


Giving Instructions in Linear Temporal Logic

Julian Gutierrez 

Monash University, Melbourne, Australia

Sarit Kraus 

Bar-Ilan University, Ramat-Gan, Israel

Giuseppe Perelli 

Sapienza University of Rome, Italy

Michael Wooldridge 

University of Oxford, UK

Abstract

Our aim is to develop a formal semantics for giving instructions to taskable agents, to investigate the complexity of decision problems relating to these semantics, and to explore the issues that these semantics raise. In the setting we consider, agents are given instructions in the form of Linear Temporal Logic (LTL) formulae; the intuitive interpretation of such an instruction is that the agent should act in such a way as to ensure the formula is satisfied. At the same time, agents are assumed to have inviolable and immutable background safety requirements, also specified as LTL formulae. Finally, the actions performed by an agent are assumed to have costs, and agents must act within a limited budget. For this setting, we present a range of interpretations of an instruction to achieve an LTL task Υ , intuitively ranging from “try to do this but only if you can do so with everything else remaining unchanged” up to “drop everything and get this done.” For each case we present a formal pre-/post-condition semantics, and investigate the computational issues that they raise.

2012 ACM Subject Classification Theory of computation \rightarrow Logic and verification; Theory of computation \rightarrow Automated reasoning

Keywords and phrases Linear Temporal Logic, Synthesis, Game theory, Multi-Agent Systems

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.15

Funding *Giuseppe Perelli*: Giuseppe Perelli is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), by the EU ICT-48 2020 project TAILOR (No. 952215), and by the Sapienza University project “BeGood” (No. AR22117A2DC7CD60).

Michael Wooldridge: Michael Wooldridge was supported by the UKRI under grant (grant EP/W002949/1).

1 Introduction

When we consider AI systems that carry out tasks on our behalf (*agents*), we can distinguish between different types. At one extreme are agents that are hard-wired to carry out a single specific function (e.g., vacuum cleaning robots). At the other extreme are *autonomous* agents, which have some control over their (mental) state and actions: we can *request* an autonomous agent to do something, but whether the agent actually does it is beyond our control. In this work, we are concerned with *taskable* agents, which lie between these two extremes. Taskable agents have a general set of capabilities, which they can draw upon to carry out a wide range of tasks on our behalf (typically within some constrained environment) – see, e.g., [19]. Unlike autonomous agents, however, taskable agents *must* try to carry out the instructions that are presented to them, although we are not in control of how they will try to do so.

In this paper we develop a semantics of instructions for taskable agents: how do we give them instructions, and how should an agent in receipt of these reconfigure itself in light of them? Our work lies within the tradition of reactive synthesis [27] and rational



© Julian Gutierrez, Sarit Kraus, Giuseppe Perelli, and Michael Wooldridge;
licensed under Creative Commons License CC-BY 4.0

29th International Symposium on Temporal Representation and Reasoning (TIME 2022).

Editors: Alexander Artikis, Roberto Posenato, and Stefano Tonetta; Article No. 15; pp. 15:1–15:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

verification [11, 15]. We assume agents are given instructions in the form of Linear Temporal Logic (LTL) formulae. A taskable agent given an LTL instruction Υ must subsequently act so as to ensure Υ is satisfied, if it can do so. At the same time, however, agents are assumed to have inviolable and immutable background *safety requirements*, which are also specified as LTL formulae. *Whatever* an agent does, it must attempt to ensure that its safety requirements are respected: *safety takes priority over obedience*. Actions performed by an agent are assumed to have costs, and agents must act within a given limited budget. For this setting, we present a range of interpretations of an instruction to achieve an LTL task Υ .

A key underlying principle to our work is the *principle of least change*: when acting on a new instruction, an agent should *try to keep everything else as close to how it was before as possible*. This is a *ceteris paribus* condition: keep all other things equal. Thus, the most basic form of instruction we consider intuitively says “try to do this but only if you can do so with everything else remaining unchanged”. An agent will adopt such an instruction only if it is able to do so while being able to guarantee its original task is achieved, ensuring that its safety requirement remains satisfied, and staying within its original budget. At the other extreme are instructions of the form “get this done whatever it takes” in which case we want the agent to get the task achieved irrespective of its original task and budget. Between these two extremes are a range of possibilities, some of which we study here. For each case, we present a pre-/post-condition semantics, and investigate the issues the semantics raise. We study the complexity of decision problems relating to our semantics, and find that these range from polynomial time decidable up to 2EXPTIME-complete.

2 The Formal Framework

Where S is a set, we denote the powerset of S by 2^S . We use various propositional languages to express properties of the systems we consider. In these languages, we let Φ be a finite and non-empty vocabulary of Boolean variables, with elements p, q, \dots . Where a and b are words (either finite or infinite) we denote the word obtained by concatenating a and b by $a \cdot b$. Where a is a word, we denote by a^ω the infinite repetition of a .

2.1 Environments

We consider environments in which a single agent is acting. We model such environments as nondeterministic finite state machines. We use *CALLIGRAPHIC* letters for elements of the environment. Our model assumes that an agent acts in an environment that can be in any of a set \mathcal{S} of possible environment states; the environment is initially in state s_0 . The agent has a repertoire \mathcal{A} of actions that it can perform: the result of performing an action $\alpha \in \mathcal{A}$ when the environment is in a state $s \in \mathcal{S}$ is to transform the environment into another state that is nondeterministically taken from $\mathcal{T}(s, \alpha)$; we refer to \mathcal{T} as the state transformer function of the environment. Finally, individual actions have real-valued costs associated with them: the cost of performing α is denoted $\mathcal{C}(\alpha)$ – thus, in the interests of simplicity, costs are fixed, and independent of the state of the system.

