Optimal Multi-Valued LTL Planning for Systems with Access Right Levels

Mohammad Hekmatnejad, and Georgios Fainekos

***Abstract*— We propose a method for optimal Linear Temporal Logic (LTL) planning under incomplete or uncertain knowledge with a minimum required truth degree. In order to introduce modeling flexibility to the framework, we propose semantics over ranges of truth values and operators that reason over such ranges of truth values. We show that the resulting optimal planning problem can be solved efficiently. In terms of application, we show how this method can be used for motion planning in autonomous driving cars for road networks with multiple degrees of accessibility.**

I. Introduction

The path planning problem for autonomous robots in well structured environments, e.g., freeways or multi-lane city roads, can be efficiently solved using classical graph search algorithms [1]. The path planning problem can be efficiently solved even when considering high level missions expressed as temporal logic specifications for a single [2] or, even, multiple robots [3]. Temporal logic specifications can help system users to easily encode safety constraints and regulations for autonomous systems [4].

However, it became evident early enough that in many cases the temporal logic requirements need to be violated or partially satisfied in order for the path planning problem to become feasible [5]–[9]. Consider as a simple example, the scenario of a one-way road merging into a road network of two way streets. If on such a road there is an accident or a parked vehicle blocking a lane or a whole direction, then the vehicles may have to temporarily violate (move against the allowed direction) the traffic regulations in order for them to plan and resolve the local deadlock. Minimum violation methods [5]–[9] try to minimize a cost or metric capturing how much or for how long the high level specification is violated.

Unfortunately, this is not enough for the aforementioned driving scenario. In particular, backing up on the one way road can be locally tolerated, but traversing in the opposite direction a whole road segment should only be allowed for emergency and first responders' vehicles. In other words, our modeling framework needs to allow for several degrees of permissions for the different road segments. This will allow to formulate planning problems where the temporal logic specification is minimally violated (or partially satisfied) while at the same time satisfying a minimum permission level.

In order to solve this problem, in this paper, we take a multi-valued temporal logic planning approach [10]–[12]. That is, first, we use multi-valued logics to capture the different degrees of permissibility for robot motions (or actions in more general terms). Then, we define and solve an optimal temporal logic planning problem in the spirit of [13]. We demonstrate our solution on a road network scenario with different permission levels for accessing the different road segments.

Beyond just combining multi-valued and optimal temporal logic planning, we also make new contributions to the automata theoretic multi-valued temporal logic planning problem. In particular, we define threshold based multi-valued temporal logic formulas interpreted over interval based multi-valued traces and, then, we solve the resulting optimal temporal logic planning problem. The newly introduced thresholds allow one to specify lower bound and upper bound truth degrees in the spirit of Signal Temporal Logic (STL) [14]. In addition, the interval based semantics for the execution traces of multi-valued Transition Systems (mv-TS) allow modeling of uncertainty even in the truth degrees of the models themselves. Therefore, overall, we introduce a temporal logic planning framework which can handle uncertainty in the model and reason on the desired uncertainty for each constraint and mission goal independently.

Related work: In terms of applications, there has been a lot of recent work on minimum violation or partial satisfaction temporal logic planning [5]–[9], but none of these works (and their derivatives) take a multi-valued logic approach to the problem. We argue that the multi-valued logic framework is the right way to reason about the problem – at least for some applications – because the degree of satisfaction of the specification naturally captures constraint and subgoal violations at various degrees. In addition, the multi-valued framework enables a more user-friendly and versatile modeling language over classical Boolean transition systems and automata.

In terms of work related to the logic that we are using in this paper, two recent works stand out [15], [16]. In [15], the authors introduce robust interval semantics for STL for signal monitoring purposes. That is, when monitoring a signal, it is desirable to be able to estimate the range of the robust valuation of the specification in the future. On the other hand, in [16], symbolic automata are proposed for monitoring the robustness estimate of STL formulas with respect to sampled signals. Both approaches, i.e., [15] and [16], develop monitoring algorithms, while our work deals with the temporal logic planning problem.

This work was partially supported by the NSF award CPS 1446730, the NSF I/UCRC Center for Embedded Systems, and the NSF grant 1361926.

M. Hekmatnejad and G. Fainekos are with CIDSE at Arizona State University, Tempe, AZ, E-mail: {mhekmatn,fainekos}@asu.edu.

II. Problem Motivation and Description

For systems that are modeled as Transition Systems (TS), in Boolean model checking [17], there are some atomic proposition symbols (from now on by symbol we mean atomic proposition symbols) labeling each state which describe the status of the system or desired goals on that state. If the model checker uses Linear Temporal Logic (LTL) formulas with Boolean semantics for the specifications, then the presence of a symbol at a state is equivalent to the assignment of $true$ to that symbol from the Boolean domain. When there is a necessity to show that the assigned value to a symbol is not certain, then there are different approaches to give a certainty level to the symbol. We first show how the concept of truth degree can be incorporated into Boolean LTL model checking, and then introduce our solution.

The LTL specification $\varphi = \Box q \wedge \Diamond p$ with Boolean semantics means that *"q must always be true with a truth degree* 1*, and p must become true eventually with truth degree* 1*"*. Now, we change the specifications to the following: *"q must always be true with truth degree at least* 0.7*, and p must become true eventually with truth degree at least* 0.9. If we wanted to model and solve the problem using Boolean semantics, then we could accommodate the new requirements by adding two new Boolean symbols named $q_{\geq 0.7}$, and $p_{\geq 0.9}$ to represent q and p with least truth degrees 0.7 and 0.9, respectively. Also, it is required to construct a new subformula that captures the relationship among the new symbols (i.e., when truth degree is 1, then it is also greater than 0.7). The modified formula that denotes the new requirements is represented as $\varphi' = \Box q_{\geq 0.7} \wedge \Diamond p_{\geq 0.9} \wedge \Box(q \implies q_{\geq 0.7}) \wedge \Box(p \implies p_{\geq 0.9})$.

Now, in the model, at each state, depending on the truth degree of p and q, we have to introduce new Boolean symbols as in the specification formula. Also, based on the relationship among the truth degrees, we have to assign $true/false$ to the symbols on the states. For example, if q has truth degree of 0.8 in a state, then $q_{=0.8}$ is added to that state as $true$, and in all the other states we have to evaluate its value based on the truth degree of q on those states. Finally, it is critical to add the new implication/causal relations (even for the negative symbols) as part of the specifications (i.e., $\Box(q_{=0.8} \implies q_{\geq 0.7} \wedge q \implies q_{=0.8})$). Otherwise, some requirement statements might not cause logical inconsistencies in the absence of causal relations. On the contrary, existence of the causal relations makes the whole specification inconsistent (i.e., $\Box(q_{\geq 0.8} \wedge q_{\leq 0.7})$ is not inconsistence without considering causal relations related to $q_{\leq 0.7}$ and $q_{\geq 0.8}$). In general, inconsistencies in the specifications is a separate different topic. Such specification debugging problems in the context of timed temporal logic formulas are discussed in [18].

