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

Tung Phan-Minh[1], 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.

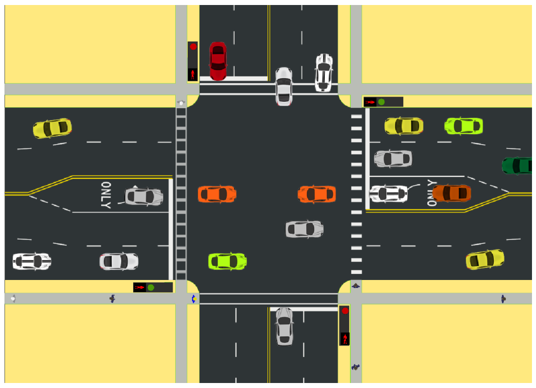

Fig. 1. A snapshot of an implementation of a networked control system of traffic presented in [10]

Interface Contract Theory

Many real-world applications ranging from online payment services to autonomous robots require networking

*This work was supported by DENSO, the European Commission under the project UnCoVerCPS (grant number 643921), and the Caltech SURF program. We would like to thank Anhminh Nguyen for contributing to the simulation implementation.

[1]Tung Phan-Minh is a graduate student in Mechanical Engineering at the California Institute of Technology tung@caltech.edu

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 $\mathscr{U}$ 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 $\mathscr{U}$ (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 radiowaves.

For any set of variables $\mathscr{V} \subseteq \mathscr{U}$, an evaluation e of $\mathscr{V}$, denoted by $e[\mathscr{V}]$ is a legal assignment of values to each of the variables in $\mathscr{V}$. 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 $\mathscr{V}$. The set $E_{\mathscr{V}} := \{e \mid e \text{ is an evaluation of } \mathscr{V}\}$ contains all possible evaluations of $\mathscr{V}$. 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 $\top := \texttt{True}$ and $\bot := \texttt{False}$. A *guard* g defined on a set of variables $\mathscr{V}$ is a predicate on the variables in this set, namely, $g : E_{\mathscr{V}} \to \{\top, \bot\}$. The set of all predicates on $\mathscr{V}$ is denoted by $G_{\mathscr{V}}$.

For example, when $\mathscr{V}$ is a set of Boolean variables, then a guard on $\mathscr{V}$ is a map from $2^{\mathscr{V}}$ 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 $\mathscr{V}$ and a tuple $\mathscr{A} = (S, s_0, \bot, A, \to)$ 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
$$A = ?A \cup !A \cup \#A,$$
where $?A, !A, \#A$ are the smallest sets containing all the input, output, and internal actions of A respectively.
(v) $\to \subset S \times [G_{\mathscr{V}} \mid A] \times \bar{S}$ is a guarded transition relation with $\bar{S} = S \cup \{\bot\}$ (note the asymmetry between the start and end sets). For all s_1, s_2, g, a such that $s_1 \times [g \mid \sim a] \times s_2 \in \to$, we say that the action $\sim a$ is only available to M in those states of the universe where g evaluates to $\top$. We also require the transitions to be *deterministic* by requiring that, from each state, there is only one transition per unmasked action.

We will be writing $q \xrightarrow{[g|\alpha]} p$ in place of $q \times [g_E \mid \alpha] \times p \in \to$ as a predicate. The fact that our interfaces are deterministic allows us also to use the shorthand $q \xrightarrow{[g|\alpha]}$ to say with little ambiguity that there exists a state $p \in \bar{S}$ such that $q \xrightarrow{[g_E|\alpha]} p$. 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 in 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 $g_1 \wedge g_2$ is satisfiable (there exists an evaluation of the variables in $g_1 \wedge g_2$ that makes it evaluate to $\texttt{True}$). 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* $t_1 = q_1 \xrightarrow{[g_1|\sim a_1]} q_1'$ and $t_2 = q_2 \xrightarrow{[g_2|\sim a_2]} q_2'$, from automata M_1 and M_2 respectively, is possible if $g_1 \wedge g_2$ is satisfiable and $a_1 = a_2$. If these conditions are satisfied, the composed transition is given by $t = (q_1, q_2) \xrightarrow{[g|a]} (q_1', q_2')$, where $g := g_1 \wedge g_2$ 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 $i \in \{1, 2\}$

$$\sim a_1 + \sim a_2 := \begin{cases} \#a_1 & \bigvee_{i=1}^{2} (\sim a_i \in !A_i \wedge \sim a_{3-i} \in ?A_{3-i}) \\ & \vee \sim a_i = \#a_i \\ ?a_1 & \sim a_1 \in ?A_1 \wedge \sim a_2 \in ?A_2 \\ !a_1 & \sim a_1 \in !A_1 \wedge \sim a_2 \in !A_2 \end{cases}$$