Formally an environment is given by a structure $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{L}, s_0)$, where:

- \mathcal{S} is a finite and non-empty set of *environment states*;
- \mathcal{A} is the finite and non-empty set of actions that may be performed in \mathcal{E} ;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow 2^{\mathcal{S}}$ is the *nondeterministic state transformer function* of \mathcal{E} ;
- $\mathcal{C} : \mathcal{A} \rightarrow \mathbb{R}$ is a *cost function*, which indicates the cost $\mathcal{C}(\alpha)$ of performing action $\alpha \in \mathcal{A}$;
- $\mathcal{L} : \mathcal{S} \rightarrow 2^\Phi$ is a *labelling function*, which associates with each state $s \in \mathcal{S}$ the set $\mathcal{L}(s) \subseteq \Phi$ of Boolean variables true in s ; and finally
- $s_0 \in \mathcal{S}$ is the initial state of the environment.

The environment \mathcal{E} is said to be *deterministic* if $\mathcal{T}(s, \alpha)$ is a singleton set for each environment state $s \in \mathcal{S}$ and action $\alpha \in \mathcal{A}$. As we will see later, deterministic environments greatly simplify many of the decision problems addressed in this paper.

2.2 Strategies

A strategy is a plan that defines how an agent will act in an environment. As is common practice in the theory of iterated games [6], and in work on synthesis and verification [13], we model strategies as finite state machines with output (i.e., Moore machines). Formally, a strategy σ for an environment \mathcal{E} is given by a finite state machine $\sigma = (Q, \tau, \delta, q^0)$ where:

- Q is the set of machine states;
- $\tau : Q \times \mathcal{S} \rightarrow Q$ is the state transition function of the strategy;
- $\delta : Q \rightarrow \mathcal{A}$ is the action selection function of the strategy, which selects an action $\delta(q)$ for every machine state $q \in Q$; and
- $q^0 \in Q$ is the initial state of the strategy.

We let Σ be the set of all such finite state machine strategies (\mathcal{E} is assumed to be clear from context). A strategy is enacted in an environment, starting from some state $s \in \mathcal{S}$, as follows: the environment starts in state s , and the strategy starts in state q^0 . The strategy then selects an action $\alpha_0 = \delta(q^0)$, and changes state to $q' = \tau(q^0, s)$. The environment responds to the performance of α_0 by moving to a state $s_1 \in \mathcal{T}(s, \alpha_0)$; the agent then chooses an action $\delta(q')$, and so on. In this way, the performance of a strategy σ in the environment \mathcal{E} traces out an infinite interleaved sequence of environment states and actions, called a *run*:

$$\rho : s \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \dots$$

The set of runs that may be generated by the enactment of σ in \mathcal{E} from state $s \in \mathcal{S}$ is denoted by $\mathcal{R}(\mathcal{E}, \sigma, s)$. We write $\mathcal{R}(\mathcal{E}, \sigma)$ as an abbreviation for $\mathcal{R}(\mathcal{E}, \sigma, s_0)$. A run $\rho \in \mathcal{R}(\mathcal{E}, \sigma, s)$ is sometimes said *compatible* with σ in \mathcal{E} – the definition is straightforward, and so we omit it here. Where ρ is a run and $u \in \mathbb{N}$, we write $s(\rho, u)$ to denote the state indexed by u in ρ – so $s(\rho, 0)$ is the first state in ρ , $s(\rho, 1)$ is the second, and so on and so forth. In the same way, we denote the first action in ρ by $\alpha(\rho, 0)$, the second by $\alpha(\rho, 1)$, etc.

Above, we defined the cost function $\mathcal{C}(\cdot)$ with respect to individual actions. In what follows, we find it useful to lift the cost function from individual actions to runs. Since runs are infinite, simply summing costs is not appropriate: instead, we consider the cost of a run to be the *average* cost incurred over the whole run; more precisely, we define the cost $\mathcal{C}(\rho)$ as the *inferior limit of means*: this is a very standard approach in automated formal verification as well as in the theory of iterated games (see, e.g., [6, p.366]). Formally, we have:

$$\mathcal{C}(\rho) = \liminf_{t \rightarrow \infty} \frac{1}{t} \sum_{i=0}^t \mathcal{C}(\alpha(\rho, i))$$

Given a run ρ , we define the *histories* associated with ρ to be the set of finite and non-empty prefixes of ρ which end in an environment state. We let $\mathcal{H}(\rho)$ denote the histories associated with ρ , and use h, h', \dots to refer to elements of \mathcal{H} . Where h is a history, we denote the word obtained from h by removing the final environment state from it by \underline{h} . We denote the final environment state occurring in h by $\text{last}(h)$.

2.3 Linear Temporal Logic

We use Linear Temporal Logic (LTL) to express properties of runs [25, 9]. LTL extends propositional logic with tense operators **X** (“in the next state...”), **F** (“eventually...”), **G** (“always...”), and **U** (“...until...”). Formally, the syntax of LTL is defined with respect to a set Φ of Boolean variables by the following grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \psi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \psi$$

where $p \in \Phi$. Other classical connectives (“ \perp ”, “ \wedge ”, “ \rightarrow ”, “ \leftrightarrow ”) are defined in terms of \neg and \vee in the conventional way. The LTL operators **F** and **G** are defined in terms of **U** as follows: $\mathbf{F}\varphi = \top \mathbf{U} \varphi$, and $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$. Given a set of variables Ψ , let $LTL(\Psi)$ be the set of LTL formulae over Ψ ; where Ψ is clear from the context, we write LTL . We interpret formulae of LTL with respect to pairs (ρ, t) where ρ is a run and $t \in \mathbb{N}$ is a temporal index into ρ . Any given LTL formula may be true at none or multiple time points on a run; for example, a formula $\mathbf{X}p$ will be true at a time point $t \in \mathbb{N}$ on a run ρ if p is true on run ρ at time $t + 1$. We will write $(\rho, t) \models \varphi$ to mean that $\varphi \in LTL$ is true at time $t \in \mathbb{N}$ on run ρ . The rules defining when formulae are true (i.e., the semantics of LTL) are defined as follows:

