Motion Planning with Temporal Logic Specifications

Murat Can Üste
Technische Universität München
Email: can.ueste@tum.de

***Abstract*—The need for integrating higher level task specifications into the motion planning is increasing due to the growing number of robotics applications in different areas. In addition to the collision avoidance and dynamical feasibility, more sophisticated specifications such as coverage, sequencing, or temporal ordering of different higher level tasks are necessary in robotics applications in fields such as autonomous driving, navigation, search-and-rescue missions, manipulation, and medicine. Integrated task and motion planning poses unique computational challenges since it involves reasoning with both discrete abstractions and continuous motions. An expansive hybrid discrete/continuous space must be searched in order to find a motion that is collision-free, dynamically-feasible, and satisfies temporal goals. There has been a growing interest in motion planing with temporal logics as a way to address this challenge. In this seminar work, we introduce some of the relevant concepts, and review the recent progress in the field of motion planning with temporal logics.**

I. Introduction

Motion planning, whether it is for simple geometric setting as in piano mover's problem [1], or for more complex setting such as autonomous vehicles that has complex robot dynamics, has generally considered just the reachability problem of computing a trajectory to the goal region. However, as the application areas of robotics expanded into different fields and more complex environments, the need for specifiying higher level tasks and integrating them into motion planning has become important. For example in [2], Maierhofer et. al. formalizes the interstate traffic rules based on the German Road Traffic Regulastion, the Vienna Convention on Road Traffic, in combination with legal decisions from courts in order to make the application of traffic rules on autonomous vehicles easier. The higher level tasks doesn't need to be only related to the regulations, as they can be formalized in order to incorporate utility functions. In [3], Kundu et. al. tries to ensure that the robot never gets depleted with battery charge using the specifications provided with LTL formulas. Another example would be work of Wilde et. al. [4], which provides an interface for learning user-preferences for complext task specifications through human-robot interactions for robotics applications in known environments.

In order to integrate task specifications into motion planing, growing body of research focuses on formalising the requirements or tasks using temporal logics [5, 6]. However, motion planning algorithms that have been designed to address reachability problem, for example sampling-based methods such as PRM [7] or RRT [8], are not designed with the high-level specifications in mind, there are significant challenges for integrating these high-level specifications into motion planning problem. Many approaches focus on leveraging the model checking tools or algorithms (or their modified versions) for verifying whether the motion of the robot satisfies the given temporal logic specifications. Model checking is a formal verification method that has been heavily used for verifying hardware or software systems. Overview of the model checking process can be seen in Figure 1. In order the leverage the power of existing model checking tools and algorithms, the specifications must be formalized using temporal logics, and the system must be modelled to a discrete abstraction. Many approaches differ in how they formalize the requirements, how they model the motion of the robot and the environment, and which specific tools or algorithms from model checking they use, or how they extend/modify them.

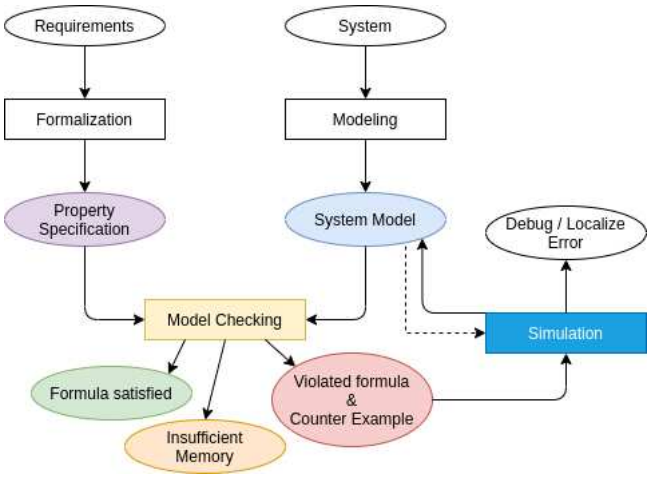


Fig. 1: Overview of Model Checking

One approach to this problem is the controller-synthesis from temporal logic specifications. This approach focuses on synthesizing controllers that satisfy given temporal logic specifications, where the behavior of the robot is captured by an automaton that is constructed using temporal logic specifications and the discrete abstraction techniques from model checking methods [9].

Controller synthesis methods usually obtain discrete abstractions of the system by decomposing the continuous state space

into polytopes. Afterwards, propositions can be defined over these polytopes, and discrete functions of the robot such as move left, right, etc. can be referred to using actions. These abstractions can then be used, along with the temporal logic specifications, as inputs for the model checking methods [10]. In these methods, the requirements, or task specifications then can be formalized using temporal logics such as LTL or CTL. Then, using these as inputs, model checking methods can be used to verify and compute a discrete path as a sequence of regions that the robot should move. However, it should be noted that in order for the constructed automaton to be viable, the controllers must be available for each action. These controllers should also be able to generate collision-free and dynamically-feasible motion.

