The Modal Transition System Control Problem [⋆]

Nicolás D'Ippolito[1], Victor Braberman[2],
Nir Piterman[3], and Sebastián Uchitel[1,2]

[1] Computing Department, Imperial College London, London, UK
[2] Departamento de Computatión, FCEyN, Universidad de Buenos Aires, Argentina
[3] Department of Computer Science, University of Leicester, Leicester, UK

Abstract. Controller synthesis is a well studied problem that attempts to automatically generate an operational behaviour model of the system-to-be such that when deployed in a given domain model that behaves according to specified assumptions satisfies a given goal. A limitation of known 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. In this paper we study the controller synthesis problem when there is partial behaviour information about the problem domain. More specifically, we define and study the controller realisability problem for domains described as Modal Transition Systems (MTS). An MTS is a partial behaviour model that compactly represents a set of complete behaviour models in the form of Labelled Transition Systems (LTS). Given an MTS we ask if all, none or some of the LTS it describes admit an LTS controller that guarantees a given property. We show a technique that solves effectively the MTS realisability problem and is in the same complexity class as the corresponding LTS problem.

1 Introduction

Michael Jackson's Machine-World model [15] establishes a framework on which to approach the challenges of requirements engineering. In this model, requirements R are prescriptive statements of the world expressed in terms of phenomena on the interface between the machine we are to build and the world in which the real problems to be solved live. Such problems are to be captured with prescriptive statements expressed in terms of phenomena in the world (but not necessarily part of the world-machine interface) called goals G and descriptive statements of what we assume to be true in the world (domain model D).

Within this setting, a key task in requirements engineering is to understand and document the goals and the characteristics of the domain in which these are to be achieved, in order to formulate a set of requirements for the machine to be built such that assuming that the domain description and goals are valid, the requirements in such domain entail the goals, more formally $R, D \models G$.

[⋆] This work was partially supported by grants ERC PBM-FIMBSE, MEALS 295261, CONICET PIP955, UBACYT X021, and PICT PAE 2272.

Thus, a key problem of requirements engineering can be formulated as a synthesis problem. Given a set of descriptive assumptions on the environment behaviour and a set of system goals, construct an operational model of the machine such that when composed with the environment, the goals are achieved. Such problem is known as the controller synthesis [24] problem and has been studied extensively resulting in techniques which have been used in various software engineering domains.

Controller synthesis [24] is a well studied problem that attempts to automatically generate an operational behaviour model of the system-to-be such that when deployed in a given environment that behaves according to specified assumptions satisfies a given goal. Controller synthesis techniques have been used in several domains such as safe synthesis of web services composition [14] or synthesis of adaptation strategies in self-adaptive systems [26].

In practice, requirements engineering is not a waterfall process. Engineers do not build a complete description for G and D before they construct or synthesise R. Typically D, G and R are elaborated incrementally. Furthermore, multiple variations of partial models of D, G and R are explored to asses risk, cost and feasibility [18]. In particular a key question that drives requirements engineering forward and consequently drives elaboration of a partial description of D, G and R is if it is feasible to extend them to D', G' and R' such that $R', D' \models G'$.

In this context, existing controller synthesis techniques are not such a good fit because they require complete domain descriptions. Typically, the domain is described in a formal language with its semantics defined as some variation of a two-valued state machine such as Labelled Transition Systems (LTS) [17] or Kripke structures. Thus, the domain model is assumed to be complete up to some level of abstraction (i.e, with respect to an alphabet of actions or propositions).

An appropriate formalism to support modelling when behaviour information is lacking is one in which currently unknown aspects of behaviour can be explicitly modelled [27]. A number of such formalisms exist such as Modal Transition Systems (MTS) [19] and Disjunctive MTS [20]. Partial behaviour models can distinguish between required, possible, and proscribed behaviour.

In this paper, we define controller synthesis in the context of partially specified domain models. More specifically, we study the problem of checking the existence of an LTS controller (i.e. controller realisability) capable of guaranteeing a given goal when deployed in a completely defined LTS domain model that conforms to the partially defined domain model given as an MTS.

The semantics of MTS is given in terms of a set of LTS implementations in which each LTS provides the required behaviour described in the MTS and does not provide any of the MTS proscribed behaviour. We define the *MTS control problem* as follows: given an MTS we ask if *all*, *none* or *some* of the LTS implementations it describes admit an LTS controller that guarantees a given goal given as a Fluent Linear Temporal Logic [11] formula. The realisability question we address in the context of MTS has a three valued answer.

From a model elaboration perspective, a *none* response indicates that there is no hope of building a system that satisfies the goals independently of the aspects of the domain that have been modelled as uncertain. This entails that either goals must be weakened or stronger assumptions about the domain must be made. An *all* response indicates that the partial domain knowledge modelled is sufficient to guarantee that the goals can be achieved, consequently further elaboration may not be necessary. Finally, a *some* response indicates that further elaboration is required. Feedback as to why in some domains which conform to the partial model the goal may not be realisable may be good indicators as to in which direction should elaboration proceed. Note that the latter, feedback on *some* realisability, is beyond the scope of this paper.