- $(\rho, t) \models \top$;
- $(\rho, t) \models x$ iff $x \in \mathcal{L}(s(\rho, t))$ (where $x \in \Phi$);
- $(\rho, t) \models \neg\varphi$ iff it is not the case that $(\rho, t) \models \varphi$;
- $(\rho, t) \models \varphi \vee \psi$ iff $(\rho, t) \models \varphi$ or $(\rho, t) \models \psi$;
- $(\rho, t) \models \mathbf{X}\varphi$ iff $(\rho, t + 1) \models \varphi$;
- $(\rho, t) \models \varphi \mathbf{U} \psi$ iff, for some $t' \geq t$, $(\rho, t') \models \psi$ and $(\rho, t'') \models \varphi$ for all $t \leq t'' < t'$.

We write $\rho \models \varphi$ for $(\rho, 0) \models \varphi$, in which case we say that ρ *satisfies* φ . A formula φ is *satisfiable* if there is some run satisfying φ . A strategy σ *realizes* φ in \mathcal{E} , written $\sigma \triangleright_{\mathcal{E}} \varphi$, if ρ satisfies φ for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma, s_0)$, that is, if every run compatible with σ satisfies the LTL formula. The formula φ is *realizable* in \mathcal{E} , written $\triangleright_{\mathcal{E}} \varphi$, if there is a strategy σ such that $\sigma \triangleright_{\mathcal{E}} \varphi$. We write $\sigma \triangleright_{\mathcal{E}, s} \varphi$ to indicate that every run consistent with σ starting from state $s \in \mathcal{S}$ satisfies φ . While LTL satisfiability is PSPACE-complete [31], LTL synthesis/reliability is 2EXPTIME-complete [27]. The following results are useful in what follows:

► **Lemma 1.** *Given some $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{L}, s_0)$, a state $s \in \mathcal{S}$, and an LTL formula φ , checking whether there is a run ρ of \mathcal{E} starting in s such that $\rho \models \varphi$ is PSPACE-complete.*

Proof. For membership, we can reduce to LTL satisfiability: we can define a new LTL formula $\psi_{\mathcal{E}}$ which represents the behaviour of the environment, and check whether $\psi_{\mathcal{E}} \wedge \varphi$ is satisfiable. For hardness, we can reduce LTL model checking [31]. ◀

► **Lemma 2.** *Given an environment $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{L}, s_0)$, a state $s \in \mathcal{S}$, and an LTL formula φ , checking whether there is a strategy σ such that $\sigma \triangleright_{\mathcal{E}, s} \varphi$ is 2EXPTIME-complete.*

Proof. For membership, we can reduce to LTL realizability: we can define a new LTL formula $\psi_{\mathcal{E}}$ which represents the behaviour of the environment, and check whether $\psi_{\mathcal{E}} \rightarrow \varphi$ is realizable. For hardness, we can reduce LTL synthesis [27]. ◀

2.4 Agents

The agents we consider are taskable: an agent is designed to operate in a specific environment \mathcal{E} , but may be assigned different tasks to carry out in the environment. The basic approach we adopt is to specify a task for an agent via an LTL formula Υ : an agent succeeds with

the task Υ if it chooses a strategy that realizes Υ . Crucially, tasks for agents are *mutable*: we might initially instruct an agent to achieve a task formula Υ_1 , but later *change* our instructions by giving the agent a task Υ_2 . Much of the remainder of this paper is concerned with understanding precisely how an agent should interpret a new instruction like this.

We intend our agents to operate safely. To this end, we assume they have a background *safety requirement*, specified as an LTL formula ζ ¹. The absolute priority for an agent is to ensure that ζ is realized. While tasks Υ are mutable, safety requirements ζ are immutable: no matter what instructions the agent receives relating to tasks, the agent *must* ensure that its safety requirement is fulfilled if it can do so in the first place.

Finally, we assume that agents have a budget, which we denote by β . This is a real number which indicates how much cost we are willing for our agent to incur, where the cost of a run is measured using the limit of means approach (see above). Thus, the budget is not an absolute limit, but indicates maximum *average* expenditure that we are willing for an agent to incur. Formally, then, an agent A is defined by a structure $A = (\Upsilon, \zeta, \sigma, \beta)$ where:

- Υ is an LTL formula which specifies the currently assigned task of the agent;
- ζ is an LTL formula representing the safety requirements of the agent;
- $\sigma \in \Sigma$ is the agent's current strategy; and finally,
- $\beta \in \mathbb{R}$ is the agent's budget.

We will say an agent $(\Upsilon, \zeta, \sigma, \beta)$ in an environment \mathcal{E} is:

- *safe* if the strategy ensures the agent's safety requirement is realized: $\sigma \triangleright_{\mathcal{E}} \zeta$;
- *sound* if the strategy achieves the task: $\sigma \triangleright_{\mathcal{E}} \Upsilon$; and
- *solvent* if the agent is within budget: $\mathcal{C}(\rho) < \beta$, for each $\rho \in \mathcal{R}(\mathcal{E}, \sigma, s_0)$.

An agent that satisfies all of the above conditions is said to be *properly configured*. A properly configured agent is one that has thus been assigned a task Υ , and that has chosen a strategy to achieve Υ that will do so while respecting both the safety requirement ζ and budget β .