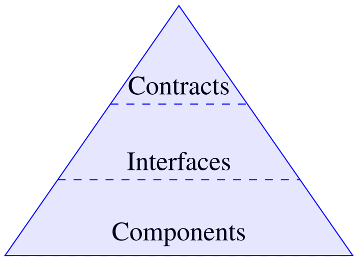

Fig. 2. Contract design hierarchy pyramid

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 $\mathscr{V}_1$ and $(S_1, s_{0,1}, \perp_2, A_1, \rightarrow_1)$ and M_2 defined by $\mathscr{V}_2$ and $(S_2, s_{0,2}, \perp_2, A_2, \rightarrow_2)$, their *composition* $M = M_1 \times M_2$ is defined by $\mathscr{V} := \mathscr{V}_1 \cup \mathscr{V}_2$ and $(S, s_0, \perp, A \rightarrow\ )$ such that

(i) $S := S_1 \times S_2$
(ii) $s_0 := s_{0,1} \times s_{0,2}$
(iii) A is partitioned into $\#A \cup !A \cup ?A$, with

$$\begin{aligned} \#A \quad &:= \#A_1 \cup \#A_2 \bigcup_{i=1}^{2} \#(u(?A_i) \cap u(!A_{3-i})) \\ ?A \quad &:= ?A_1 \cup ?A_2 \\ !A \quad &:= !A_1 \cup !A_2 \end{aligned}$$

(iv) $\rightarrow$ is the Cartesian product of transitions in $\rightarrow_1$ and $\rightarrow_2$ with respect to composition as defined in definition 3. More specifically, $\forall t_1 \in S_1 \times [G_{\mathscr{V}_1} \mid A_1] \times \bar{S_1}$, $t_2 \in S_2 \times [G_{\mathscr{V}_2} \mid A_2] \times \bar{S_2}$ such that t_1 and t_2 are composable, $t_1 \times t_2 \in \rightarrow$.
(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 transtion 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

$$M_1 \times M_2 = M_2 \times M_1 \tag{1}$$

and

$$(M_1 \times M_2) \times M_3 = M_1 \times (M_2 \times M_3). \tag{2}$$

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 $\bigcup_{i=1}^{3} \#A \bigcup_{j=1, j\neq i}^{3} \#(u(?A_i) \cap u(!A_j))$. By Definition 4(iv) it remains to be shown that the transition sets are identical, which is to prove

$$\forall t_1, t_2, t_3. \bigwedge_{i=1}^{3} (t_i \in \rightarrow_i) \implies (t_1 \times t_2) \times t_3 = t_1 \times (t_2 \times t_3) \tag{3}$$

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 # "absorbs" 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_2 *simulates* q_1, written $q_1 \lesssim q_2$, if for all $\alpha \in A$

$$q_1 \xrightarrow{[g_1|\alpha]} q_1' \implies \exists q_2' : q_2 \xrightarrow{[g_2|\alpha]} q_2' \wedge (g_1 \implies g_2) \wedge (q_1' \lesssim q_2') \tag{4}$$

M_2 simulates M_1, written $M_1 \lesssim M_2$, when $s_{0,1} \lesssim s_{0,2}$.

The following is a fact will be useful later when we define contract composition.

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

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 $q_1 \lesssim q_2$ and $q_3 \lesssim q_4$. We claim $(q_1, q_3) \lesssim (q_2, q_4)$. Suppose for some q_1', q_3' such that $(q_1, q_3) \xrightarrow{[g_1 \wedge g_3|\alpha]} (q_1', q_3')$ where $q_1 \xrightarrow{[g_1|\alpha_1]} q_1'$, $q_3 \xrightarrow{[g_3|\alpha_2]} q_3'$ and $\alpha = \alpha_1 + \alpha_2$. By Definition 5, we have $q_2 \xrightarrow{[g_2|\alpha_1]} q_2' \wedge (g_1 \Rightarrow g_2) \wedge q_1' \lesssim q_2'$

and $q_4 \xrightarrow{[g_4|\alpha_2]} q_4' \wedge (g_3 \Rightarrow g_4) \wedge q_3' \lesssim q_4'$. These two yield the transition $(q_2,q_4) \xrightarrow{[g_2 \wedge g_4|\alpha_1+\alpha_2]} (q_2',q_4')$ in $M_2 \times M_4$ and $g_1 \wedge g_3 \Rightarrow g_2 \wedge g_4$. By performing this argument inductively, we have $(q_1',q_3') \lesssim (q_2',q_4')$ and therefore $q_1 \times q_3 \lesssim q_2 \times q_4$, 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 $\mathscr{Q}$ 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 $\mathscr{Q}$ and write $K \models_{comp}^{\mathscr{Q}} M$ if there exists a state machine representation $\bar{K}$ of K modulo $\mathscr{Q}$ such that $\bar{K} \lesssim M$.
Immediately from the definition, we have for all interfaces M_1 and M_2, $M_1 \lesssim M_2 \implies M_1 \models_{comp}^{\mathscr{I}} M_2$. where $\mathscr{I}$ 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* $\mathscr{C}$ consists of a set of reference variables $\mathscr{V}$ and a tuple of the form $\mathfrak{A} = (S, s_0, \perp, A, \rightarrow, \dashrightarrow)$, where $S, s_0, \perp, A$ are defined as in the interface automaton object. $\rightarrow$ and $\dashrightarrow$ are two transition relations called *must* and *may* respectively. Intuitively, a may transition with guard g and action α in the interface contract specifies that any interface implementing the contract is *allowed but not required* to perform α 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 $q \in S, g_1, g_2 \in G_{\mathscr{V}}$, and $\alpha \in A$, we have

