Controllability in Partial and Uncertain Environments

Nicolas D'Ippolito*†, Victor Braberman*
*Universidad de Buenos Aires, Argentina

Nir Piterman‡
†Imperial College London, UK

Sebastian Uchitel*†
‡University of Leicester, UK

***Abstract*—Controller synthesis is a well studied problem that attempts to automatically generate an operational behaviour model of the system-to-be that satisfies a given goal when deployed in a given domain model that behaves according to specified assumptions. A limitation of many controller synthesis techniques is that they require complete descriptions of the problem domain. This is limiting in the context of modern incremental development processes when a fully described problem domain is unavailable, undesirable or uneconomical. Previous work on Modal Transition Systems (MTS) control problems exists, however it is restricted to deterministic MTSs and deterministic Labelled Transition Systems (LTS) implementations. In this paper we study the Modal Transition System Control Problem in its full generality, allowing for nondeterministic MTSs modelling the environment's behaviour and nondeterministic LTS implementations. Given an nondeterministic MTS we ask if all, none or some of the nondeterministic LTSs it describes admit an LTS controller that guarantees a given property. We show a technique that solves effectively the MTS realisability problem and it can be, in some cases, reduced to deterministic control problems. In all cases the MTS realisability problem is in same complexity class as the corresponding LTS problem.**

I. Introduction

Correct-by-construction is an alternative approach to construct-and-verify that makes sense in many software engineering settings including embedded [1], adaptive systems [2], and model-based development (e.g, [3], [4], [5], [6], [7], [8]). The assumption is that reasoning about system goals declaratively and then producing automatically operational descriptions of how such goals can be achieved leads to high quality systems.

Controller synthesis is a field which fits into this approach. Very abstractly, given a model of the assumed behaviour of the environment (E) and a system goal (G), controller synthesis produces an operational behaviour model for a component M that when executing in an environment consistent with the assumptions results in a system that is guaranteed to satisfy the goal – i.e., $E \| M \models G$.

Controller synthesis has been traditionally oriented towards hardware engineering, focusing mainly on a Machine-Environment model based on shared memory. Consequently, Kripke structures have been extensively used as the formal setting for behaviour modelling [9]. More recently, focus on event-based formal models, such as Labelled Transition Systems (LTS), that support other interactions models (e.g. message passing and remote procedure calls) that are commonplace in Software Engineering have also been studied [8].

In practice, software engineering is not a waterfall process. Engineers do not build a complete description of the environment E and system goals G before they start thinking about (or actually getting round to doing some) implementation. In fact, it is widely accepted that intertwining requirements and design is crucial and informs the requirements elaboration process [10]. Indeed, notions such as realisability (i.e. if it is possible to build a system that satisfies a partial enunciation of its goals and also a partial description of its environment) have been studied extensively for this purpose.

The importance of reasoning over partial specifications is at odds with the requirement of complete behaviour models that controller synthesis imposes. Typically, approaches to controller synthesis require a completely defined model of the system behaviour (e.g. an LTS) with respect to a fixed alphabet of actions (e.g. traces described by the transition system are required, all else is prohibited) and consequently do not allow reasoning about realisability or constructing controllers when only partial information about the system environment is available.

Formalisms that support partial operational descriptions of behaviour have been studied for sometime. Notably, multi-valued Kripke structures [11] and modal transition systems [12]. These models allow explicit distinction between behaviour that is required and prohibited from behaviour for which it is yet unknown in which of the two previous categories it falls.

Despite advances in verification of partial behaviour models and the close technical relation between verification and synthesis (the former can answer if $E \| M \models G$, the latter goes one step further showing what M must look like so to have $E \| M \models G$) such formalisms have been scantily studied in the context of the controller synthesis. In the context of event-based interaction models, we have studied the problem of synthesis for partial behaviour models using Modal Transition Systems (MTS) as the formal grounding for environment behaviour description. This preliminary work was limited to deterministic behaviour models, an important restriction which impedes using abstraction in models to reduce complexity and hinders modelling failures and in general non-deterministic aspects of problem domains. *In this paper we solve the problem of MTS control for non-deterministic domain models and consider non-deterministic implementations*. The framework supports goals expressed in Fluent Linear Temporal Logic (FLTL) [13] and can be used in conjunction with specialised (and more efficient) synthesis algorithms for sublogics such as GR(1) [14].

II. Preliminaries

A. Transition Systems

We fix notation for labelled transition systems (LTSs) [15], which are widely used for modelling and analysing the behaviour of concurrent and distributed systems. An LTS is a state transition system where transitions are labelled with actions. The set of actions of an LTS is called its communicating alphabet and constitutes the interactions that the modelled system can have with its environment.

Definition 2.1: (Labelled Transition Systems [15]*)* Let $States$ be the universal set of states, Act be the universal set of actions. A *Labelled Transition System* (LTS) is a tuple $E = (S, A, \Delta, s_0)$, where $S \subseteq States$ is a finite set of states, $A \subseteq Act$ is a finite alphabet, $\Delta \subseteq (S \times A \times S)$ is a transition relation, and $s_0 \in S$ is the initial state.

If for some $s' \in S$ we have $(s, \ell, s') \in \Delta$ we say that ℓ is enabled from s. For a state s we denote $\Delta(s) = \{\ell \mid \exists s' \cdot (s, \ell, s') \in \Delta\}$ and $\Delta(s, \ell) = \{s' \mid (s, \ell, s') \in \Delta\}$.

Definition 2.2: (Parallel Composition) Let $M = (S_M, A_M, \Delta_M, s_0^M)$ and $N = (S_N, A_N, \Delta_N, s_0^N)$ be LTSs. *Parallel composition* $\|$ is a symmetric operator (up to isomorphism) such that $M \| N$ is the LTS $P = (S_M \times S_N, A_M \cup A_N, \Delta, (s_0^M, s_0^N))$, where Δ is the smallest relation that satisfies the rules below, where $\ell \in A_M \cup A_N$:

$$\frac{(s,\ell,s') \in \Delta_M}{((s,t),\ell,(s',t)) \in \Delta} \ \ell \in A_M \setminus A_N \qquad \frac{(t,\ell,t') \in \Delta_N}{((s,t),\ell,(s,t')) \in \Delta} \ \ell \in A_N \setminus A_M$$

$$\frac{(s,\ell,s') \in \Delta_M, \ (t,\ell,t') \in \Delta_N}{((s,t),\ell,(s',t')) \in \Delta} \ \ell \in A_M \cap A_N$$

Definition 2.3: (Traces) Consider an LTS $L = (S, A, \Delta, s_0)$. A sequence $\pi = \ell_0, \ell_1, \ldots$ is a trace in L if there exists a sequence $s_0, \ell_0, s_1, \ell_1, \ldots$, where for every $i \geq 0$ we have $(s_i, \ell_i, s_{i+1}) \in \Delta$.

Modal Transition System (MTS) [12] are abstract notions of LTSs. They extend LTSs by distinguishing between two sets of transitions. Intuitively an MTS describes a set of possible LTSs by describing an upper bound and a lower bound on the set of transitions from every state. Thus, an MTS defines required transitions, which must exist, and possible transitions, which may exist. By elimination, other transitions cannot exist. Formally, we have the following.

Definition 2.4: (Modal Transition System [12]*)* A *Modal Transition System (MTS)* is $M = (S, A, \Delta^r, \Delta^p, s_0)$, where $S \subseteq States$, $A \subseteq Act$, and $s_0 \in S$ are as in LTSs and $\Delta^r \subseteq \Delta^p \subseteq (S \times A \times S)$ are the required and possible transition relations, respectively.

We refer to actions in $\Delta^p \setminus \Delta^r$ as maybe actions. We depict maybe transitions by suffixing actions with a question mark "?". We denote by $\Delta^p(s)$ the set of possible actions enabled in s, namely $\Delta^p(s) = \{\ell \mid \exists s' \cdot (s, \ell, s') \in \Delta^p\}$ and $\Delta^p(s, \ell)$ the set of a-successors of s, namely $\Delta^p(s, \ell) = \{s' \mid (s, \ell, s') \in \Delta^p\}$. Similarly, $\Delta^r(s)$ and $\Delta^r(s, \ell)$.