The other approach is coupling of dicrete planning and sampling-based motion planning, where the focus is on motion planning with temporal logics for high-dimensional robotic systems with nonlinear dynamics. Due to high-dimensionality and nonlinearity, the combined task and motion planning is treated as a search over hybrid continuous/discrete space. In order to incorporate temporal logics, discrete layer is constructed using the simplified and discrete representation of the system as well as an automaton representing the temporal logic formula [9]. In this approach, the planning can be separated into two distinct layers: continuous layer and discrete layer. The state space, control inputs, as well as the robot dynamics are considered in the continuous layer. A motion tree planned by the discrete layer is being used as basis for the continous layer when searching the continuous space. The temporal task specifications can be incorporated into the discrete layer in the form of an automaton that combines with the workspace decomposition of the robot. For example, Bhatia et. al. proposed the multi-layered synergistic planning consisting of dicrete and continuous space in their works [11, 12].

In the following sections we will first introduce concepts that are relevant to the integrated task and motion planning research in II, such as robot motion planning in Section II-A, transition systems in Section II-B, and timed automatons in Section II-C. Afterwards, we will look into temporal logics in Section III, which will be followed by review of the some motion planning applications using temporal logics in Section IV. In SectionV, extensions of temporal logics and their applications are explored. Finally we will conclude in Section VI.

II. Preliminaries

A. Robot Motion Planning

The motion planning problem is defined as the task of finding a *collision-free* and *dynamically-feasible* trajectory from initial state to the goal state. An example motion planning task is illustrated in Figure 2. In the Figure, we can see a CommonRoad scenario [13], where the task is to find a *dynamically-feasible* and *collision-free* vehicle trajectory from the initial state to goal state.

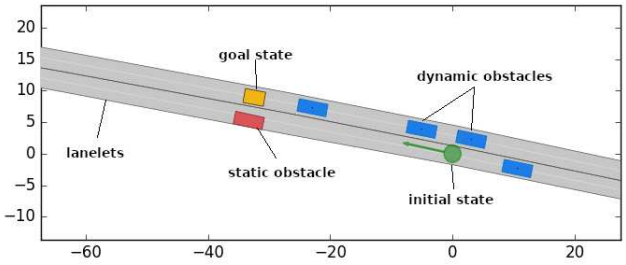


Fig. 2: A Commonroad scenario (DEU_Muc-2_1_T-1) as an example of a motion planning task. The task is to find a dynamially-feasible and collision-free vehicle trajectory from initial state to goal state.

The motion of a robot is formalized as

$$\zeta : [0, T] \to S$$

that satisfies an ordinary differential equation of the form

$$\dot{\zeta}(t) = f(\zeta(t), u(t)), \tag{1}$$

for some $u : [0, T] \to \mathcal{U}$, for all $t \in [0, T]$, where $S \subset \mathbb{R}^n$ is the state space and $\mathcal{U} \subset \mathbb{R}^m$ is the control space. The components of a state $s \in S$ depend on the model of the robot, but the common ones are position, orientation, velocity, steering angle. The values components of a state change with the motion of the robot. The control space defines the external inputs that are used to control the robot, such as accelaration and change in steering angle.

The motion planning problem poses unique computational challenges, especially in the case of complex robotic systems, such as autonomous vehicles. The challenge stems from the search over a vast discrete/continuous space while considering for complex geometries, motion dynamics which are nonlinear, and collision avoidance. In order to address this challenge, sampling-based motion planning methods have been used with great success [14, 15].

The sampling-based approaches often relax the completeness guarantees to probabilistic or resolution completeness, and optimality guarantee to guarantees on covergence towards optimal solutions. These sampling-based approaches typically expand a motion tree starting from the initial state and incrementally add collision-free and dynamically-feasible trajectories as branches, as illustrated in Figure 3.

Following the success of sampling-based motion planning approaches, incorporating temporal tasks into the discrete planning algorithms is seen as a promising way to combine task and motion planning. The temporal logic specifications can be integrated into the planning algorithm in the form of automaton in combination with the workspace decomposition obtained by the discretization of the continuous dynamics of the robot.

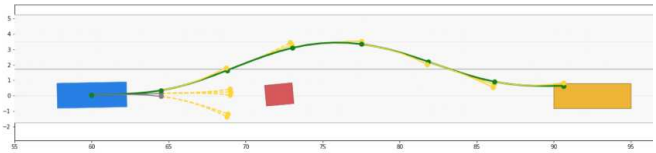


Fig. 3: Sampling-based methods can be used for searching vast hybrid continuous/discrete space for a collision-free and dynamically-feasible trajectory.

B. Transition Systems

One of the models used for representing the temporal logic specifications or the motion of the system is the *transition systems*. A *transition system* can be used to describe the potential behavior of discrete Systems. It consists of *states* and *transitions* between states, which might have labels from a set. In some cases, same label might appear on more than one transition. In the other hand, if the label set is a *singleton*, the system can be unlebeled (see Figure 4).

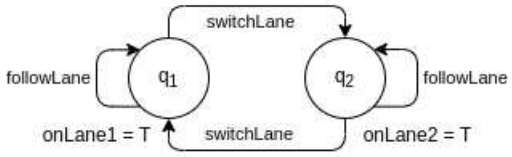


Fig. 4: Example transition system.

If the label set is singleton for the *transitions*, it can be considered as a *Kripke structure*, which is a slight variation of the *transition system*. A *Kripke structure* consists of a graph where the *nodes* represent the *reachable states*, *edges* represent the *state transitions*, and a *labeling function* that maps each node to a set of properties that hold in that state (see Figure 5).