The technique we present yields an answer to the MTS control problem showing that, despite dealing with a potentially infinite number of LTS, the MTS control problem is actually in the same complexity class as the underlying LTS synthesis problem. The results for MTS realisability can be used with controller synthesis techniques that deal efficiently with restricted yet expressive goals such as [1, 22]. Note that our results are limited to deterministic domain models.

The rest of this paper is organised as follows. In Section 2 we introduce the required concepts and notations. Then, in Section 3 we define the MTS control problem and show how to solve it. We then optimise our algorithmic solution to achieve optimal complexity bounds in Section 4. Finally, we discuss related work in Section 5 and conclude in Section 6.

Due to lack of space all proofs are omitted and given in [7].

2 Preliminaries

2.1 Transition Systems

We fix notation for labelled transition systems (LTSs) [17], which are widely used for modelling and analysing the behaviour of concurrent and distributed systems. 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 1. (Labelled Transition Systems [17]) *Let States be the universal set of states, Act be the universal set of action labels. 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.

Definition 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 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) [19] 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 4. (Modal Transition Systems [19]) *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 denote by $\Delta^p(s)$ the set of possible actions enabled in s, namely $\Delta^p(s) = \{\ell \mid \exists s' \cdot (s, \ell, s') \in \Delta^p\}$. Similarly, $\Delta^r(s)$ denotes the set of required actions enabled in s.

Definition 5. (Refinement) *Let $M = (S, A, \Delta^r_M, \Delta^p_M, s^M_0)$ and $N = (T, A, \Delta^r_N, \Delta^p_N, s^N_0)$ 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 $(s, \ell, s') \in \Delta^r_M$ then there is t' such that $(t, \ell, t') \in \Delta^r_N$ and $(s', t') \in H$.*
- *If $(t, \ell, t') \in \Delta^p_N$ then there is s' such that $(s, \ell, s') \in \Delta^p_M$ and $(s', t') \in H$.*

We say that N refines M if there is a refinement relation H between M and N such that $(s^M_0, s^N_0) \in H$, denoted $M \preceq N$.

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 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 $\mathcal{I}(M)$.*

An implementation is *deadlock free* if all states have outgoing transitions. 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 LTS E is *deterministic* if $(s, \ell, s') \in \Delta_E$ and $(s, \ell, s'') \in \Delta_E$ implies $s' = s''$. For a state s we denote $\Delta(s) = \{\ell \mid \exists s' \cdot (s, \ell, s') \in \Delta\}$. We refer to the set of all deterministic implementations of an MTS M as $\mathtt{I}^{det}[M]$.

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

Fig. 1. Semantics for the satisfaction operator.

2.2 Fluent Linear Temporal Logic

We describe properties using Fluent Linear Temporal Logic (FLTL) [11]. Linear temporal logics (LTL) [23] are widely used to describe behaviour requirements [11, 21]. The motivation for choosing an LTL of fluents is that it provides a uniform framework for specifying and model-checking state-based temporal properties in event-based models [11]. An LTL formula checked against an LTS model requires interpreting propositions as the occurrence of events in the LTS model. Some properties can be rather cumbersome to express as sequences of events, while describing them in terms of states is simpler. Fluents provide a way of defining abstract states. 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 $\dot{\ell} = \langle \ell, Act \setminus \{\ell\}, false\rangle$.

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 **X** (next), **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 $\wedge$, $\diamondsuit$ (eventually), and $\Box$ (always) 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. The interval over which a fluent holds is *closed* on the left and *open* on the right, since actions have an immediate effect on the value of fluents.

Given an infinite trace π, the satisfaction of a formula φ at position i, denoted $\pi, i \models \varphi$, is defined as shown in Figure 1. 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.

Consider P, shown in Figure 2(a), and the FLTL formula $\phi = \neg\dot{idle}\mathbf{U}Cooking$, where $Cooking = \langle\{cook\}, \{doneCooking\}, false\rangle$, and the trace $\pi = idle$, $cook$, $doneCooking$, $moveToBelt$, $cook$, $doneCooking$, $\ldots$ of the LTS shown in Figure 2(a). Since at position 1 $\dot{idle}$ holds (i.e, $\pi, 1 \models \dot{idle}$) but $Cooking$ does not (i.e, $\pi, 1 \not\models Cooking$) it follows that $\pi, 0 \not\models \phi$. On the other hand, at time 2 $\dot{idle}$

does not holds (i.e, $\pi, 2 \not\models \dot{idle}$) but *Cooking* does hold (i.e, $\pi, 2 \models Cooking$), hence, $\pi, 2 \models \phi$. Note that ϕ holds in P, i.e, $P \models \phi$.

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 *alphabetised next* version of φ, denoted $\mathcal{X}_\Gamma(\varphi)$, as follows.

