conjunction. This new contract is $C_1 \otimes C_2$. And we have the following result.

**Proposition 7:** Modal composition is equivalent to metatheoretic contract composition.

**Proof:** It suffices to show that, for $M_1 \in M_{C_1}, M_2 \in M_{C_2},

1) M_1 \times M_2 \in M_{C_1 \otimes C_2}$

2) For all $E \in E_{C_1 \otimes C_2}, E \times M_2 \in E_{C_1}$ and $E \times M_1 \in E_{C_2}$

3) $C_1 \otimes C_2$ is the least contract with respect to refinement that satisfies these.

First we show $C_1 \otimes C_2$ satisfies conditions 1 and 2. Let $C = C_1 \otimes C_2$. Then condition 1 is equivalent to $M_{\text{must}} \preceq M_1 \times M_2 \preceq M_{\text{may}}$. Since, for $i \in \{1, 2\}$, $M_i \in M_{C_i}, M_{i, \text{must}} \preceq M_{i, \text{may}}$, the desired result immediately follows from Proposition 2. Next consider some $E \in E_{C_1 \otimes C_2}, E \times M_1 \times M_2$ is closed. It follows that $(E \times M_{1, \text{may}}) \times M_{2, \text{may}}$ and $(E \times M_{2, \text{may}}) \times M_{1, \text{may}}$ are also closed. By definition, $M_1 \preceq M_{1, \text{may}}$ and $M_2 \preceq M_{2, \text{may}}$, so $(E \times M_1) \times M_{2, \text{may}}$ and $(E \times M_2) \times M_{1, \text{may}}$ are closed. It then remains to show $E \times M_1$ and $E \times M_2$ satisfy (6) and (7) of Definition 7. First, since $E$ is an environment of $C_1 \otimes C_2$, we have for any reachable state $(q, q_1, q_2)$ in $E \times M_{C_1 \otimes C_2}$

$$qE \xrightarrow{[gE] [\alpha]} q_1 \xrightarrow{[g_1] \alpha} \land (gE \Rightarrow g_1 \otimes g_2)$$

$$qE \xrightarrow{[gE] [\alpha]} q_2 \xrightarrow{[g_2] \alpha} \land (gE \Rightarrow g_1 \otimes g_2)$$

Note that since $M_{1 \otimes 2, \text{must}} = M_{1, \text{must}} \times M_{2, \text{must}}$, these are equivalent to

$$qE \xrightarrow{[gE] [\alpha]} q_1 \xrightarrow{[g_1] \alpha} \land q_2 \xrightarrow{[g_2] \alpha} \land (gE \Rightarrow g_1 \land g_2)$$
\[ q_E \overset{[g_E, \alpha]}{\rightarrow} q_1 \quad q_1 \overset{[g_1, \sim_1, \alpha]}{\rightarrow} q_2 \quad q_2 \overset{[g_2, \sim_2, \alpha]}{\rightarrow} (g_E \Rightarrow g_1 \land g_2) \]

where \( \sim_1 \) and \( \sim_2 \) are action types such that their composition matches that of \( \mathcal{C}_1 \otimes \mathcal{C}_2 \). The following chart demonstrates possible action types of this transition.

<table>
<thead>
<tr>
<th>( E )</th>
<th>( \mathcal{C}_1 )</th>
<th>( \mathcal{C}_2 )</th>
<th>( E \times M_{1\text{.must}} )</th>
<th>( E \times M_{2\text{.must}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>?</td>
<td>?</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>!</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>!</td>
<td>#</td>
<td>?</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>!</td>
<td>#</td>
<td>!</td>
<td>!</td>
<td>#</td>
</tr>
<tr>
<td>?</td>
<td>!</td>
<td>!</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>?</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>?</td>
<td>#</td>
<td>!</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>?</td>
<td>#</td>
<td>?</td>
<td>#</td>
<td>?</td>
</tr>
</tbody>
</table>

