LTL Semantics for Tumato: A Declarative Approach to Autonomous Agent Planning

Jan Vermaelen[1][0000−0002−7898−7859] and Tom Holvoet[1][0000−0003−1304−3467]

DistriNet, KU Leuven, 3001 Leuven, Belgium
{jan.vermaelen,tom.holvoet}@kuleuven.be

Abstract. This paper explores the semantics of Tumato 2.0, a constraint-based planning framework, through the lens of Linear Temporal Logic (LTL). Tumato enables the generation of policies for autonomous agents, ensuring safe and robust goal-oriented behavior. The framework guarantees that critical safety constraints hold across all potential outcomes of non-deterministic actions, while pre-computed policies eliminate the need for runtime decision-making. By translating Tumato's language constructs into LTL, we formalize its approach to handling safety, liveness, and robustness properties. This contribution offers a foundation for reliable agent behavior under real-world uncertainties, as well as improved interpretability.
We further demonstrate the semantics of Tumato's specification language through a case study, demonstrating how LTL guides system specification and supports potential formal verification efforts. These contributions align with key challenges in engineering intelligent and multi-agent systems, focusing on safety, correctness, and robust operation within complex environments. Overall, this work emphasizes the importance of declarative approaches in delivering reliable solutions for real-world applications.

Keywords: Tumato 2.0 · Linear Temporal Logic (LTL) · Autonomous Agents.

1 Introduction

Autonomous agents and multi-agent systems require rigorous planning frameworks to operate safely and effectively in complex, real-world environments. The Tumato framework, originally introduced by Hoang Tung Dinh et al. [3], is a constraint-based planning tool designed to generate safe, goal-oriented behavior policies. However, the uncertainty inherent in real-world conditions—such as unexpected obstacles and sensor and actuator inaccuracies—demands that an agent's planning framework be robust. By incorporating these non-deterministic action outcomes within its constraint-based structure, Tumato 2.0 [8] ensures that agents can reliably face these challenges, making it particularly relevant for the engineering of robust autonomous agents.

The approach adopted by Tumato—offline, complete policy generation—further enhances robustness by eliminating the need for real-time re-planning. In

safety-critical applications, where stable and reliable behavior is required, replanning under computational constraints can introduce risk and latency. By generating policies ahead of runtime, Tumato reduces computational demands during operation, enabling agents to execute predefined, guaranteed safe, and goal-oriented behavior. Furthermore, the planning approach generates complete policies, specifying which actions to execute in every potential state of the system. The approach is valuable for systems deployed in industries such as autonomous logistics, healthcare robotics, and field operations.

To fully value Tumato's capabilities, we analyze it through the lens of formal logic, which provides a structured, rigorous way to evaluate and reason about agent behavior. Tumato's declarative specification language enables users to specify desired outcomes and constraints without detailing the specific actions required to achieve them. A natural choice for such an analysis is Linear Temporal Logic (LTL), which facilitates reasoning about sequences of actions and states over time. LTL enables the specification of safety and liveness properties, providing a foundation for well-defined behaviors.

This paper formalizes Tumato's specification framework using LTL. We focus on the guarantees Tumato provides: safety (avoiding dangerous actions and states), robustness (handling non-determinism), and goal achievement (meeting specified objectives). While Tumato, the planning tool itself, is not formalized in this work, nor can it be fully expressed in LTL, this paper focuses on the semantics of Tumato's specification framework. The presented formalization improves the interpretability of Tumato's guarantees but does not aim to redefine or enhance the planning tool's underlying mechanisms. For a deeper exploration of Tumato's planning capabilities, we refer to [8]. By mapping Tumato's functionalities into LTL, we analyze its alignment with formal methods and assess its implications for reliable agent planning. To demonstrate its practical utility, we include a case study that represents Tumato's features as LTL expressions, in a robotic pick-and-place scenario. Tumato's declarative specification framework inherently supports multi-agent systems, as multi-agent interactions can be implemented in a state-action-based manner. This work does not delve into the modeling of multi-agent-specific features. Additionally, probabilistic approaches—which Tumato avoids by design—and detailed implementation aspects fall outside the scope of this work.

The remainder of this paper is structured as follows. Section 2 provides the background and related work, introducing LTL-based approaches as well as Tumato. Section 3 shows the relationship between Tumato specifications and LTL, offering insights into Tumato's approach. Section 4 presents the case study, followed by Section 5 discussing the findings. Finally, Section 6 concludes.

2 Background and Related Work

To ensure reliable and safe operation in autonomous systems, formal methods such as temporal logic are often essential for specifying and verifying system

properties. This section introduces LTL and explores its applications and adaptations in autonomous systems, before focusing on the Tumato framework.