– For a fluent $Fl \in \mathcal{F}$ we define $\mathcal{X}_\Gamma(Fl) = Fl$.
– For $\varphi \vee \psi$ we define $\mathcal{X}_\Gamma(\varphi \vee \psi) = \mathcal{X}_\Gamma(\varphi) \vee \mathcal{X}_\Gamma(\psi)$.
– For $\neg\varphi$ we define $\mathcal{X}_\Gamma(\neg\varphi) = \neg\mathcal{X}_\Gamma(\varphi)$.
– For $\varphi\mathbf{U}\psi$ we define $\mathcal{X}_\Gamma(\varphi\mathbf{U}\psi) = \mathcal{X}_\Gamma(\varphi)\mathbf{U}\mathcal{X}_\Gamma(\psi)$.
– For $\mathbf{X}\varphi$ we define $\mathcal{X}_\Gamma(\mathbf{X}\varphi) = \mathbf{X}((\bigvee_{f\in\Gamma} f)\mathbf{U}\mathcal{X}_\Gamma(\varphi))$

Thus, this transformation replaces every next operator occurring in the formula by an until operator that skips uninteresting actions that are in Γ. The transformations in Section 3 force an action not in Γ to appear after every action from Γ. Thus, the difference between $\mathbf{U}$ under even and odd number of negations is not important. Given a trace $\pi = \ell_0, \ell_1, \ldots$, we say that $\pi' = \ell'_0, \ell'_1, \ldots$ is a Γ-variant of π if there is an infinite sequence $i_0 < i_1 < \ldots$ such that $\ell_j = \ell_{i_j}$ for every j. That is, π' is obtained from π by adding a finite sequences of actions from Γ between actions in π.

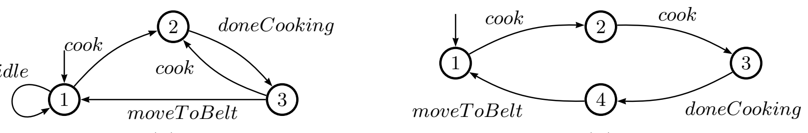
Theorem 1. *Given a trace $\pi = \ell_0, \ell_1, \ldots$ in $E = (S, A, \Delta, s_0)$, an FLTL formula φ and a set of actions $\Gamma \in Act$. If $\Gamma \cap A = \emptyset$ then the following holds. For every trace π' that is a Γ-variant of π we have $\pi \models \varphi$ iff $\pi' \models \mathcal{X}_\Gamma(\varphi)$.*

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.

2.3 LTS Controller Synthesis

Given a domain model, which is a description of what is known about the world, the problem of controller synthesis is to construct a machine / controller that will interact with the world and ensure that certain goals are fulfilled. In our context, the domain model is given as and LTS and the goal of the machine is defined as an FLTL formula. The interface between the machine and the domain model is given by partitioning the events that can occur to those that are controllable by the machine and those that are uncontrollable by it. Then, the controller restricts the occurrence of events it controls to ensure that its goals are fulfilled.

Definition 7. (LTS Control [8]) *Given a domain model in the form of a deterministic 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 $A_M = A$, from every state in S_M all actions in $A_M \backslash A_c$ are enabled, $E\|M$ is deadlock free, and every trace π in $E\|M$ is such that $\pi \models \varphi$.*



(a) P. (b) C_P

Fig. 2. Ceramic Cooking Example

That is, looking for a solution of an LTS control problem with domain model E, is to verify the existence of an LTS M such that when composed in parallel with E (i.e. $E \| M$), it does not block uncontrollable actions in E and every trace of $E \| M$ satisfies a given FLTL goal φ (i.e. $E \| M \models \varphi$).

We may refer to the solution of an LTS control problem as controller or LTS controller. Whenever such a controller exists we say that the control problem is realisable and 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 $\mathcal{E} = \langle E, \varphi, A_c \rangle$.

Theorem 2. (LTS Control [24]) *Given an LTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$ it is decidable in 2EXPTIME whether $\mathcal{E}$ is realisable. The algorithm checking realisability can also extract a controller M.*

Note that determinism of the domain model is required. As LTS controllers guarantee the satisfaction of their goals through parallel composition, having nondeterministic domain models means that the controller would not be able to know the exact state of the domain model. This leads to imperfect information, as the controller would only be able to deduce which *set* of states the domain model is in. Translation of the existing results on synthesis with imperfect information to the context of nondeterministic LTSs is out of the scope of this paper.

For example, consider E, the simple domain model in Figure 2(a), where a ceramics cooking process is described. The aim of the controller is to produce cooked ceramics by taking raw pieces from the in-tray, placing them in the oven and moving them once cooked to a conveyor belt. In addition, raw pieces have to be cooked twice before being moved to the conveyor belt. A natural solution for such a problem is to build a controller guaranteing that raw pieces are cooked twice and moved to the conveyor belt infinitely often. The solution for this simple example is shown in Figure 2(b). Note that the controller has the memory needed to remember how many times a piece has been cooked.