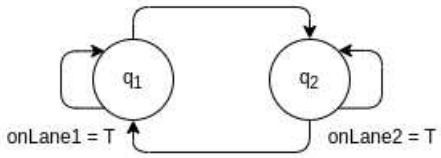


Fig. 5: Example Kripke structure.

C. Timed Automata

Another way of modeling of the continuous system seen in literature is the *timed automatons*. In timed automaton, finite automaton is extended with a finite set of real-valued clocks. The clock values increase with the same speed druing the run of the automaton. Additionally, guards are used for transitions where the clock values are compared to integers, and depending on the result, transitions are enabled or disabled. An example of a timed automaton can be seen in Figure 6

In many robotics applications, due to the real-time nature of the robotic system, timed-automatons can be a better model of the overall system. The reason is that in real-time systems, we are not just interested in the order of events, we are also interested in *when* they will happen. For example, in engineering applications such as railway systems, wrong timing can lead to catastrophic results, therefore modelling those systems with timed-automaton would be more fitting.

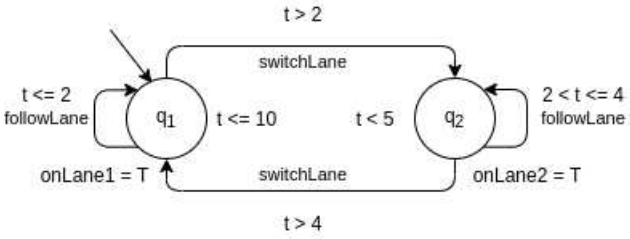


Fig. 6: Example timed automata.

III. Temporal Logics

Usually, the requirements or task specifications of a system is given in a form of regular text document. The problem with this kind of documentation is that it can be interpreted differently based on the readers prior technical or non-technical knowledge about the topic. In order to mitigate this ambiguity, we need to formalize these requirements or task specifications. Both in Model Checking related works, and motion planning related works, temporal logics have been used for the formalization process as they can express *coverage*, *sequencing*, and *temporal ordering* of tasks, which are mandatory for correct specifications.

The reason for not using propositional or predicate logic is that the formula is *statically* true or false in those logics. However, in robot motion planning, our system is *dynamic*, and we need to address this using temporal logics. In temporal logics, the models contain several states and a formula can be true in some states and false in others. Therefore, the *static* view of truth is replaced by a *dynamic* one, where the formulas may change their truth values as the system evolves from state to state.

There are various temporal logics that are proposed and used for different tasks. In general, however, we can separate them into two categories according to their view of 'time'. First category is the *Linear-time logics*, which think of time as set of paths where the path is a sequence of time instances as it is illustrated in Figure 7. The other category, which makes the non-deterministic nature of the future more explicit, is the *Branching-time logics*. In *Branching-time logics*, the time is represented as a tree, rooted at the present moment and branching out into the future as illustrated in Figure 7.

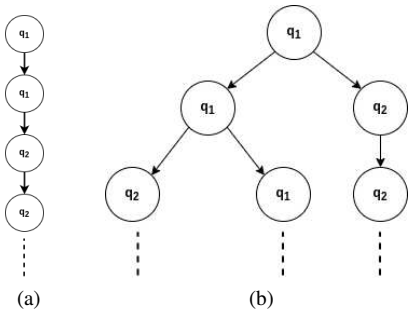


Fig. 7: Temporal logics can be categorized mainly by their view of time: a linear model of time *(a)*, and branching model of time *b*.

Different temporal logics have different expressive power as some expressions can't be defined in linear time models, but can be expressed in branching time models, and vice versa.

A. LTL — Linear-time Temporal Logic

Linear-time temporal logic (LTL) [16], is a temporal logic that allows us to refer to the future with connectives. The time is modeled as sequence of states that extend infinitely into the future. This sequence of states are also called paths (computation paths). Since the future is not determined, in general, several paths are considered in LTL. Each path represents different possible future, any of which might be the 'actual' path that is realised.

Given an atomic proposition p and a LTL formula ϕ, Backus-Naur Form of the a LTL formula is

$$\phi ::= p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid X(\phi) \mid \phi_1 U \phi_2 \tag{2}$$

where the *logical operators* are: $\wedge$ (and), $\vee$ (or), $\neg$ (not), and *temporal connectives* are: and X (ne**X**t), U or (**U**ntil).

- F (**F**uture or eventually)
- G (**G**lobally or always)
- W (**W**eak until)
- R (**R**elease)
- M (Strong release)

The expressive power of LTL can be shown using the following examples:

Coverage: 'inpect areas $A_1, \ldots, A_n$ in any order"

$$F\pi_{A_1} \wedge \cdots \wedge F\pi_{A_n} \tag{3}$$

Sequencing: 'inpect areas $A_1, \ldots, A_n$ in order"

$$F\pi_{A_1} \wedge (F\pi_{A_2} \wedge (F\pi_{A_3})) \tag{4}$$

Ordering: 'inpect areas A_1 and A_2 before A_3"

$$(\neg\pi_{A_3})\ U\ ((\pi_{A_1} \vee \pi_{A_2}) \wedge X\pi_{A_3}) \tag{5}$$

LTL implicitly quantifies over all paths by evaluating the formulas on paths, meaning that a state of a system satisfies an LTL formula if *all paths* from the given state satisfy it. Due to this implicit universal quantification, expressions which assert the *existence* of a path cannot be defined in LTL [16]. For example, LTL can't express 'the robot can stay idle on area A_3 after inpecting area A_3', which means following the state where the robot ispects A_3, there exists a path that the robot stays idle in A_3.