2.1 Linear Temporal Logic (LTL)

Temporal logic, especially LTL, has become a natural choice for specifying (and verifying) properties on safety and liveness in autonomous and multi-agent systems [1].

LTL enables reasoning about sequences of states and events over paths using propositional variables (such as *at_charger* and *object_loaded*) which can be either *true* or *false*, logical operators ($\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$), and temporal modal operators (X, F, G, U, R). The temporal operators allow for describing the progression of states over time:

- *X (Next)*: Specifies that a condition will hold in the next state. For example, Xp means that p is true in the state immediately following the current one.
- *F (Eventually)*: Specifies that a condition will hold at some point in the future. For example, Fq means that q will become true at least once.
- *G (Globally)*: Specifies that a condition holds in all future states. For example, Gr means that r is true in every state.
- *U (Until)*: Specifies that one condition must hold at least until another becomes true. For example, $p\,U\,q$ means that p must hold continuously up to the point where q becomes true.
- *R (Release)*: Specifies that one condition releases another. For example, $p\,R\,q$ means that q must hold until and including the point where p becomes true. If p never becomes true, q must hold indefinitely.

As mentioned before, the suitability of looking at LTL to understand Tumato arises from its ability to capture conditions critical to safety, liveness, and robustness. Common properties in robotics, for example in a (mobile) pick and place application, include:

- **Safety**:
  - $G\neg u$ (with an unsafe condition u, which should never occur),
  - e.g. $G\neg$collision (a collision should never happen),
  - e.g. $G($loaded $\rightarrow \neg$pickup) (a pickup should never execute if the robot is already loaded).
- **Liveness**:
  - Ft (with single task formula t) or GFg (with repeating goal formula g),
  - e.g. $G(\neg$loaded $\rightarrow F$loaded) (if the robot is not loaded, it should be loaded in the future), and
    e.g. $G($loaded $\rightarrow F\neg$loaded) (if the robot is loaded, it should be unloaded in the future),
  - e.g. $G(F$loaded $\wedge F\neg$loaded) (true liveness).

  From a more practical (yet more procedural) viewpoint, one could also consider $G($at_delivery_station $\rightarrow ($loaded $\rightarrow X$deliver$))$. That is, if the robot is at the delivery station while loaded, it should deliver, next.

- **Robustness** (regarding the *failure* of actions):
  - of the form $v\,U\,w$ (with formula v the intended operation and formula w the desired effect),
  - e.g. $G(\text{pickup} \rightarrow \text{pickup}\,U\,\text{loaded})$ (pickup should be executed until successfully loaded).

Looking at related work in this context, LTL-based approaches have focused on managing conflicting objectives and handling constraints imposed by the environment, as in the work by Tumova et al. [6, 7], where methods are developed to minimize specification violations. This line of work emphasizes practical adaptability by finding the *least-violating* control strategy. In contrast, Tumato ensures absolute safety without compromise, opting instead to only generate behavior that succeeds in maintaining and restoring safety. If safety can not be guaranteed, it refrains from generating behavior.

Handling both high-level abstraction and continuous low-level observation and execution remains a universal challenge in autonomous control [4]. This challenge has been observed using Tumato as well, and is addressed by using *monitoring modules* that translate low-level observations into high-level states [8], although the details are beyond this paper's scope.

Various approaches include probabilistic aspects in agent planning, potentially combined with LTL. An example is a method to generate a control strategy that maximizes the probability of accomplishing a task given as an LTL formula [2]. Tumato, however, avoids probabilistic dependencies to mitigate the complexities of obtaining and maintaining reliable probabilities in dynamic environments. Instead, it assumes (biased) foreseeable, non-deterministic action outcomes, enabling robust planning while maintaining a declarative approach to safety.

LTL frameworks are often capable of real-time plan (re)calculation, for dynamic tasks or changing environments [11]. Tumato, however, generates complete, sound policies offline, enabling it to preemptively address safety constraints and remove the need for runtime recalculations. This pre-planned robustness ensures that no additional runtime intervention is needed beyond executing the existing policy.

LTL has been effectively applied to task assignment and planning in multi-robot systems, leveraging techniques like *lazy collision checking* to simplify the planning problem [5]. Although multi-agent applications are beyond this paper's immediate focus, Tumato's planning structure inherently supports multi-robot scenarios. It enables flexible, high-level task allocation and, when needed, communication among agents. Since planning is handled offline, the required computational power is readily available.

Other advancements have led to the development of HyperLTL [10], which extends LTL to express planning objectives like optimality, robustness, and privacy across multiple paths. Tumato's approach to non-determinism shares a related concept by ensuring safety across multiple possible outcomes for each action, illustrating a comparable need for handling relationships among multiple possible execution paths.