3 MTS Control Problem

The problem of control synthesis for MTS is to check whether all, none or some of the LTS implementations of a given MTS can be controlled by an LTS controller [9]. 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 defined formally below.

Definition 8. (Semantics of MTS Control) *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, to solve the* MTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$ *is to answer:*

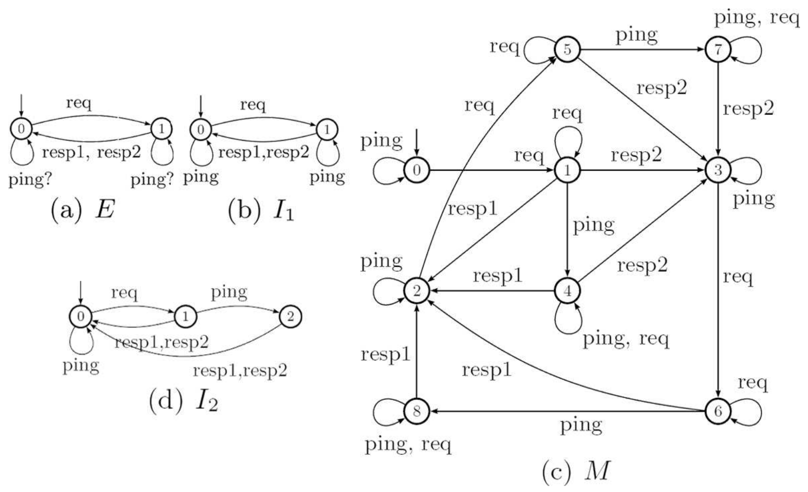


Fig. 3. Server Example.

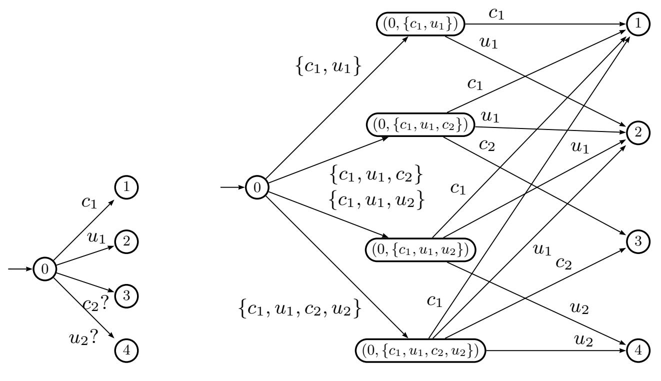
– ***All****, if for all LTS $I \in \mathcal{I}^{det}[E]$, the control problem $\langle I, \varphi, A_c\rangle$ is realisable,*
– ***None****, if for all LTS $I \in \mathcal{I}^{det}[E]$, the control problem $\langle I, \varphi, A_c\rangle$ is unrealisable,*
– ***Some****, otherwise.*

Note that, as in the case of LTS control problem, we restrict attention to deterministic domain models. This follows from the fact that our solution for MTS realisability is by a reduction to LTS realisability.

Consider E, shown in Figure 3(a), that describes the interactions between a server and clients. Note that although it is certain that client requests can be responded by the server, definitions regarding when clients may ping the server have not been made yet. Suppose that we want to build a controller for this server such that the server guarantees that after receiving a request it will eventually yield a response and if there are enough requests, responses of both kinds will be issued. We formally describe this requirement as the FLTL formula: $\varphi = \Box\Diamond\neg ResponseOwed \wedge (\Box\Diamond \dot{req} \Rightarrow (\Box\Diamond \dot{resp_1} \wedge \Box\Diamond \dot{resp_2}))$, where $ResponseOwed = \langle\{req\}, \{resp_1, resp_2\}, false\rangle$. As expected, the server can only control the response. Hence, we have the MTS control problem $\mathcal{E} = \langle E, \varphi, \{resp_1, resp_2\}\rangle$. Consider the implementation I_1, shown in Figure 3(b). The uncontrollable self loop over ping in state 1 allows the environment to flood the controller impeding it from eventually producing a response (i.e. no controller can avoid the trace $req, ping, ping, \ldots$). The implementation I_2, shown in Figure 3(d), allows only a bounded number of pings after a request, hence, the server cannot be flooded and a controller for the property exists . Since I_1 and I_2 are implementations of E such that I_2 can be controlled and I_2 cannot, it follows that the answer for the MTS control problem $\mathcal{E}$ is *some*.

A naive approach to the MTS control problem may require to evaluate an infinite number of LTS control problems. Naturally, such approach is not possible, hence, it is mandatory to find alternative ways to handle MTS control problems.

We reduce the MTS control problem to two LTS control problems. The first LTS control problem encodes the problem of whether there is a controller for each



(a) E (b) E^I

Fig. 4.

implementation described by the MTS. It does so by modelling an environment that can pick the "hardest" implementation to control. In fact, in the LTS control problem, the environment will pick at each point the subset of possible transitions of the MTS that are available. If there is a controller for this environment, there is a controller for all implementations.