Definition 2.5: (Refinement) Let $M = (S, A, \Delta_M^r, \Delta_M^p, s_0^M)$ and $N = (T, A, \Delta_N^r, \Delta_N^p, s_0^N)$ be two MTSs. Relation $H \subseteq S \times T$ is a *refinement* between M and N if the following holds for every $\ell \in A$ and every $(s, t) \in H$.

- If $(s, \ell, s') \in \Delta_M^r$ then there is t' such that $(t, \ell, t') \in \Delta_N^r$ and $(s', t') \in H$.
- If $(t, \ell, t') \in \Delta_N^p$ then there is s' such that $(s, \ell, s') \in \Delta_M^p$ and $(s', t') \in H$.

We say that N refines M if there is a refinement relation H between M and N such that $(s_0^M, s_0^N) \in H$, denoted $M \preceq N$. We say that N and M are *bisimilar* if the same refinement relation (transposed) shows that M refines N and that N refines M.

Intuitively, N refines M if every required transition of M exists in N and every possible transition in N is possible also in M. An LTS can be viewed as an MTS where $\Delta^p = \Delta^r$. Thus, the definition generalises to when an LTS refines an MTS. LTSs that refine an MTS M are complete descriptions of the system behaviour and thus are called *implementations* of M.

Definition 2.6: (Implementation and Implementation Relation) An LTS N is an *implementation* of an MTS M if and only if N is a refinement of M ($M \preceq N$). We shall refer to the refinement relation between an MTS and an LTS as an implementation relation. We denote the set of implementations of M as $\mathcal{I}_M$.

We say that an MTS is *deterministic* if there is no state that has two outgoing possible transitions on the same label. More formally, an MTS E is *deterministic* if $(s, \ell, s') \in \Delta_E^p$ and $(s, \ell, s'') \in \Delta_E^p$ implies $s' = s''$. The definition generalizes to LTSs as well. We refer to the set of all deterministic implementations of an MTS M as $\mathcal{I}_M^d$.

B. Fluent Linear Temporal Logic

We describe properties using Fluent Linear Temporal Logic (FLTL) [13]. FLTL is a linear-time temporal logic for reasoning about fluents. A *fluent Fl* is defined by a pair of sets and a Boolean value: $Fl = \langle I_{Fl}, T_{Fl}, Init_{Fl} \rangle$, where $I_{Fl} \subseteq Act$ is the set of initiating actions, $T_{Fl} \subseteq Act$ is the set of terminating actions and $I_{Fl} \cap T_{Fl} = \emptyset$. A fluent may be initially *true* or *false* as indicated by $Init_{Fl}$. Every action $\ell \in Act$ induces a fluent, namely $fl_\ell = \langle \{\ell\}, Act \setminus \{\ell\}, false \rangle$.

Let $\mathcal{F}$ be the set of all possible fluents over Act. An FLTL formula is defined inductively using the standard Boolean connectives and temporal operators $\mathbf{X}$ (next), $\mathbf{U}$ (strong until) as follows: $\varphi ::= Fl \mid \neg\varphi \mid \varphi \vee \psi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \psi$, where $Fl \in \mathcal{F}$. As usual we introduce $\wedge$, $\Diamond$ (eventually), $\Box$ (always), and $\mathbf{W}$ (weak until) as syntactic sugar. Let Π be the set of infinite traces over Act. The trace $\pi = \ell_0, \ell_1, \ldots$ satisfies a fluent Fl at position i, denoted $\pi, i \models Fl$, if and only if one of the following conditions holds:

- $Init_{Fl} \wedge (\forall j \in \mathbb{N} \cdot 0 \leq j \leq i \rightarrow \ell_j \notin T_{Fl})$
- $\exists j \in \mathbb{N} \cdot (j \leq i \wedge \ell_j \in I_{Fl}) \wedge (\forall k \in \mathbb{N} \cdot j < k \leq i \ \rightarrow \ell_k \notin T_{Fl})$

In other words, a fluent holds at position i if and only if it holds initially or some initiating action has occurred, but no terminating action has yet occurred.

For an infinite trace π, the satisfaction of a (composite) formula φ at position i, denoted $\pi, i \models \varphi$, is defined as follows:

$$
\begin{array}{lcl}
\pi, i \models \neg\varphi & \triangleq & \neg(\pi, i \models \varphi) \\
\pi, i \models \varphi \vee \psi & \triangleq & (\pi, i \models \varphi) \vee (\pi, i \models \psi) \\
\pi, i \models \mathbf{X}\varphi & \triangleq & \pi, i+1 \models \varphi \\
\pi, i \models \varphi \mathbf{U} \psi & \triangleq & \exists j \geq i . \pi, j \models \psi \wedge \forall\, i \leq k < j . \pi, k \models \varphi
\end{array}
$$

We say that φ holds in π, denoted $\pi \models \varphi$, if $\pi, 0 \models \varphi$. A formula $\varphi \in$ FLTL holds in an LTS E (denoted $E \models \varphi$) if it holds on every infinite trace produced by E.

In this paper we modify LTSs and MTSs by adding new actions and adding states and transitions that use the new actions. It is convenient to change FLTL formulas to ignore these changes. Consider an FLTL formula φ and a set of actions Γ such that for all fluents $Fl = \langle I_{Fl}, T_{Fl}, Init_{Fl} \rangle$ in φ we have $\Gamma \cap (I_{Fl} \cup T_{Fl}) = \emptyset$. We define the Γ-variant version of φ, denoted $\mathcal{X}_\Gamma(\varphi)$, by replacing every sub-formula $\mathbf{X}\psi$ in φ by $\mathbf{X}((\bigvee_{\ell \in \Gamma} fl_\ell)\mathbf{U}\mathcal{X}_\Gamma(\psi))$.

Thus, this transformation replaces every next operator occurring in the formula by an until operator that skips uninteresting actions that are in Γ.

Theorem 2.1: Given a trace $\pi = \ell_0, \ell_1, \ldots$ in $E = (S, A, \Delta, s_0)$, an FLTL formula φ and a set of actions $\Gamma \in Act$. If $\Gamma \cap A = \emptyset$ then for every trace π' that is a Γ-variant of π we have $\pi \models \varphi$ iff $\pi' \models \mathcal{X}_\Gamma(\varphi)$.

We note that our results hold for properties that describe sets of traces that can be modified easily to accept Γ-variants as above. We choose to focus on FLTL as it makes all complexity results concrete and is a well accepted standard.

C. Controller Synthesis

An LTS control problem is an LTS E whose actions are partitioned to controllable and uncontrollable. We seek a *controller* M such that $E \| M$ does not restrict uncontrollable actions of E and $E \| M$ does not have deadlocks. Formally, we have the following.

Definition 2.7: (Legal LTS) Consider E and M two LTSs. We say that M is *legal* for E with respect to controllable alphabet $A_c \subseteq A$ if for every reachable state (s, m) of $E \| M$ we have $i)$ if $(s, \ell, s') \in \Delta^E$ and $\ell \notin A_c$ then there is m' such that $(m, \ell, m') \in \Delta^M$ and, $ii)$ there is an action ℓ and states s' and m' such that $((s, m), \ell, (s', m'))$ is a transition in $E \| M$.

Definition 2.8: (LTS Control [8]*)* Given a domain model in the form of an LTS $E = (S, A, \Delta, s_0)$, a set of controllable actions $A_c \subseteq A$, and an FLTL formula φ, a solution for the LTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$ is an LTS $M = (S_M, A_M, \Delta_M, s_{0_M})$ such that M is a legal LTS for E, $E \| M$ is deadlock free, and every trace π in $E \| M$ is such that $\pi \models \varphi$.

