Teaming Linear Temporal Logic: Coordinated Behavior Specification in Heterogeneous Multi-Agent Systems

Thomas Fawcett
Computer Science Department
Brown University
Providence, Rhode Island 02912
Email: thomas_fawcett@brown.edu

George Konidaris
Computer Science Department
Brown University
Providence, Rhode Island 02912
Email: gdk@brown.edu

***Abstract*—Linear Temporal Logic (LTL) is a formal language that can be used to specify robot behaviors and goal states. We extend LTL to enable the specification of complex cooperative behaviors in a multi-agent system (MAS). The extended language, TeamingLTL, models features commonly present in MAS such as partial observability and cooperation between agents. This enables a user to specify behaviors or goals while deferring the realization of those states to a separate, automated task planning and allocation system. We show how to specify the behavior of a MAS using TeamingLTL and use this formulation in program verification and the creation of correct-by-design controllers.**

I. Introduction

Effective control of multi-agent systems (MAS) is an open problem [6]. The number of states in a MAS can be large, so simplifications and abstractions are often used. Much focus has been placed on swarms that are either homogeneous in nature or exhibit few variations within the population. This often results in control schemes wherein all agents are given the same control logic, such as Boids [28]. Other control schemes involve explicitly constructed or automatically organized hierarchies, where a few individuals make decisions driving the overall system (i.e. sparse control [26]). These, and other proposed control schemes, effectively sacrifice generality in favor of tractability. Additionally, these approaches are imperative, and the implementer/operator must define how the system will behave in any given (foreseeable) situation.

As an alternative to these imperative control schemes, temporal logics provide a means to define a family of acceptable behaviors for a system. Notably, the formal language Linear Temporal Logic [20, 27] (LTL) can specify correct and incorrect states, represented as simple boolean variables, over time. This information can be compared to actual system behavior to verify that the system behaves correctly (program verification) or used to directly construct a controller that will cause the system to behave correctly (reactive synthesis). LTL and its variants have been used successfully to specify tasks for individual robots [9, 21, 23] and swarms [14, 25, 7], and multi-robot systems without teaming [5].

Despite the flexibility that LTL provides for specifying correct behavior, it has not been used to control teaming in large heterogeneous multi-agent systems. This can be attributed to LTL's roots in formal program verification, wherein a system is modeled as a monolithic, fully-observable entity. A MAS in the real world is unlikely to possess these attributes. Instead, it is often characterized by local sensing leading to local information that must be explicitly shared. It may also exhibit redundancy where multiple agents are capable of equivalently satisfying a particular element of the overall formula. The tasking assigned to a MAS may also exceed the capabilities of any single agent, and multiple agents must cooperate to achieve the goal. Fully defining the operation of a system with all of these properties via standard LTL is possible, but may require extensive work on the part of the operator to track all of the relevant agents and states and ensure that the generated formula is consistent with both the capabilities and intended behavior of the system.

To address these limitations, we propose an extension of LTL with system partitioning semantics reflecting the partial observability, finite bandwidth, and combination of redundancy and diversity that may be found in a heterogeneous MAS. First, we will restrict ourselves to truth states that are grounded in the individual local sensing capabilities of the individual agents. Second, the designer of the behavior specification does not need to explicitly allocate agents to tasks. The problem of task allocation within the MAS must be handled automatically. Finally, the language must express the behavioral requirements in a way that can be scaled to a larger MAS without requiring a modification to the specification itself.

With these objectives in mind, we present TeamingLTL, an extension to LTL for specifying behavior in Multi-Agent Systems. We show how a TeamingLTL formula can be algorithmically decomposed into a standard (but more verbose) LTL formula while retaining the teaming requirements expressed in the original formula. We show how Teaming LTL can be used to verify the collective behavior of a MAS. Finally, we demonstrate the creation of correct-by-design controllers, further exploiting the teaming structures embedded in the TeamingLTL formula. We do this in several domains including

shift coverage, emergency services dispatch, and distributed recycling systems in both simulation and on physical robots.

II. Background

A. Standard LTL semantics

LTL is a formalism for specifying correct behavior of concurrent programs. It imposes constraints on system states as the system progresses from time step to time step. Give a set of atomic propositions (AP) describing the state of the system as boolean values at each time step, LTL has the following basic syntax:

$$\varphi := a \,|\, \neg\varphi \,|\, \varphi_1 \vee \varphi_2 \,|\, \bigcirc \varphi \,|\, \varphi_1 \mathcal{U} \varphi_2 \,.$$

Here, $a \in AP$ is a propositional variable which maps some aspect of system state to a Boolean value. The logical operators $\neg$ and $\vee$ indicate negation and disjunction, respectively. The unary temporal operator $\bigcirc$ means that formula φ must be true at the next time step. The binary temporal operator $\mathcal{U}$ means that φ_1 must be true until φ_2 becomes true.

From these basic operators, it is possible to define constants for true ($\top$) and false ($\bot$) as well as logical operators for conjunction ($\wedge$), implication ($\rightarrow$), and bicondition ($\leftrightarrow$). Unary temporal operators eventually ($\Diamond$) and always ($\Box$), along with binary temporal operators for release ($\mathcal{R}$), weak release ($\mathcal{W}$), and strong release ($\mathcal{M}$), can also be defined for convenience.

The AP and logical operators specify which states must be true, must be false, or are unspecified at a given time step. The temporal operators control how these requirements change over time. Consider the formula $a\mathcal{U} \bigcirc b$. This expresses the requirement that a must be true at least up to the point where b will true two time steps in the future, as illustrated in Figure 1.

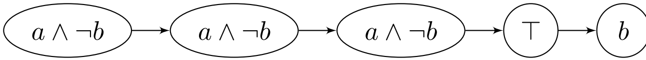


Fig. 1: Example state sequence satisfying $a\mathcal{U} \bigcirc b$ in 5 time steps.

B. Abstract Syntax Tree

Formal languages like LTL can be converted from a linear sequence of tokens into a tree structure, referred to as an abstract syntax tree (AST). This representation is exactly equivalent to the original formula and can be useful for recursive analysis and modification of the formula. For example, the formula $a\mathcal{U} \bigcirc b$ has an AST as illustrated in Figure 2.

