Temporal Logic Control under Incomplete or Conflicting Information

Georgios Fainekos, and Herbert G. Tanner

***Abstract*— Temporal logic control methods have provided a viable path towards solving the single- and multi-robot path planning, control and coordination problems from high level formal requirements. In the existing frameworks, the prevalent assumption is that there is a single stakeholder with full or partial knowledge of the environment or the system. However, this assumption may not be valid in both off-line and on-line temporal logic control problems. That is, multiple stakeholders and inaccurate sources of information may produce a self-contradictory model of the world or the system. Classical control synthesis methods cannot handle non-consistent model environments even though such inconsistencies may not affect the control synthesis problem for a given formal requirement. In this work, we show how such problems can be circumvented by utilizing multi-valued temporal logics and system models.**

I. Introduction

Temporal logic control methods have provided a viable path towards solving control, path planning, and coordination problems from high level formal requirements. The early theoretical results have now resulted in a multitude of software toolboxes that can be used in a variety of applications [1]–[3]. A particular class of control synthesis methods from Linear Temporal Logic (LTL) requirements is conceptually founded upon the theory of counterexample generation in LTL model checking [4]. This approach has led to a variety of practical methods solving control synthesis problems for linear systems [5], [6], single robot motion planning problems [7], [8] and multi-robot motion planning and coordination problems [9], [10].

A common thread in the existing frameworks is that there is a single stakeholder with full knowledge of the environment that the robots (or, more generally, the control systems) operate in. However, this assumption may not be valid in both off-line and on-line temporal logic control and synthesis problems. That is, multiple stakeholders and inaccurate sources of information may produce a self-contradictory model of the world or the system.

Classical LTL control synthesis methods cannot handle non-consistent model environments even though such inconsistencies may not affect the control synthesis problem for a given formal requirement. For example, the knowledge that a passage may not be open may not affect the discovery of a feasible path in the environment. Thus, the path planning (or, more generally, the control synthesis) method should have the mechanisms to handle such inconsistencies. In addition, the user of the system may want to take the risk and allow the robots to attempt to proceed through problematic areas.

In this paper, we propose to use multi-valued transition systems and logic to capture incomplete and, even, conflicting information about the environment and/or the requirements of the system. We show that if a desired level of uncertainty is tolerated by the users of the system, then the problem can be reduced to a classical Boolean valued LTL control synthesis problem. We also show that if the goal is to determine the sequence of control actions that maximize the satisfaction of the requirement, then the problem is also solvable in polynomial time with only a small increase in complexity over the Boolean LTL control synthesis problem.

A direct consequence of our results is that the methods in [5]–[10] and related works can immediately take advantage of our results with only minor modifications and virtually no increase in computational complexity. Furthermore, the impact of our work extends well beyond the motion planning scenarios that we use as a motivating example in this paper. In particular, one can envision requirements engineers for control systems merging requirements from different documents and sources and automatically synthesizing control software that is going to satisfy the control requirements despite the fact that there may be conflicts and inconsistencies in the documentation.

II. Preliminaries

In our framework, both the specifications and the transition systems are multi-valued. The main reason is that multi-valued transition systems can capture uncertain, incomplete and contradictory knowledge about the system. At the same time, multi-valued specification formulas enable reasoning about the desired levels of incomplete or contradictory knowledge. In this section, we proceed with a brief review of multi-valued transition systems and logics.

In the following, we will use the abbreviation "mv" in the place of "multi-valued". Furthermore, we will assume a working knowledge of control synthesis methods using model checking, e.g., [5], [7].

A. Lattices

Informally, lattices can capture an order relation between the possible truth values in the system. A detailed exposition on lattices and order can be found in [11].

A partial ordered set or poset is a set S with an ordering relation $\preceq \subseteq S \times S$. For a poset $(S, \preceq)$, we define the *join* ($\sqcup : S \times S \to S$) and *meet* ($\sqcap : S \times S \to S$) operations as $x \sqcup y := \sup(\{x, y\})$ and $x \sqcap y := \inf(\{x, y\})$, respectively. Also, we adopt the following notation. For a poset $(S, \preceq)$,

This work has been partially supported by awards NSF CPS 1446730 and ARL MAST CTA W911NF-08-2-0004.

G. Fainekos is with SCIDSE at Arizona State University, Tempe, AZ, E-mail: fainekos@asu.edu.

H. Tanner is with the Department of Mechanical Engineering, University of Delaware. Email: btannerg@udel.edu.

we define $\bigsqcup S = \sup(S) \triangleq \top$ and $\bigsqcap S = \inf(S) \triangleq \bot$ which are read as join and meet of S, respectively.

Definition 2.1 (Lattices): A poset $\mathfrak{L} = (L, \preceq)$ is called a:

- *lattice*, if for all $x, y \in L$, both $x \sqcup y$ and $x \sqcap y$ exist;
- *quasi-boolean lattice*, if it is distributive, every element $l \in L$ has a unique complement $\sim l$ such that $\sim\sim l = l$, and $l_a \preceq l_b$ implies $\sim l_b \preceq \sim l_a$.