Whenever a controller exists we say that the control problem is realisable. It is unrealisable otherwise. In case that a domain model E is given and A_c and φ are implicit we denote by $\mathcal{E}$ the control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$. We depict uncontrollable actions (i.e. actions in $A \setminus A_c$) with doubled-dashed transitions (e.g., see transition $(4, readyForPickup, 5)$). In figures we use c and u actions to denote controllable and uncontrollable actions respectively – e.g. models in Figure 2.

Theorem 2.2: (LTS Control [9], [16]*)* Given an LTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$ it is decidable in 2EXPTIME in the size of φ and EXPTIME in E whether $\mathcal{E}$ is realisable. If E is deterministic the algorithm is polynomial in the size of E. The algorithm for checking realisability can also extract a controller M.

The problem MTS control problem [17] is to check whether all, none or some of the LTS implementations of a given MTS can be controlled by an LTS controller [8]. More specifically, given an MTS, an FLTL goal and a set of controllable actions, the answer to the MTS control problem is *all* if all implementations of the MTS can be controlled, *none* if no implementation can be controlled and *some* otherwise. This is formally defined below.

Definition 2.9: (Semantics of MTS Control) Given an MTS $E = (S, A, \Delta^r, \Delta^p, s_0)$, an FLTL formula φ and a set $A_c \subseteq A$ of controllable actions, to solve the *MTS control problem* $\mathcal{E} = \langle E, \varphi, A_c \rangle$ is to answer:

- **All**, if for all LTS $I \in \mathcal{I}_E$, the control problem $\langle I, \varphi, A_c \rangle$ is realisable,
- **None**, if for all LTS $I \in \mathcal{I}_E$, the control problem $\langle I, \varphi, A_c \rangle$ is unrealisable,
- **Some**, otherwise.

A naïve approach to the MTS control problem may require to evaluate an infinite number of LTS control problems. In [17] we solved the problem of MTS control when the MTS is deterministic and restricting attention to deterministic implementations. That is, replace all quantification over LTS above by quantification over $\mathcal{I}_E^d$ instead of $\mathcal{I}_E$ – recall that $\mathcal{I}_E^d$ refers to the set of deterministic implementations of E. Technically, we proposed two LTS control problems that answer, respectively, whether all implementations are controllable and whether none of the implementations are controllable.

III. Motivation

Consider the following example inspired by the one presented originally in [18]. A library requires a system that allows the users to borrow books and guarantees that users have access to their desired ones. Books are loaned for a fixed period of time after which users must return them. Books are automatically available as soon as they are returned. The system must handle concurrent book requests by multiple users.

In Figure 1, we show a partial specification for the book-loan process up to the level requirements have been defined. Although the system is multi-user, the model shows the behaviour of the system from the perspective of one user, abstracting away multi-user behaviour using non-determinism. When a user wants to borrow a book she searches for the desired one ($queryBook$). Then, the system displays a list of available copies ($listBooks$) from which the user can choose one ($selectForPickup$) and then pick it up from the counter ($pickup$). However, as the system is used by several users at the same time, the selected copy may have been offered, chosen and taken by another user. In such a case, an alternative book is locked and offered. The user can then either accept ($acceptAlt$) and pick up the book ($pickupAlt$) or decline the

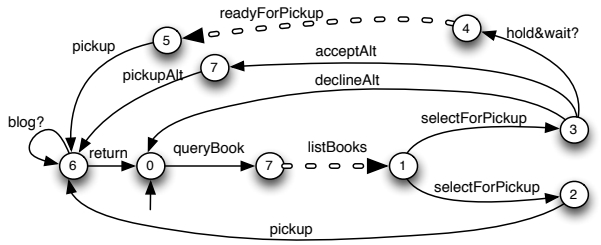


Fig. 1. Books Loan Partial Specification.

offer ($declineAlt$). For simplicity, in this example we require a user to return a book before another can be requested.

The interference between users that may make a $selectForPickup$ successful (allowing pick up) or not (proposing an alternative) is abstracted away in Figure 1 with a non-deterministic choice. The choice hides the reason for which one scenario or the other may occur. This is a standard approach to reducing model complexity.

Some libraries allow users to reserve a copy of a currently unavailable book. When a copy of the reserved book is returned to the library, the copy is locked and the user is notified for pick up ($readyForPickup$). Such a situation is modelled here as a possible but not required – i.e. maybe – transition that models the choice of the user to hold a copy as soon as it is returned ($hold\&wait$).

Another variability in this specification is that users may be provided with a blogging feature to allow them to share their impressions about a book while they are reading it. This is modelled with another maybe transition labelled $blog$.

The maybe transitions $hold\&wait$ and $blog$ underspecify the library system allowing implementations that provide different combinations of functionality (i.e. providing or not "hold and wait", providing or not "blogging"). Our aim is to understand if a controller exists that can guarantee that users will eventually get the book of choice (formalised by the FLTL property $\Box(queryBook \rightarrow \Diamond pickup)$). Note that actions In particular we are interested in knowing whether the goal will be achievable in *all* valid implementations of such specifications, in only *some* or *none* of them.

The answer to the MTS control problem stated above is *some*. This follows from the fact that there are implementations of the system that are indeed controllable but there are also implementations in which it is simply not possible to guarantee that the user will eventually get her desired book.

Consider an implementation that does not implement the $hold\&wait$ feature. As $selectForPickup$ is nondeterministic, it is impossible to guarantee that the user will eventually get the chance to pickup the book – i.e., it may always lose the race with other users and end up in state 3, being forced to choose an alternative book or cancel the request.

On the other hand, an implementation that allows the user to reserve the book to be taken when returned is indeed controllable. Naturally, this holds because we are assuming that there is a mechanism for guaranteeing that users actually return books. This assumption is coded in state 5 in which $readyForPickup$ must eventually happen.

As not all implementations are controllable this suggests that either we must strengthen our specification to force implementations with the "hold and wait" feature (provision of the "blogging" feature is irrelevant for controllability of our goal), we must weaken our goal (e.g. for instance allowing users to give up on their wanted book) or strengthen our assumptions (e.g. queries cannot fail more than 5 times in a row).

IV. MTS Control Problem

In this section we consider the problem of MTS control when the MTS could be nondeterministic and we consider nondeterministic implementations. We provide intuitions for proofs, for more detailed see [19]. We show how to construct two nondeterministic LTSs, $E^{\forall}$ and $E^{\exists}$, such that if a control problem with $E^{\forall}$ is realisable the answer to the MTS control problem for E is *all* and, dually, if a control problem with $E^{\exists}$ is unrealisable the answer is *none*. This is similar to the solution of MTS control in the deterministic case [17] in the sense that we construct an LTS that is the "hardest" implementation to control and an LTS that is the "easiest" implementation to control. Answering these two LTS control problems gives a correct answer to the MTS control problem. We start with the problem of answering the question whether *all* implementations are controllable.