This way, we can use Boolean LTL to model check a system with uncertainty. Although, the above approach works in Boolean logic domain there are some major drawbacks:

- a specification formula carries overhead rules/subformulas so that the relationship among symbols remains sound. This affects the readability of the specification formulas as well as the size of the specification which increases the computational complexity of the model checking problem in an exponential fashion [17];
- a specification formula is dependent on new symbols on the model and there may exist unnecessary symbols in many states;
- it is not easy to compare the degree of satisfaction of different plans for a given LTL formula.

In the following, we propose an approach using multi-valued temporal logics (mv-TL) that can solve model checking problems with multiple required truth degrees, while avoids the above drawbacks. In mv-TL, for example, assigning values to a symbol from a quasi-Boolean lattice is one way to represent the certainty or truth degree of that symbol. That is, the assignment of $\top$ to a symbol means that it is certainly $true$, assignment of $\bot$ means certainly $false$, while other truth values represent varying certainty between these two opposite extremes. For example, on lattice $\mathfrak{L}_3$ (see Fig. 1(c)) there are totally six valid set of values that can be assigned to a symbol: $\{\langle false\rangle, \langle true\rangle, \langle maybe\rangle, \langle false, maybe\rangle, \langle maybe, true\rangle, \langle false, maybe, true\rangle\}$. For the sake of simplicity, we consider only Totally Ordered (TO) quasi-Boolean lattices in our definitions, and name each set an interval with the lowest and highest values in the set as their representative boundaries [19]. For example, we use $[maybe, true]$ to represent interval $\langle maybe, true\rangle$ in lattice $\mathfrak{L}_3$.

Adopting the aforementioned notation, now a symbol labeling a state can take its value from a bounded set of potential values. Furthermore, in the LTL specification, it can be enforced for a symbol to have a least truth degree by defining a set of potential values for its range such that the infimum of the set is the least requested truth degree. We are only interested in the cases that in LTL specifications the truth value intervals are either bounded by $\top$ as their supremum; or they are bounded by $\bot$ as their infimum. This is because we assume that usually in the specifications we only care about the least/most truth degrees for the symbols.

From here on, for the transition systems and LTL formulas which are using intervals as the range of their Atomic Proposition (AP) symbols, we add prefix "mvs" to them, therefor mvs-TS, mvs-Automaton and mvs-LTL are used in our descriptions to refer to the multi-valued-set transition systems, automata and LTL formulas, respectively.

In this paper, we are interested in finding an optimal plan for a system modeled as an mvs-TS $\mathcal{M}$ (which corresponds to a path/trace on $\mathcal{M}$) so that the path or observable trace satisfies an mvs-LTL specification φ with a given least desired truth degree l. Later, we introduce specification based cost functions Θ and Γ that compute the minimum truth degree and maximum violating cost for a plan, respectively. For the application of the aforementioned problem, we consider a high-level path planner which is tasked to plan for n robots located at n different locations to reach their destinations with the least total cost while satisfying a given mvs-LTL formula with an optimal truth degree.

Problem 1: Given an mvs-TS $\mathcal{M}$, an mvs-LTL specification φ, and a truth degree l, compute a path p on $\mathcal{M}$ such that the corresponding trace μ satisfies the specification with the least truth degree l, and also satisfies $\Gamma_\varphi(\mu) \leq \Gamma_\varphi(\mu')$, for any other path p' with trace μ' on $\mathcal{M}$ that satisfies the specification with at least truth degree l.

III. Preliminaries

A. Lattices

Informally, lattices can represent an order relation between the possible truth degrees in the system. A detailed exposition on lattices, order and mv-logics can be found in [10], [19], [20].

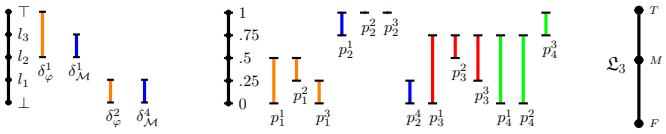
A partial ordered set or poset is a set S with an ordering relation $\preceq \subseteq S \times S$. For a poset $(S, \preceq)$, we define the *join* ($\vee : S \times S \to S$) and *meet* ($\wedge : S \times S \to S$) operations as $x \vee y := \sup(\{x, y\})$ and $x \wedge y := \inf(\{x, y\})$, respectively.

Definition 3.1 (Lattices): A poset $\mathfrak{L} = (L, \preceq)$ is called a:

- *lattice*, if for all $x, y \in L$, both $x \vee y$ and $x \wedge y$ exist;
- *totally ordered (TO)*, if for all $x, y \in L$, either $x \preceq y$ or $y \preceq x$ holds;
- *quasi-Boolean lattice*, if it is distributive, every element $l \in L$ has a unique *complement* ($\sim : L \to L$) $\sim l$, such that $\sim\sim l = l$, and $l_a \preceq l_b$ implies $\sim l_b \preceq \sim l_a$.

Also, we adopt the following notation. For a lattice $(L, \preceq)$, we define $\bigvee L = \sup(L) \triangleq \top$ and $\bigwedge L = \inf(L) \triangleq \bot$ which are read as the $join$ and the $meet$ of L, respectively.

The above definition of lattices $(L, \preceq)$ is based on partial orders, but equivalently we can define lattices as algebraic structures $(L, \vee, \wedge)$ that satisfy the *commutative*, *associative*, *absorption* and *idempotency* laws. A lattice with three values $\{\mathbf{F}alse, \mathbf{M}aybe, \mathbf{T}rue\}$ is depicted on Fig. 1(c) named $\mathfrak{L}_3$.



(a) (b) (c)

Fig. 1: (a) On the left: some intervals (labeled by δ) from the TO quasi-Boolean lattice $\mathfrak{L}_5$ on the left of the figure, where the order among values is depicted based on their position on the lattice (the upper position the higher the order); and we have $\sim \bot = \top, \sim l_1 = l_3, \sim l_2 = l_2$. (b) In the middle: some intervals aligned into four groups corresponding for four paths on lattice $\mathfrak{L}_5$ (the same as (a) but values are selected from reals). (c) On the right: TO quasi-Boolean lattice $\mathfrak{L}_3$ i.e., $F \prec M \prec T$ and $\sim F = T, \sim M = M$ (we abbreviate {False,Maybe,True} to {F,M,T}).

Next, we formally define set of values over a TO quasi-Boolean lattice as intervals which is the essence of extending the current multi-valued logic to multi-valued-set (mvs from now on) logic, and establishing new definitions.

Definition 3.2 (Interval): Given a TO quasi-Boolean lattice $\mathfrak{L} = (L, \preceq, \sim)$, we call $S \in 2^L$ an interval if $\forall l \in L$ s.t. $\inf(S) \leq l \leq \sup(S)$, we have $l \in S$. We denote by L^I the set of all intervals of a lattice $\mathfrak{L}$. We adopt the following notation to represent an interval.

$$S \triangleq [\inf(S), \sup(S)]$$

An interval here subsumes up-sets and down-sets as defined in [19] (Chapter 2, Section 3.10).