The above definition of lattices $(L, \preceq)$ is based on partial orders, but equivalently we can define lattices as algebraic structures $(L, \sqcup, \sqcap)$ that satisfy the *commutative*, *associative*, *absorption* and *idempotency* laws. Note here that the *distributive* and *neutral element laws* need not hold on a lattice.

It is sometimes necessary to combine lattices. The lattice product provides such a notion that can be computed.

Definition 2.2 (Lattice Product): Given two lattices $\mathfrak{L}_a = (L_a, \preceq_a)$ and $\mathfrak{L}_b = (L_b, \preceq_b)$, we can define the product lattice $\mathfrak{L} = \mathfrak{L}_a \times \mathfrak{L}_b$ as $(L_a \times L_b, \preceq)$ such that $(l_a, l_b) \preceq (l'_a, l'_b)$ iff $l_a \preceq_a l'_a$ and $l_b \preceq_b l'_b$.

Example 1: The classical Boolean semantics for propositional logic are defined over the lattice $\mathfrak{L}_2 = (\{0, 1\}, \leq)$.

Figure 1 presents the diagrams for three other lattices. In the figure, the nodes that are higher in the diagram have higher value under the ordering relation. Likewise, two nodes at the same height are not comparable. The lattice $\mathfrak{L}_3$ is an extension of the lattice $\mathfrak{L}_2$ with the value $1/2$ which is interpreted as *maybe*.

The lattice $\mathfrak{L}_3 \times \mathfrak{L}_3$ is the product of the lattice $\mathfrak{L}_3$ with itself. It can be thought of as representing two experts with different opinions on the satisfaction of a statement. For example, the value $(1, 0)$ represents the fact that one expert is certain that a statement is true while the other is certain that the statement is false. Moreover, $(1, 0)$ cannot be directly compared with $(0, 1)$ since these represent disputed values by the experts. However, if we were to just require that at least one of the experts interpreted the statement as true, then the supremum of the two values would give us $(1, 1)$ which represents the top element in the lattice.

The lattice $\mathfrak{L}_{4+2}$ imposes a different ordering relation and interpretation on values of the lattice $\mathfrak{L}_{3,3}$ (see also [12]).

B. Multi-Valued Transition Systems

A multi-valued Transition System (mv-TS) [13] is an extension of the classical notion of a Transition System (TS). In a TS, the atomic propositions are either true or false in every state and the transitions between states are defined using a relation. In contrast, in an mv-TS, the atoms take values over a truth lattice which describes the truth degree of an atom in a specific system state. Similarly, we can model how "true" or certain a transition between system states is. However, since LTL is interpreted over sequences of atomic propositions, it is not meaningful to allow multi-valued transitions as it is the case for example in CTL or μ-calculus [13], [14]. Therefore, we will only allow the atomic propositions that label the states to map over truth values over the lattice.

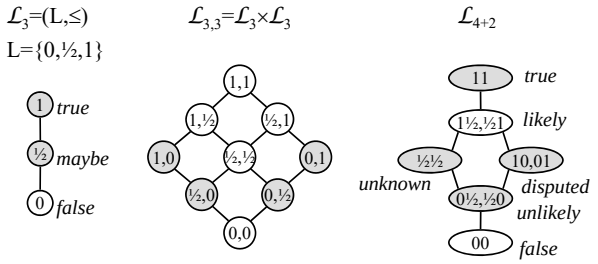

Fig. 1: Diagrams of the lattice examples in Example 1.

Definition 2.3 (Multi-Valued Transition System): (mv-TS) is a tuple $\mathcal{M} = (S, S_0, \mathfrak{L}, \rightarrow, AP, \mathcal{O})$ where:

- S is a finite set of states,
- $S_0 \subseteq S$ is the set of initial states,
- $\mathfrak{L}$ is a lattice $(L, \sqcap, \sqcup)$ or an algebra $(L, \sqcap, \sqcup, \sim, \bot, \top)$,
- $\rightarrow \subseteq S \times S$ is a transition function,
- AP is a finite set of atomic propositions, and
- $\mathcal{O} : S \times AP \rightarrow L$ is a total labeling function that maps a pair (s, a) to some truth value $l \in L$.

For the multi-valued case, we need to slightly modify the definition of traces that correspond to a path on the mv-TS. The concept of a *path* on the mv-TS is the same as in the 2-valued case. Namely, a path p is a sequence of states of an mv-TS, i.e., $p \in S^\omega$, such that for $i \geq 0$, $p_i \rightarrow p_{i+1}$. Here, S^ω denotes infinite concatenation of symbols from the set S. Alternatively, a path can be viewed as a function $p : \mathbb{N} \rightarrow S$, where $\mathbb{N}$ is the set of natural numbers (including zero). More importantly, a trace generated by the mv-TS is a sequence of functions that map atomic propositions from the set AP to their truth degree from the lattice $\mathfrak{L}$. Namely, given a path p, the corresponding trace μ is now a function $\mu : \mathbb{N} \times AP \rightarrow L$ defined as $\mu(i, \pi) = \mathcal{O}(p(i), \pi)$. It is easy to confirm that when the lattice is $\mathfrak{L}_2$, then the above definition of a trace is equivalent to the definition of a trace as a sequence of sets of atomic propositions.