Definition 4.1: Consider an MTS $E = (S, A, \Delta^r, \Delta^p, s_0)$ and a set of controllable actions $A_c \subseteq A$. We define $E^{\forall} = (S_{E^{\forall}}, A_{E^{\forall}}, \Delta_{E^{\forall}}, s_0)$ as follows:

- $S_{E^{\forall}} = S \cup \{(s,1) \mid s \in S \text{ and } \emptyset \neq \Delta^r(s) \subseteq A_c\}$
- $A_{E^{\forall}} = A \cup \{\ell_1\}$
- $\Delta_{E^{\forall}} =$
  $\{(s,\ell,s') \in \Delta^p \mid \Delta^r(s) \neq \emptyset \text{ and } \ell \notin A_c\} \quad \cup$
  $\{(s,\ell_1,(s,1)) \mid (s,1) \in S_{E^{\forall}}\} \quad \cup$
  $\{((s,1),\ell,s') \mid s' \in \Delta^p(s,\ell) \text{ and } \Delta^r(s,\ell) \neq \emptyset\}$

Intuitively, the transformation gives more control to the environment. First, the new action ℓ_1 is uncontrollable. The construction of $E^{\forall}$ proceeds by considering three cases: i) states with required uncontrollable transitions enabled, ii) states with no required uncontrollable transitions enabled, but required controllable transitions enabled, and iii) states with no required transitions (i.e. only maybe transitions). To states that have uncontrollable required transitions we add all uncontrollable transitions – note that controllable transitions are not included as they do not affect controllability. As an example, consider E_1 and $E_1^{\forall}$ shown in Figures 2(a) and 2(d) respectively. $E_1^{\forall}$ is the result of applying the transformation above to E_1. $E_1^{\forall}$ includes all possible uncontrolled implementation choices. Hence, control $E_1^{\forall}$ guarantees controllability of every implementation of E_1. To states that have no uncontrollable required transitions and some controllable required transitions we include all uncontrollable transitions and an extra uncontrollable transition to a new state (i.e. $(s,1)$) that enables all required controllable transitions from the original state. Intuitively, we give the environment a choice between all (possible) uncontrollable transitions and (through the new state $(s,1)$) a choice to implement additional controllable transitions that agree with their labels with the existing required transitions. Hence, forcing the controller to

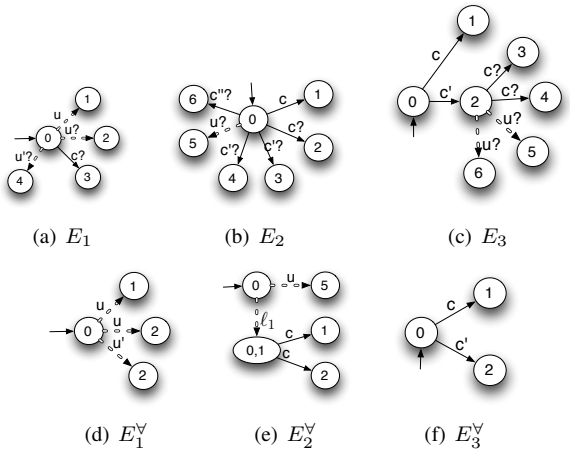


Fig. 2. Examples of $E^\forall$.

be able to control every possible nondeterministic combination of transitions with at least one required option. For example, consider E_2 and $E_2^\forall$ in Figures 2(b) and 2(e) respectively. In order to control $E_2^\forall$ the controller must have a solution for both, i) all uncontrollable possible transitions, and, as ℓ_1 is uncontrollable, ii) for every possible nondeterministic choice on the required controllable. Note that we do not include transitions on controllable actions that have no required transition (e.g., $0 \xrightarrow{c''?} 6$) as they might allow the controller to avoid enabling some required transitions. On the other hand, we do include all nondeterministic options over the actions that have some required controllable as it makes the problem harder for the controller. Finally, from states that have no required transitions we do not add transitions at all – i.e. they become deadlock states. This follows from the fact that the implementation that disables all transitions from such a state is a valid implementation. Consider E_3 in Figure 2(c) is an example of the previous case. To be sure that all implementations of R_3 can be controlled, it is necessary to have a solution that completely avoids reaching the deadlock state 2.

The LTS $E^\forall$ provides the basis for determining whether the answer to the MTS control question is **All**. The following lemma provides the key properties to compute the answer for the case *all*.

Lemma 4.1: (All) Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c\rangle$, where $E = (S, A, \Delta^r, \Delta^p, s_{0_E})$. If $E^\forall$ is the LTS obtained by applying Definition 4.1 to E, then the following holds. The answer for $\mathcal{E}$ is *all* iff the LTS control problem $\mathcal{E}^\forall = \langle E^\forall, \mathcal{X}_{\{\ell_1\}}(\varphi), A_c\rangle$ is realisable.

We give an intuition of the proof for Lemma 4.1. Assume there is no solution to $\mathcal{E}^\forall$. Then we know that the environment has a strategy to violate the controller's requirements. This follows from the fact that LTS control problems are solved by a reduction to two-player games [8] and such games are Determined [20], which means that if the game is loosing for one player (e.g. the controller) it must be the case that is winning for the other one (e.g. the environment). Note that, a game is winning for a player means that she has a strategy that wins against any strategy of her opponent. At a glance, this proof uses the environment's strategy N to violate the controller goals to build an uncontrollable implementation I of E. This is done by building an implementation relation between I and E that chooses what to implement based on the decisions of the N. Then we prove that I is uncontrollable, which follows by the fact that we built I choosing what to implement as N would have. Hence, violating the controller goals.

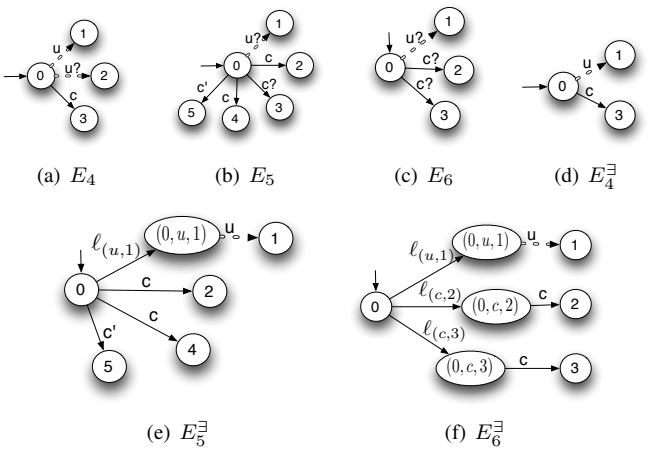
Intuitively, in the other direction the proof is based on the idea that a controller for $\mathcal{E}^\forall$encodes a strategy for all possible implementations of E. More specifically, we start by assuming that there is a controller M for $\mathcal{E}^\forall$. Then, given an implementation I of E (i.e. $I \in \mathcal{I}_E$), we show that there exists a controller M' for $\mathcal{I} = \langle I, \varphi, A_c\rangle$. We then construct M' from M following the refinement relation between E and I. Intuitively, M' can be seen as the result of pruning M by removing all implementations choices disabled from E by I. In addition, as M' is built upon M, M' is a solution to $\mathcal{I}$. More specifically, we consider states of I following cases as done in Definition 4.1, now taking into account the refinement relation between I and E. Consider a state t of I that refines state s of E. If s has uncontrollable required transitions but no controllable required, then, by construction, we know that s have all possible uncontrollable transitions enabled in $E^\forall$. Hence, M also includes all possible uncontrollable transitions from s, from which follows that M will have a transition for any enabled transition in t. If the only required transitions from s are controllable, then by definition of $E^\forall$, the controller of $\mathcal{E}^\forall$must enable a transition to, both the state $(s, 1)$ from which the required controllable transitions are enabled, and all the possible uncontrollable successors of s. It follows that the controller can control any implementation choice of uncontrollable transitions from t, and also, as t refines s, the required controllable that must be enabled in t. Note that states t of I that refine states s of E where all transitions enabled in s are maybe transitions induce deadlock states that will not be reachable in $M'\|I$ as, in $\mathcal{E}^\forall$they are deadlock states.

We now turn to the case of distinguishing between the *none* case and *all* or *some*.

