24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Automated Verification of Social Law Robustness for Reactive Agents

Alexander Tuisov[1] and **Erez Karpas**[2]

Abstract. Coordinating agents in a multi-agent system is an interesting and important challenge. One of the most effective methods of coordinating multiple agents is using a "social law", which restricts some possible behaviors in order to ensure every agent can achieve its goal. Recent work has connected social laws with automated planning, and shown how to verify if a given social law is robust, that is, ensures each agent can achieve its goal regardless of the plans chosen by the other agents. This prior work assumed the agents choose a plan offline, and never modify it in response to the other agents' actions. In this paper, we address *reactive* agents, that is, agents that can reconsider their course of action during execution. This setting presents a new challenge, as agents now have the possibility of entering an infinite loop (a livelock) in which each agent replans in the same way in response to the other agents. We show how to verify if a given social law is robust in such a setting, and our main contribution is a compilation which eliminates the need to keep track of each agent's current plan and constitutes a backbone of the verification algorithm.

1 Introduction

Systems with multiple autonomous agents are becoming more and more common (e.g., in robotic fulfillment centers) and will become more so in the future (e.g. autonomous cars, drone delivery). Designing such systems is very challenging; one of the main reasons for this is the need to coordinate all of the agents operating in the same shared environment.

Several approaches for coordination have been explored in the past. One possibility is to use a centralized controller, which controls the actions of all the agents (for example, see [19]). However, this centralized control is not feasible for a large number of agents, especially when they are owned by different entities (as is the case for autonomous cars). Another approach is to allow each agent to act autonomously, and devise "rules of encounter" for when two agents come into conflict, which usually requires some negotiation between the agents [8]. In this paper, we follow a third approach, of enacting a "social law" [23] which restricts the behavior of the agents in order to ensure each agent can achieve its own goal. One of the main advantages of social laws is that they do not require any communication between the agents.

Previous work [10, 17] has shown how to verify if a given social law is *rationally robust*, that is, ensures that each agent can achieve its goal regardless of the (goal-achieving) plans chosen by the other agents. However, the kind of robustness described in these works is extremely strict — it requires every plan chosen *offline* by every agent to work, regardless of what every other agent is doing. Importantly, this assumes that agents are blind, in the sense that they never change their plan, regardless of what they see the other agents doing. This is an extremely strong requirement, and has limited real-world applicability.

In this paper, we propose a new model, in which agents are *reactive*, that is, they are allowed to *replan* if their current plan is going to fail. We begin by formulating the execution and replanning model more precisely. We then propose a suitable definition of robustness for this model, and discuss the possible failure modes under this new model.

1.1 Contribution Statement

The contributions of this paper are threefold: first, we formalize the notions of social laws and reactive robustness. Second, we characterize the conditions under which a given social law is reactively robust, and describe a (prohibitively large) search space for finding the counterexample to the robustness. Our third contribution is an effective search procedure, which we show does not have to traverse this large search space, but a much more compact one.

2 Background

2.1 MA-STRIPS

Following [4], we define an MA-STRIPS formalism as a quadruple $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ where F is a set of predicates, A_i is a set of actions for agents numbered $1 \dots n$, $I \subseteq F$ is the initial state and $G_i \subseteq F$ is the goal of agent i. Each action $a \in A_i$ is a triplet $\langle pre(a), add(a), del(a) \rangle$, where $pre(a) \subseteq F$ is the precondition for performing a, and $add(a), del(a) \subseteq F$ are the add and delete effects of a respectively. Applying (or executing) action a to a state s transitions the system to a new state s', where $s' := (s \setminus del(a)) \cup add(a)$. For brevity, sometimes we refer to $\langle add(a), del(a) \rangle$ as $eff(a)$.

In the spirit of [10] we enrich the MA-STRIPS formalism by allowing some preconditions to be marked as *waitfor*, and by doing so, assume our agents have the ability to *wait and do nothing*. For each action a we define a set of $waitfor$ preconditions $waitfor(a) \subseteq pre(a)$ with the following semantics: if an agent is due to execute a and some $f \in waitfor(a)$ is not fulfilled, the agent performs action $wait = \langle \emptyset, \emptyset, \emptyset \rangle$ instead of applying a.

2.2 Social laws

Social laws are sets of rules that regulate the behavior of agents, such that a certain level of coordination is enforced upon the otherwise "selfish" agents. Such rules exist in human society [21] as well as in

[1] Technion, Israel, alexandt@technion.ac.il
[2] Technion, Israel, karpase@technion.ac.il

artificial systems [24, 15]. Note that social laws should forbid undesired behavior, but cannot add new capabilities to agents, or remove their goals.

Given a social law, one needs a way to assess how well it promotes the implicit coordination of agents. Karpas *et al.* [10] propose to check a social law for *rational* and *adversarial* robustness for tasks with MA-STRIPS as an underlying formalism. Rational robustness was defined as follows:

Definition 1. ***Rational Robustness*** *- A social law l for multi-agent setting $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ is rationally robust iff: for all agents i, for all individual solutions π_i of Π, for all action sequences π resulting from arbitrary interleaving $\{\pi_i\}_{i=1}^n$, $G_1 \cup \cdots \cup G_n$ is achieved.*

Enforcement of a rationally robust social law ensures that *every* agent can plan *offline* with no regard to the plans and actions of the other agents, and is still guaranteed to eventually reach its goal. Note that this requirement is very strict, as it assumes each agent is executing the plan it came up with offline with its "eyes closed". In this paper we stick to robustness as a quality certificate of a social law, but we attempt to derive a new, more liberal, notion of robustness.

Also, note that Definition 1 uses the notion of "individual solutions". These are produced by agents, where each agent solves a planning problem called the *single agent projection*. As it was defined in [10]:

Definition 2. ***Rational single agent projection*** *- Given a multi-agent setting $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$, rational single agent projection for agent i is a planning task $\Pi_i = \langle F, A_i, I, G_i \rangle$.*

The individual solutions produced by this single agent projection, however, are very restricted. For example, consider an agent that has its path blocked by another agent in the initial state. It will fail to produce a plan, even if the precondition demanding clear passage is *waitfor*. The ability to replan during execution only exacerbates the problem, since the passage has to be clear at *any* time point that one of the agent replans, otherwise their planing will fail. In Section 3 we address this problem, and suggest a more suitable single agent projection.