C. Linear Temporal Logic

In 2-valued Linear Temporal Logic (LTL), the formulas are interpreted over sequences of atoms. An LTL formula effectively describes which sequences of atomic propositions are permitted. Under the classic Boolean semantics, we can state that a system trace μ satisfies or does not satisfy the specification. In contrast, under multi-valued semantics, an LTL formula will evaluate to a degree of satisfaction from the lattice of truth values.

The syntax of the LTL formulas in the multi-valued case is the same as in the 2-valued case.

Definition 2.4 (LTL): Let AP be the set of atomic propositions, then a well formed LTL formula ϕ is generated according to the grammar $\phi ::= \pi \mid \neg\phi \mid \phi \vee \phi \mid \bigcirc\phi \mid \phi \mathcal{U} \phi$, where $\pi \in AP$, $\bigcirc$ is the next time operator and the $\mathcal{U}$ is the until operator.

In this paper, we will assume that the semantics are interpreted over a quasi-boolean lattice $\mathfrak{L}$. This will effectively enable us to use in the familiar form dual negation and the derived temporal operators. It also makes possible the translation of any LTL formula in an equivalent formula

in Negation Normal Form (NNF), i.e., where the negation symbol appears only in front of the atomic propositions.

Definition 2.5: For an LTL formula ϕ and a trace $\mu : \mathbb{N} \times AP \to L$ taking values over a quasi-boolean lattice $\mathfrak{L} = (L, \preceq, \sim)$, the mv-semantics of ϕ with respect to μ are

$$\begin{aligned}
[\![\pi]\!](\mu, i) &:= \mu(i, \pi) \text{ for } \pi \in AP \\
[\![\neg\psi]\!](\mu, i) &:= \sim [\![\psi]\!](\mu, i) \\
[\![\phi_1 \vee \phi_2]\!](\mu, i) &:= [\![\phi_1]\!](\mu) \sqcup [\![\phi_2]\!](\mu) \\
[\![\bigcirc\psi]\!](\mu, i) &:= [\![\psi]\!](\mu, i+1) \\
[\![\phi_1 \,\mathcal{U}\, \phi_2]\!](\mu, i) &:= \bigsqcup_{j \geq i} \big([\![\phi_2]\!](\mu, j) \sqcap \prod_{i \leq k < j} [\![\phi_1]\!](\mu, k) \big)
\end{aligned}$$

We denote $[\![\phi]\!](\mu, 0)$ by $[\![\phi]\!](\mu)$. Since we are using quasi-boolean algebras, we can define the derived Boolean and temporal operators with the usual equivalences, e.g., $\pi_1 \wedge \pi_2 \equiv \neg\pi_1 \vee \neg\pi_2$ (conjunction), $\Diamond\phi \equiv \top \mathcal{U} \phi$ (eventually), $\phi_1 \mathcal{R} \phi_2 \equiv \neg(\neg\phi_1 \,\mathcal{U}\, \neg\phi_2)$ (release), and $\Box\phi \equiv \neg\Diamond\neg\phi$.

III. Problem Description

Even though the concepts presented in this paper are general enough to apply to a large class of Discrete Event System (DES) or hybrid system control synthesis problems [15], we will focus the discussion to a very specific application. We will consider the path planning problem for a single robot on a graph where information about its environment is provided by different stakeholders, e.g., different firefighters observing a fire site. We remark that translations of the path planning problem on a graph to the actual trajectory planning problem for single- and multi-robot systems are independent to the results presented in this paper and they can be solved using a vast array of methods, e.g., [7], [8].

In this paper, we provide a solution to following problems under the assumption that the lattice $\mathfrak{L}$ is quasi-boolean.

Problem 1: Given m mv-TS $\mathcal{M}_i = (S_i, S_{i0}, \mathfrak{L}_i, \to_i, AP_i, \mathcal{O}_i)$ modeling the same robot environment, an LTL specification φ and a desired truth degree l from some quasi-Boolean lattice $\mathfrak{L} = (L, \preceq, \sim)$, merge the m mv-TS $\mathcal{M}_i$ into a single mv-TS $\mathcal{M}$ over the quasi-Boolean lattice $\mathfrak{L}$ and compute a path p on $\mathcal{M}$ such that the corresponding trace μ satisfies $[\![\varphi]\!](\mu) \succeq l$.