► **Theorem 3.** *Given an environment \mathcal{E} , a safety requirement ζ , a budget β , and a task Υ , checking whether there is a strategy σ such that $A = (\Upsilon, \zeta, \sigma, \beta)$ is properly configured is 2EXPTIME-complete. Moreover, given an environment \mathcal{E} and an agent $A = (\Upsilon, \zeta, \sigma, \beta)$ for \mathcal{E} , checking whether A is properly configured with respect to \mathcal{E} is PSPACE-complete.*

Proof. First, note that the problem of finding σ such that the agent is properly configured amounts to solve the synthesis problem for the formula $\Upsilon \wedge \zeta$ with the additional condition of staying within the budget β for every possible generated run in the environment. This problem corresponds to a special case of an equilibrium synthesis with combined qualitative and quantitative objectives, as it is introduced and solved in [14], and for which a 2EXPTIME procedure has been shown. For hardness, we can reduce LTL synthesis: the transformer function \mathcal{T} can be used to encode a transition relation in an instance of LTL synthesis [27]. For the second part, an agent together with an environment will trace out a Kripke structure \mathcal{K} such that $\mathcal{K} \models \zeta \wedge \Upsilon$ if, and only if, the agent is properly configured, thus membership is in PSPACE. For hardness, one can use a reduction from LTL model-checking [31]. ◀

► **Theorem 4.** *For a deterministic environment \mathcal{E} , a safety requirement ζ , a budget β , and a task Υ , checking whether there exists a strategy σ such that $A = (\Upsilon, \zeta, \sigma, \beta)$ is properly configured is PSPACE-complete. In contrast, for an environment \mathcal{E} and an agent $A = (\Upsilon, \zeta, \sigma, \beta)$ for \mathcal{E} , checking whether A is properly configured with respect to \mathcal{E} is solvable in polynomial time.*

¹ A well-known class of LTL formulae express what are technically called safety properties [32], intuitively capturing the idea that “nothing bad happens.” We do not *require* our safety requirements ζ to belong to this class, although in practice, many safety requirements will.

Proof. The first part follows from Lemma 1 (note that for deterministic environments, solvency can be checked in polynomial time). For hardness use LTL model checking: the transformer function \mathcal{T} can encode a transition relation in an instance of LTL model checking [31]. For the second part, an agent together with an environment will trace out a path of the form $a \cdot b^\omega$ with a and b finite words. We can compute a and b in polynomial time by executing the strategy in the environment and watching until we find to have repeated a configuration; we then check $\zeta \wedge \Upsilon$ on this path, which can be done in polynomial time [21], and then check solvency, which can be done in polynomial time in the deterministic case (by computing the average cost over the finite path corresponding to b). ◀

2.5 Progressing an LTL Formula

We use the concept of *progressing* an LTL formula [5]. The idea is as follows. Suppose that an agent has been operating to achieve a task Υ , and in this process has generated a (finite) history h . We then give the agent a new instruction Υ' . Now, it may be that the task Υ has already been discharged in the history h , in which case, when the agent reconfigures itself in light of the new instruction Υ' , it can disregard the original task Υ . To make this concrete, consider a (rather contrived) agent with a task $\Upsilon = \mathbf{F}$ beer (must eventually drink beer). The agent configures itself, adopting a strategy to eventually drink beer, and succeeds to do so. Sometime later, a user issues a new instruction Υ' . At this point, however, the task Υ has *already been achieved* (beer has been drunk), and nothing the agent does subsequently will change that. As the agent adjusts its configuration in light of the new instruction, it can take account of that. Similar comments apply to the agent's safety requirement ζ : it may be that the requirement has already been discharged within the finite history h . (In general, of course, we do not think of safety requirements operating in this way: they are typically ongoing properties, which must be maintained infinitely into the future. For example, suppose the agent's safety requirement was $\zeta = \mathbf{G}\neg$ crash. Such a requirement cannot be discharged within a finite history: as the agent adjusts its configuration in light of the new instructions, it must take into account these ongoing requirements.)

The notion of progressing a temporal formula φ through a history h captures the idea of transforming φ into a new formula $\mathbf{prog}(\varphi, h)$ so that it captures all those requirements of φ that have not already been satisfied within h . Formally, we have the following [16, Thm 3.2].

► **Theorem 5.** *For every LTL formula φ , finite history h , and run ρ such that $s(\rho, 0) = \mathbf{last}(h)$, there exists an LTL formula $\mathbf{prog}(\varphi, h)$ such that $\underline{h} \cdot \rho \models \varphi$ iff $\rho \models \mathbf{prog}(\varphi, h)$.*

We refer the reader to the definition of the function $\mathbf{prog}(\dots)$ in [16], and note that the function can be implemented in time polynomial in the size of the history and given formula².

We now move on to the main focus of the present article: the issue of giving instructions to agents. The precise setting we consider is as follows. We have an agent $A = (\Upsilon, \zeta, \sigma, \beta)$, operating in an environment $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{L}, s_0)$, which has traced out a history h . The agent is then presented with a new instruction Υ' . How then should the agent reconfigure itself – and in particular, what new strategy σ' should the agent adopt, taking into account the safety requirement ζ and the history h to date? As we will see, there are multiple possible answers to this question: we consider a range of possible instruction types, which differ primarily in the strength of the new instruction given to the taskable agent.

² The definition in [16] is defined with respect to states rather than histories: the extension to histories simply requires progressing the input formula progressively through each state in the given history h .

As we noted above, a very natural principle when an agent is asked to adopt a new instruction is the one of *least change*: when reconfiguring yourself, first try to do so while keeping everything else as close to how it was before as possible. Thus, on the one hand, we have instructions of the form “take onboard my new instruction Υ' only if you can do so without changing anything else.” On the other hand, we have instructions of the form “drop everything else you are doing and get this done whatever it takes.” There are several variants between these two extremes. We emphasise, however, that *no* new instructions can result in the agent releasing its safety requirement: this is immutable – but the satisfaction of the safety requirement needs to be considered in the context of the history h traced out so far.

