The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Automated Verification of Social Laws for Continuous Time Multi-Robot Systems

Ronen Nir, Erez Karpas
Technion — Israel Institute of Technology
Haifa, Israel
ronen.nir@outlook.com, karpase@technion.ac.il

Abstract

Designing multi-agent systems, where several agents work in a shared environment, requires coordinating between the agents so they do not interfere with each other. One of the canonical approaches to coordinating agents is enacting a social law, which applies restrictions on agents' available actions. A good social law prevents the agents from interfering with each other, while still allowing all of them to achieve their goals. Recent work took the first step towards reasoning about social laws using automated planning and showed how to verify if a given social law is robust, that is, allows all agents to achieve their goals regardless of what the other agents do. This work relied on a classical planning formalism, which assumed actions are instantaneous and some external scheduler chooses which agent acts next. However, this work is not directly applicable to multi-robot systems, because in the real world actions take time and the agents can act concurrently. In this paper, we show how the robustness of a social law in a continuous time setting can be verified through compilation to temporal planning. We demonstrate our work both theoretically and on real robots.

Introduction

One of the main challenges of designing systems with multiple autonomous agents is coordinating between all agents acting in a shared environment. Each agent has a goal and a set of actions it can perform and when the agent acts it's actions can affect other agents. Thus, without coordination, agents must come up with a contingent plan, which must consider all possible actions of all other agents. This can be solved using non-deterministic planning (Muise et al. 2016), which unfortunately does not scale very well. There are also no examples of non-deterministic planners in continuous time settings, the settings we consider in this paper.

There are several methods to achieve coordination. One, for example, is to establish a centralized controller, which controls all agents, ensuring that each achieves its goal. However, this method fails to scale well for a large number of agents. A different method is to apply some restrictions on the agents' actions. These restrictions are designed to make the agents mutually compatible. The union of these restrictions is called a "social law", and a system that is obliged to follow this law is called an "artificial social system" (Tennenholtz 1991; Shoham and Tennenholtz 1992a; 1992b; 1995; Moses and Tennenholtz 1995). In previous work (Karpas, Shleyfman, and Tennenholtz 2017), a social law is called *robust* if it guarantees that each agent can choose any plan it wants to achieve its goal, and the plans of different agents will not interfere with each other. This work took a first step towards reasoning about social laws using automated planning, by describing a formalism to represent social laws and a criteria for social law robustness, as well as an algorithm to verify whether a given social law is robust by compilation to classical planning.

It is important to note however, that this work assumed that actions are instantaneous, and that some external scheduler chooses which agent acts next, and thus is not applicable to real multi-robot systems where actions take time and robots can act concurrently. In fact, assuming that only one single robot acts at any given moment requires a centralized controller, which defeats the purpose of social law.

In this paper, we address a more realistic setting, where actions have durations, and agents can act concurrently. We assume the plan of each agent is a sequence of actions. The result of agents acting together is a schedule. The contributions of this paper are threefold: First, we describe a new notion of robustness of social laws in a continuous time setting where agents act concurrently. Second, we describe a compilation to temporal planning, which can verify whether a given social law is robust in such a setting. Finally, we provide an empirical evaluation which demonstrates our compilation on a real multi-robot system as well as on several PDDL 2.1 domains.

Preliminaries

We define a temporal multi-agent planning setting, which is built around a combination of MA-STRIPS (Brafman and Domshlak 2008) and PDDL 2.1 (Fox and Long 2003). As in previous work on social laws (Karpas, Shleyfman, and Tennenholtz 2017), this is different than traditional multi-agent planning in that each agent has its own goal, and we assume that each agent can achieve its goal alone, that is, cooperation is not needed. Moreover, we assume that an agent is capable of performing only one action at a time. Formally, a temporal multi-agent planning setting is a tuple $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$, where: F is the set of facts

used to describe the world, A_i is the set of actions of agent i (for $i = 1 \dots n$), $I \subseteq F$ is the initial state and $G_i \subseteq F$ is the goal of agent i.

Unlike previous work (Karpas, Shleyfman, and Tennenholtz 2017), we allow *durative* actions (Fox and Long 2003) and consider that agents can execute actions concurrently, A durative action a is described by its minimum and maximum duration $d_{min}(a)$ and $d_{max}(a)$, its start condition $cond_s(a) \subseteq F$, its invariant condition $cond_i(a) \subseteq F$, and its end condition $cond_e(a) \subseteq F$, as well as its start effect $eff_s(a)$ and its end effect $eff_e(a)$ which are both divided into add effects ($add_s(a) \subseteq F$ and $add_e(a) \subseteq F$, respectively) and delete effects ($del_s(a) \subseteq F$ and $del_e(a) \subseteq F$, respectively).

A solution to a temporal multi-agent setting is a schedule τ which consists of a set of triples $\langle a, t, d \rangle$ where a is an action, $t \in \mathbb{R}^{0+}$ is the time the action starts, and $d \in [d_{min}(a), d_{max}(a)]$ is the duration of the action. Basically, a schedule is valid if it respects all action conditions and achieves all the agents' goals in the end state. For action conditions, we require that $cond_s(a)$ holds at time t, $cond_e(a)$ holds at time $t + d$ and that $cond_i(a)$ holds from $t + \epsilon$ until $t + d$ continuously. Finally, we require that G_i is satisfied by the state which the schedule reaches after all actions have been completed, for all agents i.