Problem 2: Given m mv-TS $\mathcal{M}_i = (S_i, S_{i0}, \mathfrak{L}_i, \to_i, AP_i, \mathcal{O}_i)$ modeling the same robot environment and an LTL specification φ, merge the m mv-TS $\mathcal{M}_i$ into a single mv-TS $\mathcal{M}$ over a quasi-Boolean lattice $\mathfrak{L} = (L, \preceq, \sim)$ and compute a path p on $\mathcal{M}$ such that the corresponding trace μ does not satisfy $[\![\varphi]\!](\mu) \prec [\![\varphi]\!](\mu')$ for any other trace μ' of $\mathcal{M}$.

Note that in either problem definition, we do not formally specify the merge operation for the m mv-TS $\mathcal{M}_i$ or the resulting lattice $\mathfrak{L}$. This is an application dependent notion which we briefly discuss in Section V.

IV. Multi-Valued LTL Planning

First, we show how multi-valued LTL planning can be reduced to a finite number of classical 2-valued LTL planning problems. Next, we present planning algorithms that directly compute the "best" possible plan for the given requirement.

A. Reduction to 2-valued LTL Planning

As a first step towards the reduction, we need to abstract the mv-TS into a classical 2-valued TS (from this point on simply referred to as TS). The abstract TS actually depends both on the LTL specification φ and on the truth degree $l \in \mathfrak{L}$. In particular, in the following, we will assume that the specification φ is in Negation Normal Form (NNF) and that any negative atomic proposition $\neg\pi$ is replaced by a new atomic proposition $\tilde{\pi}$ that does not appear in the set of atomic propositions, i.e., $AP' = AP \cup \{\tilde{\pi} \mid \pi \in AP\}$. Then, the abstract TS needs to account for the negations of atomic propositions that appear in the specification φ. That is, the definition of the observation map $\mathcal{O}$ must be extended to assign values to the new atoms, i.e., $\mathcal{O}(s, \tilde{\pi}) = \sim \mathcal{O}(s, \pi)$.

The reason for not allowing arbitrary negations in the formula is technical for establishing Proposition 4.1. If arbitrary negations are allowed, then Proposition 4.1 can lead to contradictions because through the abstraction the exact truth values of the atomic propositions have been lost. However, we remark that any LTL formula can be syntactically converted into NNF by pushing the negation operators in front of the atomic propositions using the syntactic equivalences of the Boolean and temporal operators. There is no loss on the expressive power of the logic since the lattices are quasi-Boolean algebras and all the lattice values have a unique negation.

Formally, given an mv-TS $\mathcal{M} = (S, S_0, \mathfrak{L}, \to, AP, \mathcal{O})$ and a lattice value $l \in \mathfrak{L}$, we construct a TS $\mathcal{M}^l = (S, S_0, \mathfrak{L}_2, \to, AP^l, \mathcal{O}^l)$ where $\mathcal{O}^l(s, \pi) = (\mathcal{O}(s, \pi) \succeq l)$. Now, any path p on $\mathcal{M}^l$ which satisfies φ with Boolean semantics will satisfy φ with at least degree l.

Proposition 4.1: Given an mv-TS $\mathcal{M} = (S, S_0, \mathfrak{L}, \to, AP, \mathcal{O})$, a path p on $\mathcal{M}$, an LTL specification φ and a lattice value $l \in \mathfrak{L}$, then $\tilde{\mu} \models \varphi$ iff $[\![\varphi]\!](\mu) \succeq l$, where $\tilde{\mu}(i, \pi) = \mathcal{O}^l(p(i), \pi)$ and $\mu(i, \pi) = \mathcal{O}(p(i), \pi)$.

Using Proposition 4.1, we can solve the control synthesis/planning problem for at least value l by using automata based LTL planners on the abstract TS $\mathcal{M}^l$. This implies that the complexity of solving the multi-valued LTL planning problem is the same as the complexity of the LTL planning problem [16]. In brief, the search graph will be of size $|\mathcal{M}| 2^{O(\varphi)}$, in the worst case. Let V and E be the vertices and edges of the search graph and F be the set of sinks (accepting states). The plan generation process (graph search) is usually performed using nested Breadth First Search (BFS) [5], [7] which is of complexity $O(|F|(|V| + |E|))$ since a search needs to be executed to reach an accepting state and one more search is initiated from the accepting state back to itself. We remark that the computational cost of the abstraction is just $O(|S||AP|)$ since only $\mathcal{O}^l$ needs to be computed.

The computed plan will not be guaranteed to be truth-degree optimal with respect to the degree of satisfaction of the specification. Namely, when the algorithm returns a path p of degree $l' \geq l$, then it is not guaranteed that there is no other path of degree $l'' > l'$ with respect to the specification. In general, it would be desirable to find the optimal path.

Based on the reduction method the only way to find the optimal path is to execute the abstraction based algorithm for all values $l' > l$. Thus, in the worst case, $|\{l' \in L \,|\, l \prec l'\}|$ abstractions, product constructions and graph searches would be performed.