Definition 4.2: Consider an MTS $E = (S, A, \Delta^r, \Delta^p, s_0)$ and a set of controllable actions $A_c \subseteq A$. We define $E^\exists = (S_{E^\exists}, A_{E^\exists}, \Delta_{E^\exists}, s_0)$ as follows:

- $S_{E^\exists} = S \ \cup \{(s, a, s') \in \Delta^p \mid a \in A \text{ and } \Delta^r(s) = \emptyset\} \cup \left\{(s, a, s') \in \Delta^p \;\middle|\; \begin{array}{l} a \in A \text{ and } \emptyset \neq \Delta^r(s) \subseteq A_c, \\ \text{and } \Delta^r(s, a) = \emptyset \end{array}\right\}$
- $A_{E^\exists} = A \cup \{\ell_{(a,s')} \mid a \in A \text{ and } (s, a, s') \in \Delta^p \setminus \Delta^r\}$
- $\Delta_{E^\exists} = \{(s, a, s') \mid a \in A \text{ and } (s, a, s') \in \Delta^r\} \cup \{(s, \ell_{(a,s')}, (s, a, s')) \,|a \in A \text{ and } (s, a, s') \in S_{E^\exists}\} \cup \{((s, a, s'), a, s') \mid a \in A\}$

Intuitively, $E^\exists$ gives more control to the controller. As before we consider states according to the set of enabled transitions. For states with uncontrollable required transitions we add in $E^\exists$ all required transitions as they will be enabled in all implementations. We do not add other transitions as it would only result in a more complex problem because either we add new uncontrollable transitions, or we add nondeterministic choices over the (already enabled) required transitions. An example of this case is shown in the models in Figures 3(a) and 3(d). Note that to control $E_4^\exists$ it is only necessary to have a solution for the uncontrolled transition.



(a) E_4 (b) E_5 (c) E_6 (d) $E_4^{\exists}$

(e) $E_5^{\exists}$ (f) $E_6^{\exists}$

Fig. 3. Examples of $E^{\exists}$.

However, if the state were such that its only required transitions are controllable, then they should be in $E^{\exists}$ as they could be chosen by the controller.

In order to strengthen the controller, we add to $E^{\exists}$ states that allow it to choose which transitions to implement in two cases. First, from states that have required controllable transitions but no required uncontrollable transitions. Second, from states that have no required transitions from them.

From states that have no uncontrollable required transitions but have some required controllable transitions we add possible transitions that do not share an action with a required transition. This means that the controller has a choice of either implementing one maybe transition or using one of the required transitions. We do this by adding new transitions labelled with actions that uniquely identify the transition. For instance, consider $E_5^{\exists}$ from Figure 3(e), the result of applying the latter rule to E_5 in Figure 3(b). The controllable transition $0 \xrightarrow{\ell_{(u,1)}} (0,u,1)$, that comes allows the controller to choose to implement the maybe transition $0 \xrightarrow{u?} 1$ if it wants to. Otherwise, it can choose to use the (required) controllable transition labelled c.

From states that have no required transitions we allow the system to choose which maybe transition to implement. As before we use actions to identify the transition, allowing the controller to choose which one to implement. Note that all possible nondeterministic choices are included (but made deterministic). See for example models in Figures 3(c) and 3(f).

The LTS $E^{\exists}$ provides the basis for determining whether the answer to the MTS control question is **None**.

Lemma 4.2 provides the key properties to compute the answer for the case *none*.

Lemma 4.2: (None) Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c\rangle$ where $E = (S, A, \Delta^r, \Delta^p, s_{o_E})$. If $E^{\exists}$ is the LTS obtained by applying Definition 4.2 to E, then the following holds. The answer for $\mathcal{E}$ is *none* iff the LTS control problem $\mathcal{E}^{\exists} = \langle E^{\exists}, \mathcal{X}_{\overline{A}}(\varphi), A_{E^{\exists}} \setminus (A \setminus A_c)\rangle$ is unrealisable.

We give an intuition of the proof for Lemma 4.2. We first assume that there is a controller M for $\mathcal{E}^{\exists}$. Now, recall that M controls $\ell_{(a,s)}$ actions, which are only added in $E^{\exists}$ for maybe transitions in E. Thus, a transition labelled $\ell_{(a,s)}$ appearing in M represents an implementation choice that can be controlled by M. In other words, M encodes a set of controllable implementations. More specifically, we construct, from M and $E^{\exists}$, an implementation I and a controller M' for $\mathcal{I} = \langle I, \varphi, A_c\rangle$ and that will show that *none* implies the unrealisability of $\mathcal{E}^{\exists}$control problem. We consider states in cases. If the state has uncontrollable required transitions, then for each of them we include a transition in I and M'. If the state has no uncontrollable required but some controllable required, then we choose transitions to implement in I and enable in M' following what is enabled by M. If the state has no required transitions, then as all actions of the form $\ell_{(a,s)}$ are controllable we add to I and M' actions enabled by M. We then show that M' indeed controls $\mathcal{I}$. This is done by showing a correspondence between traces of $I \| M'$ and traces of $E^{\exists} \| M$, which can be proven following their construction procedure. Thus, both must satisfy the respective goals.

In the other direction, given an implementation $I \in \mathcal{I}_E$ and a controller M' for $\mathcal{I}$, we construct a controller M for $\mathcal{E}^{\exists}$. The construction of M is directed by the refinement relation between E and I, and the controllable transitions enabled by M'. Intuitively, M is an extension of M' that follows implementation choices of I by adding transitions labelled with $\ell_{(a,s)}$ that will synchronise with $E^{\exists}$– i.e. forcing $E^{\exists}$ to behave as I. More specifically, if t is a state of I that refines state s of E and implements a maybe transition (s, a, s'), then M enables the successor (s, a, s') of s in $E^{\exists}$. We then show how a correspondence between traces of $I \| M'$ and traces of $E^{\exists} \| M$ is established, from which follows that both traces satisfy the respective goals.

We now provide the algorithm to compute the solution to the MTS Control Problem. The soundness and completeness of the algorithm follows from Lemma 4.1 and Lemma 4.2.

Algorithm 1: (MTS Control) Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c\rangle$. If $E^{\forall}$ and $E^{\exists}$ are the LTSs obtained by applying Definitions 4.1 and 4.2 respectively, to E, then the answer for $\mathcal{E}$ is computed as follows.

- **All**, if there exists a solution for $\mathcal{E}^{\forall} = \langle E^{\forall}, \mathcal{X}_{\{\ell_1\}}(\varphi), A_c\rangle$
- **None**, if there is no solution for $\mathcal{E}^{\forall}$ and no solution for $\mathcal{E}^{\exists} = \langle E^{\exists}, \mathcal{X}_{\overline{A}}(\varphi), A_{E^{\exists}} \setminus (A \setminus A_c)\rangle$.
- **Some**, otherwise.

Theorem 4.1: (MTS Control) Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c\rangle$ it is decidable in 2EXPTIME in φ and EXPTIME in E whether $\mathcal{E}$ is realisable.

Proof: The algorithm for checking whether $\mathcal{E}$ is realisable calls for solving two LTS control problems. One with $E^{\forall}$ and one with $E^{\exists}$. The size of both $E^{\forall}$ and $E^{\exists}$ is linear in the size of E. The sizes of $\mathcal{X}_{\{\ell_1\}}(\varphi)$ and $\mathcal{X}_{\overline{A}}(\varphi)$ are linear in the size of φ. From Theorem 2.2 we establish the bounds of 2EXPTIME in the size of φ and EXPTIME in the size of E. ■

V. Efficient MTS Control Subproblems

The complexity of the general MTS control problem has exponential growth with respect to two factors: the size of the

formula and the size of the domain model (see Theorem 2.2). Various sublogics have been studied to reduce the (doubly) exponential growth of complexity with respect to the goal to be controlled. In particular, GR(1) goals reduce this complexity to polynomial [14].

In this section, we address the exponential growth in the size of the domain model by identifying two MTS control subproblems for which the complexity in the size of the domain model is polynomial. The first subproblem is when we restrict MTS to be deterministic. The second is when we limit the possible implementations of MTS to deterministic ones. We show that in both cases a simpler analysis that requires reasoning about deterministic LTS controllability is sufficient.

A. Deterministic MTS Non-Deterministic Implementations

We formally define the subproblem of answering whether all, some or none implementations of a deterministic MTS are controllable as follows:

Definition 5.1: (Det MTS Control) Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$, where E is deterministic. Then the answer for $\mathcal{E}$ is computed as follows.