B. CTL — Computation Tree Logic

Computation Tree Logic, introduced by Clarke and Emerson in 1981 [17], follows the branching-time model, which models the time as a tree-like structure. In CTL, future is not deterministic, therefore in order to reason with the future it adds quantifiers on top of LTL temporal and logical operators.

Given an atomic proposition p, a state formula Φ, and a path formula ϕ. Backus-Naur Form of a CTL formula is

$$\Phi ::= p \mid \neg\Phi \mid \Phi \vee \Phi \mid E(\phi) \mid A(\phi) \tag{6}$$

Path formulas can be composed by state formulas as follows:

$$\phi ::= X(\Phi) \mid \Phi_1 U \Phi_2 \tag{7}$$

CTL uses the same logical and temporal operators from LTL, but with addition of existential and universal quantifiers:

- E (**E**xists)
- A (for **A**ll)

In order to define the temporal operators in CTL, we have to define them in pairs with either existential or universal quantifiers. The first element of the pair can be either A or E, where A means 'along all paths', and E means 'along at least one path'. The second element then can be one of the temporal operators used in LTL (X, U, or others).

Using CTL we can define expressions such as $AG(EF\phi)$, which means in all paths there exists a future state where formula ϕ is satisfied. However, since we can't alternate between temporal and path quantifiers in CTL, we can't define expressions such as $A(FG\phi)$, even though it is expressed easily in LTL as $FG\phi$.

C. CTL*

As mentioned in the previous sections, there are some expressions that can't be defined using LTL, but can be expressed in CTL, and vice versa. In addition to this, there also expressions that can't be defined using neither of the languages, such as $AG(EF\phi) \vee A(FG\phi)$.

In order to address this issue, and leverage the expressiveness of both LTL and CTL, CTL* has been proposed in [18]. CTL* is a logic which combines the expressive powers of LTL and CTL, by dropping the CTL constraint that every temporal operator has to be associated with a unique path quantifier. Therefore, CTL* is more expressive then both LTL and CTL since it can explicitly quantify over paths and allow selection of a range of paths by describing them with a formula as it is done in LTL. For example, in LTL we can say 'all paths which have a p along them also have a q along them', by

writing $Fp \rightarrow Fq$. It is not possible to write this in CTL as we need to pair every F with an A or E. Same expression would have a different meaning when these connectives are added: 'if all paths have a p along them, then all paths have a q along them.', $AFp \rightarrow AFq$

Given an atomic proposition p, a state formula Φ, and a path formula ϕ. Backus-Naur Form of a CTL* formula is

$$\Phi ::= p \mid \neg\Phi \mid \Phi \vee \Phi \mid E[\phi] \mid A[\phi] \tag{8}$$

$$\phi ::= X(\Phi) \mid \Phi_1 U \Phi_2 \tag{9}$$

CTL* is more expressive than LTL or CTL, but since it is computationally much more expensive to perform model checking using CTL* specifications, mostly CTL, LTL, and their derivatives are being used in applications. The choice between LTL or CTL depends on the application at hand, and the property that needs to be modelled. In general, LTL lacks CTL's ability to quantify over paths, while CTL lacks LTL's detailied ability to describe individual paths. However, LTL or temporal logics with linear time models are more straightforward to use, easier to understand, and models motion planning problems better since we are interested in construction a trajectory that is linear time in nature. As we will see in the following sections, this is one of the main reasons for using temporal logics with linear time properties in motion planning applications.

IV. Motion Planning with Sampling-based Methods and Temporal Logics

One of the main line of research in motion planning with temporal logic focuses on motion planning with temporal logics for high-dimensional robotic systems with nonlinear dynamics. Due to high-dimensionality and nonlinearity, the combined task and motion planning is treated as a search over hybrid continuous/discrete space. In order to incorporate temporal logics, discreate layer is constructed using the simplified and discrete representation of the system as well as an automaton representing the temporal logic formula. Different methods and architectures have been proposed for integrating the discrete layer into the continuous layer.

In [19], Karaman and Frazzoli have proposed the incremental model checking approach for integrating higher level task specifications. They proposed an extention to the RRT algorithm for constructing Kripke structure that models the system by labeling the states that have been sampled. The modified RRT algorithm, named *Rapidly-exploring Random Graph (RRG)*, is used for constructing a graph with cycles instead of a tree structure. In each iteration, this Kripke structure can be used for checking whether the specification formula is satisfied using algorithms such as Tarski-Knaster iteration [20]. Final trajectory that is created incrementally, using RRG algorithm and deterministic μ-calculus, will satisfy the given specifications (see Figure 8).