B. Direct Multi-Valued Planning

In this section, we will show how to directly solve the mv-planning using mv-automata with Büchi acceptance conditions. We will first introduce some basic definitions on mv-automata which are targeted to the mv-planning problem (as opposed to a general treatment of mv-automata as in [17]).

Definition 4.1 (Multi-Valued Automaton): is a tuple $\mathcal{A} = (S, s_0, \mathfrak{L}, \Sigma, \Delta, F)$ where:

- S is a finite set of states,
- $s_0 \in S$ is the initial state,
- $\mathfrak{L}$ is a lattice $(L, \sqcap, \sqcup)$ or an algebra $(L, \sqcap, \sqcup, \sim, \bot, \top)$,
- Σ is a finite alphabet,
- $\Delta : S \times S \to 2^{\Sigma}$ labels the transitions of the automaton with sets of symbols from Σ, and
- F is the set of accepting states.

We remark that our definition of the transition function is slightly different from [17], where LTL mv-model checking is introduced, but it is similar to the Generalized Nondeterministic Finite Automata (GNFA). The reason for the modification is that now the mv-automata do not process input words over the alphabet Σ, but rather words w over functions that map symbols from Σ to values of the lattice $\mathfrak{L}$, i.e., $w \in (L^{\Sigma})^{\omega}$ or, alternatively, w is a function $w : \mathbb{N} \times \Sigma \to L$. The change leads to a straightforward definition for an accepting run on the mv-automaton.

Definition 4.2 (Run): A *run* $p \in S^{\omega}$ of an mv-automaton under word $w \in (L^{\Sigma})^{\omega}$ is infinite sequence of states such that $\forall i \geq 0$, $w_i(\Delta(p_i, p_{i+1})) \triangleq \sqcap_{\sigma \in \Delta(p_i, p_{i+1})} w(i, \sigma) \succ \bot$.

Similarly to 2-valued Büchi autotomata, an *accepting run* is one where accepting states are encountered infinitely often. We let $AR(\mathcal{A})$ denote the set of all accepting runs of $\mathcal{A}$.

The language of an mv-automaton is now a multi-valued set. Namely, for each word $w \in (L^{\Sigma})^{\omega}$, we have a degree l for which w belongs to the language of the automaton. Formally, we define the degree that a word w belongs to language $L(\mathcal{A})$ of mv-automaton $\mathcal{A}$ as

$$[\![L(\mathcal{A})]\!](w) = \bigsqcup_{p \in AR(\mathcal{A})} \prod_{i \geq 0} w_i(\Delta(p_i, p_{i+1})) \qquad (1)$$

Note that if there are no accepting runs, i.e., $AR(\mathcal{A}) = \emptyset$, then $[\![L(\mathcal{A})]\!](w) = \bot$ for any w.

As highlighted in [17], it is easy to translate an LTL formula to a Büchi automaton that will assign to a word the correct truth degree from the lattice. The only modifications needed over the translation algorithms for 2-valued automata are in the treatment of the cases $\pi \wedge \neg\pi$ and $\pi \vee \neg\pi$, which in mv-logics do not necessarily evaluate to $\bot$ and $\top$, respectively.

Theorem 4.1 (Theorem 3 in [17]): An automaton $\mathcal{A}_{\phi}$ constructed for an LTL formula ϕ assigns the value l to a word $w \in (L^{\Sigma})^{\omega}$, i.e., $[\![L(\mathcal{A})]\!](w) = l$, iff $[\![\phi]\!](w) = l$.

Moreover, in [17], it was shown that the languages accepted by mv-automata are closed under intersection. Therefore, given an LTL formula φ, we can construct an mv-automaton $\mathcal{A}_{\varphi}$, which we can now use for planning. In particular, we are looking for traces in the multi-valued intersection of the mv-automaton, which correspond to the mv-TS $\mathcal{M}$, and the specification automaton $\mathcal{A}_{\varphi}$.

In the following, we circumvent the intermediate step of converting the mv-TS $\mathcal{M}$ into an mv-automaton $\mathcal{A}_{\mathcal{M}}$ and, then, computing the mv-intersection between $\mathcal{A}_{\mathcal{M}}$ and $\mathcal{A}_{\varphi}$. Instead, we are only interested in the graph modeling the concurrent execution of $\mathcal{A}_{\mathcal{M}}$ and $\mathcal{A}_{\varphi}$ with the transitions annotated by the degree with which they are enabled.