2.3 Labeled Transition System

In this work we also make use of labeled (or named) transition systems. As defined by Keller [11], a labeled transition system is a tuple $\langle Q, \rightarrow, \Sigma \rangle$, where Q is a set of states, $\rightarrow$ is a binary relation on Q, also called the set of transitions, and Σ is a set of labels for these transitions. For any $q, q' \in Q$ we denote $q \xrightarrow{\sigma \in \Sigma} q'$ if $(q, q') \in \rightarrow$ is given a label σ. We say that σ is deterministic if there exists only one $q \xrightarrow{\sigma} q'$ for a given q and σ, and the transition system is deterministic if all $\sigma \in \Sigma$ are.

2.4 State Space Search

Our tool of choice for solving the task at hand is state space search [22]. For the purpose of this work, it can be defined as a tuple $\langle S, A, Action(s), Result(s, a), Goal(s) \rangle$ where:

- S is a set of states the system can be in.
- A is a set of action labels.
- $Action(s)$ is a function that establishes the set of actions applicable in some $s \in S$.
- $Result(s, a)$ is a function determining the state reached by applying action $a \in A$ in a state $s \in S$.
- $Goal(s)$ a function determining whether a given state $s \in S$ is a goal of the search.

3 Problem Setup

3.1 Social law formalization

Social laws are the main subject of this work, thus we start our discussion from defining a social law in a rigorous way. We formalize social laws in terms of MA-STRIPS tasks as follows:

Definition 3. ***Social law*** *- A social law l is a transformation of an MA-STRIPS task $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ to a modified task $\Pi^l = \langle F^l, \{A_i^l\}_{i=1}^n, I^l, \{G_i^l\}_{i=1}^n \rangle$ such that:*

- *F^l is a set of predicates. $F \subseteq F^l$, i.e. the social law is only allowed to add predicates.*
- *$\{A_i^l\}_{i=1}^n$ is a set of actions for each agent i. $\forall a^l \in A_i^l \; \exists! a \in A_i$:*
  - *$pre(a) \subseteq pre(a^l)$*
  - *$waitfor(a) \subseteq waitfor(a^l) \subseteq pre(a^l)$*
  - *$add(a) \subseteq add(a^l)$*
  - *$del(a) \subseteq del(a^l)$*
  - *$(add(a^l) \setminus add(a)), (del(a^l) \setminus del(a)) \subseteq F^l \setminus F$*

  i.e. the social law is not allowed to add actions, remove preconditions of actions, and to add effects affecting predicates of the original problem, but is allowed to mark preconditions as waitfor.
- *We require $I \subseteq I^l \subseteq F^l$, so that the original initial state is preserved.*
- *$\{G_i^l\}_{i=1}^n \subseteq F^l$. We require that $\forall i : G_i \subseteq G_i^l$, i.e. the social law is not allowed to remove goals.*

The reasoning behind this definition is simple: the social law promotes coordination by restricting actions and adding goals, while not allowing agents to exhibit previously impossible behavior. Since we assume the agents can wait, the labeling of some preconditions as $waitfor$ is also allowed.

3.2 Single Agent Projection

Since the overall goal of the social law is to decompose the centralized search problem into many much simpler, single-agent search tasks, we have to describe in detail the appropriate single-agent projections. Sadly, a straightforward attempt to use the single agent projection as it appears in Definition 2 is problematic. It precludes the agents from devising plans that include an action that has a waitfor precondition on some fact which is false in the initial state, e.g. to wait for some other agent to move out of the way. Thus, it may put severe limitations on the usability of reactive agents, since they may plan more than once during the execution.

As an alternative, we propose the following single-agent projection of the problem:

Definition 4. ***Reactive single agent projection*** *- given an MA-STRIPS problem $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$, the reactive single agent projection for agent i is the planning task $\Pi_i' = \langle F, A_i', I, G_i \rangle$, where $A_i' = \{\langle (pre(a) \setminus waitfor(a)), add(a), del(a) \rangle \mid a \in A_i\}$*

Informally, we ignore the *waitfor* preconditions of the actions to capture the effects of the environment and other agents that are out of i's control. For example, by ignoring the *waitfor* precondition for having some resource for an action, an agent may plan regardless of that resource currently being in use by others. This projection will allow the agent to wait until said resource is available. Note that this projection solves the problem we raised in Section 2.2.

In what follows, we say "agent i (re)plans from state s" as a shorthand for "Solving i's reactive single agent projection where the initial state is s". Likewise, we sometimes say "i's plan" as a shorthand for "solution to i's reactive single agent projection".

3.3 Planning and execution model

We now describe the planning and execution model for how the agents operate, which allows us to define a new notion of robustness. We distinguish between a number of different settings with regards to replanning while acting.

First, and perhaps the most obvious setting is the one where replanning is forbidden altogether, i.e. every agent plans exactly once before the start of execution. This model is used to derive the notion of *rational robustness* as presented in Definition 1.

Second, we mention the setting where the agent is allowed to replan after its every action. Robustness with respect to this model will be denoted as *anytime robustness*. Note that we do not put any limitations on the agent's plan, thus in many domains the setting contains a trivial livelock, akin to the Buridan's ass dilemma [25].

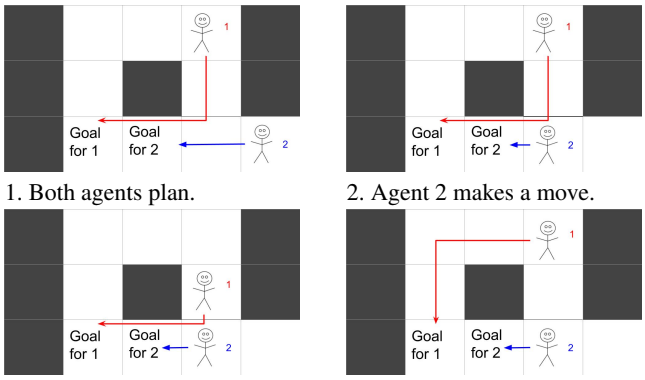
The next type of model is on a spectrum between the options presented earlier (*never* allow replanning and *always* allow it), as here we allow the agent to replan only *on need*, defined as a state where the agent deduces that the original plan is no longer valid, thus the agent must replan. Replanning *on need* presents a wide range of possible rules for deducing where the need appears, and such an inference would probably require combining multi-agent reasoning and plan recognition about other agents' actions, which is outside of the scope of this work.

Thus, we would like to focus our discussion here on a specific sub-case of *replanning on need*: *reactive replanning*. In this setting, the agents replan only when they cannot execute the next action in the original plan, i.e. the next action in the plan has an unfulfilled non-*waitfor* precondition when the agent is activated by an external scheduler. This allows us to forgo replanning in situations where the missing precondition of an un-executable action was restored by another agent. Robustness with respect to this model will be denoted as *reactive robustness*.