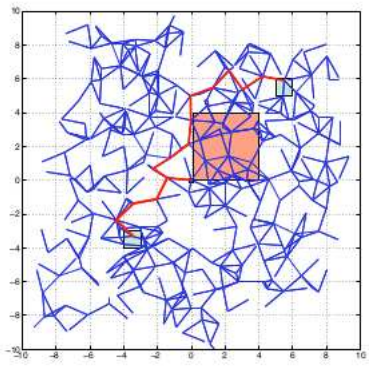


Fig. 8: Final trajectory that is created incrementally, using RRG algorithm and deterministic μ-calculus, will satisfy the given specifications, [19].

One of the main drawbacks of the work proposed in [19] is that the motion planning algorithm will not be reactive to the changes in the environment. A recent work by Vasile et. al. [21] builds on top of this approach, addressing the reactivity issue. They propose an online/offline approach, where a global Kripke structure is built for satisfying the global goals offline, and RRT-based algorithm is used for local planning in order to address online requests and locally sensed objects. For building the Kripke structure for global goals, they use RRG algorithm proposed in [19] by modifiying it for construction of a sparse graph in order to reduce the complexity, as the local changes would be addressed using the online algorithm.

In the offline planning, global specifications given as LTL specifications are translated to Büchi automaton. Afterwards, assuming partially known environment for the global goals, RRG-based algorithm is used for constructing a sparse Kripke structure. While incrementally constructing the Kripke structure, product automaton is updated and checked whether a trajectory exists satisfying global goals. For the online planning part, a modified RRT algorithm is used for constructing motion trees that satisfy the global goals as well as the online requests and locally sensed obstacles. Since the constructed motion tree paths are just checker for whether they violate the specifications, the performance of the algorithm is suitable for online planning. Additionally, they check whether the local motion tree connects back to the global transition system for satisfying global goals.

In Figure 9, an example disaster scenario can be seen, where the global transition system (in black) and the local transition system (in blue and red) defined over the goal regions A, B, and C. The robots current location is marked using the magenta disk, which is also the root node of the local transition system. Inside the local sensing area (cyan rectangle) a local (unsafe) obstacle that needs to be avoided, and a *fire* location that needs to be extinguished is detected. The red path on the local transition system indicates the path that satisfies both the local and global specifications, and connects back to the global transition system.

One possible drawback of this approach is the need for planning in highly dynamic environments where LTL specifications might not be expressive enough for timing requirements of the system, e.g. urban traffic scenarios. This approach can be extended by using a temporal language with time constraints such as MTL, however the complexity would increase for the verification parts of the algorithm.

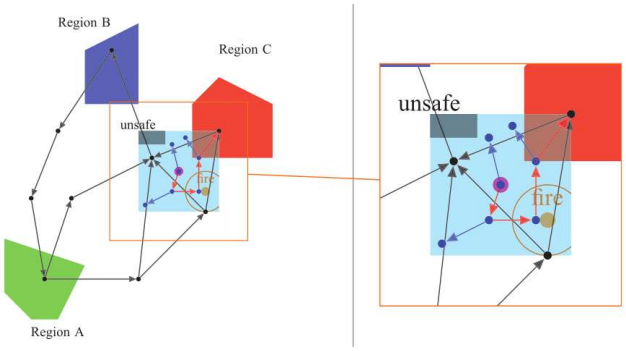


Fig. 9: A disaster scenario from. A global transition system for the global task specifications (in black) and a local transition system for reacting to online calls or detected obstacles (in blue), with the root node of it being the current robot location, can be seen in the figure. The red path on the local transition system indicates the path which satisfies bot global and local specifications. [21].

One other common approach is to have different layers for discrete planning and continuous motion planning, which work synergestically during planning to find a motion that is dynamically-feasible and collision free, and satisfies the task specifications. In this architectures, continuous and the discrete layer form a intertwined dependency where the plan is refined iteratively on both layers by receiving feedback from eachother. For example, In [22], Maly et. al. proposes an iterative temporal motion planning architecture for hybrid systems for partially known environment. In these approaches, different layers can be tweaked in a way to allow discrete layer to plan ahead as the complexity on this layer would be lower if the environment is decomposed into regions or polytopes. Then, the continuous layer planning can be guided using the discrete plan, and if no feasible plan is found, the feedback can be used on the discrete layer to refine the plan.

V. Extensions of Temporal Logics

Temporal logics have been used extensively both in motion planning with temporal specifications, and other fields. As a result, depending on the application, there have been many cases where a temporal logic has been extended to accomodate to the needs. In the following subsections, we will explore some of the extensions to the temporal logics.

A. Temporal Logics with Time Constraints

In real-time systems, whether it is controller synthesis methods or discrete planning methods for systems with high-dimensional dimensional dynamics, we are also interested in the 'timing' of the events in addition to the temporal order of them. Therefore, in order to specify these real-time properties, temporal logics have been extended.

First extension of temporal logics with time constraints is *Metric Temporal Logic (MTL)* [23]. MTL extends LTL by defining time constraints over the temporal operators. Since it assumes interleaving and fictious-clock absractions, it intuitively represents a set of points in time. It can be used for modeling task specifications together with timed automatons.