$$(q \xrightarrow{[g_1|\alpha]} \implies q \overset{[g_2|\alpha]}{\dashrightarrow}) \wedge (g_1 \implies g_2) \tag{5}$$

(5) says that the existence of a must transition implies the existence of a may transition with a weaker guard. A modal interface contract $\mathscr{C}$ naturally induces two interface automata M_{must} and M_{may} with only $\rightarrow$ and $\dashrightarrow$ as transition relations respectively and fixes a set of *environments* of the contract, denoted by $E_{\mathscr{C}}$. An environment $E \in E_{\mathscr{C}}$ 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 (unprefixed) action a

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

$$q_E \xrightarrow{[g_E|?a]} \implies q_M \xrightarrow{[g_M|!,\#a]} \wedge (g_E \implies g_M) \tag{7}$$

Here ?, # indicates that α 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_{may} can accept it. $\mathscr{C}$ also fixes a set of interfaces $M_{\mathscr{C}}$ that *implement* C, such that $M \in M_{\mathscr{C}}$ if

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

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 $\lesssim_{\text{may(must)}}$ 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 $\mathscr{C}_1$ and $\mathscr{C}_2$ be two guarded modal interface contracts. Then $\mathscr{C}_2$ *refines* $\mathscr{C}_1$, written $\mathscr{C}_2 \preceq \mathscr{C}_1$ if and only if

$$M_{2,\text{may}} \lesssim M_{1,\text{may}} \tag{9}$$

$$M_{1,\text{must}} \lesssim M_{2,\text{must}}. \tag{10}$$

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

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

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

$$q_E \xrightarrow{[g_E|!a]} \overset{E \in E_{\mathscr{C}_1}}{\implies} q_1 \xrightarrow{[g_1|?,\#a]} \overset{(9)}{\implies} q_2 \xrightarrow{[g_2|?,\#a]}$$

and

$$g_E \overset{E \in E_{\mathscr{C}_1}}{\implies} g_1 \overset{(9)}{\implies} g_2$$

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

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

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

Proof: $(\Rightarrow)$: Let $M \in M_{\mathscr{C}_2}$, by (9) we have $M \lesssim M_{2,\text{may}} \lesssim M_{1,\text{may}}$. By (10), we have $M_{1,\text{must}} \lesssim M_{2,\text{must}}$ and therefore $M_{1,\text{must}} \lesssim M$. This proves M has property (8).
$(\Leftarrow)$: First, we have $M_{2,\text{may}} \in M_{\mathscr{C}_2} \subseteq M_{\mathscr{C}_1} \implies M_{2,\text{may}} \lesssim M_{1,\text{may}}$. On the other hand, $M_{2,\text{must}} \in M_{\mathscr{C}_2} \subseteq M_{\mathscr{C}_1}$ and hence $M_{1,\text{must}} \lesssim M_{2,\text{must}}$. This shows that $\mathscr{C}_2 \preceq \mathscr{C}_1$. ■
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 $\mathscr{C}_1$ and $\mathscr{C}_2$ is defined as the greatest common lower bound (GCLB) of $\mathscr{C}_1$ and $\mathscr{C}_2$, or in other words, the most abstract contract $\mathscr{C}$ that refines both $\mathscr{C}_1$ and $\mathscr{C}_2$.