Overall, a range of LTL adaptations has been investigated before. In this work, we align Tumato with such efforts, seeking to provide a comprehensive understanding of the semantics within its specification framework.

2.2 Tumato

As introduced earlier, the Tumato 2.0 framework is a constraint-based planning tool designed to generate policies for autonomous systems, emphasizing safety, robustness, and goal-oriented behavior [8]. Tumato enables users to specify system behavior in a declarative manner, which is then automatically translated into sets of constraints. These constraints allow for the generation of a sound and complete policy that ensures safe and robust operation, even in realistic, non-deterministic environments. A Tumato-generated policy is a state-based behavior mapping, in which each state is associated with a set of actions to be executed. A recent empirical evaluation has demonstrated Tumato's effectiveness when compared to other, more ad-hoc approaches [9].

Tumato is not directly applicable to arbitrary problems in a fully observable non-deterministic (FOND) planning context, nor does it employ PDDL syntax. Instead, Tumato is a hands-on practical behavior planning tool that generates robust, constraint-based policies for autonomous agents. The focus of this work is on the formalization of its specification framework, rather than on Tumato itself.

Tumato's Specification Language Tumato employs a high-level declarative specification language for modeling autonomous systems. This language allows users to express system behavior in terms of states, actions, safety constraints, and goals. Below, we outline the core constructs of the Tumato language, along with examples to clarify their use.

States System states are defined using state variables, specifying all possible configurations of the system and its environment:

```
state <StateVarName> can be <StateValue> [, <StateValue> ...]
```

Here, *StateVarName* represents the name of the state property, and *StateValue*s define its discrete possible values. For example:

```
state location can be pickup, dropoff, corridor
```

This definition specifies that the robot's location can be one of three discrete values: `pickup`, `dropoff`, or `corridor`.

Actions Actions define the operations the system can perform and are specified as follows:

```
action <ActionName>
[duration: <Cost>]
[controlled resources: <ActionResource> [, <ActionResource> ...]]
preconditions: [none | <Predicate> [, <Predicate> ...]]
nominal effects: [none | <Predicate> [, <Predicate> ...]]
[alternative effects: [none | <Predicate> [, <Predicate> ...]]];
```

The key components of an action are:

- <**ActionName**>**:** The name of the action being defined.
- **Preconditions:** Conditions (written as predicates) that must hold before the action can be executed.
- **Nominal Effects:** Expected outcomes of the action under normal circumstances.
- **Alternative Effects:** (Zero or more) less desirable and less likely but possible deviations from the nominal effects.

Additionally, **controlled resources** can be specified, preventing two actions from executing simultaneously if they require the same resource. The optional **duration** parameter is used exclusively for prioritizing safety restoration actions: when safety must be restored, actions with shorter durations are prioritized. If safety is not violated, the duration parameter is ignored.

For example, a `pickup` action might be defined as:

```
action pickup
preconditions: location is pickup
nominal effects: object_status is loaded
alternative effects: object_status is free
```

This action requires the robot to be at the pickup location (`preconditions`) and results in the object being loaded (`nominal effects`) or failing to load (`alternative effects`).

Safety Rules Safety rules in Tumato ensure that the system maintains or restores safety. Rules are defined using the following syntax:

```
rule: <Predicate>
```

Tumato supports two types of rules:

- **Reaction Rules:** These enforce immediate actions or prohibit actions under specific conditions. They are expressed in the following form:

  ```
  rule: IF <condition> THEN [NOT] executing <action>
  ```

  This rule specifies reactive behavior triggered by the given condition.
- **State Rules:** These define conditions that must hold across all state-action outcomes. They are expressed in the following form:

  ```
  rule: IF <condition> THEN <other condition>
  ```

  State rules specify safety constraints that must hold in the next state, after the effects of the action have taken place.

Goals Goals specify the desired states or conditions the system must achieve. Tumato supports both unconditional and conditional goals:

```
goal: <goal condition> // unconditional
when <condition> then goal: <goal condition> // conditional
```

In the second case, the goal is pursued only while the condition holds.

Maximum Plan Length The maximum plan length is specified using:

```
max_plan_length: <value>
```

This parameter sets an upper bound on the number of steps considered during offline planning. By limiting the plan length, Tumato ensures computational feasibility while still focusing on generating effective plans.

Core Features of Tumato Tumato's offline policy generation process eliminates the need for real-time re-planning during execution. This precomputed approach guarantees stable and predictable behavior, which is especially important in safety-critical environments. As described, the system specifications are transformed into constraints that are solved by a constraint solver. This ensures:

- **Completeness:** Every possible state has an associated action or set of actions.
- **Soundness:** No unsafe (or resource-conflicting) actions are executed, adhering to all specified safety rules, while the goals are pursued.
- **Robustness:** The system accounts for non-deterministic outcomes by considering both nominal and alternative effects of actions.