The second LTS control problem encodes the problem of whether there is no controller for every implementation of the MTS. Similarly, this is done by modelling an LTS control problem in which the controller can pick the "easiest" implementation to control (in fact, it is now the controller that picks the subset of possible transitions of the MTS that are available at each point). If there is no controller in this setting, then for every implementation there is no controller.

The two LTS problems are defined in terms of the same LTS. The only difference is who controls the selection of the subset of possible actions, i.e. implementation choice. We now define the LTS E^I in which additional transition labels are added to model explicitly when either the controller or the environment choose which subset of possible transitions of the MTS are available.

Definition 9. *Given an MTS $E = (S, A, \Delta^r, \Delta^p, s_0)$. We define $E^I = (S_{E^I}, A_{E^I}, \Delta_{E^I}, s_0)$ as follows:*
- $S_{E^I} = S \cup \{(s, i) \mid s \in S \text{ and } i \subseteq A \text{ and } \Delta^r(s) \subseteq i\}$
- $A_{E^I} = A \cup \overline{A}$, *where* $\overline{A} = \{\ell_i \mid i \subseteq A\}$
- $\Delta_{E^I} = \begin{array}{l} \{(s, \ell_i, (s, i)) \mid s \in S \text{ and } i \subseteq \Delta^p(s) \text{ and } \Delta^r(s) \subseteq i\} \cup \\ \{((s, i), \ell, s') \mid (s, \ell, s') \in \Delta^p \text{ and } \ell \in i\} \end{array}$

States in E^I are of two kinds. Those that are of the form s with $s \in S$ encode states in which a choice of which subset of possible transitions are implemented has to be made. Choosing a subset $i \subseteq A$, leads to a state (s, i). States of latter form (s, i) have outgoing transitions labelled with actions in i. A transition from (s, i) on an action $\ell \in i$ leads to the same state s' in E^I as taking ℓ from s in E. For example, the model in Figure 4(b) is obtained by applying Definition 9 to model in Figure 4(a).

The LTS E^I provides the basis for tractably answering the MTS control question. The following algorithm shows how to compute the solution for the MTS control problem.

Algorithm 1 (MTS Control) *Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c\rangle$. If E^I is the LTS model obtained by applying Definition 9 to E, then the answer for $\mathcal{E}$ is computed as follows.*

- ***All**, if there exists a solution for $\mathcal{E}^I_A = \langle E^I, \mathcal{X}_{\overline{A}}(\varphi), A_c\rangle$*
- ***None**, if there is no solution for $\mathcal{E}^I_N = \langle E^I, \mathcal{X}_{\overline{A}}(\varphi), A_c \cup \overline{A}\rangle$*
- ***Some**, otherwise.*

Algorithm 1 shows how to compute the answer for a given MTS control problem.

Consider the case in which the answer for the MTS control problem is *all*. As stated by Algorithm 1, the answer to $\mathcal{E}$ is *all*, if there is solution to the LTS control problem $\mathcal{E}^I_A$. Intuitively, if we give control over the new actions ℓ_i to the environment, it can choose the hardest implementation to control. Thus, this solves the question of whether all implementations are controllable.

Lemma 1 proves that the case *all* in Algorithm 1 is sound and complete.

Lemma 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^I is the LTS obtained by applying Definition 9 to E, then the following holds. The answer for $\mathcal{E}$ is* all *iff the LTS control problem $\mathcal{E}^I_A = \langle E^I, \mathcal{X}_{\overline{A}}(\varphi), A_c\rangle$ is realisable.*

Consider the case in which the answer for the MTS control problem is *none*. The answer to $\mathcal{E}$ is *none*, if there is no solution to the LTS control problem $\mathcal{E}^I_N$. Intuitively, if we give control over the new actions ℓ_i to the controller, it can choose the easiest implementation to control. Thus, this solves the question of whether no implementation is controllable.

Lemma 2 proves that the case *none* in Algorithm 1 is sound and complete.

Lemma 2. ***(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 E^I is the LTS obtained by applying Definition 9 to E, then the following holds.*

The LTS control problem $\mathcal{E}^I_N = \langle E^I, \mathcal{X}_{\overline{A}}(\varphi), A_c \cup \overline{A}\rangle$ is realisable iff there exists $I \in \mathcal{I}^{det}[E]$ such that the LTS control problem $\mathcal{I} = \langle I, \varphi, A_c\rangle$ is realisable.

The answer to $\mathcal{E}$ is *some* whenever there exists an implementation of E that can be controlled and an implementation of E that cannot be controlled.