To present a mathematical definition of robustness, we introduce some auxiliary notation and definitions. For a multi-agent setting $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$, we present a transition system which allows us to reason about agents' plans, which we must do in order to determine when agents are allowed to replan.

This transition system extends the transition system defined by Π by adding an action $replan_i$ for each agent i (in addition to the "regular" actions A_i), which allows the agent to change its current plan. replan$_i$ is only applicable when the preconditions of i's next planned action do not hold. To account for each agent's plan, we also introduce a variable plan$_i$, which holds the current plan of agent i. Thus, the global state of the system is $\langle s, \pi_1, \ldots, \pi_n \rangle$, where $s \subseteq F$ describes the state of the world, and will be called the *world state*. π_i is the not-yet-executed suffix of the current plan agent i intends to execute, and we will refer to it as the *internal state of i*.

The need to keep track of agents' internal states is best illustrated by an example as given in Figure 1. One of the modes of breaching robustness is the existence of a cycle in the execution system, where agents can repeat same sequences of actions *ad infinitum*, without ever reaching their goals (see Definition 8 ahead). In this example, although the same world state is reached twice (with different internal states), there exists no cycle agents can be caught in indefinitely. This means, the global state g carries additional information, uninferrable from the world state alone.



1. Both agents plan. 2. Agent 2 makes a move. 3. Agent 1 comes to a point where it needs to replan. 4. Same world state as (2) with different internal states — no livelock.

Figure 1: Why Internal States Matter. Red arrow is the plan for agent 1, blue arrow is the plan for agent 2.

We now describe the possible transitions in this system from some global state $g = \langle s, \pi_1, \ldots, \pi_n \rangle$. First, any agent i can be chosen to act next, with the exception of agents who are waiting for some *waitfor* precondition, as described below. Assume agent i's current plan is $\pi_i = \langle a_1^i, a_2^i, \ldots, a_m^i \rangle$. We distinguish between three cases:

1. If a_1^i is applicable in the world state s, that is, pre$(a_1^i) \subseteq s$, then agent i can execute a_1^i, and is not allowed to replan. The system then transitions to the state $\langle (s \setminus del(a_1^i)) \cup add(a_1^i)), \pi_1, \ldots \pi_i', \ldots, \pi_n \rangle$. Since the a_1^i was executed, we update i's plan to be $\pi_i' = \langle a_2^i \ldots a_m^i \rangle$.
2. If the non-*waitfor* preconditions of a_1^i hold in s, but some of the *waitfor* preconditions do not, that is, (pre$(a_1^i) \setminus$ waitfor$(a_1^i)) \subseteq s$ and pre$(a_1^i) \not\subseteq s$, then agent i must wait, and another agent will act.
3. If some of the non-*waitfor* preconditions of a_1^i do not hold in s, that is, (pre$(a_1^i) \setminus$ waitfor$(a_1^i)) \not\subseteq s$, then agent i is required to replan instead of performing a_1^i. The system then transitions to the state $\langle s, \pi_1, \ldots, \pi_i', \ldots, \pi_n \rangle$ where π_i' is a possible plan for agent i in the single agent projection from world state s.

There are two facts worth mentioning here: first, an agent is required to apply the first action in its not-yet-executed suffix of a plan. Second, in order for the system to be deterministic, we need to specify the π_i' explicitly every time an agent replans. Formally:

Definition 5. ***Reactive Transition System*** *- given MA-STRIPS task $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ define the reactive transition system of Π as a labeled transition system $\langle Q, \rightarrow, \Sigma \rangle$ where:*

- $Q : \{\langle s, \pi_1 \ldots \pi_n \rangle \mid s \subseteq F, \forall i : \pi_i \in \langle a_1^i, \ldots, a_m^i \rangle, a_j^i \in A_i\}$
- $\Sigma = \cup_{i=1}^n (A_i \cup \{replan_i(\pi_i') \mid \pi_i' \in \langle a_1^i, \ldots, a_{m'}^i \rangle, a_j^i \in A_i\})$
- $\rightarrow$: $g \xrightarrow{a} g'$ *iff* $g, g' \in Q$, $a \in \{a_1^i \mid i = 1 \ldots n\}$, *and a transitions from g to g' as described in cases 1 - 3 above.*

Note that the number of legal transitions from some states in this transition system is exponential in $|F|$, because of the need to specify π_i'. This presents a computational challenge for reasoning about the reachability in this system. In Section 4 we present a way to avoid this problem.

We define a trajectory in the transition system described above as follows:

Definition 6. ***Valid Trajectory*** *- given MA-STRIPS task* $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$*, and its reactive transition system* $\langle Q, \Sigma, \rightarrow \rangle$*, a* valid trajectory *is a state – action sequence* $(g^0, a^1, g^1, a^2, \dots a^n, g^n)$ *such that:*

- $g^0 = (I, \pi_1 \dots \pi_n)$ *where* π_i *is agent* i*'s plan from* I*.*
- $\forall i : g^i \xrightarrow{a^{i+1}} g^{i+1}$

Then reactive robustness can be described as follows:

Definition 7. ***Reactive Robustness*** *– A social law* l *for multi-agent setting* $\Pi^l = \langle F^l, \{A_i^l\}_{i=1}^n, I^l, \{G_i^l\}_{i=1}^n \rangle$ *is reactively robust iff: for all agents* i*, for* ***all*** *valid trajectories* ξ*, execution of* ξ *achieves* $G_1^l \cup \dots \cup G_n^l$*.*

Note that the definition requires *any* sequence to lead the agents to their goals, which means any proof of robustness will have to assume an *adversarial* scheduler deciding which agent will act next, and which plan each agent chooses during replanning.

Let us examine the relation between *rational* and *reactive* robustness. We will establish a hierarchy of robustness types by proving the following theorem:

Theorem 1. *For any problem* Π *and social law* l*, rational robustness is strictly stronger than reactive robustness.*

Proof. Rephrasing the theorem, every Π under l that is rationally robust is also reactively robust, but not vice versa. If Π under l is rationally robust, it is guaranteed that *any* set of the initial plans $\pi_1 \dots \pi_n$ will be executed to the end without a replanning event. Thus, any valid trajectory ξ will be an arbitrary interleaving of agents' plans, guaranteed to achieve $G_1^l \cup \dots \cup G_n^l$ by rational robustness.