In [24] Zhou et. al. have used *Metric Interval Temporal Logic (MITL)*, which is a fragment of MTL. The only difference compared to MTL is that the constraints over the temporal operators has to be intervals instead of singletons. They first construct a timed automaton using cell decompositions of the environment, and using the time it takes to navigate between cells. Afterwards, the specifications given in MITL are translated into an *Input/Output Timed Automaton (IOTA)*. Finally, the IOTA from the MITL specifications and the Timed Automata of the environment is combined to construct a product IOTA, which will be used with UPPAAL model checking tool to generate a path from the outputs of the IOTA (Figure 10).

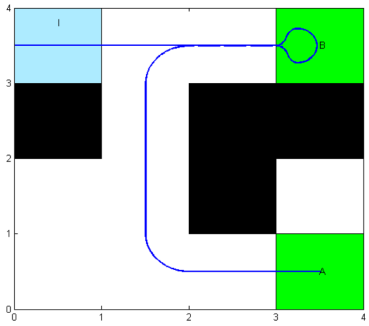
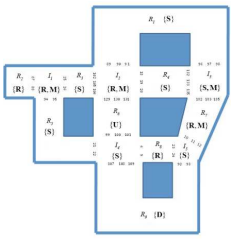


Fig. 10: Motion planning with MITL and Timed Automata model of the environment [24].

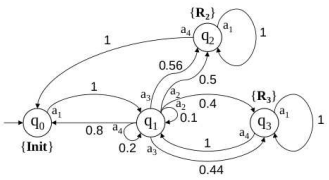
In order to model the real-time properties of CPS systems better *Signal Temporal Logic (STL)* proposed which extends MTL (and therefore LTL) by enabling defining numerical predicates over real-valued variables, instead of just integers as in the case of MTL. STL enables reasoning about real-time properties at the interface between components that exhibit both discrete and continuous dynamics [25]. STL have been mainly used hybrid systems that produce hybrid traces, and it can accomodate noise/approximations. However, since the complexity of STL is high, it is not being used for systems with high-dimensional nonlinear dynamics [26].

B. Temporal Logics with Probabilistic Quantification

Similar to the case of time constraints, real-time CPS systems might also need some probabilistic quantification over the specifications in order to model the sensors and actuators better. The probabilistic quantification extensions of



(a) Workspace decomposition



(b) MDP from temporal logic formula

Fig. 11: Motion planning with probabilistic robot actions using PCTL and Markov Decision Process [30]. In *(a)*, the environment is modeled using workspace decomposition method. In *(b)*, the actions of the robot is modeled using Markov Decision Process.

temporal logics were developed to address this need. Due to the nature of probabilistic quantification, these extensions focus on probabilistic quantification over paths of branching-time models as we can't model uncertainties in linear time models.

One of the first extensions of CTL is the *Probabilistic Temporal Logic (PTL)*, which allows defining predicates with probabilities. It was introduced in [27] with two different versions for finite models and infinite models. It allows reasoning about finite-state sequential probabilistic programs. It introduces probabilities over path quantifiers, but doesn't consider timing properties like we have seen in the case of MTL or STL. A slight variation of PTL was also introduced in [28] by Lehmann and Shelah, called *TC — Time and Chance Logic*.

Another extension of CTL is *Probabilistic Computation Tree Logic (PCTL)*, which builds on top of *PTL* [27] and TC [28] by extending them with time constraints as well [29]. PCTL allows definition of probabilities over predicates, e.g. "there is at least 98% probability that robot will inpect area A_3 successfully.".

In [30], Lahijanian et. al. uses PCTL specifications together with a Markov Decision Process model of the environment. In their work, they model the environment using workspace decomposition method, and assume the current evironment of the robot is known, however they model the robot such that the actions of the robot are probabilistic. The motion planning is performed by first constructing a Markov Decision Process from the temporal logic formula, and then running an algorithm inspired from PCTL model checking that maximizes the probability of satisfying the specificaion 11.

C. Temporal Logics with Quantitative Semantics

Another way of extending temporal logics is equipping them with quantitative semantics [31]. This extension replaces the binary satisfaction relation with the quantitative robustness degree function, while preserving the original syntax of the specification language. In essence, the robustness degree function gives a real value that indicates how far is a signal from satisfying or violating a specification. These extensions are generally used together with search methods [32], or with (reinforcement) learning methods as in [33], where Li et. al. have proposed a temporal logic with quantitative semantics called *Truncated Linear Temporal Logic (TLTL)* in order to use it as a reward function instead of heuristic reward functions for the reinforcement learning method. They have used *TLTL* for training a toast placing robot with reinforcement learning (Figure 12). As a result of using a temporal logic with quantitative semantics, they have mentioned that the model learned faster compared to using heuristic reward functions.

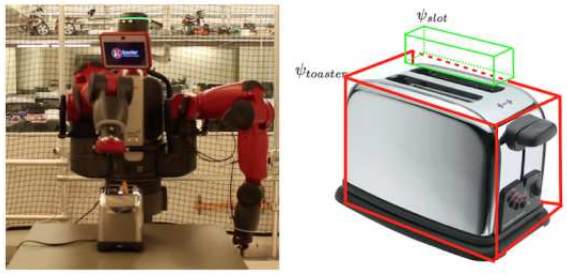


Fig. 12: Toast placing robot trained using temporal logic rewards [33].