Definition 4.3 (Planning Graph): Given an mv-TS $\mathcal{M} = (S_{\mathcal{M}}, S_{\mathcal{M}0}, \mathfrak{L}, \to, AP, \mathcal{O})$ and a specification mv-automaton $\mathcal{A}_{\varphi} = (S_{\mathcal{A}}, s_{\mathcal{A}0}, \mathfrak{L}, AP, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$, the Planning Graph $G_{\mathcal{M}\varphi}$ is the tuple (V, E, β, v_0, F), where

- $V = S_{\mathcal{M}} \times S_{\mathcal{A}}$ is the set of vertices,
- $E \subseteq V \times V$ is the set of edges and $\beta : E \to L$ is a set of weights such that for given $v = (s_{\mathcal{M}}, s_{\mathcal{A}})$ and $v' = (s'_{\mathcal{M}}, s'_{\mathcal{A}})$, it is

$$(v, v') \in E \text{ and } \beta(v, v') = \prod_{\pi \in \Delta_{\mathcal{A}}(s_{\mathcal{A}}, s'_{\mathcal{A}})} \mathcal{O}(s'_{\mathcal{M}}, \pi)$$

$$\text{iff } s_{\mathcal{M}} \to s'_{\mathcal{M}} \text{ and } \prod_{\pi \in \Delta_{\mathcal{A}}(s_{\mathcal{A}}, s'_{\mathcal{A}})} \mathcal{O}(s'_{\mathcal{M}}, \pi) \succ \bot,$$

- $v_0 = (s_{\mathcal{M}0}, s_{\mathcal{A}0})$ is the initial vertex, and
- $F = S_{\mathcal{M}} \times F_{\mathcal{A}}$ is the set of final vertices.

It is straightforward that the construction in Def. 4.3 only allows for paths on $G_{\mathcal{M}\varphi}$ that are possible on $\mathcal{M}$ and whose corresponding trace would allow a transition on $\mathcal{A}_{\varphi}$.

Note that Def. 4.3 reduces to the classical 2-valued construction when the lattice is $\mathfrak{L}_2 = (\{0, 1\}, \leq)$. For example, if a transition on the Büchi automaton is labeled by $\{\pi_1, \pi_2\}$, then it is required that both π_1 and π_2 are true on the next state on the finite state transition system. If it were the case that $\mathcal{O}(s'_{\mathcal{M}}, \pi_1) = \mathcal{O}(s'_{\mathcal{M}}, \pi_2) = 1$, then the transition would be possible on the product automaton $\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\varphi}$ since $\mathcal{O}(s'_{\mathcal{M}}, \pi_1) \sqcap \mathcal{O}(s'_{\mathcal{M}}, \pi_2) = 1 > 0$.

The mv-planning on the graph $G_{\mathcal{M}\varphi}$ exactly mirrors the corresponding construction for 2-valued automata. First, we need to compute a path from the initial vertex to a final vertex and, then, from the final vertex back to itself. Such a path will guarantee that an accepting state on automaton $\mathcal{A}_{\varphi}$ is visited infinitely often and, thus, it is an accepting run with degree greater than $\bot$. However, if the goal is to find an input word that maximizes the membership degree function in Eq. (1), then the resulting problem reduces to solving a finite number instances of that Maximum Capacity Path Problem [18] (also known as the Widest Path Problem (WPP)). In addition, it may be desirable to identify the shortest path among the paths of maximal capacity or the maximal capacity path among the shortest paths on the graph [18].

In principle, WPP can be solved by a straightforward modification of the Dijkstra's algorithm [19]. Therefore, we would need $O(|V|^2)$ time to solve the WPP from the

start vertex to any vertex on the graph. In the worst case, $|F|+1$ such problems need to be solved; one from the initial vertex and one from each final vertex. Thus, as opposed to the solution presented in Sec. IV-A, this approach avoids the multiple derivations of the product automata and the corresponding graph search for each $l' \succeq l$.

V. Merging Different Models

The problem of merging the models produced by different stakeholders is really application dependent. A general approach would necessitate an authority deciding which aspects of the models should be merged and how.

Considering the application at hand, i.e., robot path planning, we will impose some structure to the problem in order to automate the model merging process. In particular, we will assume that there is a base model $\overline{\mathcal{M}} = (S, S_0, \mathfrak{L}, \rightarrow, AP, \overline{\mathcal{O}})$ and that the users are allowed to change the values of the atomic propositions through the map $\overline{\mathcal{O}}$.

Furthermore, since the current definition of $\overline{\mathcal{M}}$ does not allow for multi-valued transitions, we can assume that each state $s \in S$ is also labeled by a special atomic proposition π_m indicating whether the robot can move into that state or not. For instance, using the 3 valued lattice $\mathfrak{L}_3$, the user could indicate that the transition to state s is not certain by letting $\mathcal{O}(s, \pi_m) = 1/2$.

If desired, it is also possible to allow users to add new states, transitions or atomic propositions to $\overline{\mathcal{M}}$. However, in this case an application dependent decision must be taken. For example, if a stakeholder adds a new state, then what should be the values of the atomic propositions assigned by the other users to that new state? It could be assumed that all the other users agree with the user adding the new state or it could be the case that logic supporting an "unconfirmed" truth value is used. Alternative, other information could be used to check the validity of adding a new state, e.g., geometric information in the case of motion planning. We leave these issues for future work.