We define the semantics of each type of instruction using pre- and post-conditions. The pre-conditions define the circumstances under which the instruction can be accommodated. If the pre-conditions of an instruction type are not satisfied, then that instruction will fail, in which case the agent is assumed to be unchanged. The post-conditions specify properties of the agent configuration after the agent has adjusted its configuration to accommodate the instruction, *under the assumption that the pre-condition was satisfied*. We consider four different types of instructions, which we refer to as Type I instructions through to Type IV instructions: Type I instructions are the weakest (in the sense that they make the least demands on an agent); Type IV are the strongest type of instructions.

2.6 Instructions of Type I

The first type of instruction we consider is the weakest form of instruction in our setting. In a Type I instruction, we ask an agent to take on a new task Υ' *only* if it can do so safely, without affecting its current task, and within the originally specified budget. This type of instruction thus embodies the notion of “least change” in a very direct way. With this type of instruction, an agent will *retain* its previous task, irrespective of whether the instruction is successful or not. Formally, our Type I semantics are as follows.

Suppose we have an agent $(\Upsilon, \zeta, \sigma, \beta)$ operating in an environment \mathcal{E} , having generated a history h . The agent is then presented with a Type I instruction Υ' . The preconditions for this Type I instruction require that there exists a strategy $\sigma' \in \Sigma$ such that:

$$(I.1) \quad \underline{h} \cdot \rho \models \zeta \wedge \Upsilon, \text{ for every } \rho \in \mathcal{R}(\mathcal{E}, \sigma', \text{last}(h));$$

$$(I.2) \quad \sigma' \triangleright_{\mathcal{E}, \text{last}(h)} \Upsilon';$$

$$(I.3) \quad \mathcal{C}(\rho) \leq \beta, \text{ for every } \rho \in \mathcal{R}(\mathcal{E}, \sigma', \text{last}(h)).$$

If these pre-conditions are satisfied, then an agent acting upon a Type I instruction will change to a configuration $(\text{prog}(\Upsilon, h) \wedge \Upsilon', \text{prog}(\zeta, h), \sigma', \beta)$ where $\sigma' \in \Sigma$ satisfies conditions (I.1)–(I.3). Several points are in order with respect to this definition. First, note that the only components modified with the new instruction will be the current task of the agent (extended to include the new instruction) and the chosen strategy: both the safety requirement and budget will remain unchanged after updating the agent.

With respect to (I.1), the conditions require that the adopted strategy will ensure that both the original task Υ and the safety requirement are satisfied by σ' . Note that we could alternatively have expressed this condition as:

$$\rho \models \text{prog}(\zeta, h) \wedge \text{prog}(\Upsilon, h) \quad \text{for every } \rho \in \mathcal{R}(\mathcal{E}, \sigma, \text{last}(h))$$

Condition (I.2) requires that σ' ensures the achievement of the new task Υ' . The main point to note here is that, unlike condition (I.1), this condition *does not* take into account the prior history h . The reason for this is as follows: suppose the original task Υ was **F**beer (eventually drink beer), and the agent succeeded to do so within the history h . Then if the

15:8 Giving Instructions in Linear Temporal Logic

new instruction Υ' was $\mathbf{F}\text{beer}$ (intuitively, “drink another beer”), then if we took h into account when considering the new strategy we would regard Υ' as being already discharged. Thus, when we consider the satisfaction of the new task Υ' , we need to ignore the history to date and focus on the run that will be generated subsequently.

Finally, observe that the solvency requirement for an agent adopting the new strategy (condition (I.3)) is given with respect to *future* costs only. This may seem counter-intuitive: should we not take into account costs already incurred? The answer stems from the way we measure costs, using the inferior limit of means. The agent will have been running for a finite time when presented with its new instruction, but will run for an *infinite* time subsequently. In the limit, costs already incurred in the finite history h are irrelevant: all that matters are the *future* costs incurred over the infinite suffix to h .

► **Example 6.** Consider the floor of an elderly home where a number of guest rooms are arranged around a corridor, and a robot deployed to patrol the surroundings. The robot can move along the corridor or in one of the guest rooms. Moreover, each room can be either occupied or empty, and the floor might be dirty or clean. The robot has two tasks. The first is the safety requirement to eventually tidy up the corridor every time it gets dirty. This can be represented by the LTL formula $\zeta = \mathbf{G}(\text{dirty} \rightarrow \mathbf{F}(\text{tidy} \mathbf{U} \neg \text{dirty}))$. The second is the assigned task to monitor the occupied room and assist a guest in room i in case they send an assistance request. This can be represented by the formula $\Upsilon_i = \mathbf{G}(\text{call}_i \rightarrow \mathbf{F}\text{assist}_i)$. Note that requests from the other rooms will be ignored by the agent. This is because they are marked as empty, and so requests from those rooms would happen for reasons that are not relevant to the robot, e.g., a contractor team is testing/maintaining the room.

Once a new guest arrives in room j , the agent needs to be reconfigured to include the task of listening to requests from the corresponding room. This can be done by the Type I instruction of the form $\Upsilon_j = \mathbf{G}(\text{call}_j \rightarrow \mathbf{F}\text{assist}_j)$. Without giving full details, for brevity, the setting above could be represented by the following environment \mathcal{E} :