However, we present an example where reactive robustness $\not\Rightarrow$ rational robustness. Consider the following task: agents $\{Alice, Bob\}$, $F = \{r, g1, g2\}$, $G_{Alice} = g_1, G_{Bob} = g_2$, $I = r$, $A_{Alice} = \{a_1\}, A_{Bob} = \{a_2, a_3\}$, where $a_1 = \langle \emptyset, g_1, r \rangle$, $a_2 = \langle r, g_2, \emptyset \rangle$, and $a_3 = \langle \emptyset, g_2, \emptyset \rangle$. This task is not rationally robust, since if $\pi_{Alice} = (a_1), \pi_{Bob} = (a_2)$ and *Alice* performs a_1 first, Bob's plan will fail. On the other hand, Bob may replan and perform a_3 instead, achieving g_2. Thus achievement of $\{g_1, g_2\}$ is guaranteed, i.e. this task is reactively robust. □

3.4 Types of failure

Having settled on the execution model, we need to explore in detail what can disrupt robustness. We now discuss the types of failure that could occur during execution. Reactive robustness can be violated in a few ways:

- Deadend - an agent should act but does not have a defined action to execute for the current state, i.e. an agent cannot execute the next action from its current plan, and the single agent projection is unsolvable. [3]
- Deadlock - A state where no agent can perform any action, i.e. every agent is either waiting or finished and at least one agent is not finished.
- Livelock - a condition where one or more agents change the state of the system continuously, but no agent makes progress towards its goal. The oscillations of the system states continue *ad infinitum*, but the goal of some agents is never achieved.

[3] It is possible that in the future other agents' actions can make $i's$ plan for the goal possible again, but we still regard a possibility of a deadend as a violation of robustness.

Note that the types of failure in this work differ from those presented in [10]. A *rationally robust* system cannot enter a livelock, since no replanning occurs. On the other hand, in a reactive system an agent cannot fail because of a missing precondition, and will replan instead of declaring failure in that case. Formally, we define them as follows:

Definition 8. *Consider a valid trajectory* ξ*. We define the above cases as follows:*

Livelock *-* ξ *leads to a livelock if it is infinite.*

Otherwise, ξ *is finite. Denote its last state by* g^n*.*

Deadend *-* ξ *leads to a deadend if at* g^n *there exists an agent* i *such that* i*'s next planned action* $\pi_i(g^n)_{[1]}$ *is not applicable in the world state* $s(g^n)$*, i.e.* $(pre(\pi_i(g^n)_{[1]}) \setminus waitfor(\pi_i(g^n)_{[1]})) \not\subseteq s(g^n)$*, and* i*'s single agent projection is unsolvable.*

Deadlock *-* ξ *leads to a deadlock if there are no transitions from* g^n *and* $G_1 \cup \dots \cup G_n \not\subset s(g^n)$*.*

Since we want our robustness verification to be complete, we need to show that the aforementioned failures are the only possible types of failure in our setting. This is done by stating and proving the next theorem:

Theorem 2. *Let* l *be a social law for multi-agent setting* $\Pi^l = \langle F^l, \{A_i^l\}_{i=1}^n, I^l, \{G_i^l\}_{i=1}^n \rangle$*. If there does not exist a valid trajectory* ξ *which leads to a deadend, deadlock, or livelock, then* Π^l *is reactively robust.*

Proof. Consider a valid trajectory ξ formed by complete execution of a reactive transition system. If it is infinite - the system is in a livelock by definition. Otherwise, let us examine the final state g^n of ξ. If the system is not in a deadlock, there are two possibilities:

- All the agents have achieved their goals
- There exists at least one agent i that is not waiting or finished, which means it can perform an action. On one hand, if i's next planned action is applicable - i will perform it, contrary to our assumption that g^n was a final state of the system. On the other hand, if i's next action is not applicable - i will replan. If the replanning yielded a new plan - the first action of this plan will be executed, still contrary to our assumption that g^n is a final state. If the replanning did not yield a plan - the system is in a deadend.

To summarize, g^n can only be the final state of the system, if a system is in a deadlock, deadend, or every agent had achieved its goal. □

As we have established the failures that lead to a breach in robustness, in the following section we propose a way to check whether a given social law is robust by using state-space search, where the aforementioned failure modes are the goals of the search.

3.5 Robustness Verification as Reachability Problem

Having all theoretical groundwork covered, we can now describe reactive robustness verification task as a graph reachability problem. Our goal is to verify there exists a valid trajectory that leads to one of three failure modes. This approach is similar to directed model checking (for example, see [7]). Here we only describe the graph, while in the next section we proceed to describe the full solution. We now explain the vertices, edges, starting vertex and the goal vertices.

Vertices: Vertices in the graph correspond to the global states g the system can be in. The starting vertex corresponds to the starting state g^0 as appears in Definition 6.

Edges: We create a directed edge between two vertices corresponding to g, g' if there exists action a such that $g \xrightarrow{a} g'$ as per Definition 5.

Goals: We search for a counterexample to robustness, thus the reachability goals correlate with the three modes of failure described earlier. The checks for deadend and deadlock are straightforward and done exactly according to Definition 8. We assume that the plans produced by the agents have no loops, therefore are of bounded length. This restricts our transition system to be finite too. Thus, an infinite execution, and therefore livelock, can form if and only if the graph contains a reachable cycle. This directly correlates to the result shown shown by Patrizi *et. al.* [20] for planning with LTL goals.

Unfortunately, there number of global states (therefore vertices) is $n^{2^{|F|}}$, and the outdegree of each vertex can reach $2^{|F|}$. The sheer size of the graph precludes us from using straightforward approaches to solve the reachability problem. In the next section we show how to overcome this obstacle.

4 Towards Solving the Reactive Robustness

Our main tool for solving the reachability problem described in Section 3.5 is state-space search. However, the state space and the branching factor are prohibitively large, thus a straightforward search in that space will not work.

In order to achieve computational feasibility, we propose a much smaller search space, with a much smaller branching factor while preserving correctness. Specifically, we propose to search in the state space of the *original* MA-STRIPS task (with some minor technical additions), using only actions that correspond to the *original* actions of the agents.

Recall that, as illustrated by Figure 1, checking for a livelock requires keeping track of agents' internal states. Additionally, we must make sure that each action an agent executes is one it could have planned to execute when it last replanned. Thus, in order to preserve correctness, we use a sophisticated goal test, which checks whether the sequence of actions which led to the current state is a valid trajectory (Definition 6), and whether one of the failure conditions (Definition 8) occurs.

The complete compilation can be found in Section 4.2. However, we first describe the most complex part of our approach, the planning problem we use to check whether a given trajectory which visits the same world state twice, is a real livelock.