Definition 9 (Contract Conjunction): Conjunction is defined for two modal interface contracts $\mathscr{C}_1$ and $\mathscr{C}_2$ if A_1 and A_2 are equal and have the same decomposition. Then the pre-conjunction $\mathscr{C}_1 \underline{\triangle} \mathscr{C}_2$ 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 $\mathscr{C}_1$ and $\mathscr{C}_2$, 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

$$(q_1,q_2) \overset{[g_1 \wedge g_2|\alpha]}{\dashrightarrow} (q_1',q_2') \Leftrightarrow q_1 \overset{[g_1|\alpha]}{\dashrightarrow} q_1' \wedge q_2 \overset{[g_2|\alpha]}{\dashrightarrow} q_2' \tag{13}$$

$$q_1 \xrightarrow{[g_1|\alpha]} q_1' \vee q_2 \xrightarrow{[g_2|\alpha]} q_2' \Leftrightarrow (q_1,q_2) \xrightarrow{[g_1 \vee g_2|\alpha]} (q_1',q_2') \tag{14}$$

A state (q_1,q_2) of $\mathscr{C}_1 \underline{\triangle} \mathscr{C}_2$ is illegal if it inconsistent, that is, the "must implies may" condition in (5) does not hold. Specifically, if there exists $\alpha \in A$ such that $(q_1,q_2) \xrightarrow{[g_1|\alpha]} \wedge (q_1,q_2) \overset{[g_2|\alpha]}{\dashrightarrow}$ but $g_1 \not\Rightarrow g_2$, 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 $\mathscr{C}_1 \wedge \mathscr{C}_2$ (note that the deletion must terminate because the number of states is finite).

Proposition 5: $\mathscr{C}_1 \wedge \mathscr{C}_2$ has a start state if and only if $\mathscr{C}_1$ and $\mathscr{C}_2$ have a common lower bound.

Proof: $(\Rightarrow)$: We prove $\mathscr{C}_1 \wedge \mathscr{C}_2 \preceq \mathscr{C}_1, \mathscr{C}_2$, by showing (9) and (10). Letting (q_1,q_2) be the start state of $\mathscr{C}_1 \wedge \mathscr{C}_2$, we have for $i \in \{1,2\}$ and any (q_1',q_2') in $\mathscr{C}_1 \wedge \mathscr{C}_2$ such that $(q_1,q_2) \overset{[g_1' \wedge g_2'|\alpha]}{\dashrightarrow} (q_1',q_2')$, we have by (13) that $q_i \overset{[g_i'|\alpha]}{\dashrightarrow} q_i'$ and continuing inductively, we conclude $(q_1,q_2) \lesssim_{\text{may}} q_i$. Fixing i, for any q_i'', β such that $q_i \xrightarrow{[g_i''|\beta]} q_i''$, by (14), $(q_1,q_2) \xrightarrow{[g_1'' \vee g_2''|\beta]} (q_1'',q_2'')$ in $\mathscr{C}_1 \underline{\triangle} \mathscr{C}_2$. Since (q_1,q_2) is not illegal, so is (q_1'',q_2''), because otherwise, the may transition that performs β 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 $q_i \lesssim_{\text{must}} (q_1,q_2)$.
$(\Leftarrow)$: Suppose $\mathscr{C} \preceq \mathscr{C}_1, \mathscr{C}_2$. Instead of showing the start state of $\mathscr{C}_1 \wedge \mathscr{C}_2$ is not pruned, we will prove a stronger result, namely that $\mathscr{C} \preceq \mathscr{C}_1 \wedge \mathscr{C}_2$. Indeed, if q is the start state of $\mathscr{C}$, then by definition, the start state q_i of $\mathscr{C}_i$ for $i = 1,2$ satisfies for $\alpha \in A$, $(q \overset{[g'|\alpha]}{\dashrightarrow} q' \Rightarrow q_i \overset{[g_i'|\alpha]}{\dashrightarrow} q_i') \wedge (g' \Rightarrow g_i') \wedge (q' \lesssim_{\text{may}} q_i')$. Thus $q \overset{[g'|\alpha]}{\dashrightarrow} q' \Rightarrow (q_1 \overset{[g_1'|\alpha]}{\dashrightarrow} q_1' \wedge q_2 \overset{[g_2'|\alpha]}{\dashrightarrow} q_2')$ with $g' \Rightarrow g_1' \wedge g_2'$ and $q' \lesssim_{\text{may}} q_i'$. By (9), we have $q \overset{[g'|\alpha]}{\dashrightarrow} q' \Rightarrow (q_1,q_2) \overset{[g_1' \wedge g_2'|\alpha]}{\dashrightarrow} (q_1',q_2')$ where (q_1,q_2) and (q_1',q_2') are states of $\mathscr{C}_1 \underline{\triangle} \mathscr{C}_2$. Fixing i, for any β

$$q_i \xrightarrow{[g_i''|\beta]} q_i'' \Rightarrow q \xrightarrow{[g''|\beta]} q'' \wedge (g_i'' \Rightarrow g'') \wedge (q_i'' \lesssim_{\text{must}} q'')$$