Lemma 3. ***(Some)*** *Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c\rangle$. The answer for $\mathcal{E}$ is* some *iff $\mathcal{E}^I_A$ is unrealisable and $\mathcal{E}^I_N$ is realisable.*

4 Linear Reduction into LTS Control Problems

Algorithm 1 shows that the MTS control problem can be reduced to two LTS control problems. Hence, our solution to the MTS control problem is, in general, doubly exponential in the size of E^I (cf. [24, 8]). Unfortunately, the state space of E^I is exponential in the branching degree of E, which in turn is bounded by the size of the alphabet of the MTS. More precisely, for a state $s \in E^I$ the number of successors of s is bounded by the number of possible combinations of labels of maybe transitions from s in E. In this section we show that to compute the answer for $\mathcal{E}^I_A$and $\mathcal{E}^I_N$it is enough to consider only a small part of the states of E^I. Effectively, it is enough to consider at most linearly (in the number of outgoing transitions) many successors for every state. This leads to the MTS control problem being 2EXPTIME-complete.[4]

First, we analyse E^I in the context of $\mathcal{E}^I_A$. We define a fragment $\mathcal{E}^{I^+}_A$of $\mathcal{E}^I_A$. Let $E^{I^+} = (S_{E^I}, A_{E^I}, \Delta^+, s_{0_{E^I}})$, where only the following transitions from Δ_{E^I} are included in Δ^+.

1. Consider a state $s \in E$ that has at least one required uncontrollable successor. In Δ^+ we add to s only the transition $(s, \ell_i, (s,i))$, where $i = \Delta^r_E(s) \cup (\Delta^p_E(s) \cap A_\mu)$. That is, in addition to required transitions from s we include all uncontrollable possible successors of s.
2. Consider a state $s \in E$ that has no required uncontrollable successors but has a required controllable successor. In Δ^+ we add to s only the transitions $(s, \ell_i, (s,i))$, where i is either $\Delta^r_E(s)$ or i is $\Delta^r_E(s) \cup (\Delta^p_E(s) \cap A_\mu)$. That is, we include a transition to all required transitions from s as well as augmenting all required transitions by all uncontrollable possible transitions.
3. Consider a state $s \in E$ that has no required successors. In Δ^+ we add to s a transition to $(s, \ell_i, (s,i))$, where $i = \Delta^p_E(s) \cap A_\mu$, and for every $\ell \in \Delta^p_E(s) \cap A_c$ we add to s the transition $(s, \ell_{\{\ell\}}, (s, \{\ell\}))$. That is, we include a transition to all possible uncontrollable transitions from s and for every possible controllable transition a separate transition.
4. For a state (s,i) we add to Δ^+ all the transitions in Δ_{E^I}.

Lemma 4. *The problem $\mathcal{E}^I_A$is realisable iff $\mathcal{E}^{I^+}_A = \langle E^{I^+}, \mathcal{X}_{\overline{A}}(\varphi), A_c\rangle$ is realisable.*

We now analyse E^I in the context of the $\mathcal{E}^I_N$. We define a fragment $\mathcal{E}^{I^-}_N$ of $\mathcal{E}^I_N$. Let $E^{I^-} = (S_{E^I}, A_{E^I}, \Delta^-, s_{0,E^I})$, where only the following transitions from Δ_{E^I} are included in Δ^-.

1. Consider a state $s \in E$ that has at least one required uncontrollable successor. In Δ^- we add to s only the transition $(s, \ell_i, (s,i))$, where $i = \Delta^r_E(s)$. That is, include only the required transitions from s.
2. Consider a state $s \in E$ that has no required uncontrollable successors. In Δ^- we add to s a transition to $(s, \ell_i, (s,i))$, where $i = \Delta^r_E(s) \cup (\Delta^p_E(s) \cap A_c)$, and for every $\ell \in \Delta^p_E(s) \cap A_\mu$ we add to s the transition to $(s, \ell_{\Delta^r_E(s) \cup \{\ell\}}, (s, \Delta^r_E(s) \cup \{\ell\}))$. That is, we include a transition to all controllable transitions from s and for every possible uncontrollable transition a separate transition.
3. For a state (s,i) we add to Δ^- all the transitions in Δ_{E^I}.

[4] We can avoid adding states altogether by having a per state definition of what are controllable and uncontrollable actions. For simplicity of presentation we choose to add states. The modification is not complicated. In an enumerative implementation of game analysis this would be our suggested treatment.

Lemma 5. *The problem $\mathcal{E}^I_N$is realisable iff $\mathcal{E}^{I^-}_N = \langle E^{I^-}, \mathcal{X}_{\overline{A}}(\varphi), A_c \cup \overline{A} \rangle$ is realisable.*

Using $\mathcal{E}^{I^+}_A$ and $\mathcal{E}^{I^-}_N$ can establish the complexity of the MTS control problem.

Theorem 3. (MTS Control Complexity) *Given an MTS control problem $\mathcal{E} = \langle E, \varphi, A_c \rangle$ it is 2EXPTIME-complete to decide whether the answer to $\mathcal{E}$ is all, none, or some.*

5 Discussion and Related Work

Automated construction of event-based operational models of intended system behaviour has been extensively studied in the software engineering community.