Tumato's declarative approach simplifies system modeling by abstracting procedural details. Users can modify high-level constraints as needed, and Tumato automatically regenerates the corresponding policy. This flexibility makes Tumato suitable for evolving system requirements.

Another key strength is Tumato's handling of non-deterministic action outcomes. Actions are modeled with both nominal and alternative effects to account for possible deviations. For example, a robot tasked with moving to a location might encounter obstacles or actuator malfunctions, resulting in delays or failures. Tumato considers such foreseeable alternatives during planning, ensuring that the policy remains safe. The completeness of the policy ensures that, regardless of the outcomes of actions, the system can keep progressing toward its goals.

By combining offline policy generation, a declarative specification language, and a robust constraint-based backbone, Tumato provides a comprehensive solution for generating safe, goal-oriented policies for autonomous systems operating in uncertain environments.

Further details on Tumato's capabilities (and a case study) can be found in [8].

3 Mapping Tumato to LTL and CTL

We use LTL (and CTL) to formalize the semantics of Tumato's specification constructs. While these logics provide a way to reason about the behavior, they can not capture the decision-making process. They are suited for specifying properties over paths of states (and actions) during execution. By translating elements of a Tumato specification into LTL, we enable formal reasoning about the properties of the generated policy.

The completeness of Tumato's generated policy refers to the existence of a set of actions for every possible state—unless explicitly assumed otherwise in the specification. This completeness can be trivially verified since each state is guaranteed to have a corresponding state-action mapping generated by Tumato.

The specification of a maximum plan length in Tumato contains a parameter that guides the offline planning process, defining the execution horizon over which the planning system reasons. However, this does not constrain the infinite execution traces of the policy at runtime.

Using LTL formulas for state transitions, goals, and safety, Tumato policies can be verified to be sound, as claimed by the tool. The entries in a policy can also be represented using LTL as:

$$G(s \rightarrow a),$$

meaning that in a state s, action(s) a is (are) executed. In turn, potential state transitions resulting from actions can be expressed as

$$G((s_1 \wedge a) \rightarrow X(s_{2a1} \vee s_{2a2} \vee \ldots)),$$

where $s_{2a1}, s_{2a2}, \ldots$ represent all possible individual next states after executing action a in s_1. More details about the effects of actions can be found in Section 3.1.

3.1 Modeling Actions

In Tumato, actions are associated with preconditions and effects that can be mapped into LTL.

Preconditions Preconditions specify conditions that must hold before an action can be executed. If an action a_1 has preconditions $c_1, c_2, \ldots$, they can be represented in LTL as:

$$G(a_1 \rightarrow (c_1 \wedge c_2 \wedge \ldots)).$$

This ensures that action a_1 is only taken when its preconditions are satisfied.

Resource Constraints In Tumato, actions that use the same resources are mutually exclusive. LTL can also enforce that no two actions with overlapping resource needs execute simultaneously. For a pair of actions a_i and a_j (with $i \neq j$) that require the same resource, mutual exclusion is expressed as:

$$G(a_i \rightarrow \neg a_j), \quad \text{or equivalently} \quad G\neg(a_i \wedge a_j).$$

This guarantees that resources can be allocated without conflict.

Effects of Actions Effects describe the outcomes of executing an action. The resulting next state depends on the effects of the action(s)—executed in a specific state.

In an idealized setting (without external interference and no intermediary states), we assume that executing an action a leads directly to its nominal effect $\text{effect}_{\text{nom}}$ in the next state. This idealized effect can be expressed in LTL as:

$$G(a \rightarrow X(\text{effect}_{\text{nom}})).$$

In realistic settings, action outcomes are often non-deterministic, meaning that an action a may lead to one of several possible effects $\text{effect}_{\text{nom}}$, $\text{effect}_{\text{alt1}}$, $\text{effect}_{\text{alt2}}$, ... To model this uncertainty, we can express that one of these effects will eventually occur if the action is executed:

$$G(a \rightarrow F(\text{effect}_{\text{nom}} \vee \text{effect}_{\text{alt1}} \vee \text{effect}_{\text{alt2}} \vee \ldots)).$$

However, this formulation assumes that the effects occur regardless of whether the action continues to be executed, which may not always be accurate.

A more realistic formulation incorporates the condition that the effects only occur as long as the action a is being executed. This relationship can be captured as:

$$G(a \rightarrow (a \; U \; (\text{effect}_{\text{nom}} \vee \text{effect}_{\text{alt1}} \vee \text{effect}_{\text{alt2}} \vee \ldots \vee \neg a))).$$

This formula states that while a is being executed, it must continue until one of its effects occurs, unless a is stopped.