The proof proceeds as follows. For \( M_1 \) implementing \( \mathcal{C}_1 \) and \( M_2 \) implementing \( \mathcal{C}_2 \), note that for \( i \in \{1, 2\} \)

\[ q_i \overset{[g_i, \alpha]}{\rightarrow} q_{M_i} \quad \overset{[g_{M_i}, \alpha]}{\rightarrow} q_2 \quad \overset{[g_2, \alpha]}{\rightarrow} \land (g_E \Rightarrow g_{M_1} \land g_2), \]

so the composition of the transitions from \( q_E \) and \( q_{M_1} \) in \( E \times M_1 \) yields

\[ q_{(E,M_1)} \overset{[g_E, g_{M_1}, \alpha]}{\rightarrow} q_2 \quad \overset{[g_2, \alpha]}{\rightarrow} \land (g_E \Rightarrow g_{M_1} \land g_2), \]

and the equivalent result for \( E \times M_2 \) yields

\[ q_{(E,M_2)} \overset{[g_E, g_{M_2}, \alpha]}{\rightarrow} q_1 \quad \overset{[g_1, \alpha]}{\rightarrow} \land (g_E \Rightarrow g_{M_2} \land g_1), \]

The latter result is precisely (7) with respect to \( E \times M_2 \) and \( \mathcal{C}_1 \). It can be easily verified in a similar manner that the rest of the combinations yield similar results. For condition 3, it suffices to show that \( \mathcal{C}_1 \otimes \mathcal{C}_2 \) is the greatest lower bound of all contracts that satisfy conditions 1 and 2. Thus, for any \( \mathcal{C}_s \) satisfying 1 and 2, then \( \mathcal{C}_1 \otimes \mathcal{C}_2 \succeq \mathcal{C}_s \). This follows immediately from condition 1 and Proposition 2, since \( M_1 \times M_2 \in M_{\mathcal{E}} \), yields, as desired

\[ M_{s, \text{must}} \subseteq M_1 \times M_2 \subseteq M_{s, \text{may}} \]
\[ \Rightarrow M_{s, \text{must}} \subseteq M_{1, \text{must}} \times M_{2, \text{must}} \subseteq M_1 \times M_2 \]
\[ \leq M_{1, \text{may}} \times M_{2, \text{may}} \subseteq M_{s, \text{may}} \]
\[ \Rightarrow M_{s, \text{must}} \leq M_{1, \text{may}} \times M_{2, \text{may}} \subseteq M_{s, \text{may}}. \]

So far, we have only defined contract operations for contracts with matching alphabet conditions. Alphabet equalization is achieved via the same procedure described in [3]. May self-loops are temporarily added during the computation of the conjunction and must self-loops added for composition both having \( \top \) as their guards.

**AN APPLICATION IN TRAFFIC CONTROL**

We will apply the developed theory to the contract-based design of the real-time networked control traffic system illustrated in Fig. 1. A full simulation of this system is presented at [10]. This system consists of 4 interacting components whose temporal behaviors are described by the contracts \( \mathcal{E}_\text{lights}, \mathcal{E}_\text{pedestrian}, \mathcal{E}_\text{vehicle} \) and \( \mathcal{E}_\text{scheduler} \) shown in Fig. 3. These specify the desired models for pedestrians, traffic lights, cars, and a scheduler in the intersection. We note that the many continuous variables involved in the timers and execution conditions of these components would have made producing and deciphering their contracts in the vanilla modal interface framework significantly more challenging due to the need for numerous potentially confounding auxiliary states and actions.

The traffic lights, in addition to some timing constraints on the duration of the “red”, “green”, “yellow” signals, are also required to have an “all red” phase that lasts for \( t_c \) seconds, a period long enough for cars to clear the intersection before the walk signal with duration \( t_w \) is turned on. Pedestrians should only attempt to cross when they are capable of successfully landing on the other island for the duration of the walk signal. All (or at least some) vehicles involved are robots that can be informed by a centralized planner on how to proceed past the intersection without causing accidents. These directions must be requested by the robots upon entrance. The traffic lights and the pedestrians form a subsystem \( \mathcal{E}_\text{lights} \otimes \mathcal{E}_\text{pedestrian} \) that operates orthogonally to the subsystem \( \mathcal{E}_\text{vehicle} \otimes \mathcal{E}_\text{scheduler} \) defined by the cars and the scheduler. By orthogonality, the overall system is simply \( (\mathcal{E}_\text{lights} \otimes \mathcal{E}_\text{pedestrian}) \land (\mathcal{E}_\text{vehicle} \otimes \mathcal{E}_\text{scheduler}) \).