Also by (14)

$$(q_1,q_2) \xrightarrow{[g_1'' \vee g_2''|\beta]} (q_1'',q_2'')$$

Since q is not illegal, there is an $\alpha \in A$ such that $\beta = \alpha$ and also $g'' \Rightarrow g'$, then by determinism $q' = q''$ and $q_i' = q_i''$. Clearly, $g_1'' \vee g_2'' \Rightarrow g'' \Rightarrow g' \Rightarrow g_1' \wedge g_2'$ so that the must transition from (q_1,q_2) to (q_1',q_2') doing β is also legal. Finally, continuing this chain of inductive reasoning, we obtain (9) and (10) for $\mathscr{C}$ and $\mathscr{C}_1 \wedge \mathscr{C}_2$, 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 $\mathscr{C}_1 \underline{\otimes} \mathscr{C}_2$ of two contracts $\mathscr{C}_1$ and $\mathscr{C}_2$ is given by

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

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

A state (q_1,q_2) of $\mathscr{C}_1 \underline{\otimes} \mathscr{C}_2$ 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 $SAT_{\mathscr{V}}$ to be the set of satisfiable predicates over $\mathscr{V}$. For $i \in \{1,2\}$, assuming g_i is satisfiable, then (q_i, q_{3-i}) is illegal if either there exists $\alpha_i \in A_i$ such that

$$(q_i \overset{[g_i|\alpha_i]}{\dashrightarrow} \wedge \alpha_i \in ?A_{3-i}) \not\Rightarrow q_{3-i} \xrightarrow{[g_{3-i}|\alpha_i]} \wedge (g_i \wedge g_{3-i} \in SAT_{\mathscr{V}})$$

Pruning of states is done as in contract conjunction. This new contract is $\mathscr{C}_1 \otimes \mathscr{C}_2$. And we have the following result.

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

Proof: It suffices to show that, for $M_1 \in M_{\mathscr{C}_1}$, $M_2 \in M_{\mathscr{C}_2}$,

1) $M_1 \times M_2 \in M_{\mathscr{C}_1 \otimes \mathscr{C}_2}$
2) For all $E \in E_{\mathscr{C}_1 \otimes \mathscr{C}_2}$, $E \times M_2 \in E_{\mathscr{C}_1}$ and $E \times M_1 \in E_{\mathscr{C}_2}$
3) $\mathscr{C}_1 \otimes \mathscr{C}_2$ is the least contract with respect to refinement that satisfies these.

First we show $\mathscr{C}_1 \otimes \mathscr{C}_2$ satisfies conditions 1 and 2. Let $\mathscr{C} = \mathscr{C}_1 \otimes \mathscr{C}_2$. Then condition 1 is equivalent to $M_{\text{must}} \lesssim M_1 \times M_2 \lesssim M_{\text{may}}$. Since, for $i \in \{1,2\}$, $M_i \in M_{\mathscr{C}_i}$, $M_{i,\text{must}} \lesssim M_i \lesssim M_{i,\text{may}}$, the desired result immediately follows from Proposition 2. Next consider some $E \in E_{\mathscr{C}_1 \otimes \mathscr{C}_2}$, so $E \times (M_{1,\text{may}} \times M_{2,\text{may}})$ 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, $M_1 \lesssim M_{1,\text{may}}$ and $M_2 \lesssim M_{2,\text{may}}$, 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 $\mathscr{C}_1 \otimes \mathscr{C}_2$, we have for any reachable state $(q_E, q_{1\otimes 2})$ in $E \times M_{1\otimes 2,\text{may}}$

$$q_E \xrightarrow{[g_E|!\alpha]} \Rightarrow q_{1\otimes 2} \xrightarrow{[g_{1\otimes 2}|?,\#\alpha]} \wedge (g_E \Rightarrow g_{1\otimes 2})$$

$$q_E \xrightarrow{[g_E|?\alpha]} \Rightarrow q_{1\otimes 2} \xrightarrow{[g_{1\otimes 2}|!,\#\alpha]} \wedge (g_E \Rightarrow g_{1\otimes 2})$$

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

$$q_E \xrightarrow{[g_E|!\alpha]} \Rightarrow q_1 \xrightarrow{[g_1|\sim_1 \alpha]} \wedge q_2 \xrightarrow{[g_2|\sim_2 \alpha]} \wedge (g_E \Rightarrow g_1 \wedge g_2)$$

$$q_E \xrightarrow{[g_E|?\alpha]}\Rightarrow q_1 \xrightarrow{[g_1|\sim_1\alpha]} \wedge q_2 \xrightarrow{[g_2|\sim_2\alpha]} \wedge (g_E \Rightarrow g_1 \wedge g_2)$$

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

E	$\mathscr{C}_1$	$\mathscr{C}_2$	$E \times M_{1,\text{must}}$	$E \times M_{2,\text{must}}$
!	?	?	#	#
!	#	#	#	#
!	#	?	#	#
!	#	!	#	!
?	!	!	#	#
?	#	#	#	#
?	#	!	#	#
?	#	?	#	?