Given m functions $\mathcal{O}_i$, we need to merge them into a single lattice value mapping function $\mathcal{O}$. One conservative choice would be to take the minimum user provided value for each state and each atomic proposition. That is, for all $s \in S$ and $\pi \in AP$, $\mathcal{O}(s, \pi) = \sqcap_{i=1}^{m} \mathcal{O}_i(s, \pi)$. However, here, we will take a different approach. We will define tuples of truth values ranging over the products of the lattices. Namely, for all $s \in S$ and $\pi \in AP$, $\mathcal{O}(s, \pi) = (\mathcal{O}_1(s, \pi), \ldots, \mathcal{O}_m(s, \pi))$. In this way, it is possible to construct plans where all the users are in agreement with the same degree of certainty or where the experts are given higher priority.

VI. Motion Planning Examples

In this section, we will demonstrate the flexibility of the proposed framework using the LTL motion planning algorithm from [7]. We consider a mobile robot which operates in the planar environment of Fig. 2. The robot is moving in a convex polygonal environment with four areas of interest denoted by $\alpha_1, \alpha_2, \alpha_3, \alpha_4$. Initially, the robot is placed at the black dot in region α_0.

Two firefighters provide information regarding the current status. Firefighter A thinks that site α_4 has fires while sites α_0 to α_3 do not have any fires. Firefighter A also thinks that the water reservoirs at sites α_3 and α_1 are full while the other sites do not have any reservoirs. Firefighter B has the same information regarding fires as Firefighter A. Firefighter B also knows that the water reservoir at α_1 is full. However, Firefighter B is not sure whether the water reservoir at site α_3 has enough water.

The firefighting robot must accomplish the task: "Periodically load water and extinguish a fire and never go through site α_2 (so as not to interfere with the firefighters)". The informal requirement is captured by the LTL specification $\psi_1 = \square\diamond water \wedge \square\diamond fire \wedge \square\neg\alpha_2$. All the atomic propositions in the set take values over the lattice $\mathfrak{L}_{3,3}$ (see Fig. 1) since we want to capture the information provided by both firefighters. The LTL planning method works on a discrete abstraction of the environment which has 44 states in total. For the discrete states s that correspond to the area α_3, we have $\mathcal{O}(s, water) = (1, 1/2)$ since Firefighter A is sure that there is water in the reservoir while Firefighter B is not sure.

If we assume that the robot user would like to execute a plan with the best chances of success, then she would request a plan with degree $l_1 = (1, 1)$. Using the abstraction based method with degree $l = (1, 1)$, we get robot trajectory 1 in Fig. 2. Namely, the robot starts from the dot and then circulates between the areas α_1 (full water) and α_4 (certain fire). On the other hand, if the robot user is willing to accept some uncertainty in the success of the plan, then she can request a plan with degree $l_2 = (1/2, 1/2)$. Alternatively, she could believe that Firefighter A is more experienced and, therefore, a truth degree of $l_2' = (1, 1/2)$ would be sufficient for the mission success. If the abstraction based planning method is utilized with degree l_2 or l_2', then trajectory 2 is created in Fig. 2. In this case, the robot visits area α_3 and, then, it periodically visits α_4 and α_3.

We remark that in all the aforementioned examples the planning time is of the order of 0.15sec on an Intel i7 at 2.7GHz with 8GB memory running a Matlab implementation of the LTL planner (not including the LTL to Büchi automata translation). If we run the modified Dijkstra algorithm from Section IV-B on the above problem instance, then the algorithm will compute the same solution as the abstraction method for $l_1 = (1, 1)$ (trajectory 1 in Fig. 2). However, now, the planning time is about 0.9sec on the same platform.

VII. Related Work

Our work was primarily inspired by related work in model checking inconsistent requirements in software engineering [12]. However, the main results derived in our paper and in [12] are different both in terms of theory and in terms of application. For example, [12] does not consider the counterexample generation problem.

Much of the work in planning when the environment dynamics is uncertain can be found in the context of reinforcement learning [20] in which the entire system, capturing the dynamics of interaction between the controlled process and

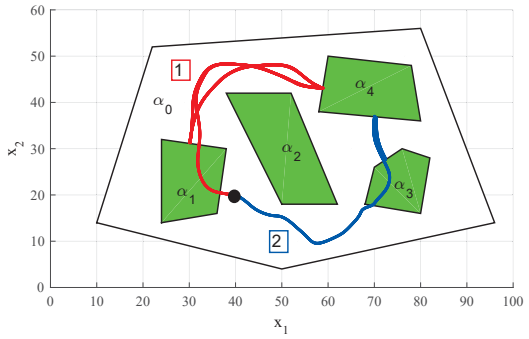

Fig. 2: The environment and two trajectories of the robot which satisfy the specification ψ_1 with different truth values.

its environment, is modeled as a Markov Decision Process (MDP). Formulations that treat the environment interaction probabilistically and aim at satisfying LTL specifications include [21], [22]. In the formulation of our paper however, uncertainty is assumed to be rooted in lack of knowledge – not in chance – and the underlying models for controlled systems and their environment are not MDPs, but rather transition systems with state labels that take values over a lattice. Moreover, even if we ignore the advantage of handling conflicting requirements, our approach is more computationally efficient and practical in case the exact transition probabilities are not known.