4.1 Hindsight Intent Attribution

To detect a livelock without storing internal states explicitly, we say that a trajectory leads to a *potential livelock* if it encounters the same *world* state for the second time. We then check whether this is a true livelock by finding a plan for each relevant point along the current path where an agent replanned, which justifies the current path and could lead to a livelock.

In more detail, we will require auxiliary definitions of three distinct states for each agent: s_i^0, s_i', s_i'', and one common state: s. These are defined as follows:

- s - a world state where a potential livelock occurred. By definition of the potential livelock it is a world state that occurred more than once during the execution. Therefore, we can divide the execution to periods according to the visits to s, i.e. refer to the time point of the last visit to the state s as t, and time point of a past visit as $t-1$. We repeat this procedure for each past visit to s, so we allow s to be visited between $t-1$ and t.
- s_i^0 - the last state where agent i planned before $t-1$. Can stem from either i replanning or i planning for the first time in the initial state.
- s_i' - the state where agent i *first* invoked replanning after $t-1$ and before t.
- s_i'' - the state where agent i *last* invoked replanning after $t-1$ and before t. Can be the same as s_i' if i replanned only once between $t-1$ and t.

Since true livelock is equivalent to reaching the same global state for the second time, it is sufficient to show in a potential livelock that the system *could* have been in the same global state, i.e. $\langle s, \pi_1(s_t), \ldots, \pi_n(s_t)\rangle = \langle s, \pi_1(s_{t-1}), \ldots, \pi_n(s_{t-1})\rangle$.

From here on, we look at each individual agent i and reason whether there is *a possibility* that $\pi_i(s_{t-1}) = \pi_i(s_t)$. For that, we divide the agents into three categories with regard to their behavior between s_{t-1} and s_t:

- Agents that did not perform any action between $t-1$ and t : since there was no action performed by those agents, their internal state remains the same. These will be called *irrelevant agents*.
- Agents that performed some actions between $t-1$ and t, but did not replan in this interval : their internal state *must* have changed, which means that if there is at least one agent in this category, s is not a true livelock. These will be called *advancing agents*.
- Agents that have both performed actions and replanned between $t-1$ and t: for each such agent i we propose to create a planning problem that has a solution if and only if $\pi_i(s_{t-1})$ *could* have been equal to $\pi_i(s_t)$. These will be called *loop agents*.

Any loop agent arrives at s_{t-1} with the plan it conceived at s_i^0. The suffix of this plan is $\pi_i(s_{t-1})$. A second time around, it arrives at s_t with the plan it conceived at s_i'' and has $\pi_i(s_t)$ as a suffix. This means we have to check for the existence of two plans: one from s_i^0 with some prefix up to s (which we will call $\pi_i(s_i^0, s)$) and suffix $\pi_i(s_{t-1})$, and a second plan from s_i'' via s, whose prefix we denote by $\pi_i(s_i'', s)$, such that their suffixes match. Of course, we also require the plans to be consistent with the actions already observed (i.e., executed on the current path). Figure 3 provides graphical intuition. We say that the state s had passed the *hindsight intent attribution test* iff such a pair of plans was found for *every* loop agent.

For each loop agent i independently, given its single agent projection Π_i' and the states s, s_i^0, s_i', s_i'' we will construct a classical planning problem Π_i'' that will have a solution iff a pair of plans as described above exists. First, we need both plans to have different prefixes and the same suffix. We split the plans into 4 phases: in phase 1 we follow the observed path $(\pi_i(s_i^0, s))$ from s_i^0 to s, and in phase 2 we follow the observed path $(\pi_i(s_i'', s))$ from s_i'' to s. Then, in phase 3, the plans merge and reach state s_i' following the observed path $\pi_i(s, s_i')$, and finally in phase 4, we diverge from the actions that have been observed, and simply need to find a plan that reaches the goal. To keep track of these separate plans, we create 2 copies of the state variables. Actions in phases 1 and 2 affect only one of these copies, while actions in phases 3 and 4 affect both copies. This is similar to the GRD compilation [12], except with merging instead of splitting. We denote the lengths $|\pi_i(s_i^0, s)|$ by l_1, $|\pi_i(s_i'', s)|$ by l_2, and $|\pi_i(s_i'', s)|$ by l_3, and define the hindsight intent attribution planning compilation in Figure 2.

To prove the correctness of this compilation, note that the existence of this pair of plans (e.g., a solution to Π_i) indicates that it is possible that $\pi_i(s_{t-1}) = \pi_i(s_t)$ if the agent was acting alone. Moreover, by definition of s_i'' we know that agent i did not replan from s_i'' to s, which means $\pi_i(s_i'', s)$ can be executed regardless of the plans of the other agents. This decoupling gives us the ability to reason

Given an single agent projection $\Pi = \langle F, A', I, G\rangle$ and states $s, s_i^0, s_i', s_i'',$, create a classical planning task $\Pi = \langle F'', A'', I'', G''\rangle$

- $F'' = \{f_1, f_2 \mid f \in F\} \cup \{fin\text{-}ph\text{-}k \mid k \in \{1,2,3\}\} \cup \{allow\text{-}a_j^{ph_k} \mid k \in \{1,2,3\}, j \in \{1,\ldots,l_k\}\}$
- $A'' = \{a_j^{ph_k} \mid a \in A, k \in \{1,2,3\}, j \in \{1,\ldots,l_k\}\} \cup \{a^{ph_4} \mid a \in A'\}$

  where:
  - for $k \in \{1,2\}$:

    $pre(a_j^{ph_k}) = \{f_k \mid f \in \text{pre}(a)\} \cup \{allow\text{-}a_j^{ph_k}\}$,

    $add(a_j^{ph_k}) = \{f_k \mid f \in \text{add}(a)\} \cup \begin{cases}\{allow\text{-}a_{j+1}^{ph_k}\} & j < l_k\\ \{fin\text{-}ph\text{-}k, allow\text{-}a_1^{ph_{k+1}}\} & j = l_k\end{cases}$

    $del(a_j^{ph_k}) = \{f_k \mid f \in \text{del}(a)\} \cup \{allow\text{-}a_j^{ph_k}\}$,
  - $pre(a_j^{ph_3}) = \{f_1, f_2 \mid f \in \text{pre}(a)\} \cup \{allow\text{-}a_j^{ph_3}\}$,

    $add(a_j^{ph_3}) = \{f_1, f_2 \mid f \in \text{add}(a)\} \cup \begin{cases}\{allow\text{-}a_{j+1}^{ph_3}\} & j < l_3\\ \{fin\text{-}ph\text{-}3\} & j = l_3\end{cases}$