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

$$q_i \xrightarrow{[g_i|?\alpha]}\Rightarrow q_{M_i} \xrightarrow{[g_{M_i}|?\alpha]} \wedge (g_i \Rightarrow g_{M_i})$$

So in $E \times M_1$, state (q_E, q_{M_1}) is reachable if state (q_E, q_1) is reachable in $E \times \mathscr{C}_{1,must}$. Consider the first row of the chart, where the environment is outputting α. Then from our result above, we have

$$q_E \xrightarrow{[g_E|!\alpha]}\Rightarrow q_{M_1} \xrightarrow{[g_{M_1}|?\alpha]} \wedge q_2 \xrightarrow{[g_2|?\alpha]} \wedge (g_E \Rightarrow g_{M_1} \wedge g_2),$$

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

$$q_{(E,M_1)} \xrightarrow{[g_E \wedge g_{M_1}|\#\alpha]}\Rightarrow q_2 \xrightarrow{[g_2|?\alpha]} \wedge (g_E \wedge g_{M_1} \Rightarrow g_2)$$

and the equivalent result for $E \times M_2$ yields

$$q_{(E,M_2)} \xrightarrow{[g_E \wedge g_{M_2}|!\alpha]}\Rightarrow q_1 \xrightarrow{[g_1|?\alpha]} \wedge (g_E \wedge g_{M_2} \Rightarrow g_1)$$

The latter result is precisely (7) with respect to $E \times M_2$ and $\mathscr{C}_1$. 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 $\mathscr{C}_1 \otimes \mathscr{C}_2$ is the greatest lower bound of all contracts that satisfy conditions 1 and 2. Thus, for any $\mathscr{C}_*$ satisfying 1 and 2, then $\mathscr{C}_1 \otimes \mathscr{C}_2 \lesssim \mathscr{C}_*$. This follows immediately from condition 1 and Proposition 2, since $M_1 \times M_2 \in M_{\mathscr{C}_*}$ yields, as desired

$$\begin{aligned} & M_{*,\text{must}} \lesssim M_1 \times M_2 \lesssim M_{*,\text{may}} \\ & \Rightarrow M_{*,\text{must}} \lesssim M_{1,\text{must}} \times M_{2,\text{must}} \lesssim M_1 \times M_2 \\ & \lesssim M_{1,\text{may}} \times M_{2,\text{may}} \lesssim M_{*,\text{may}} \\ & \Rightarrow M_{*,\text{must}} \lesssim M_{1\otimes 2,\text{must}} \lesssim M_{1\otimes 2,\text{may}} \lesssim M_{*,\text{may}}. \end{aligned}$$

■

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 $\mathscr{C}_{lights}, \mathscr{C}_{pedestrian}, \mathscr{C}_{vehicle}$ and $\mathscr{C}_{scheduler}$ 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 $\mathscr{C}_{lights} \otimes \mathscr{C}_{pedestrian}$ that operates orthogonally to the subsystem $\mathscr{C}_{vehicle} \otimes \mathscr{C}_{scheduler}$ defined by the cars and the scheduler. By orthogonality, the overall system is simply $(\mathscr{C}_{lights} \otimes \mathscr{C}_{pedestrian}) \wedge (\mathscr{C}_{vehicle} \otimes \mathscr{C}_{scheduler})$.

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, $\mathscr{C}_{horizontal\_lights}$ and $\mathscr{C}_{vertical\_lights}$, for the east-west and north-south directions respectively, that is $\mathscr{C}_{lights} = \mathscr{C}_{horizontal\_lights} \otimes \mathscr{C}_{vertical\_lights}$. Since $\mathscr{C}_{horizontal\_lights}$ and $\mathscr{C}_{vertical\_lights}$ 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 $\mathscr{C}_{horizontal\_lights}$ are `h`, `h'`, `h_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 `r` for red, `y` for yellow, and `g` for green. The output actions are `!r_h` and `!h_walk` 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 `?r_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 $\mathscr{C}_{pedestrian}$ is more simple. Its variable is `t_cross` which denotes the minimum time it takes the pedestrian to cross the street and the input actions are `?h_walk` and `?v_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 $\mathscr{C}_{horizontal\_lights} \otimes \mathscr{C}_{vertical\_lights}$ was computed automatically with the code in [10] and shown in Fig. 4. Though not included here to economize space, composing this with $\mathscr{C}_{pedestrian}$ closes all the remaining output actions in Fig. 4.