Further, Tumato addresses the frame problem implicitly. While we do not represent this in LTL, it is assumed that any state variables not affected by an action's outcome remain unchanged. In Tumato, all relevant variables are updated explicitly in the effects of actions, and those not mentioned are understood to persist by default.

If multiple actions are executed simultaneously, Tumato treats each action's effects as a distinct potential outcome.[1] At runtime, the effects of one action occur first, followed by an evaluation of the new state and corresponding actions. When modeling in LTL, simultaneous actions can be modeled by a composite action with all possible effects. However, in Tumato we should keep actions separate to maintain the notion of bias toward nominal effects.

[1] If actions directly interfere with each other, they should be modeled with shared resources to prevent conflicts.

Indeed, Tumato introduces a bias toward nominal outcomes, which represent the expected result of actions, while alternatives model less likely deviations. This bias is not probabilistic but semantic: Tumato assumes that if an action is executed repeatedly, its nominal effect will occur. In logical terms, this reflects a fairness assumption—namely, that the system is fair with respect to nominal outcomes and does not indefinitely fail. The bias can not be expressed directly in LTL.

3.2 Modeling Goals

Liveness properties in Tumato ensure that certain desirable states or conditions will eventually be reached or maintained:

- **Basic Goals**: The simplest form of a goal in Tumato can be interpreted as GFg, meaning that a goal condition g will eventually be reached and reoccur (or be maintained) indefinitely. This is useful for specifying tasks that should continue to be achieved, unconditionally.
  Multiple goals can be conjoined using Tumato's *constraint* goal option. The conjunction of all goals g_i for $i = 1, 2, \ldots, n$ is represented as $g \leftrightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_n)$.
- **Conditional and Prioritized Goals**: Tumato enables more complex goal structures, where goals are prioritized and active only under certain conditions. These prioritized conditional goals can be represented in LTL as follows:

$$G(c_1 \rightarrow (c_1 U (g_1 \vee \neg c_1)))$$

$$G((c_2 \wedge \neg c_1) \rightarrow ((c_2 \wedge \neg c_1) U (g_2 \vee \neg c_2)))$$

$$\ldots$$

$$G((c_n \wedge \neg c_1 \wedge \neg c_2 \wedge \ldots \wedge \neg c_{n-1}) \rightarrow$$

$$((c_n \wedge \neg c_1 \wedge \neg c_2 \wedge \ldots \wedge \neg c_{n-1}) U (g_n \vee \neg c_n))).$$

  These formulas ensure that a goal g_i is pursued when its corresponding condition c_i holds, provided no higher-priority goals (depending on their order) are active under their respective conditions. This ensures that goals are pursued in priority order when their conditions are met.

Under the fairness assumption (with respect to nominal outcomes), Tumato's planner synthesizes strong cyclic plans. Such plans guarantee that the goals will eventually be reached from all states. The argument can be made by contradiction: assume there exists an infinite execution where the goal is never reached. Consider the closest state to the goal—according to some distance metric—that is revisited infinitely often without progressing toward the goal. The planner associates each state with an action whose nominal effect moves closer to the goal. Therefore, the only way to never reach the goal is if the nominal effects never occur, which contradicts the fairness assumption. It follows that, under fairness, the generated policy ensures goal reachability.

3.3 Handling Safety

Safety constraints in Tumato take the form of either reaction rules or state rules:

- **Reaction Rules**: These are typically expressed as $G(c \rightarrow a)$, where c is a condition under which action a should (or should not: $G(c \rightarrow \neg a)$) be executed. For example, if a robot detects an obstacle, it should execute a braking action.
- **State Rules**: State rules push the constraint to *the next state* and are used to specify conditions that must hold after state transitions. State rules can be interpreted as GXb in LTL, where b represents a safe condition. Whichever actions are executed, they should always lead to a state where a safety constraint b holds.

When including potential alternative outcomes of actions, LTL theoretically falls short, as it can not reason about different possible futures or guarantee safety across all outcomes. Branching-time logic, such as CTL, may be more suitable, as it can express properties over different possible execution paths. For example, $A(GXb)$ specifies that b must always hold for all possible next outcomes, ensuring safety across all paths. While LTL cannot fully express Tumato's bias toward nominal outcomes or its handling of safety regarding potential alternative outcomes, it does still provide a foundational framework for reasoning about Tumato's (safety) semantics.

4 Case Study: a Mobile Pick-and-place Robot

In this case study, we examine a pick-and-place robot tasked with moving objects from a pickup location to a drop-off location. The robot operates in a well-defined environment with a corridor between the two locations, and it must adhere to specific safety and liveness requirements. This example highlights Tumato's semantics, expressed in LTL, and its approach to generating robust policies. These policies ensure safety across all potential action outcomes while achieving liveness goals.