D. Temporal Logics for Spatio-Temporal Behaviors

One of the main research areas in Cyber Physical Systems is the distibuted and connected systems. In these systems, the components are generally distributed across space, and they are connected via a communication infrastructure. This leads to emergent spatio-temporal behaviors being produced by these components, which is often impossible to predict at design time. Some of the examples include smart grids, robotic teams, or collections of genetically engineered living cells. In these applications, it might be hard to capture topological-spatial requirements using temporal logics. For example, the notion of being surrounded is not available in the standart temporal logics, and encoding them with specific functions may be hard [34].

There have been works focusing on spatio-temporal extensions of temporal logics, however many of them have high complexity and they are mostly used for monitoring applications [35]. There is less focus on motion planning for distributed and networked systems, one example being the work in [36]. However, as it can be seen from the Figure 13

Other spatio-temporal extenstions of temporal logics include *Signal Spatio Temporal Logic (SSTL)*, which extends STL with three spatial operators: *somewhere*, *everywhere*, and *surround* [37]. These operators can nested arbitrarily with the original STL temporal operator. SSTL can be interpreted as a *discrete model of space* represented as a *finite undirected graph* [38, 39]. It also provides a *metric structure to the space* by labeling each edge of the graph with a *positive weight* that can be used to represent the distance between two nodes. The weights of edges can also be used to encode other kinds of information, i.e. average travelling time between two cities.

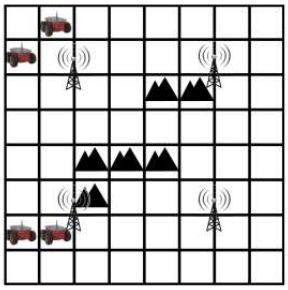


Fig. 13: Networked Multi-Agent Communication-Aware motion plannig [36].

In [35], Bartocci et. al. proposed *Spatio Temporal Reach and Escape Logic (STREL)* which generalises SSTL by introducing two new spatial operators: *reach* and *escape*. Using these operators, the same spatial modalities in SSTL can be expresssed, and monitoring procedure can be locally computed at each location using the information from neighbours. One other difference is that STREL can handle dynamic/mobile networks in addition to the static spatial-temporal models (position of the locations always fixed), unlike SSTL which can only operate the latter.

VI. Conclusion

We have reviewed previous and recent works for motion planning with temporal logics, and introduced some of the concepts used in these works. We have also looked into recent applications of motion planning with temporal logics. As we have seen, despite the significant progress made in the direction of integrating task specifications into motion planning, especially by leveraging model checking methods, there are still challanges remaining. Current applications can't still adress the computational complexity of the task specifications with timing constrains, which is necessary for high-level planning in highly dynamic environments. Especially in the case of systems with high-dimensional and nonlinear dynamics, there are still challenges remaining for successfull integration, as most of the current approaches address known or partially known environments with simpler robot dynamics.

References

[1] J. T. Schwartz and M. Sharir, "A survey of motion planning and related geometric algorithms," vol. 37, no. 1, pp. 157–169. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0004370288900537

[2] S. Maierhofer, A.-K. Rettinger, E. C. Mayer, and M. Althoff, "Formalization of Interstate Trafi¬c*RulesinTemporalLogic*," *p.* 8.

[3] T. Kundu and I. Saha, "Energy-Aware Temporal Logic Motion Planning for Mobile Robots," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8599–8605.

[4] N. Wilde, D. KuliÄ, and S. L. Smith, "Learning User Preferences in Robot Motion Planning Through Interaction," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 619–626.

[5] F. Bacchus and F. Kabanza, "Planning for temporally extended goals," vol. 22, no. 1, pp. 5–27. [Online]. Available: https://doi.org/10.1023/A:1018985923441

[6] M. C. Mayer, C. Limongelli, A. Orlandini, and V. Poggioni, "Linear temporal logic as an executable semantics for planning languages," vol. 16, no. 1, pp. 63–89. [Online]. Available: https://doi.org/10.1007/s10849-006-9022-1

[7] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," vol. 12, no. 4, pp. 566–580.

[8] S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning."

[9] E. Plaku and S. Karaman, "Motion planning with temporal-logic specifications: Progress and challenges," vol. 29, no. 1, pp. 151–162. [Online]. Available: https://content.iospress.com/articles/ai-communications/aic682

[10] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," vol. 45, no. 2, pp. 343–352. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000510980800455X

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

[12] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion Planning with Complex Goals," vol. 18, no. 3, pp. 55–64.

[13] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 719–726.

[14] J. Bruce and M. Veloso, "Real-Time Multi-Robot Motion Planning with Safe Dynamics," in *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, L. E. Parker, F. E. Schneider, and A. C. Schultz, Eds. Springer-Verlag, pp. 159–170. [Online]. Available: http://link.springer.com/10.1007/1-4020-3389-3_13

[15] J. J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue, "Dynamically-Stable Motion Planning for Humanoid Robots," vol. 12, no. 1, pp. 105–118. [Online]. Available: https://doi.org/10.1023/A:1013219111657

[16] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (Sfcs 1977)*, pp. 46–57.