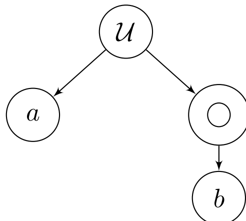


Fig. 2: AST of $a\mathcal{U} \bigcirc b$.

C. Büchi Automata and ω-regular Languages

Generalized Nondeterministic Büchi automata (typically referred to simply as Büchi automata) are automata-theoretic representations of systems and their behavior. Formally, a Büchi automata is defined as

$$\mathcal{G} = (Q, \Sigma, \delta, Q_0, \mathcal{F}) \,,$$

where Q is a finite state space, Σ is an alphabet, $Q_0 \in Q$ is the set of initial states, $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition relation of the system, and $\mathcal{F} \subset \delta$ is the acceptance sets. The symbols in alphabet Σ represent states in the automaton, and an *infinite word* is a member of Σ^ω which is an infinite sequence of symbols drawn from Σ. A given infinite word is accepted by the automaton if it is expressible by transitions δ and it enters the set of accepting states infinitely often. The set of all accepted infinite words is the language L of the automaton.

Any LTL formula can be converted into an equivalent Büchi automaton [2, 1]. The alphabet $\Sigma \in 2^{AP}$ is all reachable states, and the transition relation δ encodes all transitions permissible for the system. Since Büchi automata are representable as graphs as shown in Figure 3, we can use graph-based techniques to analyze the behavior of a system operating according to an LTL formula.

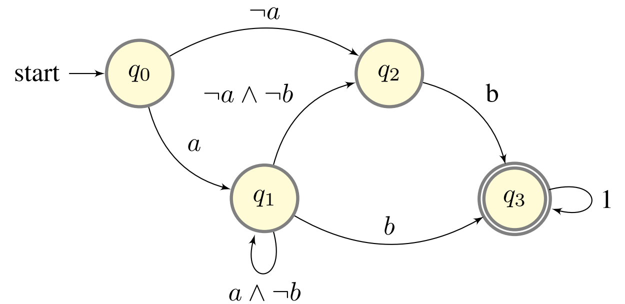


Fig. 3: Büchi Automaton of $a\mathcal{U} \bigcirc b$.

The language of a Büchi automaton is a set of infinite words, and set operations such as union and intersection can apply to the languages of two automata. New automata accepting these derived sets can be constructed from the definitions of the two input automata. This is useful when expressing the joint behavior of multiple real agents as a MAS, and computing the transition system of a MAS under the constraints of a TeamingLTL formula. The use of Büchi automata also highlights a subtlety of LTL, which is that the negation operator $\neg$ does not simply negate AP and logical expressions; it represents the complement of the words that would otherwise be accepted by the associated formula.

D. Program Verification and Controller Synthesis

We will make extensive use of LTL-derived automata to represent different aspects of a system. First, we have the agents themselves and their interactions with the environment, which we refer to as the *system*. The system may be directly

modeled via LTL and converted to an automaton, or we may use a labeling function to convert from real states S of the system to symbols in the alphabet of the LTL formula: $S \rightarrow 2^{AP}$. This gives us the *System Automaton*. We also use LTL — expressed compactly via TeamingLTL — to represent the correct behavior of a system. This can be converted into a *Behavior Specification Automaton*. We can use these two automata in a number of different ways.

The first is program verification [27]. In this context, the actual behavior of the system is checked against the requirements expressed by the behavior specification. A correctly behaving system will never violate the constraints of the LTL formula. In other words, given language L_s of a real system and language L_f of an LTL formula, there should be no intersection between the language of the first and the complement of the language of the second: $L_s \cap \neg L_f = \varnothing$. If the intersection is not empty, it is possible for the system to violate the constraints and the system fails the verification. Additionally, correct operation implies that the language of a correctly behaving system must intersect with the language of the formula: $L_s \cap L_f \neq \varnothing$.

We can also use the two automata to create a controller that is "correct by design" through a process called reactive synthesis [24]. In this process, we compute the product of the system automaton and the LTL-derived automaton to yield an new automaton that accepts a language $L_s \cap L_f$ which is both achievable by the system and correct with respect to the LTL formula. This is the *Controlled System Automaton* or CSA. This new automaton is then converted into a parity game where the first player sets "uncontrollable" AP (representing external environmental states) and the second player responds by setting the remaining "controllable" AP (representing choices or actions the controller can take). Every edge in the joint system-LTL automaton can be converted into one or more pairs of environment/controller state transitions. The resultant game expresses the same transitions of the system-LTL automaton while also encoding how the system should behave in response to a given set of inputs from the environment.

There are numerous variations on these ideas of validation and control in literature. Most focus on robust control [3, 4], efficient controller creation [12, 13, 22, 19, 29], or solutions to specific problem domains [11] for single-agent or monolithic systems. A relatively small number of works deal with multi-agent systems. Unlike some prior works, we do not restrict ourselves to the General Reactivity1 (GR1) [18, 17], which is a fragment of LTL for which controllers can be derived in polynomial time. An equivalent fragment can be derived from the temporal logic presented in this paper, if controller creation is important to the user. We also do not restrict ourselves to solving a particular problem type, such as agents moving in a discretized 2D space [15, 16, 8]. Our work is complementary to these (and other) works in that we provide a compact way to represent teaming behaviors in multi-agent systems. The problem spaces, labeling schemes, and controller synthesis techniques described in these other works can all be used in conjunction with TeamingLTL, with the added benefit that task assignment can be automated, redundancy can be exploited, and teams with distinct behaviors can be defined.

III. TeamingLTL

While standard LTL has proven useful in numerous contexts, it does not make any allowances for the redundancy and flexibility of multi-agent systems. Consequently, application of LTL to a MAS requires extensive work to properly express all of the different combinations of truth states that comprise correct behavior of such a complex system. This work modifies and extends standard LTL with notation to define the behavior of a MAS, making allowances for partial or complete interchangeability of agents as well as the need to construct subteams for controllability and tractability. The result is a substantially more compact way of expressing complex MAS behavior. In this section, we discuss the syntax and semantics of our new TeamingLTL formalism. We also illustrate the utility of the new operators with examples drawn from a warehouse staffed by mobile robots.

A. New Syntax

The new formalism starts with standard LTL formulas φ as discussed in II-A. It modifies the definition of the atomic proposition symbol a and adds operators to create exclusive groups of agents $[]$, require c repeated instances (count), and identify single agents $\mathcal{A}$.