The scheduler contract automaton has access to a variable `len(request_queue)`, which is the length of the request queue. In addition, $\mathscr{C}_{scheduler}$ has one input action, `?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 `!reject` and `!accept` denoting whether the scheduler decides to accept or reject the request, and `!primitives` denoting the sending of controlling signals to the requesting vehicle. The internal action is `#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. $\mathscr{C}_{vehicle}$ 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 `!request` and receive signals from the scheduler with the action `?reject`, `?accept`, and `?primitives`. Composing $\mathscr{C}_{scheduler}$ with $\mathscr{C}_{vehicle}$ 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, `!primitives` and `?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 $\dot{v} = a + w_1$, $\dot{\theta} = \frac{v}{L} tan(\delta + w_2)$, $\dot{x} = v cos(\theta)$, $\dot{y} = v sin(\theta)$, with velocity v, orientation θ, positions x and y as state variables; acceleration $a \in [-9.8, 9.8]\frac{m}{s^2}$ and steering angle $\delta \in [-0.9, 0.9]\frac{rad}{s}$ as controllable inputs; $w_1 \in [-1.1, 1.1]\frac{m}{s^2}$ and $w_2 \in [-0.065, 0.065]\frac{rad}{s^2}$ 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 $\mathscr{X}_0(p)$ and $\mathscr{X}_f(q)$ denote the initial and final sets around p and q respectively. By construction, the primitive controller steers in a fixed time $t_{1,2}$ from the initial set $\mathscr{X}_0(p_1)$ around a nominal waypoint p_1 to a final set $\mathscr{X}_f(p_2)$ around p_2. Constraining $\mathscr{X}_f(p_2) \subseteq \mathscr{X}_0(p_2)$ allows any trajectory in the edge that starts from $\mathscr{X}_0(p_1)$ and ends in $\mathscr{X}_f(p_2)$ to be concatenated with any trajectory starting in $\mathscr{X}_0(p_2)$. 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 SCHEDULE(*request_queue*, *timetabel*) 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, `!primitives` and `?primitives` actions therefore correspond to the 2 SEND_PRIMITIVES($\cdot$) calls in the pseudocode.