To simplify the process of writing the interface contract for the traffic lights, we specify and compose two separate subcontracts for traffic lights in each direction, \( \mathcal{E}_\text{horizontal_lights} \) and \( \mathcal{E}_\text{vertical_lights} \), for the east-west and north-south directions respectively, that is \( \mathcal{E}_\text{lights} = \mathcal{E}_\text{horizontal_lights} \otimes \mathcal{E}_\text{vertical_lights} \). Since \( \mathcal{E}_\text{horizontal_lights} \) and \( \mathcal{E}_\text{vertical_lights} \) are symmetric with the exception of the start state (the former starts at node 0 while the latter starts at node 3), we only show the former in Fig. 3. The variables for \( \mathcal{E}_\text{horizontal_lights} \) are \( \mathcal{h}, \mathcal{h'}, \mathcal{h}_\text{timer} \), which represent the traffic lights’ current state, the next state after performing a related action and a special timer to specify the minimum durations to allow for vehicles to finish clearing the intersection \( t_c \) and for the walk signal \( t_w \). The traffic light states are \( \mathcal{r} \) for red, \( \mathcal{y} \) for yellow, and \( \mathcal{g} \) for green. The output actions are \( \mathcal{r}_h, \mathcal{y}_h, \mathcal{w}_h \), which serve to announce that the current state is red or that the walk sign for lanes in the north-south directions is on. The input signal is \( ?x_v \), denoting a safety check with the state of the lights in the north-south direction. As can be seen in the automaton, via the may transition, we also allow the traffic lights to potentially bypass the yellow phase in transitioning from green to red. The contract \( \mathcal{E}_\text{pedestrian} \) is more simple. Its variable is \( t_\text{cross} \) which denotes the minimum time it takes the pedestrian to cross the street and the input actions are \( ?x_v \text{ walk} \) and \( ?v \text{ walk} \) which, in that
order, denote a crossing action in the north-south and east-west directions of the pedestrian (both of these actions need to synchronize with a walk signal from the corresponding traffic lights). Note that both transitions in this contract are optional. The composition $c_{\text{horizontal}_\text{lights}} \otimes c_{\text{vertical}_\text{lights}}$ was computed automatically with the code in [10] and shown in Fig. 4. Though not included here to economize space, composing this with $c_{\text{pedestrian}}$ closes all the remaining output actions in Fig. 4.

The scheduler contract automaton has access to a variable $\text{len}(\text{request\_queue})$, which is the length of the request queue. In addition, $c_{\text{scheduler}}$ has one input action, $\text{?request}$, which denotes a check for whether there is a new request from a vehicle trying to travel through the intersection. Its output actions are $!\text{reject}$ and $!\text{accept}$ denoting whether the scheduler decides to accept or reject the request, and $!\text{primitives}$ denoting the sending of controlling signals to the requesting vehicle. The internal action is $\#\text{processing}$, corresponding to the internal computation of the controller. Observe that the scheduler must be able to accept requests under any condition (by the True guard) but can only process the request if the queue length is greater than 0. $c_{\text{vehicle}}$ has a variable not_done which keeps track of whether the original request has been carried out to completion. As can be expected, the car can make a request with $!\text{request}$ and receive signals from the scheduler with the action $\text{?reject}$, $\text{?accept}$, and $\text{?primitives}$. Composing $c_{\text{scheduler}}$ with $c_{\text{vehicle}}$ yields the third system shown in Fig. 3. Observe that this system is also closed.

By Definition 6, checking that an implementation is a component whose interface satisfies the corresponding contract involves finding action equivalence maps between the implementation and the interface. To illustrate this process, consider the action of sending and receiving primitive commands of the scheduler and the vehicle, $!\text{primitives}$ and $\text{?primitives}$. For a reasonable autonomous traffic intersection, the class of actions that qualify as the action primitives are those control signals that result in a safe and deadlock-free operation of all vehicles. We propose an implementation based on computing robust controllers or “primitives” that can restrict the vehicles to a waypoint graph structure even in the presence of stochastic disturbance. In particular, the vehicle dynamics are given by

$$
\dot{v} = a + w_1, \quad \theta = \frac{\pi}{2} \tan(\delta + w_2), \quad \dot{x} = v \cos(\theta), \quad \dot{y} = v \sin(\theta),
$$

with velocity $v$, orientation $\theta$, positions $x$ and $y$ as state variables; acceleration $a \in [-9.8, 9.8] \frac{\text{m}}{\text{s}^2}$ and steering angle $\delta \in [-0.9, 0.9] \frac{\text{rad}}{\text{s}}$ as controllable inputs; $w_1 \in [-1.1, 1.1] \frac{\text{rad}}{\text{s}}$ and $w_2 \in [-0.065, 0.065] \frac{\text{rad}}{\text{s}}$ as uncontrollable disturbances; and vehicle length $L = 2.8m$.