    $del(a_j^{ph_3}) = \{f_1, f_2 \mid f \in \text{del}(a)\} \cup \{allow\text{-}a_j^{ph_3}\}$
  - $pre(a^{ph_4}) = \{f_1, f_2 \mid f \in \text{pre}(a)\} \cup \{fin\text{-}ph\text{-}3\}$

    $add(a^{ph_4}) = \{f_1, f_2 \mid f \in \text{add}(a)\}$

    $del(a^{ph_4}) = \{f_1, f_2 \mid f \in \text{del}(a)\}$
- $I'' = \{f_1 \mid f \in s_i^0\} \cup \{f_2 \mid f \in s_i''\} \cup \{allow\text{-}a_1^{ph_1}\}$
- $G'' = \{f_1, f_2 \mid f \in G\} \cup \{fin\text{-}ph\text{-}3\}$

Figure 2: Hindsight Intent Attribution Planning Compilation

about each agent independently, as we can follow the already existing interleaving of the individual plans to achieve the loop we have already observed. Thus, to reach the real livelock it is sufficient to: a) find such a pair of plans for each loop agent and b) show there are no advancing agents.

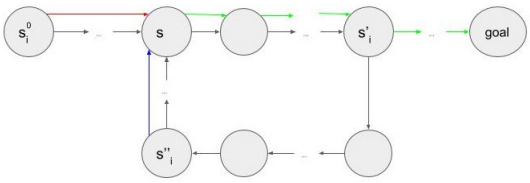


Figure 3: Graphical representation of the livelock loop from i's point of view. $\pi_i(s_i^0, s)$ is in red, $\pi_i(s_i'', s)$ is in blue, common suffix $\pi_i(s_t) = \pi_i(s_{t-1})$ is in green. Observed actions are in gray.

Theorem 3. *Let s be some state with a potential livelock (that is, search found a path π leading to s, and then looping back to s). Then the hindsight intent attribution for path π is solvable for all agents i iff there really is a true livelock possible from s.*

Proof. If a true livelock is possible from s, it implies that there exists a schedule of actions such that $\langle s, \pi_1(s_t), \ldots, \pi_n(s_t)\rangle = \langle s, \pi_1(s_{t-1}), \ldots, \pi_n(s_{t-1})\rangle$. This in turn implies that $\forall i : i \notin$ *advancing agents*. Also, for each $i \in$ *loop agents*, there exist two plans: π_i^1 planned before $t-1$ with suffix $\pi_i(s_{t-1})$, and π_i^2 planned after $t-1$ and before t, with suffix $\pi_i(s_t) = \pi_i(s_{t-1})$. Moreover, prefixes of those plans have been already executed, thus are compatible with observed actions. Thus, $\forall i : \pi_i^1, \pi_i^2$ is pair of plans that compose a solution for the hindsight intent attribution search procedure in the following way: from the points of the last replans before $t-1$ and t, the actions observed will be executed in their respective copy of facts, until both copies arrive to s. From s, $merge$ will be executed, and after that, merged version of $\pi_i(s_t)$ will complete the solution. We have shown a possible solution for each agent, therefore s passes the hindsight intent attribution test.

Proving the other side of *if and only if*, assume s passed the hindsight intent attribution test. It means, $\forall i \in agents : i \notin$ *advancing agents*. For each $i \in$ *irrelevant agents* trivially $\pi_i(s_t) = \pi_i(s_{t-1})$. For each $i \in$ *loop agents* there exist two plans found by the search procedure: $\pi_i(s_i^0, s) \cdot \pi_i(s_{t-1})$ and $\pi_i(s_i'', s) \cdot \pi_i(s_t)$, where $\cdot$ denotes concatenation, and $\pi_i(s, s')$ denotes a path from s to s'. Moreover, these plans are consistent with the observed actions, and $\pi_i(s_t) = \pi_i(s_{t-1})$ by the correctness of the search procedure. This, in turn, means that each loop agent *could* have chosen plan $\pi_i(s_i^0, s) \cdot \pi_i(s_{t-1})$ in s_i^0, and plan $\pi_i(s_i'', s) \cdot \pi_i(s_t)$ from s_i'', and still remain consistent with the actions observed, independently of other agents' plans. The independence comes from the fact that both $\pi_i(s_i^0, s)$ and $\pi_i(s_i'', s)$ were executed fully without replanning, i.e. there exists a schedule such that $\forall a \in \pi_i(s_i^0, s), \pi_i(s_i'', s)$, $pre(a)$ are fulfilled regardless of plans of other agents. Thus, there exists a schedule s.t. $(s, \pi_1(s_t), \ldots, \pi_n(s_t)) = (s, \pi_1(s_{t-1}), \ldots, \pi_n(s_{t-1}))$, which means true livelock is possible from s.

□

4.2 Complete Compilation

We present the complete compilation in Figure 4. In what follows we explain in detail the states, actions, and goals:

States – We keep track of the world states in the set F, auxiliary flags fin_i to denote that i had achieved its goal, and a vector of previously encountered replanning events $\vec{p}$ along the path, which is used in the hindsight intent attribution procedure described earlier.

Actions – we call the actions available to the planner $activate{-}action{-}a_i$, with the semantics of modelling the effect of each a_i available to the agents on the state.

Goals – The goal test examines a search node (as opposed to examining a state), and consists of two parts: in the first we verify that a path to this node is *justifiable*, i.e. the agent *could* have planned to execute the sequence of actions leading it to the node $v(s)$. For that, for agent i that performed a_i leading to $v(s)$ we check if there is a single agent projection solution that starts at the state where i last replanned, and has a prefix consistent with a's observed actions. In the second part of the goal test, we check for the failures from Definition 8. Every action may lead to a deadend, which is checked upon application of each action. The precondition for declaring deadlock follows Definition 8, and the precondition for declaring livelock is arriving to potential livelock and passing the hindsight intent attribution test.

Note that this compilation is much more compact than a reachability problem described in 3.5, both in the size of the state space and in the branching factor. The proof of correctness of the complete compilation is quite voluminous, but it closely follows the structure of the compilation itself, and is omitted for the sake of brevity. The main point of the proof is that every solution of Π^{search} is a valid trajectory, and that the goals capture their respective modes of failure correctly.

5 Empirical Evaluation

We have implemented the compilation described in Section 4, as well as a custom state space planner. The classical planning tasks created for the single agent projection and hindsight intent attribution were solved using *pyperplan*[4].