[17] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Logics of Programs*, ser. Lecture Notes in Computer Science, D. Kozen, Ed. Springer-Verlag, vol. 131, pp. 52–71. [Online]. Available: http://link.springer.com/10.1007/BFb0025774

[18] E. A. Emerson and A. P. Sistla, "Deciding full branching time logic," vol. 61, no. 3, pp. 175–201. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019995884800479

[19] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic Î¼-calculus specifications," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) Held Jointly with 2009 28th Chinese Control Conference*, pp. 2222–2229.

[20] K. Schneider, *Verification of Reactive Systems: Formal Methods and Algorithms*, ser. Texts in Theoretical Computer Science. Springer-Verlag.

[21] C. I. Vasile, X. Li, and C. Belta, "Reactive sampling-based path planning with temporal logic specifications," vol. 39, no. 8, pp. 1002–1028. [Online]. Available: https://doi.org/10.1177/0278364920918919

[22] 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 *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '13. Association for Computing Machinery, pp. 353–362. [Online]. Available: https://doi.org/10.1145/2461328.2461380

[23] R. Koymans, "Specifying real-time properties with metric temporal logic," vol. 2, no. 4, pp. 255–299. [Online]. Available: https://doi.org/10.1007/BF01995674

[24] Y. Zhou, D. Maity, and J. S. Baras. Timed Automata Approach for Motion Planning Using Metric Interval Temporal Logic. Comment: Full Version for ECC 2016. [Online]. Available: http://arxiv.org/abs/1603.08246

[25] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *In: Proceedings of FORMATS-FTRTFT. Volume 3253 of LNCS*. Springer, pp. 152–166.

[26] S. Edelkamp, M. Lahijanian, D. Magazzeni, and E. Plaku, "Integrating Temporal Reasoning and Sampling-Based Motion Planning for Multi-goal Problems With Dynamics and Time Windows," vol. 3, no. 4, pp. 3473–3480.

[27] S. Hart and M. Sharir, "Probabilistic propositional temporal logics," vol. 70, no. 2, pp. 97–155. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019995886800018

[28] D. Lehmann and S. Shelah, "Reasoning with time and chance," vol. 53, no. 3, pp. 165–198. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019995882910221

[29] H. Hansson and B. Jonsson, "A Logic for Reasoning about Time and Reliability," vol. 6, pp. 102–111.

[30] M. Lahijanian, S. B. Andersson, and C. Belta, "Temporal Logic Motion Planning and Control With Probabilistic Satisfaction Guarantees," vol. 28, no. 2, pp. 396–409.

[31] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," vol. 410, no. 42, pp. 4262–4291. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397509004149

[32] A. Rizk, G. Batt, F. Fages, and S. Soliman, "On a Continuous Degree of Satisfaction of Temporal Logic Formulae with Applications to Systems Biology," in *Computational Methods in Systems Biology*, ser. Lecture Notes in Computer Science, M. Heiner and A. M. Uhrmacher, Eds. Springer Berlin Heidelberg, vol. 5307, pp. 251–268. [Online]. Available: http://link.springer.com/10.1007/978-3-540-88562-7_19

[33] X. Li, C.-I. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3834–3839. [Online]. Available: http://ieeexplore.ieee.org/document/8206234/

[34] I. Haghighi, A. Jones, Z. Kong, E. Bartocci, R. Gros, and C. Belta, "SpaTeL: A novel spatial-temporal logic and its applications to networked systems," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '15. Association for Computing Machinery, pp. 189–198. [Online]. Available: https://doi.org/10.1145/2728606.2728633

[35] E. Bartocci, L. Bortolussi, M. Loreti, and L. Nenzi, "Monitoring mobile and spatially distributed cyber-physical systems," in *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*, ser. MEMOCODE '17. Association for Computing Machinery, pp. 146–155. [Online]. Available: https://doi.org/10.1145/3127041.3127050

[36] Z. Liu, B. Wu, J. Dai, and H. Lin, "Distributed communication-aware motion planning for multi-agent systems from STL and SpaTeL specifications," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 4452–4457.

[37] L. Bortolussi and L. Nenzi, "Specifying and monitoring properties of stochastic spatio-temporal systems in signal temporal logic," in *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, ser. VALUETOOLS '14. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 66–73. [Online]. Available: https://doi.org/10.4108/icst.Valuetools.2014.258183

[38] E. Bartocci, L. Bortolussi, D. Milios, L. Nenzi, and G. Sanguinetti, "Studying Emergent Behaviours in Morphogenesis Using Signal Spatio-Temporal Logic," in *Hybrid Systems Biology*, ser. Lecture Notes in Computer Science, A. Abate and D. Å afrÃ¡nek, Eds. Springer International Publishing, vol. 9271, pp. 156–172. [Online]. Available: http://link.springer.com/10.1007/978-3-319-26916-0_9

[39] L. Nenzi, L. Bortolussi, V. Ciancia, M. Loreti, and M. Massink, "Qualitative and Quantitative Monitoring of Spatio-Temporal Properties," in *Runtime Verification*, ser. Lecture Notes in Computer Science, E. Bartocci and R. Majumdar, Eds. Springer International Publishing, vol. 9333, pp. 21–37. [Online]. Available: http://link.springer.com/10.1007/978-3-319-23820-3_2