- $\mathcal{S} = \text{Pos} \times 2^{\{1,2,3,4\}} \times \{\text{dirty}, \text{not-dirty}\}$, where Pos represents the possible positions of the robot (i.e., locations in the corridor or one of the rooms), the set $2^{\{1,2,3,4\}}$ represents the set of rooms from which an assistance request has been made, whereas the third component in $\{\text{dirty}, \text{not-dirty}\}$ represents the fact that the corridor is dirty in some location or clean all around.
- $\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}, \text{stop}\}$ allows the robot to navigate along the floor by either moving along the four directions or by stopping.
- $\mathcal{C}(\text{up}) = \mathcal{C}(\text{down}) = \mathcal{C}(\text{left}) = \mathcal{C}(\text{right}) = 1$, while $\mathcal{C}(\text{stop}) = 0$, that is, the robot consumes one unit of power only when it moves along the floor.
- \mathcal{T} is deterministic on Pos , meaning that the value in Pos depends only on the current position and the robots action, and not on the environment reaction to this. Whereas \mathcal{T} is nondeterministic on the other components of the states, which depends on whether the environment calls for assistance in any of the room and whether it activates the signal that the corridor is somewhere dirty.
- $\mathcal{L}(\text{room}_i, C, \iota) = \{\text{assist}_i\} \cup C \cup \{\iota\}$ meaning that if the robot is in room room_i , then it is assisting the guest in the same room, while the assistance requests and the cleaning status of the corridor is copied from the other two components of the state.
 $\mathcal{L}(\text{corridor}, C, \iota) = C \cup \{\text{not-dirty}\}$ meaning that whenever the robot is in any location of the corridor, it is not assisting any guest but efficiently cleaning the corridor.
- $s_0 = (\text{corridor}_0, \emptyset, \text{not-dirty})$ is an arbitrary taken initial state of the environment.

► **Theorem 7.** *Given an environment \mathcal{E} and a properly configured agent $A = (\Upsilon, \zeta, \sigma, \beta)$, together with a history h generated by σ in the environment, checking whether a Type I instruction Υ' allows for a proper reconfiguration of an agent A after history h is 2EXPTIME-complete.*

Proof. Following Theorem 5, the problem can be reduced to checking whether there exists a strategy σ' such that the agent $A' = (\text{prog}(\Upsilon, h) \wedge \Upsilon', \text{prog}(\zeta, h), \sigma', \beta)$ is properly configured over the environment \mathcal{E}' obtained from \mathcal{E} by replacing the initial state s_0 with \underline{h} . Following Theorem 3 it holds that such problem can be solved in 2EXPTIME. For hardness, we can encode LTL synthesis [26]. ◀

2.7 Instructions of Type II

Type II instructions are stronger than Type I: whereas in Type I instructions an agent is asked to adopt the new instruction within the original budget, with a Type II instruction an agent is given a new budget – intuitively, more resources to accomplish the task. Otherwise a Type II instruction is as a Type I.

Suppose we have an agent $(\Upsilon, \zeta, \sigma, \beta)$ operating in an environment \mathcal{E} , having generated a history h . The agent is then presented with a Type II instruction (Υ', β') . The pre-conditions for this Type II instruction require that there is a strategy $\sigma' \in \Sigma$ such that:

(II.1) $\underline{h} \cdot \rho \models \zeta \wedge \Upsilon$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \text{last}(h))$;

(II.2) $\sigma' \triangleright_{\mathcal{E}, \text{last}(h)} \Upsilon'$;

(II.3) $\mathcal{C}(\rho) \leq \beta'$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \text{last}(h))$.

If the pre-conditions are satisfied, then an agent acting upon a Type II instruction will change to a configuration $(\text{prog}(\Upsilon, h) \wedge \Upsilon', \text{prog}(\zeta, h), \sigma', \beta')$ where $\sigma' \in \Sigma$ satisfies conditions (II.1)–(II.3). Since β' can be set to any value, using a Type II instruction we can in effect say “forget about your budget limits”. (We can also use Type II instructions to give *reduced* budgets.)

► **Example 8.** Consider again the setting in Example 6. After a given period of time, the components of the robots might deteriorate, including the battery, which might reduce its maximum capacity. Therefore, the budget β must be adjusted to accommodate this requirement. A Type-II instruction is suitable for this purpose. This can be of the form (Υ, β') in case we are updating the cost requirement only, or of the form (Υ', β') if we are including a new task requirement to the new configuration.

► **Theorem 9.** *Given an environment \mathcal{E} and a properly configured agent $A = (\Upsilon, \zeta, \sigma, \beta)$, together with a history h generated by σ in the environment, checking whether a Type II instruction (Υ', β') allows for a proper reconfiguration of an agent A after history h is 2EXPTIME-complete.*

2.8 Instructions of Type III

In a Type III instruction, we ask an agent to take on a task even if that means dropping its original task; but it should stay within its original budget. Suppose we have an agent $(\Upsilon, \zeta, \sigma, \beta)$ operating in an environment \mathcal{E} , having generated a history h . The agent is then presented with a Type III instruction Υ' . The pre-conditions for this Type III instruction require that there is a strategy $\sigma' \in \Sigma$ such that:

(III.1) $\underline{h} \cdot \rho \models \zeta$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \text{last}(h))$;

(III.2) $\sigma' \triangleright_{\mathcal{E}, \text{last}(h)} \Upsilon'$;

(III.3) $\mathcal{C}(\rho) \leq \beta$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \text{last}(h))$.

15:10 Giving Instructions in Linear Temporal Logic

If the pre-conditions are satisfied, then an agent acting upon a Type III instruction will change to a configuration $(\Upsilon', \text{prog}(\zeta, h), \sigma', \beta)$ where $\sigma' \in \Sigma$ satisfies conditions (III.1)–(III.3).

► **Example 10.** Consider again the setting described in Example 6. Recall that the task requirement is of the form $\Upsilon = \bigwedge_{i \in N'} \Upsilon_i$, for a given subset $N' \subseteq N$ of the rooms and Υ_i being the requirement for room i .

After a guest left the elderly home, say from room j , it is no longer necessary for the robot to monitor such room. We can use a Type III instruction to remove this task requirement. This can be obtained by using the formula $\Upsilon' = \bigwedge_{i \in N' \setminus \{j\}} \text{prog}(\Upsilon_i, \underline{h})$ that drops room j from monitoring and reinstates the other rooms at the same time.