Following [10], we checked our compilation on the BLOCKSWORLD and DRIVERLOG domains under the random goal distribution procedure, where we created an instance with a different goal set for each agent by allocating each goal to one of the agents at random. Although a direct comparison to the reports provided in [10] is unwarranted, as the rational and reactive robustness are different conditions, we still can draw some parallels

[4] https://bitbucket.org/malte/pyperplan

Given an MA-STRIPS problem $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$, define a state space search problem
$\Pi^{search} = \langle S, A, Action(s), Result(s,a), Goal(v(s)) \rangle$, where $v(s)$ is a

- S: $s = \langle F, \vec{p}, \{fin_i | i = 1 \dots n\} \rangle$, where $\vec{p}$ is the vector of replanning events along the path, and fin_i is the flag signaling whether agent i achieved its goal.
- A: $\{activate{-}action{-}a_i \mid a_i \in \{A_i\}_{i=1}^n\}$.
- $Action(s)$: $activate{-}action{-}a_i$: is applicable if $\neg fin_i$.
- $Result(s,a)$ – Given action $activate{-}action{-}a_i$, resulting state s' is constructed as follows:
  - if a_i is applicable in s, i.e. $((pre(a_i) \setminus waitfor(a_i)) \in F(s)) : s' = \langle (F(s) \setminus del(a_i)) \cup add(a_i), \vec{p}(s), fin_i = True$ if $G_i \subseteq (F(s) \setminus del(a_i)) \cup add(a_i), \vec{p}(s), fin_i = False$ otherwise $\rangle$.
  - if a_i is not applicable in s:
    - if i single agent projection is solvable: $s' = \langle F(s), \vec{p}(s) \cdot (i,s), fin_i(s) \rangle$
    - else: $s' = [s, "deadend"]$.
- $Goal(v(s))$: declares $v(s)$ as a goal if $v(s)$ is *justified* and one of the following is true:
  - deadend happens:"$deadend$" in s.
  - deadlock happens: $Action(s) = \emptyset, G_1 \cup \dots \cup G_n \not\subseteq F(s)$.
  - livelock happens: $s = s'$ for some s' on the path to $v(s)$ and s passes *hindsight intent attribution test*.

Figure 4: Complete Compilation

BLOCKSWORLD			DRIVERLOG		
Problem	Time (sec)	Result	Problem	Time (sec)	Result
9-0	1.49	deadend	1	0.01	robust
9-1	22.75	deadend	2	0.08	deadlock
9-2	12.5	deadend	3	0.06	deadend
10-0	82.07	deadend	4	0.11	robust
10-1	5.28	deadend	5	0.27	deadlock
10-2	170.40	deadend	6	0.43	deadlock
11-0	294.26	deadend	7	1.15	deadlock
11-1	–	timeout	8	0.78	deadlock
11-2	–	timeout	9	0.85	deadend
12-0	–	timeout	10	0.91	robust
12-1	–	timeout	11	0.77	robust
			12	5.16	deadend
			13	47.85	deadlock
			14	16.99	deadlock
			15	81.57	deadend
			16	–	timeout
			17	–	timeout
			18	–	timeout
			19	–	timeout

Table 1: Results on BLOCKSWORLD and DRIVERLOG domains with 300 seconds timeout

since experiments for both were run on the same computers. For the BLOCKSWORLD domain, verifying reactive robustness is indeed much slower, yet in DRIVERLOG, verifying reactive robustness is sometimes faster (with the caveat that some problems were robust under the reactive setting, yet not robust for the rational setting).

Since livelocks were not detected in these instances (not because of lack thereof, just because other failures were encountered sooner), we also created a custom set of problems where encountering a livelock is guaranteed, while no other failures exist. The domain is a variant of grid navigation, where an agent can only move on a predefined grid. Movement can be performed only to four adjacent tiles. The agents are forbidden from moving to a tile occupied by another agent, and have a goal of standing on a specific tile.

The set of problems we created features two agents, a corridor of width 2 and of variable length. The agents start on the opposite corners of the corridor, and their goal is to switch places, as illustrated by Figure 3. In Table 2 we show the running time of the planner until livelock detection.

agent 1		.....		
		.....		agent 2

Figure 3: Experimental setup - grid navigation domain

Typically, the deadend and deadlock detection occurs much faster than livelock detection, thus for practical use it might be useful to decompose the procedure into two separate steps, where the check for livelock occurs only after no deadlock or deadend has been found.

Note that this evaluation is by no means exhaustive, but it shows that our approach is feasible in practice.

corridor length	5	10	15	20	25	35	45
time (sec)	0.13	3.09	6.46	15.41	28.92	151.44	–

Table 2: Results on corridor problems with 300 seconds timeout

6 Related Work

Several techniques to tackle roughly similar tasks have been made in the past, with varying degrees of success. Thus, here we provide a quick overview of the existing literature in the field.

6.1 Multi-agent coordination as FOND

A popular approach to solving various multi-agent problems is to examine the system from an agent's perspective. One can model the behavior of other agents as a non-deterministic environment (for example, see [16]), and use fully observable non-deterministic planning (FOND) to find a solution for a particular agent. Concerning the systems robustness, one can deduce that if *every* agent has a strong solution, i.e. a solution that succeeds with no regard to outcomes of the non-deterministic action effects, the system is robust.

There are several approaches to single agent non-deterministic planning, and here we will limit our discussion to two of these: *conformant* [5] and *k-fault tolerant* [6] planning.

Conformant planning is an approach to planning under uncertainty that guarantees goal achievement regardless of observations, and by extension, any combination of possible external actions. If every agent in a multi-agent system has a conformant plan, then the eventual arrival of every agent to its goal is guaranteed. On the other hand, k-fault tolerant planning is guaranteed to guide an agent to the goal under the assumption that no more than *k* faults can occur during the execution. Both k-fault tolerant and conformant planning tasks can be solved using compilation to a heuristic search [9].

Both of these methods, however, are efficient only when there are few possible outcomes for each action, while in multi-agent tasks presented as FOND there can be a lot of different outcomes for some actions. Moreover, in k-fault tolerant planning, some outcomes are considered faults, while other are considered primary (or desired). The compilation of k-fault tolerant planning to classical planning assumes each outcome has one primary effect. This is different than our setting, where other agent's actions are their own. Likewise in the multi-agent to FOND compilation, a fixed round-robin execution order between the agents is assumed, which decreases the computational effort significantly, and can not be applied to reactive robustness verification. Thus, these methods are of very limited use to us.