Social law l is based on the restrictions it applies to the agents' actions. Thus, it seems that it can be formally described only by the modifications it applies to each action $a \in \cup\{A_i\}_{i=1}^n$. It turns out this is not the case, some restrictions require more complex conditions that cannot be described with the original facts set F. Other restrictions are based on a wait-for mechanism which requires an agent to wait for a certain condition $f \in F$ to hold before it performs an action a. Therefore, social law l is described by the following: (a) The modifications it applies to the agents' actions. (b) The modifications it applies to the facts, the initial state or the goals. (c) it's annotations of certain $f \in cond_s(a)$ as a wait-for precondition. We denote the wait-for conditions of action a by $cond_s^w(a)$, and the other start conditions by $cond_s^f(a)$. We assume these modifications make sense and do not make the problem meaningless (e.g., by changing the goal to an empty set). Flagging $f \in cond_s(a)$ as a wait-for precondition requires a more in-depth discussion which will take place in the following section.

Waiting in Temporal Social Laws

An agent is waiting for certain facts $f \in cond_s^w(a)$ if it avoids performing a by being inactive until these facts hold. For example, consider two agents moving in a grid. If an agent tries to enter an occupied position it will certainly fail, The agent can avoid such a failure by waiting for the position to be vacated before it tries to enter. To implement this a social law can flag certain facts $f \in cond_s(a)$ as a wait-for precondition and thus, an agent who respects the social law will not perform a until f holds.

One might also consider annotating also some end conditions as wait-for simply by flagging some $f \in cond_e(a)$ as a wait-for precondition. However, if we annotate an end condition of some action as wait-for, the action's duration becomes uncontrollable. This makes planning much more complicated and requires us to choose which notions of controllability make sense: strong controllability (Vidal and Fargier 1999), dynamic controllability (Morris 2006), or some hybrid between them. Thus, in this paper we only consider the case where an agent can wait to *start* an action, and reserve handling waiting for end conditions for future work.

Although, waiting is effective in terms of preventing certain types of failures, but it is also possible for it to lead to a *deadlock*. We will declare *deadlock* when there is an agent who waits for a certain $f \in cond_s(a)$ and it is guaranteed that f will never hold and thus, this agent will wait forever.

We now describe what is a robust social law in the temporal setting we propose.

Properties Of Temporal Social Laws

A robust social law enables coordination by ensuring each agent i is safe from any interference by his fellow agents, as long as all agents pursue their own goal and follow this law. Of course, to design an automatic robustness verification tool we need more formal descriptions about the possible interference and how exactly agents pursue their goal.

First, we define the projection of a multi-agent temporal planning setting $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ for agent i as the temporal planning problem $\Pi_i = \langle F, A_i, I, G_i \rangle$. To achieve G_i each agent comes up with an individual sequential plan $\pi_i = \{\langle a_j, d_j \rangle\}_{j=1}^k$, where $a_j \in A_i$ is an action and $d_j \in [dur^{min}(a_j), dur^{max}(a_j)]$ is its duration, the index $j = 1, 2, ..., k$ denotes the actions execution order. We now define what is a possible schedule for a individual sequential plan.

Definition 1. *τ is a possible schedule for a individual sequential plan π_i, that is $\tau = \langle \tau_1, \tau_1, ..., \tau_k \rangle$ so that $\tau_j = \langle a_j, t_j, d_j \rangle$ if the following conditions hold: (a) all wait-for conditions of an action $a_j \in \tau_j$ are met at time t_j and, (b) the sequential nature of π_i is kept, meaning $t_{i+1} > t_i + d_i$ for all $i = 1...(j-1)$.*

In a possible joint schedule, which describes what might actually happen in a multi agent system, we assume that: (a) All agents respect the social law and its wait-for conditions. (b) There are no two conflicting instantaneous happenings (start or end effects of an action). Note that although instantaneous happenings are theoretically possible, we argue that this requires perfect synchronization between different agents, which is not feasible. (c) When different agents are waiting for some common fact f which is achieved at a certain time, the decision about which one starts first is arbitrary, and thus, treated as adversarial.

Definition 2. *Given a set of individual sequential plans for the agents $\{\pi_i\}_{i=1}^n$, τ^u is a* possible joint schedule *iff $\tau^u = \langle \tau_1, \tau_2, ..., \tau_{k^u} \rangle$, so that $\tau_j \in \tau^u$ is $\tau_j = \langle a_j, t_j, d_j \rangle$ and the following conditions apply. (a) the projection of τ^u on an agent i is an individual possible schedule. (b) all wait-for conditions of any action $a_j \in \tau_j$ are met at time t_j. (c) there are no $\{j, k\}$ such that $t_j = t_k$ or $t_j = t_k + d_k$.*

The main reason we allow agents to perform only one action at a time is that concurrent actions might cause uncontrollable action durations which are very hard to handle.