Some examples of intervals over the TO quasi-Boolean lattice $\mathfrak{L}_5$ are presented in Fig. 1(a). For example, $\delta^1_\mathcal{M} = [l_2, l_3]$, and $\delta^1_\varphi = [l_2, \top]$ are two intervals.The following result is immediate.

Proposition 3.1: Any interval is a TO lattice.

In the following we formally minimally extend the definition of some lattice operators over intervals.

Definition 3.3 (Lattice Operators): Having a TO quasi-Boolean lattice $\mathfrak{L} = (L, \preceq, \sim)$, we extend the definition of lattice operators to operate on intervals L^I of $\mathfrak{L}$. For intervals, we define the *join* ($\sqcup : L^I \times L^I \to L^I$), *meet* ($\sqcap : L^I \times L^I \to L^I$), and *negation* ($\sim : L^I \to L^I$) operations as follows, given $(l_1, l_2, h_1, h_2 \in L)$:

$$[l_1, h_1] \sqcup [l_2, h_2] := [l_1 \vee l_2, h_1 \vee h_2]$$
$$[l_1, h_1] \sqcap [l_2, h_2] := [l_1 \wedge l_2, h_1 \wedge h_2]$$
$$\sim [l, h] := [\sim h, \sim l]$$

For example, on lattice $\mathfrak{L}_5$ in Fig. 1(a), we have $[l_1, l_3] \sqcup [l_1, l_2] = [l_1, l_3]$, and $[l_1, \top] \sqcap [l_2, l_3] = [l_1, l_3]$.

B. mvs-TS and mvs-LTL

We can now define transition systems whose traces take interval values over a truth lattice.

Definition 3.4 (Multi-Valued-Set Transition System): (mvs-TS) is a tuple $\mathcal{M} = (S, S_0, \mathfrak{L}, \to, V, \mathcal{O})$ where: S is a finite set of states; $S_0 \subseteq S$ is the set of initial states; $\mathfrak{L}$ is a lattice $(L, \wedge, \vee)$; $\to \subseteq S \times S$ is a transition relation; V is a finite set of variables; and $\mathcal{O} : S \times V \to L^I$ is a total labeling function.

A path $p \in S^\omega$ on an mvs-TS has an equivalent trace $\mu : \mathbb{N} \times V \to L^I$ such that $\mu(i, \pi) := \mathcal{O}(p_i, \pi)$, for all $i \geq 0$.

Next, it is desirable to formalize some LTL formulas in which we use $\pi \succeq l$ or $\pi \preceq l$ to imply π must have truth degree at least l or at most l, respectively. Consider the specification $\Box(\neg\pi)$ in Boolean semantics. In this case, a satisfying trace would be $\mu(\pi, i) = \bot,\ \forall i \geq 0$ (meaning that every state of trace μ satisfies the formula $\neg\pi$), and $[\![\neg\pi]\!](\mu, i) = \top,\ \forall i \geq 0$. If we were to write the same specification in our logic over the Boolean lattice ($L = \{\bot, \top\}$), we would write $\Box(\pi \preceq \bot)$, and using our semantics we would get $[\![\pi \preceq \bot]\!](\mu, i) = \sim [\bot, \bot] = [\top, \top],\ \forall i \geq 0$ assuming that now $\mu(\pi, i) = [\bot, \bot],\ \forall i \geq 0$. Similarly, if we had a lattice with more truth values, e.g., $\mathfrak{L}_5$ over set $\{\bot, l_1, l_2, l_3, \top\}$ (see Fig. 1(a)), and an interval over $\mathfrak{L}_5$ such that $\mu(\pi, i) = [\bot, l_1]$, then the specification $\Box(\pi \preceq l_2)$ would evaluate $[\![\pi \preceq l_2]\!](\mu, i) = \sim [\bot, l_1] = [l_3, \top],\ \forall i \geq 0$.

Here, we define mvs-LTL formulas and their semantics without considering negation in the formulas. That is, we consider formulas in Negation Normal Form (NNF) since we utilize quasi-Boolean truth lattices and since any negative expression such as $\neg(\pi \preceq l)$ or $\neg(\pi \succeq l)$ can be modeled with $\pi \succ l$ and $\pi \prec l$, respectively.

Definition 3.5 (mvs-LTL): Let V be the set of variables, and $\mathfrak{L} = (L, \preceq, \sim)$ a TO quasi-Boolean lattice, then a well formed mvs-LTL formula ϕ is generated according to the grammar $\phi ::= \pi \preceq l \mid \pi \succeq l \mid \phi \vee \phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi \mid \phi \mathcal{R} \phi$, where $\pi \in V$, $l \in L$, $\bigcirc$ is the next time operator, $\mathcal{U}$ is the until operator, and $\mathcal{R}$ is the release

operator. The $\succeq$ operator enforces a minimum truth degree for a symbol, and the $\preceq$ operator enforces a maximum truth degree for its symbol.

Next, we introduce a satisfaction evaluator function Ψ to check if the value of a symbol from a model (source) satisfies the minimum required value of the same symbol from the specification (reference) or not. If the source interval satisfies the required least/most degree of truth then Ψ returns either the source interval or its negation depending on which one has a upper bound. For the case that the source interval does not satisfy the required least/most degree of truth, then Ψ returns singleton interval $[\bot, \bot]$. For example in Figure 1(a), we have the following results:

$$\Psi(\delta^1_\varphi, \delta^1_\mathcal{M}) = \delta^1_\mathcal{M}, \Psi(\delta^2_\varphi, \delta^2_\mathcal{M}) = \sim \delta^2_\mathcal{M}, \Psi(\delta^1_\varphi, \delta^2_\mathcal{M}) = [\bot, \bot]$$

Definition 3.6: The satisfaction evaluator function Ψ is defined as $\Psi([l_{base}, h_{base}], [l, h]) =$

$$\begin{cases} [l, h] & \text{if } l_{base} \preceq l \text{ and } h_{base} = \top \\ \sim [l, h] & \text{if } l_{base} = \bot \text{ and } h \preceq h_{base} \prec \top \\ [\bot, \bot] & \text{otherwise} \end{cases}$$

The function Ψ takes two intervals, the first as the reference and the second as the source. It is important to note that for the reference interval, either $l_{base} = \bot$ or $h_{base} = \top$. This is due to the fact that we want to differentiate between the two following cases: if the reference interval demands for intervals with higher infimum than itself; or if it demands for intervals with lower supremum than itself.