- **All**, if all deterministic implementation of E are realisable.
- **None**, if no deterministic implementation of E is realisable, and
- **Some**, otherwise.

We note that the case of determinstic MTS realisability where only deterministic implementations are considered was handled in [17]. We show that in this case, MTS control reduces to the simpler problem of considering only deterministic implementations. We need to establish correspondence between the three possible answers of the two problems.

The following Lemma states that if the answer to the *Det MTS* control problem is some or all, then the answer cannot be none for the problem restricted to deterministic implementations as solved in [17].

Lemma 5.1: Given a deterministic MTS $E = (S, A, \Delta^r, \Delta^p, s_0)$, an FLTL formula φ, and a set $A_c \subseteq A$ of controllable actions; If for some $N \in \mathcal{I}_E$ we have $\langle N, \varphi, A_c \rangle$ is realisable then for some $D \in \mathcal{I}_E^d$ we have $\langle D, \varphi, A_c \rangle$ is realisable.

The following Lemma states that if the answer to the control problem is all when restricting attention to deterministic implementations then all nondeterministic implementations are controllable as well.

Lemma 5.2: Given a deterministic MTS $E = (S, A, \Delta^r, \Delta^p, s_0)$, and FLTL formula φ and a set $A_c \subseteq A$ of controllable actions; If for all $D \in \mathcal{I}_E^d$ we have $\langle D, \varphi, A_c \rangle$ is realisable then for every $N \in \mathcal{I}_E$ we have $\langle N, \varphi, A_c \rangle$ is realisable.

It follows from the two lemmata above that the following algorithm correctly computes the answer to the MTS control problem when the MTS is deterministic.

Algorithm 2: Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$, where E is deterministic. The answer to the control problem $\mathcal{E}$ is as follows.

- **All**, if the answer for the deterministic MTS control problem $\mathcal{E}$ is all.
- **None**, if there is no solution for the deterministic MTS control problem $\mathcal{E}$.
- **Some**, otherwise.

Theorem 5.1: (Deterministic MTS Control) Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$, where E is deterministic, it is decidable in 2EXPTIME in φ and polynomial in E whether $\mathcal{E}$ is realisable.

Proof: We have shown that in this case it is enough to consider the case of deterministic implementations. From [17] this MTS control can be reduced to two LTS control problems where the LTSs are deterministic. From Theorem 2.2 it follows that it is 2EXPTIME in φ and polynomial in E. ■

B. Non-Deterministic MTS Deterministic Implementations

The case of searching for deterministic implementations of a nondeterministic MTS is handled differently. First, we extract from the MTS E a sub-MTS D. If D and E are not bisimilar, then E has no deterministic implementations. Then, we apply a specialized control check on D and the answer to the realisability check on D is the answer to whether *all*, *some*, or *none* of the deterministic implementations of E can be controlled to satisfy the formula.

Definition 5.2: Consider an MTS $E = (S, A, \Delta^r, \Delta^p, s_0)$. We extract from E a sub-MTS D, where $D = (T, A, \Delta^r, \Delta^p, s_0)$. We construct $T \subseteq S$ by induction. Consider a state $s \in T \cap S$. For every label ℓ such that $\ell \in \Delta^r(s)$ we add to T *one* required a-successor of s. For every label ℓ such that $\ell \in \Delta^p(s) \setminus \Delta^r(s)$ we add to T *all* maybe ℓ successors of s.

Lemma 5.3: Given a nondeterministic MTS $E = (S, A, \Delta^r, \Delta^p, s_0)$, E has deterministic implementations if and only if D is bisimilar to E.

Proof: Assume that there is a deterministic implementation I of E. From the implementation relation between I and E we can construct an implementation relation between I and D. Indeed, the only transitions missing from D are cases where a state s in E has two required successors (s, ℓ, s') and (s, ℓ, s''). But as s is implemented by some state of I, then, by definition of refinement, s' and s'' must be bismilar and whatever choice in E to continue from s to either (s, ℓ, s') or (s, ℓ, s'') would be sufficient to implement the state s in E. Thus D is bisilimar to E. The other direction is trivial and omitted. ■

Clearly, if E has no deterministic implementations then the answer to the question whether all deterministic implementations satisfy a certain formula is vacuously true. In the case that D is bisimilar to E every implementation of E is an implementation of D and vice versa. Thus, the answer to the MTS control problem with D replacing E is the same answer as that of E. We analyse D to check whether all its implementations are controllable, some of them are

controllable, or none of them are controllable. We construct variants $D^{\forall}$ and $D^{\exists}$ of D where all nondeterminism is removed by renaming actions. In $D^{\forall}$ the environment chooses which maybe transitions to implement and in $D^{\exists}$ the controller chooses which maybe transitions to implement. Using $D^{\forall}$ and $D^{\exists}$ we create control problems that check the "all" and "some" cases of the control problem.

Definition 5.3: Consider an MTS $E = (S, A, \Delta^r, \Delta^p, s_0)$ and a set of controllable actions $A_c \subseteq A$. Let $D = (S', A, \rho^r, \rho^p, s_0)$ be the sub-MTS extracted from E as in Definition 5.2. We define $D^{\forall} = (S_{D^{\forall}}, A_{D^{\forall}}, \Delta_{D^{\forall}}, s_0)$ as follows:

- $S_{D^{\forall}} = S' \cup \{(s,1) \mid \emptyset \neq \rho^r(s) \subseteq A_c\}$
- $A_{D^{\forall}} = A \times S \cup \{\ell_1\}$
- $\Delta_{D^{\forall}} =$
  $\{(s,(a,s'),s') \mid (s,a,s') \in \rho^r \text{ and } \rho^r(s) \not\subseteq A_c\} \cup$
  $\{(s,(a,s'),s') \mid (s,a,s') \in \rho^p, a \notin A_c \text{ and } \rho^r(s) \not\subseteq A_c\} \cup$
  $\{(s,\ell_1,(s,1)) \mid \emptyset \neq \rho^r(s) \subseteq A_c\} \cup$
  $\{((s,1),(a,s'),s') \mid (s,a,s') \in \rho^r \text{ and } a \in A_c\} \cup$
  $\{(s,(a,s'),s') \mid (s,a,s') \in \rho^p, a \notin A_c \text{ and } \emptyset \neq \rho^s(s) \subseteq A_c\} \cup$
  $\{(s,(a,s'),s') \mid \rho^r(s) = \emptyset \text{ and } (s,a,s') \in \rho^p\}$ ESTA ESTA MAL, ESTO NO DEBERIA AGREGAR NADA Y DEJAR AL ESTADO COMO DEADLOCK.

All the new actions are uncontrollable. We note that $D^{\forall}$ is very similar to $E^{\forall}$ except that $D^{\forall}$ is made effectively deterministic by renaming all actions. As before, we extend the definition of all fluents in φ to include the copies (a,s) of actions in A. That is, if a fluent includes the action ℓ then its modified version includes also the action (ℓ, s).

Lemma 5.4: (Deterministic All) Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$ where $E = (S, A, \Delta^r, \Delta^p, s_0)$. If D is the MTS obtained from E by applying Definition 5.2 and $D^{\forall}$ is obtained from D by Definition 5.3, then the following holds. The answer for $\mathcal{E}$ is *all* iff the LTS control problem $\mathcal{D}^{\forall} = \langle D^{\forall}, \mathcal{X}_{\{\ell_1\}}(\varphi), A_c \rangle$ is realisable.

The proof is very similar to that of Lemma 4.1 except that we restrict attention to deterministic implementations.

We now turn to the case of distinguishing between the *none* case and *all* or *some*. In this case, we can apply Definition 4.2 to D. Denote the resulting LTS as $D^{\exists}$. We note that as D has no nondeterminism on required transitions it follows that $D^{\exists}$ is deterministic.