4.1 Model in LTL

To express this system in LTL, we define atomic propositions representing actions and states:

Actions:

- *pickup*: object pickup action (precondition: at pickup location),
- $drop\_off$: object unload action (precondition: at drop-off location),
- $move\_to\_a$: move to pickup location,
- $move\_to\_b$: move to drop-off location,
- *secure*: secure object if one is present on the robot,
- *release*: release (or *unsecure*) the secured object, if any.

States:

- a: robot is at the pickup location,
- b: robot is at the drop-off location,
- c: robot is in between locations, in the corridor,
- obj: an object is present on the robot,
- sec: the object on the robot is secured for transport,
- bat: the battery level is in range for normal operation.

Please note that a, b, and c are mutually exclusive. The robot will always be at one (and only one) of those locations.

These propositions enable us to formalize both liveness properties—ensuring transport goals are reached—and safety properties—ensuring objects are secured before transport.

The robot's states and actions (state transitions) can be represented in an automaton, see Figure 1, including nominal and alternative outcomes. The transitions show the non-deterministic nature of actions, where the robot cannot predict which outcome will occur at runtime but ensures safety regardless of the result. Battery levels have been omitted in this overview for readability. Please note, the action *move_to_b* (moving to the drop-off location) could be executed after *pickup* and before *secure* as moving has no preconditions. Since (one outcome of) this action is not safe—see later in this section—it has not been included in the figure. Similarly, moving to a workstation without (un)loading first does not progress the system and is not included—although again, possible.

For goal-oriented planning, Tumato assumes nominal outcomes of actions over the far less likely and less desirable alternative ones. When guaranteeing safety, however, also the alternative outcomes are considered. Tumato's policies ensure that actions exist for all states, enabling continuous operation without the need for runtime decision-making.

Action Preconditions For some actions, certain preconditions must be satisfied to ensure appropriate execution. For instance, the robot must be at the pickup location before it can execute the *pickup* action. In LTL, we can enforce this precondition by requiring that the action only occurs when the robot is indeed at the pickup location:

$$G(pickup \rightarrow a)$$

This formula states that *pickup* implies a (the robot being at the pickup location), ensuring that the action *pickup* can only occur when a is *true*.

Robust Operation To ensure robust operation, the robot must handle potential failures, requiring retries of actions when necessary. For example, if the robot attempts to pick up an object, the object may occasionally fall back down. To handle such failures, one can include robustness in that the robot will persist in attempting the pickup action until it succeeds.

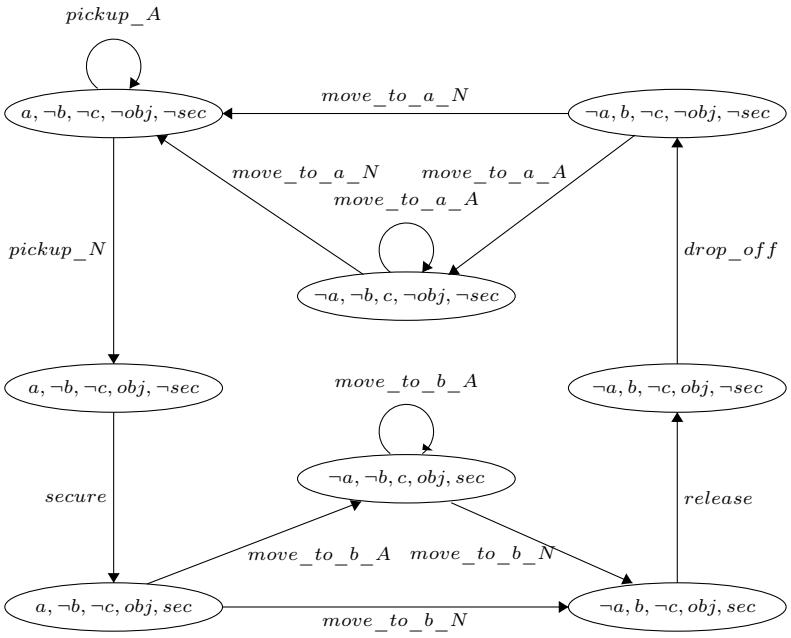

Fig. 1. An automaton showing the state transitions, visualizing both nominal and alternative effects of actions as $<action>\_N$ and $<action>\_A$ respectively

Using LTL, we define this as follows:

$$pickup\,U\,obj$$

This expression denotes that the robot will continue attempting the pickup action *pickup* until the object is successfully loaded onto the robot, indicated by *obj*.