The extended syntax to TeamingLTL is as follows:

$$\psi := \varphi \,|\, \mathcal{A}\varphi \,|\, [\psi] \,|\, c\psi \,|\, \neg\psi \,|\, \psi_1 \vee \psi_2 \,|\, \bigcirc \psi \,|\, \psi_1 \,\mathcal{U}\, \psi_2.$$

Every standard LTL formula is also a Teaming LTL formula. There are three new operators $\mathcal{A}$, $[]$, and c for single agent, group, and count respectively. From these, we also define secondary convenience operators $\mathcal{E}\varphi$ for every agent and $\{\psi\}$ for an optional group of agents. Logical and temporal operators from standard LTL apply to TeamingLTL as well.

This section describes each of these features and its relationship to standard LTL. The following supporting notation will be used in describing the aforementioned syntax. N is the set of all agents and $G \subseteq N$ is a subset of agents working together as a team. The expression $1 : n$ indicates the set of integers $\{1, \ldots, n\}$.

1) Modified Atomic Propositions: This work defines three different types of atomic propositions for use in TeamingLTL contexts. The first is *ungrounded* AP $a \in AP$. These are the AP that appear in a TeamingLTL formula specifying correct behavior of a system. They are not connected to any specific agent, and only serve to indicate that some agent must satisfy the associated criteria.

The next is *grounded* AP $a^{(i)} \in AP \times N$. These express the same truth states as the ungrounded AP, but on an agent-by-agent basis. This is consistent with the idea that AP are grounded in the local sensing and decision capabilities of individual agents. Implicit in the definition is the idea that $a^{(i)}$ is defined for every agent $i \in N$, even if it is always true or false for that agent. Also note that while it is convenient to refer to the agent-specific instances of $a^{(i)}$ when explaining the modified meaning of atomic proposition a and grounding

TeamingLTL in a real system, direct reference to a specific $a^{(i)}$ is not permitted in a TeamingLTL formula. Instead, we leave it to the automated processes described below to decide which agent will be used to satisfy each AP of the TeamingLTL formula.

The third type of AP is the *synthetic* AP, which is not connected to truth states of the system but is instead used to encode the logic of any teaming structure imposed on a MAS. We will use the atomic proposition $m_{i,G}$ to indicate that agent i is a member of some group G. The default group is all agents N, and the partition operator (discussed below) can be used to create new subgroups.

In keeping with the premise that it is sufficient for any real agent to satisfy an (ungrounded) AP specified in a formula:

$$a \equiv \bigvee_{i \in G} a^{(i)} \equiv \bigvee_{i \in N} (m_{i,G} \wedge a^{(i)}) \,.$$

This equivalence means that there is at least one agent that is a member of group G and for which $a^{(i)}$ is true. It follows from De Morgan's Law that

$$\neg a \equiv \bigwedge_{i \in G} \neg a^{(i)} \equiv \bigwedge_{i \in N} (m_{i,G} \rightarrow \neg a^{(i)}) \,.$$

This translates to "if agent i is a member of group G, then the corresponding $a^{(i)} =$ *False*."

2) Partitioning Operator: It is often the case that there are multiple tasks that must be executed by a MAS, and each task is beyond the capability of any single agent. In these cases, it makes sense to form teams of agents to address each task. Forming groups can also make it easier interpret the system's behavior, as well as reduce inefficiencies of frequent task switching. In our warehouse example, we may want to allocate a team of agents to stock shelves with products, while another team of agents is responsible for collecting products from the shelves to fulfill orders. TeamingLTL provides a Partitioning operator $[\psi]$ to identify expressions within a formula that must be satisfied by a discrete set of agents (i.e. a team). This only declares the existence of a team satisfying an expression; it does not place any restrictions of which agents are ultimately assigned to the team.

There is an implied Root scope enclosing the entire TeamingLTL formula, and all agents N are members of this group. A Partition operator declares a new group and its membership is drawn from the membership of the parent group. This means teams can be nested within teams, and the teams form a tree structure mirroring their positioning in the AST of the TeamingLTL formula. The set of agents assigned to a group is non-intersecting with the set of agents assigned to every other group that is not a parent to itself. Consider the following example formula:

$$[[\psi_1] \wedge [\psi_2]] \wedge [\psi_3] \,.$$

There are a total of five identifiable scopes as shown in Figure 4. This shows that $G_1 \subset G_4 \subset N$. Consequently, $G_1 \cup G_3 = \varnothing$ because $G_4 \cup G_3 = \varnothing$.

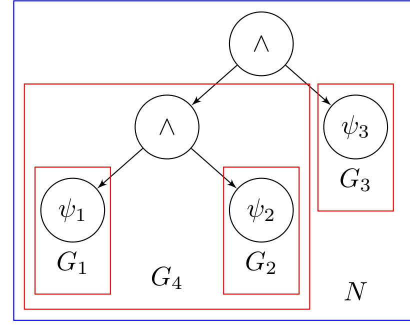


Fig. 4: Teams and their scopes in $[[\psi_1] \wedge [\psi_2]] \wedge [\psi_3]$.

The Partition operator can be rewritten as

$$[\psi] \equiv D_G \wedge \psi \,,$$

where $D_G \equiv \bigvee_{i \in N} m_{i,G}$. In plain language, this maps to "there is a non-empty group G and the agents in that group satisfy ψ". Note: while the AP listed in ψ do also have their own membership logic, it is still necessary to specify the membership requirement here. Without it, an expression like $[\neg a]$ could be satisfied by $G = \varnothing$.

The membership constraints imposed by the group operator, previously expressed via set notation, may also be encoded as standard LTL. Given a partition G (explicitly declared or the implicit Root partition) with a set of child partitions P_G, we can encode the non-intersection requirement as

$$X_G \equiv \bigwedge_{p1,p2 \in \binom{P_G}{2}} \bigwedge_{i \in 1:n} \Box \neg (m_{i,p1} \wedge m_{i,p2}) \,.$$

Additionally, all members of a declared Partition G are implicitly a members of the parent partition H, representable as

$$M_{G,H} \equiv \bigwedge_{i \in 1:n} \Box (m_{i,G} \rightarrow m_{i,H}) \,.$$

As noted previously, all agents are members of Root scope 0:

$$R \equiv \bigwedge_{i \in N} \Box m_{i,0} \,.$$

If a TeamingLTL formula does not explicitly show that two expressions are exclusive teams, then agents may participate in the satisfaction of both expressions. Given two AP a and b for example, $[a] \wedge [b]$ requires two separate agents. Comparatively, $[a] \wedge b$ could be satisfied by a single agent.