Synthesis from scenario-based specifications (e.g. [27, 6]) allows integrating a fragmented, example-based specification into a model which can be analysed via model checking, simulation, animation and inspection, the latter aided by automated slicing and abstraction techniques. Synthesis from formal 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 [16]. 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.

Synthesis is also used to automatically construct plans that are then straightforwardly enacted by some software component. For instance, synthesis of glue code and component adaptors has been studied in order to achieve safe composition at the architecture level [14], and in particular in service oriented architectures [2]. Such approaches cannot be applied when a fully specified domain model is not available, hence their application is limited in earlier phases of development. Our approach allows the construction of glue code and adaptors earlier without necessarily requiring the effort of developing a full domain model.

In the domain of self-adaptive systems there has also been an increasing interest in synthesis as such systems must be capable of designing at run-time adaptation strategies. Hence, they rely heavily on automated synthesis of behaviour models that will guarantee the satisfaction of requirements under the constraints enforced by the environment and the capabilities offered by the self-adaptive system [26, 5]. 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.

Partial behaviour models have been extensively studied. A number of such modelling formalisms exist, e.g., Modal Transition Systems (MTSs) [19] and variants such as Disjunctive MTS [20]. 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.

The formal treatment of MTSs started with model checking, which received a lot of attention (cf. [3, 4, 12]). Initially, a version of three-valued model checking was defined [3] and shown to have the same complexity as that of model checking. Generalised model checking [4] improves the accuracy of model checking of partial specifications. Indeed, three-valued model checking may yield that the answer is unknown even when no implementations of an MTS satisfy the formula. However, complexity of generalised model checking is much higher [12].

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 [13] abstraction refinement is generalised to the context of control in order to reason about larger games. Their main interest is in applying abstraction on existing games. Thus, they are able to make assumptions about which states are reasoned about together. We, on the other hand, are interested in the case that an MTS is used as an abstract model. In this case, the abstract MTS is given and we would like to reason about it.

Of the huge body of work on controller synthesis and realizability of temporal logic we highlight two topics. First, we heavily rely on LTS control. For example, we use the 2EXPTIME-completeness of LTL controller synthesis [24]. Second, we would like to mention explorations of restricted subsets of LTL in the context of synthesis (cf. [1, 22]). These results show that in some cases synthesis can be applied in practice. Similar restrictions, if applied to MTS control combined with our reductions, would 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 [10]. We have implemented a solver to GR(1) [22] formulas in the context of the LTS control problem [8].

More specifically, from a descriptive specification of the domain model in the form of an LTS and a set of controllable actions, the solver constructs an LTS controller that when composed with the domain model satisfies a given FLTL [11] formula of the form $\Box I \wedge (\bigwedge_{i=1}^{n} \Box \Diamond A_i \rightarrow \bigwedge_{j=1}^{m} \Box \Diamond G_j)$ where $\Box I$ is a safety system goal, $\Box \Diamond A_i$ represents a liveness assumption on the behaviour of the environment, $\Box \Diamond G_j$ models a liveness goal for the system and A_i and G_j are non-temporal fluent expressions, while I is a system safety goal expressed as a Fluent Linear Temporal Logic formula. We have implemented the reductions proposed in this paper and extended MTSA to support MTS control. The tool implements the conversion of the MTS E to the LTSs E^{I^+} and E^{I^-} (cf. Section 4) and calls our implementation for the LTS control solution on both problems. The structure of specification does not change when introducing the additional actions. Thus, starting from GR(1) formulas, we can call our GR(1) LTS solver.

6 Conclusions and Future Work

We present a technique that solves the MTS control problem showing that, despite dealing with a potentially infinite number of LTS implementations, the MTS problem is actually in the same complexity class that the underlying LTS synthesis problem.

Specifically, we have defined the MTS control problem that answers if all, none or some implementations of a given MTS, modelling a partially defined domain model, admit an LTS controller that guarantees a given goal. Although an MTS has a potentially infinite number of implementations, we provide an effective algorithm to compute the answer for the MTS control problem without requiring going through all the implementations described by the MTS.

The algorithm reduces the MTS control problem to two LTS control problems in which the controller or the environment get to choose which implementation described by the MTS must be LTS controlled. In principle, both LTS control problems are exponentially larger than the original MTS model. Nevertheless, we show that the number of states of each LTS problem that must be considered is in fact linear in the alphabet of the input MTS. Hence, the MTS control problem remains in the same complexity as the LTS control problems. In fact, as mentioned before we have implemented a solver for GR(1) [22] style formulas applied to LTS control [8]. Hence, our tool checks realisability of an expressive class of MTS control problems in polynomial time.

As mentioned in previous sections, having nondeterminism in the domain model leads to synthesis with imperfect information. Such a setting is much more computationally complex than synthesis with full information. Only in recent years a few approaches towards imperfect information have started to emerge [25]. However, most of them are far from actual applications. In a setting of a nondeterministic domain model but giving the controller full information of actions and states, our technique works with no changes. Similarly, our technique can handle the setting of nondeterministic MTSs and considering only deterministic implementations. Solving the synthesis problem for nondeterministic MTS, which corresponds to imperfect information games, is not straightforward. Nevertheless, we believe that it would reduce to synthesis for nondeterministic LTS in much the same way as with the deterministic variant.