We use a formal, set-based algorithm [13], [14], [15] to obtain controllers that steer cars from one node to another on the waypoint graph with each node being a set of states of the car’s dynamics around a nominal state. The reason a set of states is used is because of the disturbance present. This low-level controller ensures the satisfaction of input constraints and provides the occupancy sets of the vehicles, each of which represents a directed edge in the graph. The set-based controller computes a reference trajectory, a feedback controller to track this reference trajectory, and the corresponding reachable set of states. For any states $p, q$ of the vehicle, let $\mathcal{X}_0(p)$ and $\mathcal{X}_f(q)$ denote the initial and final sets around $p$ and $q$ respectively. By construction, the primitive controller steers in a fixed time $t_{12}$ from the initial set $\mathcal{X}_0(p_1)$ around a nominal waypoint $p_1$ to a final set $\mathcal{X}_f(p_2)$ around $p_2$. Constraining $\mathcal{X}_f(p_2) \subseteq \mathcal{X}_0(p_1)$ allows any trajectory in the edge that starts from $\mathcal{X}_0(p_1)$ and ends in $\mathcal{X}_f(p_2)$ to be concatenated with any trajectory starting in $\mathcal{X}_0(p_2)$. In this way, long chains of primitive commands that span multiple (directed) edges can be formed from unit commands spanning a single edge, this justifies treating the scheduling problem as a graph routing problem to which we propose Algorithm 1 as a solution. In Algorithm 1, the $\text{SCHEDULE}(\text{request\_queue}, \text{timetable})$ function, taking two variables representing a queue of requests and a scheduling timetable is called repeatedly to rendezvous with new requests. Each time, it extracts the request from a certain car in the form of a starting configuration and an ending configuration. From this information, the scheduling algorithm finds a path in the primitive graph that connects these configurations and consults with the scheduling timetable to see if the path is safe and legal. If it is, the scheduler will send the primitives (each is of a fixed, known time length) to the requesting car, otherwise, to improve efficiency, it will attempt to find a safe and legal transit node along the path to temporarily send the car to. If such a node is found, it will send the corresponding primitives. If not the request will be rejected. Proof details regarding the correctness of this algorithm mainly rely on the use of the timetable to avoid conflicts and illegal actions. Under this algorithm, $!\text{primitives}$ and $\text{?primitives}$ actions therefore correspond to the $\text{SEND\_PRIMITIVES}()$ calls in the pseudocode.

**Algorithm 1** The scheduling algorithm

```python
function SCHEDULE(request_queue, timetable) {
    path, car ← EXTRACT_REQUEST(request_queue)
    if IS_SAFE(path, timetable) then
        SEND_PRIMITIVES(path, car, timetable)
    else if EXISTS_TRANSIT_NODE(path, timetable) then
        transit ← FIND_TRANSIT_PATH(path, timetable)
        SEND_PRIMITIVES(transit, car, timetable)
        request_queue.ADD_LEG(path, car, transit)
    else
        request_queue.READD(path, car)
}
```

**Conclusion and future work**

In this work, we introduce and show that our theory of guarded modal interface contracts does retain many metatheoretic properties from [2], which is not only evidence that it has reasonable semantics but also a step closer to unifying many different existing contract protocols under a common formal structure. We then demonstrate the expressiveness of
this theory by using it to specify in a compact manner various components involved in a traffic intersection with variables taking on a continuous range of values that the vanilla modal interface theory would have struggled or failed to capture.

For future work, we would like to consider methods to further automate contract synthesis and verification which may or may not involve defining new objects and expanding the algebra to include new operators. Extending the current framework to include specifications in rich specification languages like TLA+ [7] is also an interesting direction.

REFERENCES