3) Optional Group: Sometimes we may want to define a task that should be pursued independently of other tasks, but only if the system has the bandwidth to allocate resources to task. For this, we define the optional group operator $\{\psi\}$ as

$$\{\psi\} \equiv \neg[\neg\psi] \equiv D_G \rightarrow \psi \,.$$

An optional group satisfies ψ, but only if it has members assigned to it. Because they are built on the partition operator, optional groups are exclusive of each other and regular groups.

4) Count Operator: It is sometimes the case that we need to execute multiple instances of the same task concurrently. For example, we may want to designate several teams of robots to cooperatively unload heavy freight if multiple deliveries arrive at the same time. This work also adds a unary Count operator to the basic LTL syntax to express that c unique but identical instances of the associated subformula must be satisfied by c non-intersecting sets of agents. Leveraging the Partition operator, we can define

$$c\psi \equiv \bigwedge_{i \in 1:c} [\psi] \,.$$

This creates c teams, each of which satisfies ψ.

Consistent with the Partition operator, a given agent may not participate in more than one instance of the associated ψ. If an agent were permitted to participate in multiple instances, then there would effectively be fewer than c instances of ψ being satisfied concurrently. Formally, the set of agents G_i satisfying ψ_i must be non-intersecting with every other set also satisfying an instance of ψ: $G_i \cap G_j = \varnothing \, \forall \, i, j \in \binom{c}{2}$.

5) Single Agent Operator: There are times when it is important to express that a singular agent is responsible for satisfying a particular set of AP in a formula. For example, we may want to require that a single agent picks up three different items in one run instead of dispatching three different robots. When a particular subformula must be satisfied by a singular agent, the relevant LTL may be identified as $\mathcal{A}\varphi$. This means that some agent i in the current group must individually satisfy the standard LTL formula φ. This can be expressed as

$$\mathcal{A}\varphi \equiv \bigvee_{i \in G} \varphi^{(i)} \equiv \bigvee_{i \in N} (m_{i,G} \wedge \varphi^{(i)}) \,,$$

where $\varphi^{(i)}$ is a version of φ that has had all ungrounded AP a replaced by the corresponding AP $a^{(i)}$ of agent i. Because the subformula is applied to individual agents, it is restricted to standard LTL.

Use of multiple Single Agent Operators does not strictly mean that the associated expressions must be satisfied by different agents. Using this section's example, $\mathcal{A}(h \wedge p)$ would require a single agent to hold a can and push a box at the same time. On the other hand, $\mathcal{A}(h) \wedge \mathcal{A}(p)$ could be satisfied by one agent performing both tasks or two agents each performing one of the tasks.

6) Every Agent Operator: At time, it is also useful to require that every agent in a system follow the same rules. For example, all robots in a warehouse should adhere to the same "rules of the road" and avoid collisions. Building on the Single Agent Operator, we can define the convenience operator $\mathcal{E}$ for every agent as

$$\mathcal{E}\varphi \equiv \neg\mathcal{A}\neg\varphi \equiv \bigwedge_{i \in N} (m_{i,G} \rightarrow \varphi^{(i)}) \,.$$

This represents the idea that every agent in group G individually satisfies φ.

B. Synthesis of Equivalent Standard LTL

TeamingLTL can be algorithmically converted to standard LTL, making it usable with existing libraries and algorithms. The easiest way to do this is through direct manipulation of the AST representing the TeamingLTL. This is done by walking the TeamingLTL AST in depth-first order, recursively swapping TeamingLTL operators with equivalent standard LTL as describe in the preceding subsection.

The membership logic encoded in the TeamingLTL requires special handling, as it is globally constant for the life of the system. This information, encapsulated in the R, X_G, and $M_{G,H}$ terms, can be accumulated in a separate tree while walking the TeamingLTL AST. When the recursion on the AST is complete, the accumulated expressions can be joined together as.

$$\psi_{const} = R \wedge \bigwedge_{G \in AST} X_G \wedge \bigwedge_{G \in AST} M_{G,H} \,.$$

The final result of converting to standard LTL is

$$\textsc{RecursiveReplaceTeaming}(root) \wedge \psi_{const} .$$

The recursive replacement of TeamingLTL terms with equivalent standard LTL terms of size $O(\|N\|)$ does result in a $O(\|N\|^d)$ increase in the number of total terms, where d is the depth of nesting of TeamingLTL operators. This mirrors the complexity of the physical system and is in line with other representations of MASs such as decentralized partially observable Markov decision processes. Optimization of the conversion process is possible, but left to a future work.

C. Differences from Standard LTL

While TeamingLTL is grounded in standard LTL, there are some important distinctions to keep in mind. The first is the modification to the definition of AP as discussed in Section III-A1. AP in a TeamingLTL formula are always ungrounded, and may be satisfied by any agent.

The second is the more subtle and has to do with the handling of group membership. As noted in Section III-B, the R, X_G, and $M_{G,H}$ terms are collected while traversing the AST and aggregated at the root of the standard LTL AST. Consequently TeamingLTL formulas ψ and $\neg\psi$ express complementary behavior of a MAS, but are not logical complements in the traditional LTL sense. Specifically, the negation operator only negates the AP and temporal logic of ψ but not the logic governing group membership.

One can think of TeamingLTL as always expressing the correct behavior of a system, even if that behavior is defined as the complement of some other behavior. The impact is that when performing program verification and checking $L_s \cap \neg L_f = \varnothing$, the overall complementation operation must be applied to the entire standard LTL formula after it has been derived from the TeamingLTL formula. This also requires that the program being verified output the membership of each agent at each step. If a program is verified only with respect to the grounded AP, it is possible for valid runs to appear invalid and vice versa.