► **Theorem 11.** *Given an environment \mathcal{E} and a properly configured agent $A = (\Upsilon, \zeta, \sigma, \beta)$, together with a history h generated by σ in the environment, checking whether a Type III instruction Υ' allows for a proper reconfiguration of an agent A after history h is 2EXPTIME-complete.*

Proof. The proof is similar to that of Theorem 7, where the synthesis problem is cast by replacing $\text{prog}(\Upsilon, h) \wedge \Upsilon'$ by just Υ' . ◀

2.9 Instructions of Type IV

In a Type IV instruction, we present an agent with a new task and budget: the agent drops its original task and budget completely, and completely replaces them with those newly presented. This represents the strongest type of instruction we consider here: it may result in all components of an agent other than the safety condition (which is progressed through the prior history) being reconfigured. Suppose we have an agent $(\Upsilon, \zeta, \sigma, \beta)$ operating in an environment \mathcal{E} , having generated a history h . The agent is then presented with a Type IV instruction (Υ', β') . The pre-conditions for this Type IV instruction require that there is a strategy $\sigma' \in \Sigma$ such that:

(IV.1) $\underline{h} \cdot \rho \models \zeta$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \text{last}(h))$;

(IV.2) $\sigma' \triangleright_{\mathcal{E}, \text{last}(h)} \Upsilon'$;

(IV.3) $\mathcal{C}(\rho) \leq \beta'$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \text{last}(h))$.

If the pre-conditions are satisfied, then an agent acting upon a Type III instruction will change to a configuration $(\Upsilon', \text{prog}(\zeta, h), \sigma', \beta')$ where $\sigma' \in \Sigma$ satisfies conditions (IV.1)–(IV.3).

► **Theorem 12.** *Given an environment \mathcal{E} and a properly configured agent $A = (\Upsilon, \zeta, \sigma, \beta)$, together with a history h generated by σ in the environment, checking whether a Type IV instruction (Υ', β') allows for a proper reconfiguration of an agent A after history h is 2EXPTIME-complete.*

We know from the optimisation and planning literature that giving instructions to an agent as it operates may deliver solutions that are inferior to solutions computed *before* the agent executes any plan. For example, consider the taskable agent presented in [17, 1], where an autonomous agent is required to move in a grid-like world, much like the robot in our running example, and is requested to find the shortest path between two places in the grid, say X and Y , but under the restriction that some elements must be collected before going from X to Y . While the same situation may arise here, our approach ensures that new tasks will be undertaken *safely*.

3 Related Work & Discussion

We hope to have illustrated that the natural and intuitively simple problem of reconfiguring a taskable agent to accommodate new LTL instructions is actually a rather subtle and complex problem. Formally, we have also shown that solving this general problem, in most settings and for most types of instructions we consider, is 2EXPTIME-complete, with only a few exceptions where the decision problems are either PSPACE-complete or solvable in polynomial time in some of the simplest cases.

Our work is somewhat related to (and influenced by) work in the semantics of speech acts and agent communication. This work traces its origins least as far back to Wittgenstein's 1953 study of language games, which marked the beginning of the *pragmatic* tradition of language understanding [20]. The best-known work in this tradition is the speech acts paradigm, initiated by [4] and [30]. Within AI, Searle's work led to [8] showing how the pre- and post-conditions of speech acts could be formulated using an AI planning formalism, which paved the way for the development of AI systems that could plan to perform speech acts as part of a natural language generation process [3], and then the semantics of agent communication languages [23, 10]. The main difference with our work is that speech act semantics pre-suppose that agents are *autonomous*, in the sense that they have complete control over their own state and actions. For example, the FIPA **request** communicative act represents an attempt to get an agent to perform an action: it is not an *instruction*, which is conceptually and technically different. Our work differs in that we assume agents are *taskable*: we give one of our agents an instruction and they will attempt to accommodate it as per the various different models we have discussed.

Research on human-robot interactions addresses the problem of robots performing tasks requested by a human through natural language [33, 28, 22, 24]. In [2], it is suggested that robots that can understand and perform human instructions will consist of three levels: (i) language-based semantic reasoning on the instruction (high level); (ii) formulation of goals in robot symbols and planning to achieve them (mid-level); and (iii) action execution (low level). We focus on level (ii), and our work can exploit previous works that translate high-level natural language tasks to LTL [29, 7]. Another interesting line of research is the use of LTL instructions in Reinforcement Learning [34, 18]. In [34], LTL is used as a language for specifying multiple tasks to speed up learning by generating the tasks in a way that supports the composition of learned skills. We assume that no learning is needed by our agents: our work is closer to the LTL synthesis paradigm (although combining our work with previous work on LTL-based reinforcement learning [34, 18] would present many interesting research questions). Our work has focused on the idea of *directly* instructing agents with regard to tasks. There are of course other *indirect* ways to influence the activities of agents such as sharing information [12] and setting taxation or rewards [35]. The main difference with our work is that we consider agents that are obligated to follow the instructions that they receive, given that they are consistent with their safety requirements; in contrast, [12] and [35] both assume self-interested agents.

So far, we have said nothing about how an agent will *operationalise* our semantics. In the single agent case, one very natural possibility, given an instruction Υ , is to start by attempting to interpret it in the "least intervention" semantics of Type I instructions: try to accommodate it while continuing with your existing goal. If this is not possible, then move on to semantics that result in greater changes. We might, for example, imagine an agent coming back and asking for an increased budget ("if you give me a bigger budget I can get your original task accomplished *and* the new task"). The multi-agent case is more complex,

because the operationalisation will have to take into account how other agents respond to the adoption of a new strategy: it would thus be useful to consider the dynamics of a system after an agent reconfigures itself (how agents might respond). Such issues have been investigated in the game theory literature, but not in the settings we consider here. In this latter case, we might, e.g., look at “responsible” agents that take into account the social welfare of a system as they adopt new strategies. The multi-agent case is the most obvious avenue for future research, wherein ideas from both cooperative and non-cooperative game theory could shed some light into how to approach that more challenging problem.