Tumato's policy structure inherently guarantees robustness by ensuring that each state has associated actions. This ensures the system can retry failed actions or correct deviations caused by unintended action outcomes, without additional runtime decision-making.

Liveness Properties The robot is expected to continuously fulfill its objective of picking up and delivering objects, represented in LTL by goals that must be repeatedly achieved.

General Liveness of Pickup and Delivery: $G(Fpickup)$ denotes that the robot will always eventually pick up an object. Similarly, $G(Fdrop\_off)$ denotes that the robot will always eventually deliver an object.

Refined Liveness Properties: Using conditions to define when specific goals must be met:

– If the robot is not currently carrying an object, it should obtain an object. We use "$condition\,U\,goal$" rather than "$F\,goal$" to emphasize a required continuous condition:

$$G(\neg obj \rightarrow (\neg obj\,U\,obj)).$$

– If the robot has an object loaded, it should eventually unload it:

$$G(obj \rightarrow (obj\,U\,\neg obj)).$$

After also including the battery level:

$$G((\neg obj \wedge bat) \rightarrow ((\neg obj \wedge bat)\,U\,(obj \vee \neg bat))),$$
$$G((obj \wedge bat) \rightarrow ((obj \wedge bat)\,U\,(\neg obj \vee \neg bat))),$$
$$G(\neg bat \rightarrow (\neg bat\,U\,bat)).$$

These formulas ensure the robot continuously attempts to pick up and deliver objects while prioritizing battery constraints and hence charging.

Safety Properties The safety requirements aim to prevent the robot from entering unsafe states, such as moving (in the corridor) with an unsecured object. Most practically, with a procedural mindset, we write:

$$G(obj \rightarrow ((secure \vee sec)\,U\,drop\_off))$$

where we have to assume that securing happens instantly since moving is not prohibited.

In practice, however, actions do not effectuate instantly. Furthermore, unlike in this *secure* example, action outcomes are seldom purely deterministic, making it necessary to include robustness to account for these potential failures or delays.

To more directly address this safety property, we can require the robot to explicitly (successfully) secure any object before movement is permitted. This can be captured as:

$$G((obj \wedge \neg sec) \rightarrow X(sec\,R\,\neg c)).$$

Or more generally applicable, reflecting Tumato's state rules, if an object is present but not secured, the robot is not allowed to be in the corridor:

$$G(X((obj \wedge \neg sec) \rightarrow \neg c))$$

4.2 Enabling Policy Verification

The derived LTL formulas provide the necessary elements to formally reason about state transitions, goal satisfaction, and safety constraints in Tumato's policies. In principle, model-checking techniques could be employed to verify that Tumato's policies comply with specified safety and liveness properties.

However, explicit verification is unnecessary as Tumato's constraint-solving approach inherently guarantees compliance with these properties by construction. The policy generation process ensures:

- All specified constraints are enforced, ensuring soundness.
- All reachable states have corresponding actions, ensuring completeness.
- Robustness is achieved through precomputed policies that account for non-deterministic outcomes.

Nonetheless, the formalization provided here enables verification if needed for specific cases or additional validation.

By addressing safety and liveness requirements while accounting for non-deterministic outcomes, Tumato effectively tackles key challenges in generating reliable policies for autonomous systems. These contributions also align with broader challenges in multi-agent engineering, particularly for systems deployed under real-world uncertainties.

5 Discussion

The mapping of Tumato's specifications into LTL highlights its contribution to generating sound and robust policies for autonomous systems. This formalization reveals how Tumato aligns with key safety and liveness properties while managing uncertainties, offering valuable insights for engineering intelligent agents.

Tumato's policy generation relies on constraint-solving to meet safety and goal requirements by construction. It ensures policies are sound, adhering to safety rules, and complete, covering all possible states. This inherent reliability guarantees that the generated policies align with specifications whose semantics can be appreciated through LTL, eliminating the need for (runtime) verification. The approach is particularly effective in environments where action outcomes are non-deterministic.

The offline nature of Tumato's policy generation eliminates the need for dynamic, on-the-fly planning. By pre-computing policies that address all reachable states, all contingencies are accounted for. This avoids reliance on ad-hoc re-planning, which poses challenges in real-world environments and distributed settings. As such, Tumato is particularly advantageous for applications where safety guarantees are critical, such as autonomous robotics.

The alignment of Tumato's policies with the safety and liveness properties expressed in LTL is observed. Safety properties, such as ensuring that objects are secured before transport, map directly to constraints that must always hold. Similarly, liveness properties, such as continuous pickup and delivery of objects, are represented as recurring goals within Tumato. This connection underscores Tumato's suitability for systems requiring operational guarantees.