IV. DEMONSTRATIONS

In this section, we show how TeamingLTL can be applied in several contexts. We use the Spot library [10] and its associated toolkit for manipulating standard LTL, creating B"uchi automata, and creating parity games after we have converted the TeamingLTL formulas to standard LTL. We would highlight that the following examples are meant to illustrate utility, and are not necessarily optimal in their design. As TeamingLTL converts to standard LTL prior to program verification or controller synthesis, any useful pattern in existing literature related to standard LTL and MASs can be brought to bear. This includes but is not limited to techniques for addressing distributed timing, robustness, consensus, and domain-specific controller design for MASs.

A. *Shift Coverage*

In this section, we will illustrate the use of TeamingLTL to specify correct behavior of a system and evaluate whether a particular distributed MAS complies with that specification. We will use a relatively simple example for tractability. Suppose we have a grocery store with several checkout lanes that may be open or closed. Cashiers are scheduled for 4 hours shifts with a 30 minute break. Break times may generally be freely scheduled according to the individual worker's preference, but there must always be a minimum number of lanes open. Breaks must be taken continuously and and start on the hour or half hour. Suppose that we have 6 employees with preferred break times as shown in Table I, and we must keep at least 4 lanes open.

TABLE I: Preferred Break Times

Count	Work time prior to break (hours)
2	0.0
1	1.0
3	2.0

After building individual automata for each employee, we can take the product of all automata to create an automata representing the total behavior of the system. We can also create a simple TeamingLTL formula to represent the desired behavior of the system: $\Box 4w$. Taking the product of the system automaton and the automaton created from the flattened TeamingLTL formula results in an automaton with no accepting edges. This shows that the system does not meet the specification. This is verifiable by inspection and noting that we can have no more than two employees on break at any given time, but three employees want to take their break after 2 hours. By setting a break schedule where no more than two employees can take a break at the same time, such as the one shown in Table II, we can rerun the verification process and see that this schedule satisfies the requirement.

TABLE II: Scheduled Break Times

Count	Work time prior to break (hours)
2	0.0
1	1.0
2	2.0
1	3.5

B. *Emergency Services Dispatch*

In this example, we will use TeamingLTL to define teams of agents that should be dispatched when an emergency occurs. Each kind of emergency corresponds to a specific type or response. As each instance of an emergency requires a unique response team, we must preallocate atomic propositions for as many concurrent instances of an emergency that we need to be able to handle. The following table of symbols will be used to represent the atomic propositions in this example.

TABLE III: Emergency Services AP symbols

AP	Definition
st	minor traffic accident
lt	major traffic accident
sf	small fire
lf	large fire
m	medical emergency
h	hazardous material spill
pd	field agent dispatched as police
fd	field agent dispatched as fire
ed	field agent dispatched as EMS

Using subscript i to indicate the ith concurrent instance of a particular emergency type, we have the following basic rules:

- $st_i \rightarrow [2pd]U\mathcal{E}\neg st_i$
- $lt_i \rightarrow [3pd \wedge ed \wedge fd]U\mathcal{E}\neg lt_i$
- $sf_i \rightarrow [2fd]U\mathcal{E}\neg sf_i$
- $lf_i \rightarrow [3fd \wedge pd \wedge ed]U\mathcal{E}\neg lf_i$
- $m_i \rightarrow [ed]U\mathcal{E}\neg m_i$
- $h_i \rightarrow [fd]U\mathcal{E}\neg h_i$

These rules essentially translate to "a given emergency requires that a team with a particular composition must be dispatched until the emergency is universally declared to be over."

We assume some maximum number of concurrent emergencies that will need to be handled, and some number of available field agents (police, fire, EMS) available to handle the emergencies. From this, we use this to create a TeamintLTL description of the system and automatically generate a controller. We can then examine its behavior when presented with a particular sequence of emergencies. In Tables IV and V, we show the number of field agents dispatched as a function of emergencies being called in to the dispatcher. The first table is restricted to simple emergencies that require one type of agent, while the second table contains all types of emergencies.

TABLE IV: Simple Emergency Services Rollout

	Num Active Emergencies			Num Dispatched Agents		
step	st	sf	h	pd	fd	ed
1	1			2		
2	1			2		
3	1	1		2	2	
4		1			2	
5		1	1		2	1
6	1	1	1	2	2	1
7	1		1	2		1

TABLE V: Complex Emergency Services Rollout

	Num Active Emergencies				Num Dispatched Agents		
step	lt	lf	m	h	pd	fd	ed
1		1			1	3	1
2		1	1		1	3	2
3	1		1		3	1	2
4	1			1	3	2	1

C. Recycling Robots

In this section, we will illustrate the applicability of TeamingLTL to the reactive synthesis problem We will use TeamingLTL to specify correct behavior of a distributed system and construct a correct-by-design centralized controller from the specification. To keep the problem tractable for illustration purposes, we will use a gridworld environment with randomly spawned cans, boxes, and agents. Agents can recover both cans and boxes, but can only hold one or the other at a time. Agents do not know where the recyclable items are located, but do know their current location and the location of the recycling point. Cans and boxes dropped in this cell are "recycled" and removed from the system. Agents can choose to move up, down, left, or right in the space. They can also choose to grab a can or box colocated in the current cell. The last action is to drop whatever item is currently being held in an empty space (including the recycling point).

While this is not an especially complicated environment, representing every cell of the gridworld with unique APs would become computationally prohibitive. It would also lock in the resultant controller to a particular grid size. For that reason, we will also create a labeling function to map from from these physical system states and actions to a reduced set of AP. The following table of symbols will be used to represent the atomic propositions in this example.

TABLE VI: Resource recovery AP symbols

AP	Definition	AP	Definition
sc	agent senses a can	p	agent is at the recycling point
sb	agent senses a box	gc	agent is grabbing a can
hc	agent is holding a can	gb	agent is grabbing a box
hb	agent is holding a box	dc	agent is dropping a can
s	agent is in search mode	db	agent is dropping a box
r	agent is in recover mode		

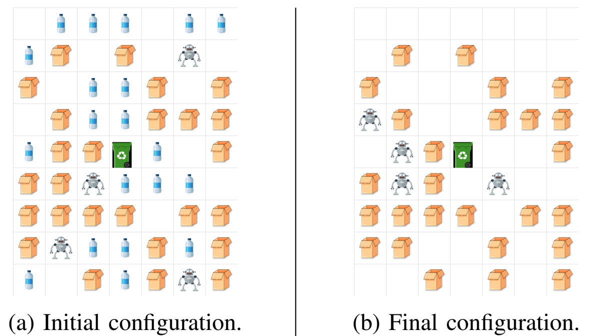
The labeling function sets environmental AP p true iff the agent is at the recycling point. AP sc and sd are true iff there is a can or box in the same cell as the agent respectively. AP hc and hb are true iff the agent is holding a can or box respectively. The search AP s causes the agent to take a random movement action, while the recover AP r causes the agent to take an action that reduces the distance to the recovery point. To avoid deadlock, the movement actions are stochastic and a different movement may occur with low probability. The action AP gc, gb, dc, and db map directly to grab and drop actions of cans and boxes, and these are deterministic. The complete LTL description of the agents is given in the Appendix.