Finally, in our model, we have a single safety requirement ζ . It is natural to extend our model to support hierarchies of logically-defined safety requirements $(\zeta_1, \dots, \zeta_k)$, where an agent is required to first ensure that ζ_1 is satisfied, then ζ_2 , and so on.

References

- 1 Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In Doina Precup and Yee Whye Teh, editors, *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 166–175. PMLR, 2017. URL: <http://proceedings.mlr.press/v70/andreas17a.html>.
- 2 Alexandre Antunes, Lorenzo Jamone, Giovanni Saponaro, Alexandre Bernardino, and Rodrigo Ventura. From human instructions to robot actions: Formulation of goals, affordances and probabilistic planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5449–5454. IEEE, 2016.
- 3 D. E. Appelt. *Planning English Sentences*. Cambridge University Press: Cambridge, England, 1985.
- 4 J. L. Austin. *How to Do Things With Words*. Oxford University Press: Oxford, England, 1962.
- 5 Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artif. Intell.*, 116(1-2):123–191, 2000. doi:10.1016/S0004-3702(99)00071-5.
- 6 K. Binmore. *Fun and Games: A Text on Game Theory*. D. C. Heath and Company: Lexington, MA, 1992.
- 7 Adrian Boteanu, Thomas Howard, Jacob Arkin, and Hadas Kress-Gazit. A model for verifiable grounding and execution of complex natural language instructions. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2649–2654. IEEE, 2016.
- 8 P. R. Cohen and C. R. Perrault. Elements of a plan based theory of speech acts. *Cognitive Science*, 3:177–212, 1979.
- 9 E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, pages 996–1072. Elsevier, 1990.
- 10 FIPA. The foundation for intelligent physical agents, 2001. See <http://www.fipa.org/>.
- 11 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In *TACAS*, volume 6015 of *LNCS*, pages 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2_16.
- 12 John Grant, Sarit Kraus, Michael John Wooldridge, and Inon Zuckerman. Manipulating boolean games through communication. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.
- 13 J. Gutierrez, P. Harrenstein, and M. Wooldridge. Iterated Boolean games. *Information and Computation*, 242:53–79, 2015.
- 14 Julian Gutierrez, Aniello Murano, Giuseppe Perelli, Sasha Rubin, Thomas Steeples, and Michael Wooldridge. Equilibria for games with combined qualitative and quantitative objectives. *Acta Informatica*, pages 1–26, 2020.
- 15 Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. Automated temporal equilibrium analysis: Verification and synthesis of multi-player games. *Artif. Intell.*, 287:103353, 2020. doi:10.1016/j.artint.2020.103353.

- 16 Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Teaching multiple tasks to an RL agent using LTL. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 452–461. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018.
- 17 Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Teaching multiple tasks to an RL agent using LTL. In Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar, editors, *AAMAS*, pages 452–461. IFAAMAS, Richland, SC, USA / ACM, 2018. URL: <http://dl.acm.org/citation.cfm?id=3237452>.
- 18 León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 540–550, 2020.
- 19 Pat Langley, Nishant Trivedi, and Matt Banister. A command language for taskable virtual agents. In G. Michael Youngblood and Vadim Bulitko, editors, *Proceedings of the Sixth AAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, October 11-13, 2010, Stanford, California, USA*. The AAI Press, 2010. URL: <http://aaai.org/ocs/index.php/AIIDE/AIIDE10/paper/view/2153>.
- 20 S. C. Levinson. *Pragmatics*. Cambridge University Press: Cambridge, England, 1983.
- 21 Nicolas Markey and Philippe Schnoebelen. Model checking a path. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR 2003 – Concurrency Theory, 14th International Conference, Marseille, France, September 3-5, 2003, Proceedings*, volume 2761 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2003. doi:10.1007/978-3-540-45187-7_17.
- 22 Pedro Henrique Martins, Luís Custódio, and Rodrigo Ventura. A deep learning approach for understanding natural language commands for mobile service robots. *arXiv preprint*, 2018. arXiv:1807.03053.
- 23 J. Mayfield, Y. Labrou, and T. Finin. Evaluating KQML as an agent communication language. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 347–360. Springer-Verlag: Berlin, Germany, 1996.
- 24 Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35(1-3):281–300, 2016.
- 25 A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth IEEE Symposium on the Foundations of Computer Science (FOCS’77)*, pages 46–57. The Society, 1977.
- 26 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the Sixteenth ACM Symposium on the Principles of Programming Languages (POPL)*, pages 179–190, January 1989.
- 27 A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programs (ICALP’89)*, pages 652–671, 1989.
- 28 Pradip Pramanick, Hrishav Bakul Barua, and Chayan Sarkar. Decomplex: Task planning from complex natural instructions by a collocating robot. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6894–6901. IEEE, 2021.
- 29 Vasumathi Raman, Constantine Lignos, Cameron Finucane, Kenton CT Lee, Mitchell P Marcus, and Hadas Kress-Gazit. Sorry dave, i’m afraid i can’t do that: Explaining unachievable robot tasks using natural language. In *Robotics: Science and Systems*, volume 2, pages 2–1. Citeseer, 2013.
- 30 J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press: Cambridge, England, 1969.
- 31 A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

15:14 Giving Instructions in Linear Temporal Logic

- 32 A. Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects Comput.*, 6(5):495–512, 1994. doi:10.1007/BF01211865.
- 33 Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:25–55, 2020.
- 34 Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 452–461, 2018.
- 35 Michael Wooldridge, Ulle Endriss, Sarit Kraus, and Jérôme Lang. Incentive engineering for boolean games. *Artificial Intelligence*, 195:418–439, 2013.