Our work is certainly not the first purely deterministic approach to handling unknown environment dynamics – see for instance [23]–[26]. Here, however, the introduction of multi-valued logic not only enables far more granularity in the degree of uncertainty than a Boolean view of "what I haven't seen does not exist," but it also allows the satisfaction of LTL specifications at varying degrees of certainty.

VIII. Conclusions

We have presented an approach for Linear Temporal Logic (LTL) controller synthesis under incomplete or even contradictory information. In order to address the problem, we utilize multi-valued logics and multi-valued finite state transition systems. We show that the computational cost of LTL control synthesis using multi-valued logics is effectively the same as the classical LTL control synthesis methods over Boolean semantics. This result is important because it implies that existing LTL planning and control synthesis methods can utilize the techniques presented in this paper without paying an excessive computational cost. Finally, the results presented in this paper open the possibility to develop new results for on-line planning in multi-robot LTL specified missions as well as for new approaches in system control synthesis when conflicting requirements are provided by different engineers.

References

[1] P. Roy, P. Tabuada, and R. Majumdar, "Pessoa 2.0: a controller synthesis tool for cyber-physical systems," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*, ser. HSCC '11. New York, NY, USA: ACM, 2011, pp. 315–316.

[2] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "Tulip: a software toolbox for receding horizon temporal logic planning," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*. ACM, 2011, pp. 313–314.

[3] C. Finucane, G. Jing, and H. Kress-Gazit, "LTLMoP: Experimenting with language, temporal logic and robot control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010.

[4] G. J. Holzmann, D. Peled, and M. Yannakakis, "On nested depth first search," in *Proceedings of the Second Workshop on the SPIN Verification System*, ser. DIMACS, vol. 32. American Mathematical Society, 1997.

[5] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[6] E. M. Wolff, U. Topcu, and R. M. Murray, "Automaton-guided controller synthesis for nonlinear systems with temporal logic," in *International Conference on Intelligent Robots and Systems*, 2013.

[7] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, Feb. 2009.

[8] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *International Conference on Robotics and Automation*. IEEE, 2010, pp. 2689–2696.

[9] J. Tumova and D. V. Dimarogonas, "Decomposition of multi-agent planning under distributed motion and task LTL specifications," in *54th IEEE Conference on Decision and Control*, 2015.

[10] L. Bobadilla, O. Sanchez, J. Czarnowski, K. Gossman, and S. LaValle, "Controlling wild bodies using linear temporal logic," in *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[11] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*, 2nd ed. Cambridge University Press, 2002.

[12] S. Easterbrook and M. Chechik, "A framework for multi-valued reasoning over inconsistent viewpoints," in *Proceedings of the 23rd International Conference on Software Engineering*, 2001.

[13] G. Bruns and P. Godefroid, "Model checking with multi-valued logics," in *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, ser. LNCS, vol. 3142, 2004.

[14] M. Chechik, A. Gurfinkel, and B. Devereux, "χ-chek: A multi-valued model-checker," in *CAV*, 2002, pp. 505–509.

[15] P. Tabuada, *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.

[16] G. D. Giacomo and M. Y. Vardi, "Automata-theoretic approach to planning for temporally extended goals," in *European Conference on Planning*, ser. LNCS, vol. 1809. Springer, 1999, pp. 226–238.

[17] M. Chechik, B. Devereux, and A. Gurfinkel, "Model-checking infinite state-space systems with fine-grained abstractions using SPIN," in *8th International SPIN Workshop*, ser. LNCS, vol. 2057. Springer, 2001.

[18] E. Martins, M. Pascoal, D. Rasteiro, and J. Dos Santos, "The optimal path problem," *Investigacão Operacional*, vol. 19, pp. 43–60, 1999.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press/McGraw-Hill, Sep. 2001.

[20] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, "Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers," *IEEE Control Systems Magazine*, vol. 32, no. 6, pp. 76–105, 2012.

[21] Y. Chen, J. Tumova, and C. Belta, "Ltl robot motion control based on automata learning of environmental dynamics," in *IEEE International Conference on Robotics and Automation*, 2012.

[22] J. Fu and U. Topcu, "Pareto efficiency in synthesizing shared autonomy policies with temporal logic constraints," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[23] K. Leahy, P. Kannappan, A. Jardine, H. G. Tanner, J. Heinz, and C. Belta, "Integration of deterministic inference with formal synthesis for control under uncertainty," in *American Control Conference*, 2016.

[24] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *Proceedings of the IEEE Conference on Robotics and Automation*, 2013.

[25] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative temporal motion planning for hybrid systems in partially unknown environments," in *International Conference on Hybrid Systems: Computation and Control*, 2013, pp. 353–362.

[26] A. I. M. Ayala, S. B. Andersson, and C. Belta, "Temporal logic motion planning in unknown environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.