Using TeamingLTL, we can compactly express desired behavior for the agents (illustrated in the subsequent subsections). The formula is converted into a controller that accepts uncontrollable AP $\{p_i, sc_i, sb_i, hc_i, hb_i\} \forall i \in N$ as inputs and generates $\{s_i, r_i, gc_i, gb_i, dc_i, db_i\} \forall i \in N$ as output to set the state of each agent as appropriate.

1) Recycle Cans Only: In this example, the TeamingLTL requires that the agents collect cans while ignoring boxes. The desired behavior for all agents can be assembled from the following expressions:

- $\psi_1 \models (\neg hc \land \neg hb \land \neg sc) \leftrightarrow s$
- $\psi_2 \models (\neg hc \land \neg hb \land sc) \leftrightarrow gc$
- $\psi_3 \models (\neg p \land (hc \lor hb)) \leftrightarrow r$
- $\psi_4 \models (p \land hc) \leftrightarrow dc$

The complete formula for specifying the desired behavior can be expressed as $\mathcal{E}\Box \bigwedge_{i \in 1:4} \psi_i$. This TeamingLTL formula can be converted to an automaton and used with the system automaton to produce a CSA and finally a controller. The end result of running this controller is illustrated in Figure 5 for a randomly initialized environment and running for 250 steps. As expected, the boxes are left alone and the agents only collect cans.



(a) Initial configuration. (b) Final configuration.

Fig. 5: Recycle Cans Only.

2) Specialized Tasking: In this example, we specify two sets of instructions: one to collect cans and one to collect boxes. We use TeamingLTL to require that two agents follow the first instruction set set and a different two agents follow the second instruction set. The first instruction set for collecting cans can be constructed from the following expressions:

- $\psi_1 \models (\neg hc \land \neg hb \land \neg sc) \leftrightarrow s$
- $\psi_2 \models (\neg hc \land \neg hb \land sc) \leftrightarrow gc$
- $\psi_3 \models (\neg p \land (hc \lor hb)) \leftrightarrow r$

- $\psi_4 \models (p \wedge hc) \leftrightarrow dc$

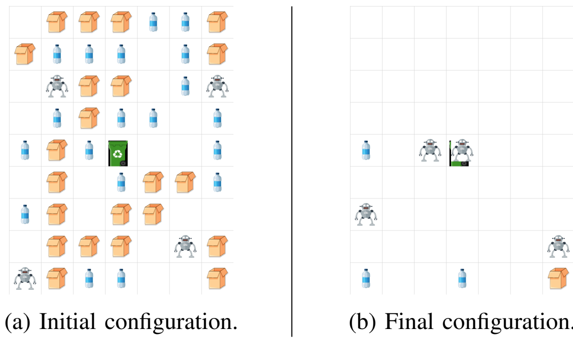
The second set is defined by the following expressions:

- $\psi_5 \models (\neg hc \wedge \neg hb \wedge \neg sb) \leftrightarrow s$
- $\psi_6 \models (\neg hc \wedge \neg hb \wedge sb) \leftrightarrow gb$
- $\psi_7 \models (\neg p \wedge (hc \vee hb)) \leftrightarrow r$
- $\psi_8 \models (p \wedge hb) \leftrightarrow db$

The complete instruction set is given by

$$2\mathcal{A}\Box \bigwedge_{i \in 1:4} \psi_j \wedge 2\mathcal{A}\Box \bigwedge_{i \in 5:8} \psi_j \, .$$

This TeamingLTL formula can be converted to an automaton and used with the system automaton to produce a CSA and finally a controller. The end result of running this controller is illustrated in Figure 6 for a randomly initialized environment and running for 250 steps.



(a) Initial configuration. (b) Final configuration.

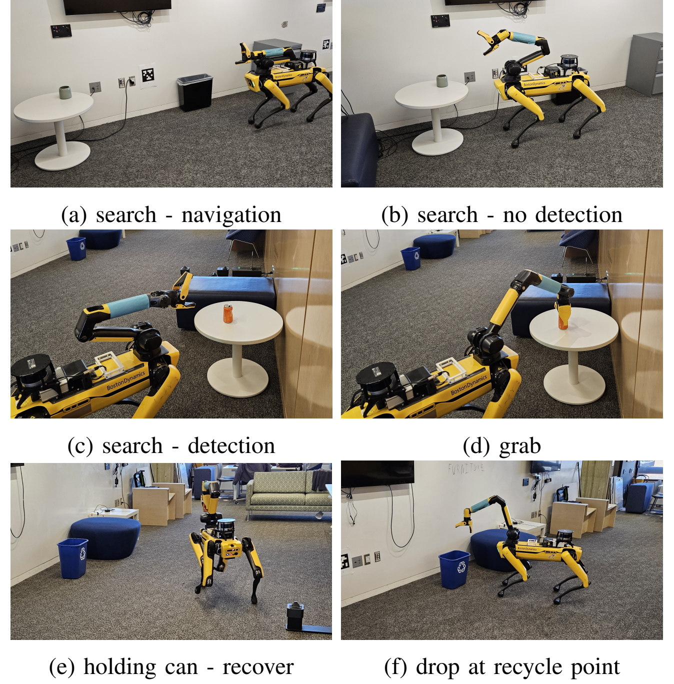
Fig. 6: Distinct Assignments.