Lemma 5.5: (Deterministic None) Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$ where $E = (S, A, \Delta^r, \Delta^p, s_0)$. If D is the MTS obtained from E by applying Definition 5.2 and $D^{\exists}$ is obtained from D by Definition 4.2, then the following holds. The answer for $\mathcal{E}$ is *none* iff the LTS control problem $\mathcal{D}^{\exists} = \langle D^{\exists}, \mathcal{X}_{\overline{A}}(\varphi), A_{D^{\exists}} - (A - A_c) \rangle$ is not realisable.

The proof is very similar to that of Lemma 4.2 except that we (naturally) restrict attention to deterministic implementations.

It follows that the following algorithm provides the answer to the MTS control problem in the case that we restrict attention to deterministic implementations.

Algorithm 3: (MTS Deterministic Control) Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$. If D is the LTS obtained from E by applying Definition 5.2 and $D^{\forall}$ is the LTS obtained by applying Definition 5.3 to D and $D^{\exists}$ is the LTS obtained by applying Definition 4.2 to D, then the answer for $\mathcal{E}$ is computed as follows.

- **All**, if there exists a solution for $\mathcal{D}^{\forall} = \langle D^{\forall}, \mathcal{X}_{\{\ell_1\}}(\varphi), A_c \rangle$
- **None**, if there is no solution for $\mathcal{D}^{\forall}$ and no solution for $\mathcal{D}^{\exists} = \langle D^{\exists}, \mathcal{X}_{\overline{A}}(\varphi), A_{D^{\exists}} - (A - A_c) \rangle$.
- **Some**, otherwise.

Theorem 5.2: (MTS Deterministic Control) Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$ the answer to the control problem of $\mathcal{E}$, where implementations are restricted to deterministic implementations, is decidable in 2EXPTIME in φ and polynomial in E.

Proof: The Theorem follows from noticing that both $D^{\forall}$ and $D^{\exists}$ are deterministic. Thus, the simpler algorithm for checking realisability of deterministic LTS from Theorem 2.2 applies. ■

VI. Discussion and Related Work

The software engineering community has studied the construction of event-based operational models in various forms. For instance, construction of such models from scenario-based specifications(e.g. [21]) has received much attention as example-based descriptions are close to wide-spread specification approaches such as message sequence charts and use-cases. Integration of fragmented, example-based specification into a state-based model can be analysed via model checking, simulation, animation and inspection, the latter aided by automated slicing and abstraction techniques.

Synthesis that also combines some form of declarative specification (e.g. temporal logics) has also been studied with the aim of providing an operational model on which to further support requirements elicitation and analysis [22]. The work presented herein shares the view that model elaboration can be supported through synthesis and analysis. Furthermore, analysis of a partial domain model for realisability of system goals by means of a controller allows prompting further elaboration of both domain model and goals.

Enacting automatically synthesised plans is a strategy adopted in various software engineering domains. For instance, Inverardi et al. automatically build glue code and component adaptors in order to achieve safe composition at the architecture level [4], and in particular in service oriented architectures [23]. Such approaches build on classical controller synthesis and consequently require fully specified domain models, hence their application is limited in earlier phases of development. The results that we present in this paper allows reasoning about the feasibility of constructing such glue code and adaptors without necessarily requiring the effort of developing a full domain model.

Architectural design of self-adaptive systems often includes layers that must deal with mission planning. In these layers, algorithms capable of producing at run-time strategies that adapt to changes in the environment, in the system's capabilities or goals is required. Hence, these systems can leverage automated

controller synthesis [24], [2]. We speculate that controller synthesis techniques that support partial domain knowledge, such as the one presented here, may allow deploying self-adaptive systems that work in environments for which there is more uncertainty.

In [17], we solved the MTS control problem for a restricted setting that considers only the case where both the MTS specification and its implementations are deterministic. Such case can be considered an even simpler case of the one described in Section V-A. However, as we have shown, the problem of MTS control with deterministic MTS specification and nondeterministic implementations is essentially reduced to the one presented in [17].

Our work builds on a particular formalism for describing partial behaviour models (MTS [12]), however, there are many formalisms that have been developed with the notion of partiality explicitly embedded. Notably, there exist many (more expressive) variants of MTS including Disjunctive [25], and Parametric [26] MTS. The results presented in this work would have to be revisited in the context of other partial behaviour formalisms. However, since many complexity results for MTS hold for extensions such as DMTS, we believe that our results could also extend naturally to these extensions.

Initial attention to partial models was focused on property verification (cf. [11], [27], [28]). First, three-valued model checking was defined [11] and shown to have the same complexity as that of model checking. Subsequently, generalised model checking [27] improves the accuracy of results: three-valued model checking may result in the answer "some" even when no implementations satisfy the property. Generalised model checking resolves this but at a high computational complexity [28].

In order to reason about generalised model checking one has to go from the model of transition systems (for 3-valued model checking) to that of a game. Our definition of MTS control is more similar to generalised model checking than to 3-valued model checking. We find it interesting that both MTS and LTS control problems are solved in the same model (that of a game) and that MTS control does not require a more general model.

Another related subject is abstraction of games. For example, in [29] abstraction is applied to games in order to enable reasoning about infinite games. Similarly, in [30] abstraction refinement is generalised to reason about larger games. Applying abstraction allows making assumptions about which states can be reasoned about together. In our approach, we work with a given abstraction from the start, the MTS, rather than abstracting a more detailed model.

We have mentioned some of the many results that exist on controller synthesis and realizability of temporal logic. Our work builds on LTS control using the 2EXPTIME-completeness of LTL controller synthesis [9] and also allows use of more efficient synthesis defined over restricted subsets of LTL (cf. [31], [14]). The latter results show that in some cases synthesis can be applied in practice. Similar restrictions, if applied to MTS control combined with our reductions, produce the same reduction in complexity.

Our previous work on usage of controller synthesis in the context of LTSs has been incorporated in the MTSA toolset [32]. We have implemented a solver to GR(1) [14] formulas in the context of the LTS control problem [8].

In addition to dealing with partial behaviour models, our work allows non-deterministic behaviour. Non-determinism and partial observation (the existence of actions which can neither be controlled nor monitored by the controller) are closely related challenges in the realm of games. Partial observation boils down to safety imperfect-information games [33]. Very recently, we have presented two techniques for synthesis of LTS controllers in the context of nondeterministic environments [34]. Each technique produces controllers for different interactions modes between the environment and the controller. One similar to that of IA legal environments and the other inspired in the interaction among players in an imperfect-information game.

Partial observable Markov Decision Processes (MDPs) are used in robotics for planning purposes (e.g. [35]). A key difference is that the environment is given in terms of stochastic behaviour of actions and that the aim is the maximization of cumulative payoffs, there are no hard goal. Optimal policies under hard co-safety properties has only been recently addressed in the robotics community ([36]).

In [5] safety properties (and bounded-liveness) are used as goals in an LTS-like framework. The technique assumes input enabledness but the resulting controller is legal, in the sense of interface automata – i.e. the controller cannot block uncontrollable actions, nor the environment block controllable ones. Thus, the controller disables all controllable transitions that are not enabled from every possible nondeterministic successors. Hence, the approach in [5] cannot be reused for the general interface automata control problem since liveness requires special treatment as shown, e.g., in [8].

VII. Conclusions and Future Work

We have presented a general technique for answering the MTS control problem. In other words, an algorithm for answering if all, some or none of the implementations described by a partial behaviour model in the form of an MTS can be controlled to achieve a given goal. Our technique is general in the sense that it does not restrict MTS to deterministic specifications nor does it limit the implementations considered to deterministic ones.

We show that the answer to the MTS control problem can be computed by considering two LTS control problems representing the "hardest" and "easiest" implementations to control. In addition, we identified restricted versions of the MTS control problem which can be answered more efficiently as it is not necessary to consider control of non-deterministic implementations.

For the general case, the two LTS control problems must be applied to nondeterministic LTS which means an exponential complexity in the size of the domain model. However, for the restricted MTS control problems identified, we show that it is enough to consider the two deterministic LTS control problems, reducing the complexity of the problem to polynomial in the size of the domain model.