Definition 3.7: For an mvs-LTL formula ϕ and a trace $\mu : \mathbb{N} \times V \to L^I$ on an mvs-TS that takes intervals over a quasi-boolean lattice $\mathfrak{L}^I = (L^I, \preceq, \sim)$, and satisfaction evaluator function $\Psi : L^I \times L^I \to L^I$ the mvs-Semantics of ϕ with respect to μ are:

$$\begin{aligned}
[\![\pi \preceq l]\!](\mu, i) &:= \Psi([\bot, l], \mu(i, \pi)) \;\; \pi \in V,\; l \in L \\
[\![\pi \succeq l]\!](\mu, i) &:= \Psi([l, \top], \mu(i, \pi)) \;\; \pi \in V,\; l \in L \\
[\![\phi_1 \vee \phi_2]\!](\mu, i) &:= [\![\phi_1]\!](\mu, i) \sqcup [\![\phi_2]\!](\mu, i) \\
[\![\phi_1 \wedge \phi_2]\!](\mu, i) &:= [\![\phi_1]\!](\mu, i) \sqcap [\![\phi_2]\!](\mu, i) \\
[\![\bigcirc \psi]\!](\mu, i) &:= [\![\psi]\!](\mu, i + 1) \\
[\![\phi_1 \mathcal{U} \phi_2]\!](\mu, i) &:= \bigsqcup_{j \geq i} \big([\![\phi_2]\!](\mu, j) \sqcap \bigsqcap_{i \leq k < j} [\![\phi_1]\!](\mu, k) \big) \\
[\![\phi_1 \mathcal{R} \phi_2]\!](\mu, i) &:= \bigsqcap_{j \geq i} \big([\![\phi_1]\!](\mu, j) \sqcup \bigsqcup_{i \leq k < j} [\![\phi_2]\!](\mu, k) \big)
\end{aligned}$$

Intuitively by $[\![\pi]\!](\mu, 0) = [l, h],\; l, h \in L$, we mean that the formula ϕ is satisfiable under trace μ with truth degree at least equal to l, and at most equal to h. Based on the above semantics, we define (*eventually*) $\Diamond\phi$ and (*globally*) $\Box\phi$ as $(\top \mathcal{U} \phi)$ and $(\bot \mathcal{R} \phi)$, respectively.

Lastly, we define merging of n mvs-TS with single initial state, as the n-init merged mvs-TS, mainly because for robot motion planning problems, the environment abstraction is typically the same for each robot, but the initial position of robots are different. Later, we need to have the product of the model and the specification for model checking and planning purposes. Therefore, we merge different model transition systems into one for the sake of simplicity.

Definition 3.8 (n-init Merged mvs-TS): Given an mvs-TS $\mathcal{M} = (S, S_0, \mathfrak{L}, \to, V, \mathcal{O})$ modeling a multi robot environment, where each state in the S_0 (where S_0 is a n-tuple) is considered as an initial state for a robot in the system, we build n mvs-TS $\mathcal{M}_1, ..\mathcal{M}_n$ which are the same as $\mathcal{M}$, but each has a single unique initial state s_{0_i} from S_0 (i.e., $\bigcup_{0<i\leq n} s_{0_i} = S_0$), and build a merged mvs-TS $\mathcal{M}^n$ which is the synchronous composition of $\mathcal{M}_1 \times .. \times \mathcal{M}_n$ [17].

A merged-state of $\mathcal{M}^n$ is a tuple of n states, which represent the current states of all robots at each merged-state. Based on each robot's motion strategy and for the sake of performance we can omit unreachable merged-states such as those that have more than one robot located at the same state and so forth.

IV. Multi-Valued-Set LTL Planning

In this section, we introduce planning under uncertainty by extending the concept of multi-value-set over automaton, and product of them with mvs-TS (mvs-LTL planning automaton). Later, for finding optimal plans on the mvs-LTL planning automaton, we define cost functions that consider traversing costs based on modeled system and satisfying robustness.

A. mvs-Automaton

In Boolean model checking, we can construct an automaton $\mathcal{A}_\varphi$ for a given LTL formula φ [21]. Its language $L(\mathcal{A}_\varphi)$ is defined as the set of words w that produce accepting runs over the automaton, such that $[\![\varphi]\!](w, 0) = \top$. Using similar techniques, we can build an mvs-Automaton for an mvs-LTL formula without any negations in it [12]. We need to represent interval expressions symbolically (i.e., $\pi \succeq \top$ is treated as a propositional symbol), and apply the classic Boolean translation. In the constructed automata, we have edges with DNF logical formulas on them. If, as a result of translation, any of interval expressions became negative, then we remove the negations by pushing them in their expressions. If there was a contradictory expression in the mvs-LTL formula, such as $\Box(q \succeq 0.8 \wedge q \preceq 0.7)$, then we still get a valid mvs-Automaton, but its language is $\emptyset$ with respect to mvs-Semantics. For expressions with the same variables with the same supremum $\top$, or infimum $\bot$, further simplification is possible. For example, $p \succeq 0.9 \wedge p \succeq 1$ is the conjunction of two expressions of the same variable p with $\top$ as their supremum. This conjunction can be simplified to $(p \succeq 1)$ since $p \succeq 1$ implies $p \succeq 0.9$. In the following, we formally define an mvs-Automaton, a run, an accepting run, and state the equivalence of a constructed mvs-Automaton from an mvs-LTL formula.

Definition 4.1 (Multi-Valued-Set Automaton): is a tuple $\mathcal{A} = (S, s_0, \mathfrak{L}, \Sigma, \Delta, \overline{\mathcal{O}}, F)$ where: S is a finite set of states; $s_0 \in S$ is the initial state; $\mathfrak{L}$ is a TO quasi-Boolean lattice $(L, \preceq, \sim)$; Σ is a finite alphabet; $\Delta : S \times S \to 2^\Sigma$ labels the transitions of the automaton with sets of symbols from Σ; $\overline{\mathcal{O}} : (S \times S \times \Sigma \to L^I)$ maps each symbol to an interval; and F is the set of accepting states.

Definition 4.2 (Run and Accepting Run): A *run* $p \in S^\omega$ of an mvs-Automaton $\mathcal{A} = (S, s_0, \mathfrak{L}, \Sigma, \Delta, \overline{\mathcal{O}}, F)$ under word $w \in (L^{I^\Sigma})^\omega$ (or alternatively $w : \mathbb{N} \times \Sigma \to L^I$ which has the same behavior as a trace) is an infinite sequence of states such that $\forall i \geq 0$, $w_i(\Delta(p_i, p_{i+1})) \triangleq \sqcap_{\sigma \in \Delta(p_i, p_{i+1})} \Psi(\overline{\mathcal{O}}(p_i, p_{i+1}, \sigma), w(i, \sigma))$

$\iff \inf(\sqcap_{\sigma\in\Delta(p_i,p_{i+1})}\ \Psi(\overline{\mathcal{O}}(p_i,p_{i+1},\sigma),w(i,\sigma))) \succ \bot$. We call a run an accepting run if it contains infinite states from F in it, and we denote the set of all accepting runs of $\mathcal{A}$ by $AR(\mathcal{A})$.

Remark 4.1: We call an ultimately periodic path as a run/path $p = p^f \circ p^c$ with a finite prefix p^f and infinite executing loop/cycle p^c.

Theorem 4.1 (Theorem 3 in [12]): An mvs-Automaton $\mathcal{A}_\phi = (S, s_0, \mathfrak{L} = (L, \preceq, \sim), V, \Delta, \overline{\mathcal{O}}, F)$ constructed for an mvs-LTL formula ϕ assigns the interval $[l, h] \in L^I$ from the TO quasi-Boolean lattice $\mathfrak{L}$ to a word $w \in (L^{I^V})^\omega$, $\bigsqcup_{p\in AR(\mathcal{A})}\bigsqcap_{i\geq 0} w_i(\Delta(p_i, p_{i+1})) = [l, h] \iff [\![\phi]\!](w, 0) = [l, h]$.