LTL provides a solid foundation for specifying safety and liveness properties. In this work, it was used to provide the semantics of Tumato's specification constructs. However, LTL can not represent the explicit accounting for alternative effects of actions in terms of safety nor the bias toward nominal outcomes. Also, the approach of restoring safety the most preferred way, based on a *duration* is outside LTL's scope. These discrepancies are due to Tumato's origin of addressing practical needs in engineering (robotic) agents. Rather than focusing on LTL's expressiveness, Tumato employs a constraint-solving mechanism

that directly integrates robustness into policy generation, ensuring safety even when actions have multiple possible outcomes. The balance between formal logic and practical needs sets Tumato apart from prior works, which often prioritize theoretical guarantees over real-world applicability.

Tumato also differs from approaches based on probabilistic models, which rely on accurate probabilistic information to guide decision-making. While these models are mathematically powerful, they are challenging to apply in uncertain environments where probabilities are difficult to estimate. By avoiding such dependencies, Tumato ensures greater reliability in dynamic and non-deterministic scenarios.

6 Conclusion

This work formalizes the semantics of Tumato's constraint-based planning framework using LTL. We investigated the specification used to generate sound, complete, and robust policies for autonomous systems by satisfying safety and liveness requirements. The obtained semantics form a structured way to interpret Tumato's specifications and understand its guarantees, particularly in non-deterministic and safety-critical settings.

Tumato's focus on nominal outcomes, while accounting for other contingencies, provides a practical solution for real-world applications. Although LTL cannot fully capture every aspect of Tumato's safety handling, it provides a valuable framework for understanding specifications and verifying policies. By combining explicit robustness with pre-computed, complete policies that eliminate runtime checks, Tumato effectively balances theoretical rigor with practical applicability, making it highly effective for engineering safe and robust autonomous (robotic) systems.

Future work should investigate Tumato's capabilities regarding dynamic goal assignment and explicit support for multi-agent collaboration, which would enhance its utility in evolving tasks and cooperative planning. Additionally, the use of machine learning techniques could be explored to assist users in adequately representing real-world systems without relying on probabilistic models. Furthermore, prioritizing safety rules could enable Tumato to weigh more critical constraints more heavily when resolving unsafe situations. Finally, exploring the possibility of temporarily allowing (transient) *less safe* states before fully restoring safety could expand Tumato's flexibility in highly constrained environments. This trade-off must be carefully examined to ensure alignment with safety-critical requirements. Such advancements would broaden Tumato's applicability and contribute to ongoing efforts to engineer intelligent agents capable of operating in complex, real-world scenarios.

Acknowledgments. This research is partially funded by the Research Fund KU Leuven.

Throughout this work, the authors used *Grammarly* and, to a lesser extent, *ChatGPT* for grammar and readability improvements. All edits were reviewed and refined by the authors, who take full responsibility for the publication's content.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
2. Ding, X.C.D., Smith, S.L., Belta, C., Rus, D.: LTL control in uncertain environments with probabilistic satisfaction guarantees. IFAC Proceedings Volumes **44**(1), 3515–3520 (2011)
3. Dinh, H.T., Cruz Torres, M.H., Holvoet, T.: Sound and complete reactive UAV behavior using constraint programming (2017), https://lirias.kuleuven.be/retrieve/470086
4. Fainekos, G.E., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for mobile robots. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation. pp. 2020–2025. IEEE (2005)
5. Luo, X., Zavlanos, M.M.: Temporal logic task allocation in heterogeneous multi-robot systems. IEEE Transactions on Robotics **38**(6), 3602–3621 (2022)
6. Tumova, J., Castro, L.I.R., Karaman, S., Frazzoli, E., Rus, D.: Minimum-violation LTL planning with conflicting specifications. In: 2013 American Control Conference. pp. 200–205. IEEE (2013)
7. Tumova, J., Karaman, S., Belta, C., Rus, D.: Least-violating planning in road networks from temporal logic specifications. In: 2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS). pp. 1–9. IEEE (2016)
8. Vermaelen, J., Holvoet, T.: Tumato 2.0-a constraint-based planning approach for safe and robust robot behavior. Annals of Mathematics and Artificial Intelligence pp. 1–27 (2024)
9. Vermaelen, J., Holvoet, T.: An empirical evaluation of a formal approach versus ad hoc implementations in robot behavior planning. Science of Computer Programming **241**, 103226 (2025)
10. Wang, Y., Nalluri, S., Pajic, M.: Hyperproperties for robotics: Planning via HyperLTL. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). pp. 8462–8468. IEEE (2020)
11. Xu, N., Li, J., Niu, Y., Shen, L.: An LTL-based motion and action dynamic planning method for autonomous robot. IFAC-PapersOnLine **49**(5), 91–96 (2016)