3) Hardware Demo: This section shows a practical demonstration of TeamingLTL in the context of recycling materials found in an regular human environment environment. We can assemble the desired behavior for all agents from the following expressions:

- $\psi_1 \models (\neg hc \wedge \neg sc) \leftrightarrow s$
- $\psi_2 \models (\neg hc \wedge sc) \leftrightarrow gc$
- $\psi_3 \models (\neg p \wedge hc) \leftrightarrow r$
- $\psi_4 \models (p \wedge hc) \leftrightarrow dc$

The complete formula specifying the desired behavior can be expressed as $\mathcal{E}\Box \bigwedge_{i \in 1:4} \psi_i$. This TeamingLTL formula can be converted to an automaton and used with the system automaton (created from a labeled transition system) to produce a CSA and finally a controller. The execution of each of these behaviors is deferred to low-level controllers on each agent.

Figure 7 shows a single agent engaging in each of the behaviors defined in the TeamingLTL formula. The sequence of behaviors is governed by the TeamingTLT formula applied to a MAS of size 1. Figure 8 shows multiple agents concurrently performing each of the behaviors defined in the MAS as a function of their own individual sensing and state tracking. It is important to note that the only difference between this run and the single agent run was the number of agents assigned to the MAS. The scaling of behaviors to multiple agents occurred automatically as a function of the teaming logic embedded in the TeamingLTL formula. While this demo was limited by hardware and available space, we were able to generate controllers for up to 8 agents operating in parallel before the compute time became prohibitive. Figure 9 shows the exponential grown of the complexity of the controllers.



(a) search - navigation (b) search - no detection

(c) search - detection (d) grab

(e) holding can - recover (f) drop at recycle point

Fig. 7: Single agent operation

V. Conclusion

This work expands LTL with syntax for describing team-oriented behaviors in a heterogeneous MAS. This extended syntax, along with the restriction that all atomic propositions are locally observable by individual agents, creates an avenue for mission specification which respects the partial observability and redundancy common in a MAS. The formulation can be reduced to standard LTL, enabling it to be used with existing tools and techniques.

Acknowledgements

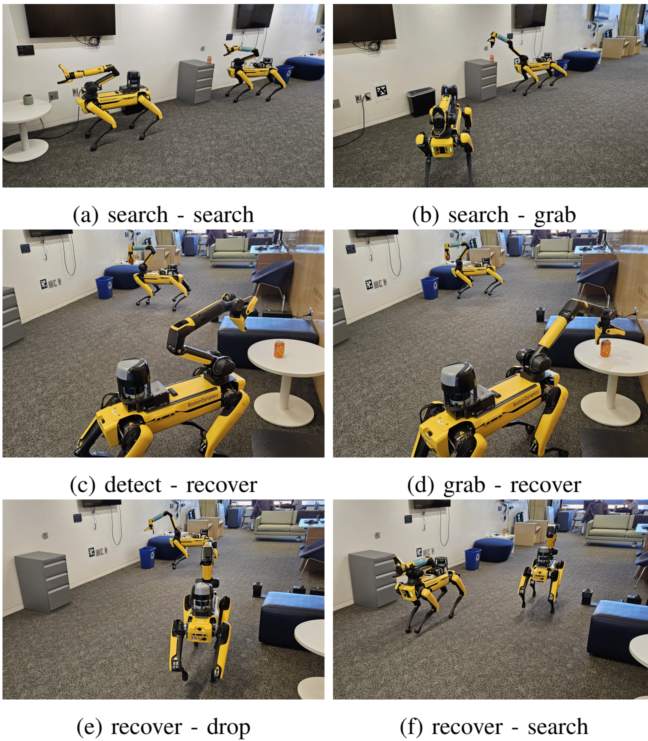
This work was supported in part by ONR REPRISM MURI N00014-24-1-2603 and ONR grant 00014-22-1-2592.

References

[1] Tomáš Babiak, Mojmír Křetínský, Vojtěch Řehák, and Jan Strejček. LTL to Büchi Automata Translation: Fast and More Deterministic. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214, pages 95–109. Springer Berlin Heidelberg, 2012.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press Ser. 2008.

[3] Benoît Barbot, Damien Busatto-Gaston, Catalin Dima, and Youssouf Oualhadj. Controller synthesis in timed büchi automata: Robustness and punctual guards. In *Quantitative Evaluation of Systems and Formal Modeling*



(a) search - search (b) search - grab

(c) detect - recover (d) grab - recover

(e) recover - drop (f) recover - search

Fig. 8: Multi agent operation

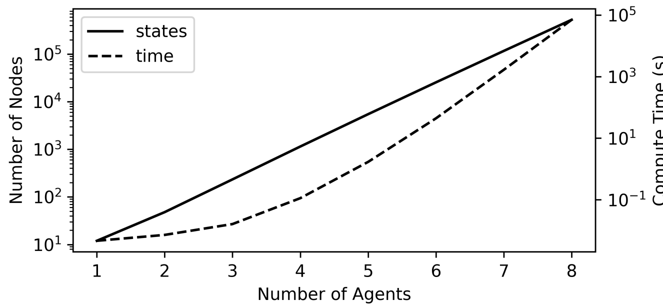


Fig. 9: Controller automata complexity

and Analysis of Timed Systems, pages 268–283. Springer Nature Switzerland, 2024.

[4] Damien Busatto-Gaston, Benjamin Monmege, Pierre-Alain Reynier, and Ocan Sankur. Robust Controller Synthesis in Timed Büchi Automata: A Symbolic Approach. In *Computer Aided Verification*, July 2019.

[5] Gustavo A. Cardona and Cristian-Ioan Vasile. Planning for heterogeneous teams of robots with temporal logic, capability, and resource constraints. *The International Journal of Robotics Research*, 43(13), November 2024.

[6] Fei Chen and Wei Ren. On the Control of Multi-Agent Systems: A Survey. *Foundations and Trends® in Systems and Control*, 6(4), 2019.

[7] Ji Chen, Salar Moarref, and Hadas Kress-Gazit. Verifiable Control of Robotic Swarm from High-level Specifications. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, July 2018.

[8] Yushan Chen, Xu Chu Ding, Alin Stefanescu, and Calin Belta. Formal Approach to the Deployment of Distributed Robotic Teams. *IEEE Transactions on Robotics*, 28(1), February 2012.

[9] Xu Chu (Dennis) Ding, Stephen L. Smith, Calin Belta, and Daniela Rus. LTL Control in Uncertain Environments with Probabilistic Satisfaction Guarantees*. *IFAC Proceedings Volumes*, 44(1), January 2011.

[10] Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's New? In *Computer Aided Verification*, volume 13372. 2022.