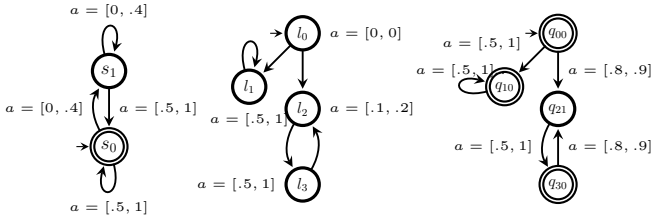


Fig. 2: (a) On the left: an automaton for $\varphi = \square\diamond(a \succeq 0.5)$. We abbreviate $\overline{\mathcal{O}}(s_0, s_0, a) = [0.5, 1]$ with $a = [0.5, 1]$ on the loop on s_0. Similar abbreviation is used for the rest. (b) In the middle: a TS for robot motion planning. (c) On the right: an mvs-LTL planning automaton with respect to formula $\varphi = \square\diamond(a \succeq 0.5)$, where we ignored representing mapping functions.

In Fig. 2(a), an equivalent automaton for $\varphi = \square\diamond(a \succeq 0.5)$ is presented. Variable a takes values from truth lattice $\mathfrak{L}_{10}$ that is a TO quasi-Boolean lattice with ten truth values $0, 0.1, ..., 0.9, 1$ with the usual order relation over the reals. It is not difficult to check that any accepting run on the automaton has to visit state s_0 infinitely often; and for doing that, it has to visit transitions with $a \succeq 0.5$ on it (if a became less than 0.5, then it goes to state s_1, but eventually it has to take the other transition from s_1 to s_0 with $a \succeq 0.5$). This is semantically the same as what the formula states: *a always must eventually have truth degree at least* 0.5.

B. mvs-LTL Planning Automaton

Now that we have introduced mvs-TS and mvs-Automaton, we are going to show the construction of product of them with respect to a given threshold as the minimum truth level.

Definition 4.3 (mvs-LTL Planning Automaton): Given an mvs-TS $\mathcal{M} = (S_\mathcal{M}, s_{\mathcal{M}0}, \mathfrak{L} = (L, \preceq, \sim), \rightarrow, V, \mathcal{O})$ and a specification mvs-Automaton $\mathcal{A}_\varphi = (S_\mathcal{A}, s_{\mathcal{A}0}, \mathfrak{L} = (L, \preceq, \sim), V, \Delta_\mathcal{A}, \overline{\mathcal{O}}_\mathcal{A}, F_\mathcal{A})$, and minimum truth level $l \in L$, the mvs-LTL Planning Automaton is $\mathcal{A}_{\mathcal{M}\varphi} = (S, s_0, \mathfrak{L} = (L, \preceq, \sim), V, \Delta, \overline{\mathcal{O}}, F, l)$, where $S = S_\mathcal{M} \times S_\mathcal{A}$ is the set of states; $s_0 = (s_{\mathcal{M}0}, s_{\mathcal{A}0})$ is the initial state; $\Delta : S \times S \to 2^V$ labels the transitions; $F = S_\mathcal{M} \times F_\mathcal{A}$ is the set of final vertices, and $\overline{\mathcal{O}} : S \times S \times V \to L^I$ is a mapping function that maps symbols on a transition to an interval such that for given $s = (s_\mathcal{M}, s_\mathcal{A})$, $s' = (s'_\mathcal{M}, s'_\mathcal{A})$,

$$\Delta(s, s') \triangleq \begin{cases} \Delta_\mathcal{A}(s_\mathcal{A}, s'_\mathcal{A}), & \text{if } s_\mathcal{M} \to s'_\mathcal{M} \\ \emptyset, & \text{otherwise} \end{cases}$$

$$\forall \pi \in \Delta(s, s') : \overline{\mathcal{O}}(s, s', \pi) = \Psi\big(\overline{\mathcal{O}}_\mathcal{A}(s_\mathcal{A}, s'_\mathcal{A}, \pi), \mathcal{O}(s'_\mathcal{M}, \pi)\big)$$
$$\iff s_\mathcal{M} \to s'_\mathcal{M} \text{ and}$$
$$\inf\Big(\bigsqcap_{\pi\in\Delta_\mathcal{A}(s,s')} \Psi\big(\overline{\mathcal{O}}_\mathcal{A}(s_\mathcal{A}, s'_\mathcal{A}, \pi), \mathcal{O}(s'_\mathcal{M}, \pi)\big)\Big) \succ l;$$

Note that in φ, we remove the negation sign in front of symbols (φ is in NNF) by applying the negation to its interval. The same can be done for symbols on $\mathcal{A}_\varphi$. For example, the automaton in Fig. 2(a), where the mapping function for a on the transition from s_1 to itself and from s_0 to s_1 are changed to $\overline{\mathcal{O}}(s_1, s_1, a) = \overline{\mathcal{O}}(s_0, s_1, a) = [0, 0.5) = [0, 0.4]$. Note that all symbols that are used in the $\mathcal{A}_\varphi$ have interval values that are bounded either by $\bot$ from below or by $\top$ from above. For the above mvs-LTL formula and the mvs-TS model depicted in Fig. 2(b), the resulting mvs-LTL planning automaton is shown in Fig. 2(c). The planning automaton has two accepting plans $p^1 : q_{00}, q_{10}, (q_{10})^\omega$, and $p^2 : q_{00}, q_{21}, q_{30}, (q_{21}, q_{30})^\omega$. In the next section, we introduce cost functions by which one can distinguish the optimal plans in the previous planning automaton.

C. Cost Functions

In this paper, we are not concerned about computing an optimal path based on the infinite trace for the system as for example done in [22]. In [22], they define a cost resetting proposition which resets the total cumulative cost in the mission specified by the temporal logic specification. Here, instead, we compute ultimately periodic paths that satisfy the specification with an optimal truth degree [10]. Among these optimal truth degree paths, we select one which optimizes an additive cost function.

In the following, by transition minimum-capacity, we mean an interval which its infimum is equal to the minimum of all infimums of the intervals of transition variables, and its supremum is the minimum of all the supremums of the intervals of the transition variables. Assume that there are only two interval variables $\delta_1 = [l_2, \top], \delta_2 = [\top, \top]$ from lattice $\mathfrak{L}_5$ on a transition of an mvs-Automaton. In this case, the minimum-capacity of the transition is $[l_2, \top]$. Informally, it means that all the runs which pass through this transition have at most minimum truth degree l_2 and maximum truth degree $\top$.