The semantics of MTS is given in terms of a set of LTS implementations. In the context of controller synthesis an MTS is interpreted as the characterisation of a set of possible problem domain models. Hence, the MTS control problem could be used as a mechanism to explicitly identify which are the behaviour assumptions in problem domain that guarantee realisability of a certain goal: We are interested in studying the problem of characterising the set of realisable implementations whenever the answer for a given MTS control problem is *some*.

Having solved the realisability of the MTS control problem, the next logical step is to an algorithm that will produce controllers if the MTS control problem is realisable. We expect that synthesising controllers for the two LTS control problems derived from the MTS control problem should serve as templates to construct controllers for specific LTSs that are implementations of the MTS.

References

1. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: Proc. of the IFAC Symposium on System Structure and Control (1998)
2. Bertolino, A., Inverardi, P., Pelliccione, P., Tivoli, M.: Automatic synthesis of behavior protocols for composable web-services. In: ESEC/FSE, ACM (2009)
3. Bruns, G., Godefroid, P.: Model checking partial state spaces with 3-valued temporal logics. In: CAV'99. Springer (1999)
4. Bruns, G., Godefroid, P.: Generalized model checking: Reasoning about partial state spaces. In: CONCUR'00. Springer (2000)
5. Dalpiaz, F., Giorgini, P., Mylopoulos, J.: An architecture for requirements-driven self-reconfiguration. In: CAiSE, Springer-Verlag (2009)
6. Damas, C., Lambeau, B., van Lamsweerde, A.: Scenarios, goals, and state machines: a win-win partnership for model synthesis. In: SIGSOFT FSE, ACM (2006)
7. D'Ippolito, N.: Technical Report, http://www.doc.ic.ac.uk/∼srdipi/techfm2012
8. D'Ippolito, N., Braberman, V., Piterman, N., Uchitel, S.: Synthesising non-anomalous event-based controllers for liveness goals. TOSEM, (2013)
9. D'Ippolito, N., Braberman, V.A., Piterman, N., Uchitel, S.: Synthesis of live behaviour models for fallible domains. In: Taylor, R.N., Gall, H., Medvidovic, N. (eds.) ICSE. pp. 211–220. ACM (2011)
10. D'Ippolito, N., Fischbein, D., Chechik, M., Uchitel, S.: Mtsa: The modal transition system analyser. In: ASE, IEEE Computer Society (2008)
11. Giannakopoulou, D., Magee, J.: Fluent model checking for event-based systems. In: Proc. ESEC/FSE-11, ACM (2003)
12. Godefroid, P., Piterman, N.: Ltl generalized model checking revisited. In: Proc. VMCAI '09, Springer-Verlag (2009)
13. Henzinger, T.A., Jhala, R., Majumdar, R.: Counterexample-guided control. In: ICALP'03, Springer (2003)
14. Inverardi, P., Tivoli, M.: A reuse-based approach to the correct and automatic composition of web-services. In: ESEC/FSE joint meeting. ESSPE '07, ACM (2007)
15. Jackson, M.: The world and the machine. In: Proceedings of the 17th international conference on Software engineering. pp. 283–292. ICSE '95, ACM (1995)
16. Kazhamiakin, R., Pistore, M., Roveri, M.: Formal verification of requirements using spin: A case study on web services. In: SEFM, IEEE Computer Society (2004)
17. Keller, R.M.: Formal verification of parallel programs. Comm. ACM 19. (1976)
18. van Lamsweerde, A.: Requirements Engineering - From System Goals to UML Models to Software Specifications. Wiley (2009)
19. Larsen, K., Thomsen, B.: "A Modal Process Logic". In: Proc. LICS'88. IEEE (1988)
20. Larsen, K.G., Xinxin, L.: Equation solving using modal transition systems. In: LICS. IEEE Computer Society (1990)
21. Letier, E., van Lamsweerde, A.: Agent-based tactics for goal-oriented requirements elaboration. In: Proc. ICSE '02, ACM (2002)
22. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive (1) designs. LNCS, (2006).
23. Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science, 1977., 18th Annual Symposium on. pp. 46–57. IEEE (1977)
24. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. POPL, ACM (1989)
25. Raskin, J.F., Chatterjee, K., Doyen, L., Henzinger, T.A.: Algorithms for omega-regular games with imperfect information. Logical Methods in CS (2007)
26. Sykes, D., Heaven, W., Magee, J., Kramer, J.: Plan-directed architectural change for autonomous systems. In: SAVCBS. pp. 15–21 (2007)
27. Uchitel, S., Brunet, G., Chechik, M.: Synthesis of partial behavior models from properties and scenarios. IEEE Trans. on Software Engineering 35, (May 2009)