6.2 Robust normative systems

Agotnes *et al.* [1] explore robustness of normative systems. Normative systems are sets of constraints on agents' behavior, and essentially are a special case of social laws. Their notion of robustness is different from what appears in our work: it originates from the assumption that agents can choose not to abide by norms/laws imposed on the system. The authors proceed to explore different aspects of non-compliance and propose ways to deal with it, and define *k-robustness* as a property of a normative system which can handle up to *k* agents that do not comply, and still remain effective. We, however, are not concerned with non-compliance, and assume that agents cannot break the social law.

Agotnes and Woolridge [2] also described a setting where imposing any social law incurs cost to the system designer, but also provides some benefit. In this context, the authors proceed to describe

optimal social laws with respect to a given cost function. Thus, designing a social law becomes an optimization problem. In this paper, however, we reason about the properties of an externally given social law, and do not attempt to minimize costs of any kind.

6.3 Control via LTL

Another related line of research is LTL synthesis, which is concerned with verification and synthesis of robot control programs for tasks such as safe navigating and grasping objects (e.g., [13]). In their work, the authors try to specify the agent's behavior in response to an uncertain environment using different variations and fragments of LTL formulae. These include regular LTL, *GR1* fragment [3] and *Co-safe* linear logic [14] among others. Specifications are given as LTL formulae, and different control-generating algorithms are presented. In the multi-agent setting, however, this approach seems to be less applicable, as the execution model and goals are hard to compactly describe as LTL formula. The hardness stems from the fact that we allow replanning under very specific conditions only. These conditions include keeping track of replanning events, currently executing plan, and so on. We conjecture that the size of the resulting LTL formula needed to capture these conditions will be exponential, and hope to prove this conjecture in a follow-up work.

7 Discussion and Conclusion

We have described an execution model which allows agents to adjust their plans *online* via replanning. We then proposed and formally described the notion of reactive robustness of a system under a social law. We also described the problem of checking reactive robustness property as a graph reachability problem. Finally, we have shown how to verify robustness via a compilation to a state space search problem, while keeping the state relatively compact. Finally we have presented a proof-of-concept empirical evaluation, and have shown that our procedure works on some benchmark problems.

In a recent work [18], authors propose a method to procedurally synthesise subset of rationally robust social laws given a problem description. We will proceed to tackle the challenge on generating reactively robust social laws in the follow-up work.

REFERENCES

[1] Thomas Agotnes, Wiebe van der Hoek, and Michael Wooldridge, 'Robust normative systems', in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '08, pp. 747–754, Richland, SC, (2008). International Foundation for Autonomous Agents and Multiagent Systems.

[2] Thomas Ågotnes and Michael Wooldridge, 'Optimal social laws', in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 667–674. International Foundation for Autonomous Agents and Multiagent Systems, (2010).

[3] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Saar, 'Synthesis of reactive (1) designs', *Journal of Computer and System Sciences*, **78**(3), 911–938, (2012).

[4] Ronen I Brafman and Carmel Domshlak, 'From one to many: Planning for loosely coupled multi-agent systems.', in *ICAPS*, pp. 28–35, (2008).

[5] Alessandro Cimatti and Marco Roveri, 'Conformant planning via symbolic model checking', *Journal of Artificial Intelligence Research*, **13**, 305–338, (2000).

[6] Carmel Domshlak, 'Fault tolerant planning: Complexity and compilation.', in *ICAPS*, (2013).

[7] Stefan Edelkamp, Alberto Lluch Lafuente, and Stefan Leue, 'Directed explicit model checking with hsf-spin', in *Proceedings of the 8th international SPIN workshop on Model checking of software*, pp. 57–79. Springer-Verlag, (2001).

[8] Michael Georgeff, 'Communication and interaction in multi-agent planning', in *Readings in distributed artificial intelligence*, 200–204, Elsevier, (1988).

[9] Jörg Hoffmann and Ronen I Brafman, 'Conformant planning via heuristic forward search: A new approach', *Artificial Intelligence*, **170**(6-7), 507–541, (2006).

[10] Erez Karpas, Alexander Shleyfman, and Moshe Tennenholtz, 'Automated verification of social law robustness in strips', in *Twenty-Seventh International Conference on Automated Planning and Scheduling*, (2017).

[11] Robert M Keller, 'Formal verification of parallel programs', *Communications of the ACM*, **19**(7), 371–384, (1976).

[12] Sarah Keren, Avigdor Gal, and Erez Karpas, 'Goal recognition design.', in *ICAPS*, (2014).

[13] Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman, 'Synthesis for robots: Guarantees and feedback for robot behavior', *Annual Review of Control, Robotics, and Autonomous Systems*, **1**, 211–236, (2018).

[14] Orna Kupferman and Moshe Y Vardi, 'Model checking of safety properties', *Formal Methods in System Design*, **19**(3), 291–314, (2001).

[15] Yoram Moses and Moshe Tennenholtz, 'Artificial social systems', *Computers and Artificial Intelligence*, **14**, 533–562, (1995).

[16] Christian J Muise, Paolo Felli, Tim Miller, Adrian R Pearce, and Liz Sonenberg, 'Planning for a single agent in a multi-agent environment using fond.', in *IJCAI*, pp. 3206–3212, (2016).

[17] Ronen Nir and Erez Karpas, 'Automated verification of social laws for continuous time multi-robot systems', in *AAAI*, (2019).

[18] Ronen Nir, Alexander Shleyfman, and Erez Karpas, 'Automated synthesis of social laws in strips', in *AAAI*, (2020).

[19] Raz Nissim and Ronen I Brafman, 'Multi-agent A* for parallel and distributed systems', in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pp. 1265–1266. International Foundation for Autonomous Agents and Multiagent Systems, (2012).

[20] Fabio Patrizi, Nir Lipovetzky, Giuseppe De Giacomo, and Hector Geffner, 'Computing infinite plans for LTL goals using a classical planner', in *IJCAI*, (2011).

[21] Jean-Jacques Rousseau. The social contract, 1762, 1762.

[22] Stuart J Russell and Peter Norvig, *Artificial intelligence: a modern approach*, Malaysia; Pearson Education Limited,, 2016.

[23] Yoav Shoham and Moshe Tennenholtz, 'On the synthesis of useful social laws for artificial agent societies (preliminary report)', in *AAAI*, pp. 276–281, (1992).

[24] Yoav Shoham and Moshe Tennenholtz, 'On social laws for artificial agent societies: off-line design', *Artificial intelligence*, **73**(1-2), 231–252, (1995).

[25] The Oxford Dictionary of Phrase and Fable. Buridan's ass, 2006. [Online; accessed 22-August-2019].