Definition 4.4 (Transition Minimum Capacity): Given an mvs-Automaton $\mathcal{A} = (S, s_0, \mathfrak{L} = (L, \preceq, \sim), \Sigma, \Delta, \overline{\mathcal{O}}, F)$, we define the minimum-capacity of a transition between two states as $\theta : S \times S \to L^I$ such that

$$\theta(s_i, s_j) = \begin{cases} \bigsqcap_{\pi\in\Delta(s_i,s_j)} \overline{\mathcal{O}}(s_i, s_j, \pi), & \text{if } \Delta(s_i, s_j) \neq \emptyset \\ [\bot, \bot], & \text{otherwise} \end{cases}$$

In order to distinguish between transitions with higher minimum capacities, we define our desirability function so that it maps values from lattice $\mathfrak{L}$ to a set of real numbers between 0 and 1, such that the order of the values is being reflected on their desirability.

Definition 4.5 (Desirability): Given a quasi-Boolean lattice $\mathfrak{L} = (L, \preceq, \sim)$, we define a desirability function $\lambda : L \to \mathbb{R}_{[0,1]}$, such that $\forall l_1, l_2 \in L$, we have $(l_1 \preceq l_2) \implies (\lambda(l_1) \leq \lambda(l_2))$, and $\lambda(\top) = 1, \lambda(\bot) = 0$.

For example, for $\mathfrak{L}_3$ we can have $\lambda(false) = 0, \lambda(maybe) = 0.5$, and $\lambda(true) = 1$.

Next, we define two heuristic-cost functions for a transition on an mvs-Automaton, one as the primary, and other as the secondary. By primary, we mean it has higher weight for comparison purposes; similarly, by secondary, we mean it has lower weight for comparison purposes. The reason for having two functions is that in the case two transitions have the same cost based on their lower minimum-capacity, then we compute and compare their cost based on their upper minimum-capacity. In our cost function, we mix the cost of the model with the cost of least satisfiability for specifications. For the model, $c_{i,j}$ represents the cost for transition between states i and j; and for the specification satisfiability cost, we reduce the desirability of the infimum of the minimum-capacity of the transition from one. Therefore, for the higher minimum-capacities we get lower costs.

Definition 4.6 (Primary and Secondary Transition Costs): Given an mvs-Automaton $\mathcal{A} = (S, s_0, \mathfrak{L} = (L, \preceq, \sim), \Sigma, \Delta, \overline{\mathcal{O}}, F)$, we define the primary and the secondary transition cost functions between two states by $\gamma_1, \gamma_2 : S \times S \to \mathbb{R}_{>0}$, respectively such that

$$\gamma_1(s_i, s_j) = \quad c_{i,j} + 1 - \lambda\big(\inf(\theta(s_i, s_j))\big), \quad \Delta(s_i, s_j) \neq \emptyset$$
$$\gamma_2(s_i, s_j) = \quad c_{i,j} + 1 - \lambda\big(\sup(\theta(s_i, s_j))\big), \quad \Delta(s_i, s_j) \neq \emptyset$$

where γ_1 and γ_2 calculate the primary and secondary transition costs, respectively; and $c_{i,j}$ is a positive real number as a cost for transition from s_i to s_j. Both functions return $+\infty$ if Δ is empty.

For example, in Fig. 1(b), if we assume that the desirability of values on lattice $\mathfrak{L}_5$ are their values, the fixed cost is 1, and the minimum-capacity of two arbitrary transitions are p_1^1 and p_1^3, respectively; then their primary costs are 2 and 2, and their secondary cost are 1.5 and 1.75, respectively. Based on our cost functions, the first transition is less expensive than the second one. From the figure, one can determine which interval has lower cost, by checking which one has a higher lower bound, and then higher upper bound.

In the following, we introduce the cost of a path/run on an mvs-Automaton, where we use the summation of transition costs for the non-cyclic and finite cyclic part of the path.

Definition 4.7 (Primary and Secondary Path Cost): Given an mvs-Automaton $\mathcal{A} = (S, s_0, \mathfrak{L} = (L, \preceq, \sim), \Sigma, \Delta, \overline{\mathcal{O}}, F)$, and an ultimately periodic path $p = p^f \circ p^c$ with a finite length $n = |p^f| + |p^c|$, and its corresponding trace μ, we define primary and secondary path cost functions $\Gamma_1, \Gamma_2 : S^* \to \mathbb{R}_{>0}$ as

$$\Gamma_1(p) = \sum_{0 \leq i < n} \gamma_1(\mu_i, \mu_{i+1}), \;\; \Gamma_2(p) = \sum_{0 \leq i < n} \gamma_2(\mu_i, \mu_{i+1})$$

where S^* is the set of all finite sequences of states over S. Here, Γ_1 calculates the primary cost of a path, and Γ_2 calculates the secondary cost.

A similar concept for the cost of paths can be defined for the capacity of paths. For calculating the primary minimum-capacity of a path, we simply find the minimum of the lower bounds and the minimum of the upper bounds for all the minimum capacities of the transitions of the path. We use the two minimums to build the interval as the primary minimum-capacity of the path. The same way that we calculated the cost of paths, we heuristically calculate the secondary minimum-capacity of a path, but without considering the fixed cost for transitions, and taking the average cost per transition. Again, we use the secondary result for comparison purposes when the primary capacities are the same.

Definition 4.8 (Primary and Secondary Path Capacity): Given an mvs-Automaton $\mathcal{A} = (S, s_0, \mathfrak{L} = (L, \preceq, \sim), \Sigma, \Delta, \overline{\mathcal{O}}, F)$, and an ultimately periodic path $p = p^f \circ p^c$ with a finite length $n = |p^f| + |p^c|$, and its corresponding trace μ, we define primary and secondary path capacity functions $\Theta_1 : S^* \to L^I$, and $\Theta_2 : S^* \to \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$, respectively as

$$\Theta_1(p) = \prod_{0 \leq i < n} \theta(\mu_i, \mu_{i+1}), \qquad \Theta_2(p) =$$
$$\Big\langle \sum_{0 \leq i < n} \frac{\lambda(\inf(\theta(\mu_i, \mu_{i+1})))}{n}, \sum_{0 \leq i < n} \frac{\lambda(\sup(\theta(\mu_i, \mu_{i+1})))}{n} \Big\rangle$$

One can use these functions to find optimal plans based on maximizing the minimum capacity of plans. Based on the planning problem, one can mix the cost and capacity functions to achieve certain optimality objectives.

Assume that each set of intervals in Fig. 1(b) is corresponding to the minimum-capacity of transitions of a path/plan (i.e., there are four plans p_1, p_2, p_3, and p_4). Using the path capacity functions, we can say p_1, p_2 with the same primary capacity are less trustworthy paths/plans in comparison to p_3, p_4. Also, between p_1 and p_2, p_2 is more desirable as the optimal plan; and the same way, p_4 is more desirable in comparison to p_3.

Finally, we can use the introduced cost functions to calculate optimal plans on an mvs-LTL planning automaton.

Proposition 4.1: Given an mvs-LTL planning automaton $\mathcal{A}_{\mathcal{M}\varphi}$ as in Def 4.3, the least expensive run $r^{optimal}$ (aka most trustworthy run) can be calculated by using a modified version of *Dijkstra* or *A* algorithm*, and the introduced cost and capacity functions.