[11] G.E. Fainekos, H. Kress-Gazit, and G.J. Pappas. Hybrid Controllers for Path Planning: A Temporal Logic Approach. In *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005.

[12] Felipe Galarza-Jimenez, Vishnu Murali, and Majid Zamani. Compositional Synthesis of Controllers via Co-Büchi Barrier Certificates. *IFAC-PapersOnLine*, 58(11), 2024.

[13] Pushpak Jagtap and Dimos V. Dimarogonas. Controller synthesis against omega-regular specifications: A funnel-based control approach. *International Journal of Robust and Nonlinear Control*, 34(11), 2024.

[14] Marius Kloetzer and Calin Belta. Temporal Logic Planning and Control of Robotic Swarms by Hierarchical Abstractions. *IEEE Transactions on Robotics*, 23(2), April 2007.

[15] Marius Kloetzer and Calin Belta. Automatic Deployment of Distributed Teams of Robots From Temporal Logic Motion Specifications. *IEEE Transactions on Robotics*, 26(1), February 2010.

[16] Marius Kloetzer, Xu Chu Ding, and Calin Belta. Multi-robot deployment from LTL specifications with reduced communication. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, December 2011.

[17] Hadas Kress-Gazit, Nora Ayanian, George J. Pappas, and Vijay Kumar. Recycling controllers. In *2008 IEEE International Conference on Automation Science and Engineering*, August 2008.

[18] Hadas Kress-Gazit, David C. Conner, Howie Choset, Alfred A. Rizzi, and George J. Pappas. Courteous Cars. *IEEE Robotics & Automation Magazine*, 15(1), March 2008.

[19] M. Lahijanian, S. B. Andersson, and C. Belta. A probabilistic approach for control of a stochastic system from LTL specifications. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) Held Jointly with 2009 28th Chinese Control Conference*, December 2009.

[20] Leslie Lamport. "Sometime" is sometimes "not never": On the temporal logic of programs. In *Proceedings of the*

7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '80, 1980.

[21] Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-Independent Task Specifications via GLTL, April 2017.

[22] Rupak Majumdar, Kaushik Mallik, and Sadegh Soudjani. Symbolic controller synthesis for Büchi specifications on stochastic systems. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, April 2020.

[23] Eleni Mandrali. Weighted LTL with Discounting. In *Implementation and Application of Automata*, volume 7381. 2012.

[24] Thibaud Michaud and Maximilien Colange. Reactive Synthesis from LTL Specification with Spot.

[25] Salar Moarref and Hadas Kress-Gazit. Decentralized control of robotic swarms from high-level temporal logic specifications. In *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, December 2017.

[26] Benedetto Piccoli. Control of multi-agent systems: Results, open problems, and applications, February 2023.

[27] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (Sfcs 1977)*, September 1977.

[28] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, August 1987.

[29] Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. Automaton-guided controller synthesis for nonlinear systems with temporal logic. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, November 2013.

APPENDIX

The dynamics of an individual agent in the recycling grid world can be represented with the following LTL expressions:

1) $\varphi_1 \models \Box((sc \wedge gc) \rightarrow \bigcirc hc)$. If the agent detects a can in its vicinity and grabs, then it will be holding the can at the next time step.
2) $\varphi_2 \models \Box((sb \wedge gb) \rightarrow \bigcirc hb)$. If the agent detects a box in its vicinity and grabs, then it will be holding the box at the next time step.
3) $\varphi_3 \models \Box((hc \wedge dc) \rightarrow \bigcirc \neg hc)$. If the agent is holding a can and drops it, then it will not be holding a can at the next time step.
4) $\varphi_4 \models \Box((hb \wedge db) \rightarrow \bigcirc \neg hb)$. If the agent is holding a box and drops it, then it will not be holding a box at the next time step.
5) $\varphi_5 \models \Box((db \vee dc) \rightarrow (\neg dc \wedge \neg db))$. Cannot drop anything in an occupied area.
6) $\varphi_6 \models \Box(\neg sc \rightarrow \neg gc)$. If there is no can, the agent cannot grab a can.
7) $\varphi_7 \models \Box(\neg sb \rightarrow \neg gb)$. If there is no box, the agent cannot grab a box.
8) $\varphi_8 \models \Box((hc \vee hb) \rightarrow (\neg gc \wedge \neg gb))$. If the agent is holding anything, it cannot grab another item.
9) $\varphi_9 \models \Box(\neg hc \rightarrow \neg dc)$. If the agent is not holding a can, it cannot drop a can.
10) $\varphi_{10} \models \Box(\neg hb \rightarrow \neg db)$. If the agent is not holding a box, it cannot drop a box.
11) $\varphi_{11} \models \Box(hc \rightarrow dc \mathcal{M} hc)$. If the agent is holding a can, it will continue to hold the can until it drops the can.
12) $\varphi_{12} \models \Box(\neg hc \rightarrow gc \mathcal{M} \neg hc)$. If the agent is not holding a can, it will continue not holding a can until it grabs one.
13) $\varphi_{13} \models \Box(hb \rightarrow db \mathcal{M} hb)$. If the agent is holding a box, it will continue to hold the box until it drops the box.
14) $\varphi_{14} \models \Box(\neg hb \rightarrow gb \mathcal{M} \neg hb)$. If the agent is not holding a box, it will continue not holding a box until it grabs one.
15) $\varphi_{15} \models \Box\neg(hc \wedge hb)$. Cannot hold a can and a box at the same time.
16) $\varphi_{16} \models \Box\neg(dc \wedge db)$. Cannot detect a can and a box at the same time (guaranteed by the environment).
17) $\varphi_{17} \models \neg hc$. The agent never starts out holding a can.
18) $\varphi_{18} \models \neg hb$. The agent never starts out holding a box.
19) $\varphi_{19} \models \bigwedge_{a_1, a_2 \in \binom{s,r,gc,gb,dc,db}{2}} \Box\neg(a_1 \wedge a_2)$. Never take two actions at the same time.
20) $\varphi_{20} \models \Box(s \vee r \vee gc \vee gb \vee dc \vee db)$. Always take an action.

The complete formula for a single agent is the conjunction of all the individual expressions: $\bigwedge_{i \in 1:20} \varphi_i$. This formula can be converted into an automaton. The product of multiple instances of this automaton with itself represents the aggregate behavior of multiple agents in a shared environment.