The complexity of deciding controllability also depends on the complexity of the goal to be controlled. In the general case (general FLTL formula) the complexity is double exponential in the size of the goal. However, our technique can build on existing results for solving more efficiently sub-logics. Hence, for GR(1) formulae the complexity is remains polynomial.

We believe that from the LTS control problems that are presented in this paper it is possible to construct "templates" for the control of all implementations and "templates" for the control of "controllable" implementations when only some are controllable. Extraction of these templates and their usage for the refinement of the domain models or their restriction is an interesting area for further studies.

We are currently working on implementation of these algorithms and their incorporation within the realisability framework of MTSA [32]. In spite of the complexity of the control problem for nondeterministic LTS our initial experiments are encouraging and we hope that the cases that arise in practice will not incur the full theoretical complexity of the solution.

References

[1] M. Mazo, A. Davitian, and P. Tabuada, "Pessoa: A tool for embedded controller synthesis," in *Proceedings of the 22Nd International Conference on Computer Aided Verification*, ser. CAV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 566–569. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14295-6_49

[2] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "An architecture for requirements-driven self-reconfiguration," in *Proceedings of the 21st International Conference on Advanced Information Systems Engineering*, ser. CAiSE '09. Springer-Verlag, 2009, pp. 246–260.

[3] W. Heaven, D. Sykes, J. Magee, and J. Kramer, "Software engineering for self-adaptive systems," B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. A Case Study in Goal-Driven Architectural Adaptation, pp. 109–127. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02161-9_6

[4] P. Inverardi and M. Tivoli, "A reuse-based approach to the correct and automatic composition of web-services," in *International workshop on Engineering of software services for pervasive environments: in conjunction with the 6th ESEC/FSE joint meeting*, ser. ESSPE '07. New York, NY, USA: ACM, 2007, pp. 29–33. [Online]. Available: http://doi.acm.org/10.1145/1294904.1294908

[5] E. Letier and W. Heaven, "Requirements modelling by synthesis of deontic input-output automata," in *35th International Conference on Software Engineering*, ser. ICSE 2013. Norwell, MA, USA: Kluwer Academic Publishers, 2013.

[6] M. Jackson, "The world and the machine," in *Proceedings of the 17th international conference on Software engineering*, ser. ICSE '95. ACM, 1995, pp. 283–292.

[7] N. D'Ippolito, V. A. Braberman, N. Piterman, and S. Uchitel, "Synthesis of live behaviour models for fallible domains," in *ICSE*, R. N. Taylor, H. Gall, and N. Medvidovic, Eds. ACM, 2011, pp. 211–220.

[8] N. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel, "Synthesising non-anomalous event-based controllers for liveness goals," *ACM Tran. Softw. Eng. Methodol.*, vol. 22, 2013.

[9] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ser. POPL '89. ACM, 1989, pp. 179–190.

[10] W. Swartout and R. Balzer, "On the inevitable intertwining of specification and implementation," *Communications of the ACM*, vol. 25, no. 7, pp. 438–440, 1982.

[11] G. Bruns and P. Godefroid, "Model checking partial state spaces with 3-valued temporal logics," in *11th International Conference on Computer Aided Verification*, ser. Lecture Notes in Computer Science, vol. 1633. Springer, 1999, pp. 274–287.

[12] K. Larsen and B. Thomsen, ""A Modal Process Logic"," in *Proceedings of 3rd Annual Symposium on Logic in Computer Science (LICS'88)*. IEEE Computer Society Press, 1988, pp. 203–210.

[13] D. Giannakopoulou and J. Magee, "Fluent model checking for event-based systems," in *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, ser. ESEC/FSE-11. ACM, 2003, pp. 257–266.

[14] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," *J. Comput. Syst. Sci.*, vol. 78, no. 3, pp. 911–938, 2012.

[15] R. M. Keller, "Formal verification of parallel programs," *Communications of the ACM*, vol. 19, pp. 371–384, July 1976.

[16] J. H. Reif, "Universal games of incomplete information," in *STOC*, M. J. Fischer, R. A. DeMillo, N. A. Lynch, W. A. Burkhard, and A. V. Aho, Eds. ACM, 1979, pp. 288–308.

[17] N. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel, "The modal transition system control problem," in *19th International Symposium on Formal Methods*, ser. Lecture Notes in Computer Science, vol. 7436. Paris, France: Springer-Verlag, 2012, pp. 155–170.

[18] A. van Lamsweerde, *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[19] N. D'Ippolito, "Technical Report."

[20] D. Martin, "Borel determinacy," *The annals of Mathematics*, vol. 102, no. 2, pp. 363–371, 1975.

[21] Y. Bontemps, P.-Y. Schobbens, and C. Löding, "Synthesis of open reactive systems from scenario-based specifications," *Fundamenta Informaticae - Application of Concurrency to System Design (ACSD'03)*, vol. 62, pp. 139–169, February 2004.

[22] C. Damas, B. Lambeau, and A. van Lamsweerde, "Scenarios, goals, and state machines: a win-win partnership for model synthesis," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, ser. SIGSOFT '06/FSE-14. ACM, 2006, pp. 197–207.

[23] A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli, "Automatic synthesis of behavior protocols for composable web-services," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ser. ESEC/FSE '09. ACM, 2009, pp. 141–150.

[24] D. Sykes, W. Heaven, J. Magee, and J. Kramer, "Plan-directed architectural change for autonomous systems," in *SAVCBS*, 2007, pp. 15–21.

[25] K. G. Larsen and L. Xinxin, "Equation solving using modal transition systems," in *LICS*. IEEE Computer Society, 1990, pp. 108–117.

[26] N. Benes, J. Kretínský, K. G. Larsen, M. H. Møller, and J. Srba, "Parametric modal transition systems," in *ATVA*, ser. Lecture Notes in Computer Science, T. Bultan and P.-A. Hsiung, Eds., vol. 6996. Springer, 2011, pp. 275–289.

[27] G. Bruns and P. Godefroid, "Generalized model checking: Reasoning about partial state spaces," in *11th International Conference on Concurrency Theory*, ser. Lecture Notes in Computer Science, vol. 1877. Springer, 2000, pp. 168–182.

[28] P. Godefroid and N. Piterman, "Ltl generalized model checking revisited," in *Proceedings of the 10th International Conference on Verification, Model Checking, and Abstract Interpretation*, ser. VMCAI '09. Springer-Verlag, 2009, pp. 89–104.

[29] P. Stevens, "Abstract games for infinite state processes," in *9th International Conference on Concurrency Theory*, ser. Lecture Notes in Computer Science, vol. 1466. Springer, 1998, pp. 147–162.

[30] T. A. Henzinger, R. Jhala, and R. Majumdar, "Counterexample-guided control," in *30th International Colloquium on Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, vol. 2719. Springer, 2003, pp. 886–902.

[31] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, "Controller synthesis for timed automata," in *Proceedings of the IFAC Symposium on System Structure and Control*, 1998.

[32] N. D'Ippolito, D. Fischbein, M. Chechik, and S. Uchitel, "Mtsa: The modal transition system analyser," in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '08. IEEE Computer Society, 2008, pp. 475–476.

[33] K. Chatterjee, T. A. Henzinger, and B. Jobstmann, "Environment assumptions for synthesis," in *Proceedings of the 19th international conference on Concurrency Theory*, ser. CONCUR '08. Springer-Verlag, 2008, pp. 147–161.

[34] N. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel, "Synthesis of event-based controllers for non-deterministic environments," in *Submitted to 20th International Symposium on Formal Methods. available at http://www.doc.ic.ac.uk/ srdipi/techacsd2014/dbpu2014a.pdf*, ser. Lecture Notes in Computer Science, vol. 7436. Paris, France: Springer-Verlag, 2012, pp. 155–170.

[35] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[36] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *I. J. Robotic Res.*, vol. 32, no. 8, pp. 889–911, 2013.