V. Motion Planning Examples

Our approach can be used for model checking with multi-valued logics in general, but in this section, we use it for optimal LTL motion planning in mission critical autonomous driving cars where the underlying road network has multiple degrees of accessibility. We show how the planner can be used to find an optimal plan complies with some specifications which are represented in mvs-LTL. Due to space limitations, we cannot present all the details in our example use case, but rather we focus on important concepts.

A. Application Scenario

Assume that there are two autonomous driving ambulances (robots) that have a critical rescue mission. The main idea is that we have represented the road network for the mission as a block world, where the allowed direction of entering and exiting the blocks is represented on them. It is required that in some situations the autonomous cars be able to move against the allowed direction of entering/exiting a block. We will use our introduced multi-valued logics here to represent the desirability of moving along each direction. Although entering/exiting against the specified direction is

not allowed, in general, these cars should be able to override the restrictions when needed.

We want to assign two target locations to each car, one as the location to load the rescuees, and the other to unload them. It is crucial that they complete the task in the least amount of time, while their priority changes according to whether they are loaded or empty. For example, we want them to be able to take the risk of moving against the regular allowed directions while they are empty, but as soon as the one with higher privilege (car number two) gets the patient loaded (done with the first part of its task), we want both to not take any risks anymore. Alternatively, we may want the robots to take risks when loaded in case a patient needs to be transported to a hospital as soon as possible. In the following, we demonstrate modeling and planning of the aforementioned scenario using our framework.

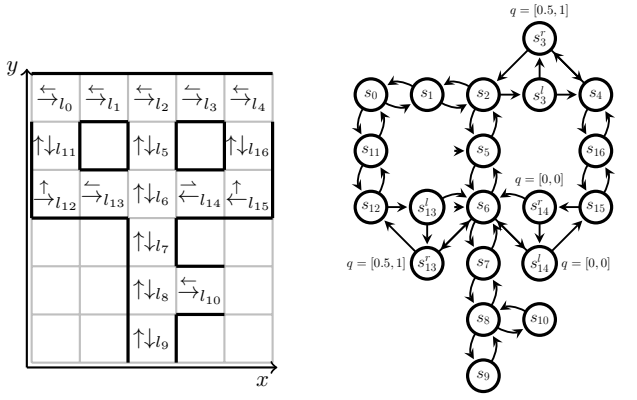


Fig. 3: (a) On the left: a block-road system in which blocks with $\leftharpoondown$ or $\rightharpoonup$ arrow on them represent road sections that cars can move toward the arrow direction with cautious, and for the normal arrows, there is no restriction. The indexed letter l on each block represents the location. The capacity of each block at each time allows only one car to be in. (b) On the right: the corresponding mvs-TS of the block-road system on (a) is depicted. For the states that $q = [0.5, 1]$ or $q = [0, 0]$, it is shown as label outside them, but for the rest of the states in which $q = [1, 1]$, the symbols are omitted.

B. Modeling

A road system is depicted in the Fig. 3(a), in which a regular arrow in each block represents the direction of movement without any caution sign for that. On the other hand, the harpoon shape arrows are one-way roads that can be accessed by emergency vehicles when required. In this example, in the modeling, we use $q \in V$ (V is the set of interval variables) taking values from lattice $\mathfrak{L}_3$ (we change the values of F, M, T in Fig. 1(c) to $0, 0.5$, and 1, respectively) to represent desirability/accessibility of entering a block such that a state with $q = [1, 1]$ is desirable to enter, a state with $q = [0.5, 1]$ needs caution to enter, and a state with $q = [0, 0]$ is not allowed to enter. As part of priori knowledge, we know that block l_{14} is temporarily not allowed to enter, and all blocks except for l_3, l_{13}, and l_{14} are open and not directionally restricted for cars to enter. Also, each block only allows the presence of one car at each time. The 2-init mvs-TS version of the system is illustrated on Fig. 3(b). Please note that other possible design preference symbols (e.g., collision avoidance) are not shown on the mvs-TS, and that we omit displaying self loops on the mvs-TS for readability. The initial states s_5, s_6, correspond to the location of the two cars C^1 and C^2, and they are equal to l_5 and l_6, respectively. Here, because there are only two directions allowed for entering/exiting each restricted block (i.e., l_3, l_{13}, and l_{14}), there are two states with the same name but superscripted by l or r (abbreviation for left and right) on top of them to show the direction of entrance into them.

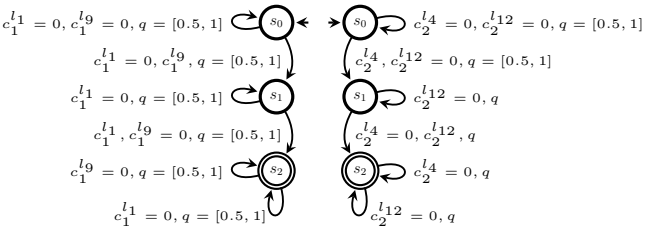


Fig. 4: (a) On the left: an mvs-LTL automaton for ϕ_1 in Def.5.1. For simplicity, when we don't put range for a non-negative symbols they are $[1, 1]$. (b) On the right: an mvs-LTL automaton for ϕ_2. We abbreviated interval $[0, 0]$ by 0 for readability.

C. Specification Requirements

Assume that the goal for for C^1 and C^2 is to satisfy some mvs-LTL specification formulas ϕ_1 and ϕ_2, respectively, while we know that block l_{14} is not allowed to be entered temporarily. The informal narrative mission specification is:

Definition 5.1 (Specification Goals): $\phi_1 :=$ *always avoid entering not allowed blocks, and eventually visit* l_9*, and then eventually visit* l_1 *after visiting* l_9 , and $\phi_2 :=$ *always avoid entering not allowed blocks, and eventually visit* l_4*, and then eventually visit* l_{12} *after visiting* l_4 *while always not entering caution marked blocks.*

For each specification goal there are three variables (symbols with interval ranges), from which a global variable q is used for enforcement of level of accessibility, and the rest represent if the specified blocks in the specification are visited by specific cars. The new variables are $c_1^{l_1}, c_1^{l_9}, c_2^{l_4}, c_2^{l_{13}}$, and q, such that $c_1^{l_1} = [1, 1]$ means that certainly the current location of C^1 is l_1. A similar interpretation applies to the rest. The specifications ϕ_1 and ϕ_2 are formalized as follows:

$$\phi_1 = \Box(q \succeq 0.5) \wedge \Diamond c_1^{l_1} \wedge \Diamond c_1^{l_9} \wedge (c_1^{l_1} \preceq 0)\,\mathcal{U} c_1^{l_9} \wedge \Box(c_1^{l_1} \preceq 0 \vee c_1^{l_9} \preceq 0)$$

$$\phi_2 = \Box(q \succeq 0.5) \wedge \Diamond c_2^{l_4} \wedge \Diamond c_2^{l_{12}} \wedge (c_2^{l_{12}} \preceq 0)\,\mathcal{U} c_2^{l_4} \wedge \Box(c_2^{l_4} \preceq 0 \vee c_2^{l_{12}} \preceq 0) \wedge \Box(c_2^{l_4} \implies \bigcirc\Box(q \succeq 1))$$

We highlight that the requirement $\Box(q \succeq 0.5)$ captures the fact that the ambulances are allowed to enter an opposite direction road when needed. This is now controlled by the existence of feasible plans with $q = 1$, i.e., without any violations (truth value optimality) and other relevant cost functions (i.e, distance traversed) to compute the optimal plans.