Algorithm 1 The scheduling algorithm

```
function SCHEDULE($request\_queue, timetable$)
    $path, car \leftarrow$ EXTRACT_REQUEST($request\_queue$)
    if IS_SAFE($path, timetable$) then
        SEND_PRIMITIVES($path, car, timetable$)
    else if EXISTS_TRANSIT_NODE($path, timetable$) then
        $transit \leftarrow$ 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

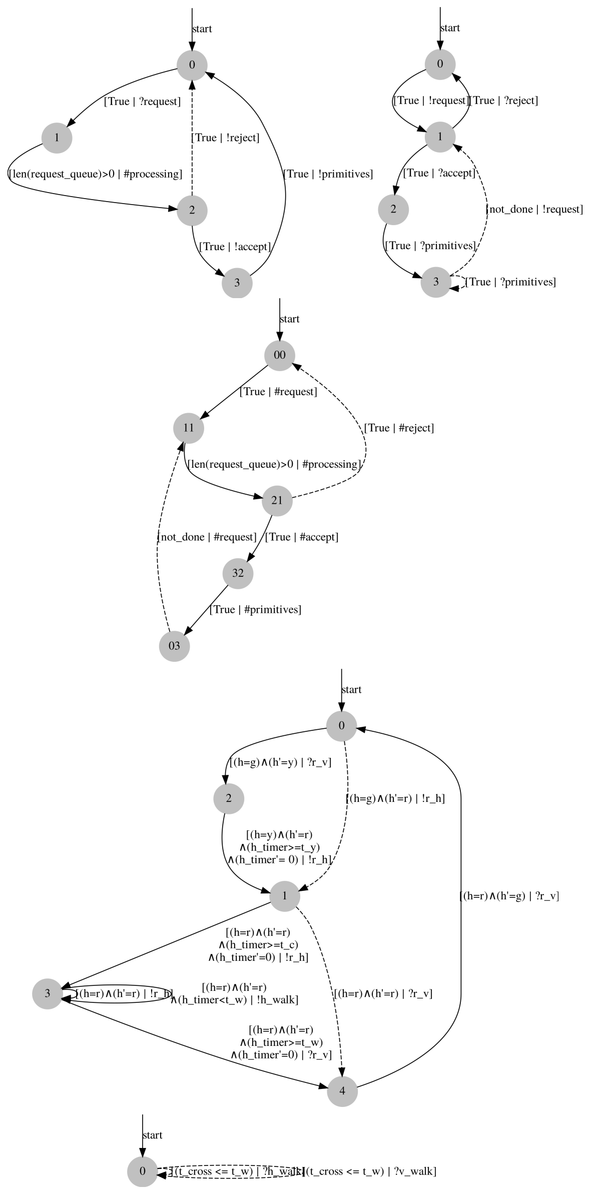

Fig. 3. Reading from left to right, then top to bottom: $\mathscr{C}_{scheduler}$, $\mathscr{C}_{car}$, $\mathscr{C}_{scheduler} \otimes \mathscr{C}_{car}$, $\mathscr{C}_{horizontal\_lights}$, $\mathscr{C}_{pedestrian}$.

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.

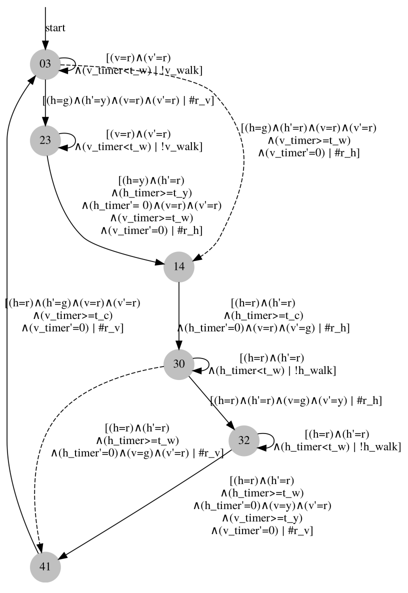
start
03
23
14
30
32
41
[(v=r)∧(v'=r) ∧(v_timer<t_w) | !v_walk]
[(h=g)∧(h'=y)∧(v=r)∧(v'=r) | #r_v]
[(h=g)∧(h'=r)∧(v=r)∧(v'=r) ∧(v_timer>=t_w) ∧(v_timer'=0) | #r_h]
[(v=r)∧(v'=r) ∧(v_timer<t_w) | !v_walk]
[(h=y)∧(h'=r) ∧(h_timer>=t_y) ∧(h_timer'= 0)∧(v=r)∧(v'=r) ∧(v_timer>=t_w) ∧(v_timer'=0) | #r_h]
[(h=r)∧(h'=g)∧(v=r)∧(v'=r) ∧(v_timer>=t_c) ∧(v_timer'=0) | #r_v]
[(h=r)∧(h'=r) ∧(h_timer>=t_c) ∧(h_timer'=0)∧(v=r)∧(v'=g) | #r_h]
[(h=r)∧(h'=r) ∧(h_timer<t_w) | !h_walk]
[(h=r)∧(h'=r)∧(v=g)∧(v'=y) | #r_h]
[(h=r)∧(h'=r) ∧(h_timer>=t_w) ∧(h_timer'=0)∧(v=g)∧(v'=r) | #r_v]
[(h=r)∧(h'=r) ∧(h_timer<t_w) | !h_walk]
[(h=r)∧(h'=r) ∧(h_timer>=t_w) ∧(h_timer'=0)∧(v=y)∧(v'=r) ∧(v_timer>=t_y) ∧(v_timer'=0) | #r_v]

Fig. 4. $\mathscr{C}_{lights} = \mathscr{C}_{horizontal\_lights} \otimes \mathscr{C}_{vertical\_lights}$

References

[1] Carliss Y Baldwin and Kim B Clark. Modularity in the design of complex engineering systems. In *Complex engineered systems*, pages 175–205. Springer, 2006.

[2] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Tom Henzinger, and Kim Larsen. Contracts for systems design. 2012.

[3] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Tom Henzinger, and Kim Larsen. Contracts for systems design: Methodology and application cases. 2015.

[4] Nikolaj Bjorner and Margus Veanes. Symbolic transducers. Technical report, January 2011.

[5] Luca de Alfaro and Thomas A Henzinger. Interface automata. *ESEC/FSE*, pages 109–120, 2001.

[6] Chun-Che Huang and Andrew Kusiak. Modularity in design of products and systems. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 28(1):66–77, 1998.

[7] Leslie Lamport. *The TLA+ Hyperbook*. 2015.

[8] Kim Guldstrand Larsen. Modal specifications. In *International Conference on Computer Aided Verification*, pages 232–246. Springer, 1989.

[9] Nancy A Lynch and Mark R Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 137–151. ACM, 1987.

[10] Tung M Phan. Traffic intersection. `https://github.com/tungminhphan/traffic-intersection`, 2018.

[11] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. Modal interfaces: unifying interface automata and modal specifications. In *Proceedings of the seventh ACM international conference on Embedded software*, pages 87–96. ACM, 2009.

[12] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. A modal interface theory for component-based design. *Fundamenta Informaticae*, 108(1-2):119–149, 2011.

[13] Bastian Schürmann and Matthias Althoff. Guaranteeing constraints of disturbed nonlinear systems using set-based optimal control in generator space. In *Proc. of the 20th IFAC World Congress*, pages 12020–12027, 2017.

[14] Bastian Schürmann and Matthias Althoff. Optimal control of sets of solutions to formally guarantee constraints of disturbed linear systems. In *Proc. of the American Control Conference*, pages 2522–2529, 2017.

[15] Bastian Schürmann, Daniel Heß, Jan Eilbrecht, Olaf Stursberg, Frank Köster, and Matthias Althoff. Ensuring drivability of planned motions using formal methods. In *Proc. of the Intelligent Transportation Systems Conference*, pages 1661–1668, 2017.