We remark that collision avoidance can be directly modeled on the merged mvs-TS. The mvs-Automaton translation of the above formulas are illustrated in Fig. 4. Note that here for the sake of simplicity we represent the specifications separately while in reality we conjunct them (q is a shared symbol among them), and build a single mvs-Automaton for this use case.

D. Planning

Beginning with the initial situation, both cars have to reach their first targets l_9 and l_4, and then finally visit l_1, and l_{12}, respectively. Before C^1 loads its rescuees, both vehicles are free to take risks by riding against the allowed road direction to get into the blocks l_3 and l_{13}, only if it helps them to optimize their global plan. It is clear from the specifications that they cannot get into l_{14} which is not an allowed location. After C^2 was loaded, then l_3 and l_{13} become only-one-way to the cars, which limits their freedom of movement by minimizing taking any risks.

We implemented a planner framework based on our approach, and modeled the above use case. The execution result returned the below trace of an optimal plan with a cost of 27 using the cost functions in Def. 4.7.

$r_1^{optimal} := (s_5, s_6) \overset{2.5}{\rightarrow} (s_6, s_{13}^r) \rightarrow (s_7, s_6) \rightarrow (s_8, s_5) \rightarrow (\mathbf{s_9}, s_2) \rightarrow (s_8, s_3) \rightarrow (s_7, \mathbf{s_4}) \overset{2.5}{\rightarrow} (s_6, s_3^r) \rightarrow (s_5, s_2) \rightarrow (s_2, s_1) \rightarrow (\mathbf{s_1}, s_0) \rightarrow (s_1, s_{11}) \rightarrow (s_0, \mathbf{s_{12}}) \rightarrow (s_{11}, s_{12})^*$.

In the above path the bold states are those that must be traversed with the predefined order (i.e, first s_9 and then s_1), and by $s_i \overset{v}{\rightarrow} s_j$ we mean $\gamma_1(s_i, s_j) = v$, and by $s_i \rightarrow s_j$ we mean $\gamma_1(s_i, s_j) = 2$. The resulting plan is depicted in Fig. 5 demonstrating the motion plan of each car separately.

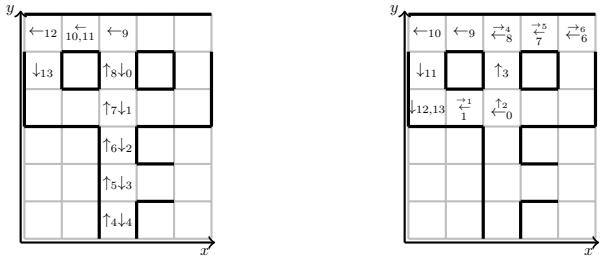


Fig. 5: (a) On the left, an optimal plan for car C^1. (b) On the right, an optimal plan for car C^2. In the picture the numbers at each location represent the time-step in which the car was at that location, and the direction of arrows represents the direction of movement for cars.

VI. Conclusions and Future Work

In this paper, we introduced a model checking based optimal multi valued LTL planning. The framework can be used to model uncertainty or incomplete knowledge about the environment, different access rights, etc. For representing uncertainty about events in a system modeled as a TS, we introduced multi-valued-set logics for reasoning over atomic proposition symbols. To facilitate the use of mvs-Logics over LTL specification formulas, we extend the semantics of LTL for using symbols that have ranges over set of values from TO quasi-Boolean lattices. Based on the aforementioned definitions, we defined an mvs-LTL planning automaton, and a monotonic cost function to find optimal plans.

In the future, we plan to improve the scalability of our framework in large scale multi-robot scenarios by utilizing algorithmic frameworks such as in [23].

References

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms and Implementations*. MIT Press, March 2005.

[2] 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.

[3] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimal multi-robot path planning with temporal logic constraints," in *IEEE/RSJ International Conference on Intelligent Robots and Systems,*, 2011, pp. 3087 –3092.

[4] H. Kress-Gazit and G. J. Pappas, "Automatically synthesizing a planning and control subsystem for the darpa urban challenge," in *IEEE International Conference on Automation Science and Engineering*, 2008.

[5] G. E. Fainekos, "Revising temporal logic specifications for motion planning," in *Proceedings of the IEEE Conference on Robotics and Automation*, May 2011.

[6] K. Kim, G. Fainekos, and S. Sankaranarayanan, "On the minimal revision problem of specification automata," *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1515–1535, 2015.

[7] J. Tumova, G. C. Hall, S. Karaman, E. Frazzoli, and D. Rus, "Least-violating control strategy synthesis with safety rules," in *16th international conference on Hybrid systems: computation and control*. ACM, 2013.

[8] 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.

[9] M. Lahijanian, S. A. D. Fried, L. E. Kavraki, and M. Y. Vardi, "This time the robot settles for a cost: a quantitative approach to temporal logic planning with partial satisfaction," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[10] G. Fainekos and H. G. Tanner, "Temporal logic control under incomplete or conflicting information," in *American Control Conference*, 2017.

[11] O. Kupferman and Y. Lustig, "Lattice automata," in *International Conference on Verification, Model Checking, and Abstract Interpretation*, ser. LNCS, vol. 4349. Springer, 2008, pp. 199–213.

[12] 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.

[13] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, pp. 1695–1708, 2011.

[14] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications," in *Lectures on Runtime Verification*. Springer, 2018, pp. 135–175.

[15] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," *Formal Methods in System Design*, vol. 51, no. 1, pp. 5–30, 2017.

[16] S. Jaksic, E. Bartocci, R. Grosu, and D. Nickovic, "Quantitative monitoring of stl with edit distance," in *Runtime Verification*, ser. LNCS. Springer, 2016.

[17] C. Baier, J.-P. Katoen, and K. G. Larsen, *Principles of model checking*. MIT press, 2008.

[18] A. Dokhanchi, B. Hoxha, and G. Fainekos, "Formal requirement debugging for testing and verification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 2, p. 34, 2018.

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

[20] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel, "Multi-valued symbolic model-checking," *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 4, pp. 1–38, Oct. 2004.

[21] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, Massachusetts: MIT Press, 1999.

[22] A. Ulusoy, S. L. Smith, and C. Belta, "Optimal multi-robot path planning with ltl constraints: guaranteeing correctness through synchronization," in *Distributed Autonomous Robotic Systems*. Springer, 2014, pp. 337–351.

[23] J. Yu and D. Rus, "An effective algorithmic framework for near optimal multi-robot path planning," in *Robotics Research*. Springer, 2018, pp. 495–511.