Previous work (Karpas, Shleyfman, and Tennenholtz 2017) defines two notions of robustness: rational robustness, which assumes each agent is rational and wants to achieve its goal eventually, and adversarial robustness, which assumes all other agents want to interfere and do not care about achieving their goal. It was also shown that verifying rational robustness is the more general case, and thus in this paper we only focus on rational robustness. We now define rational robustness, with the necessary modifications:

Definition 3. *A social law l in temporal multi-agent setting $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ is robust to* rational *iff for all agents $i = 1 \dots n$, for all possible solutions π_i to Π_i, for any possible joint schedule τ^u consisting of $\{\pi_i\}_{i=1}^n$, the state after executing τ^u satisfies $G_1 \cup \dots \cup G_n$.*

Verifying Rational Robustness

With a definition of rational robustness in hand, we now describe an algorithm that verifies if a social law l in a given multi-agent temporal planning problem Π^l is rationally robust. The algorithm we present is based on a compilation to temporal planning. Specifically, the input to our algorithm is Π^l, a temporal multi-agent planning problem with a certain social law that is already encoded in it. Our compilation creates a temporal planning problem Π' such that Π' is unsolvable iff Π^l is robust to rational.

We construct $\Pi' = \langle F', V', A', I', G' \rangle$ such that any solution to Π' will be composed of individual plans π_i for each agent i, as well as a joint schedule for these individual plans, such that π_i solves the projection of Π^l on agent i alone, but the joint schedule fails.

The possible causes for the joint schedule failing are: (a) FAILURE: An agent executed action a without meeting one of its conditions (either start, end, or invariant). (b) DEADLOCK: An agent is waiting forever for a fact f. (c) GOAL NOT ACHIEVED: Agent i did not achieve its goal at the end of the joint schedule.

We begin with a high-level description of Π'. Π' keeps track of the state of the world for each agent i had it been acting alone, and of the global state of the world in which all agents act together. Thus, for each fact $f \in F$, we create $n + 1$ facts in Π': f_i (for $i = 1 \dots n$) which describes the value of f had agent i been acting alone, and f_g which keeps track of the value of f in the joint schedule. The goal of the compilation is to achieve the fact $failure$ which indicates that the joint schedule failed.

For each action a we create multiple copies, corresponding to several possible outcomes. We have one successful version of a and one version of a to flag the deadlock situation. Other copies of a deal with the following outcomes: (a) a fails to meet its start condition. (b) a fails to meet its invariant condition immediately after it starts. (c) a fails to meet its end condition. (d) a deletes the invariant condition of a currently executing action when it starts. (e) a deletes the invariant condition of a currently executing action when it ends. The duration of a, that is $dur(a)$ remains the same in all actions' copies.

While handling most of these outcomes can be achieved in a similar manner to the previous work (Karpas, Shleyfman, and Tennenholtz 2017), a few require extra insights which are only relevant to the continuous time setting. First, the invariant condition of action $cond_i(a)$ could be violated by some other action a'. Therefore, we check if there is an agent that can violate an invariant condition of some other agent's current action. If that is the case, we achieve $failure$. An important point here is that in order to keep track of how many actions with invariant condition f are currently being executed we use a numerical variable inv_f.

Additionally, this compilation differs from the previous work (Karpas, Shleyfman, and Tennenholtz 2017) in the way it handles waiting. As mentioned, we assume the agents act without any scheduler. Moreover, we assume that a waiting agent will act as soon as his wait-for condition holds. Therefore, before our compilation declares that an agent that wants to execute action a with wait-for condition f will wait forever it needs to ensure that f will never hold. In the previous work (Karpas, Shleyfman, and Tennenholtz 2017), the assumption that there is an adversarial external scheduler who controls the agents made exploiting the possibility of a deadlock fairly easy — once an agent started waiting it could be made to wait until all other agents have finished acting. This allowed other agents to achieve f as long as they deleted f before the end. In the following subsections we define the compilation formally.

Facts, Initial State and Goal State

We begin our formal description of the compilation by listing its set of facts, F'. First, as mentioned above, for each fact $f \in F$, F' contains f_g, a copy of f for keeping track of the global state of the world, as well as f_i for each agent i, a copy of f for keeping track of the state of the world assuming agent i acts alone. Furthermore, the compilation indicates a deadlock in which agent i waits for fact f forever with the fact $wt_{f,i}$. The compilation also include the fact $waiting_i$, which indicates agent i is waiting forever for some fact to become true. Additionally, for each agent i the fact fin_i indicates whether agent i finished executing its plan. The proposition act is used to indicate that agents are free to act. It is initially true, and becomes false once the agents finish executing "real" actions, allowing only "bookkeeping" actions to be executed. Finally, $failure$ indicates there was some kind of failure, and thus the social law is not robust, and is part of the goal.

As mentioned above, we also need the numeric variable inv_f, which counts the number of currently executing actions that have f as an invariant condition, for each fact f. We remark that n is an upper bound on the number of concurrently executing actions, and thus it is possible to replace inv_f with a set of n propositions. However, in our implementation we used a numerical variable, as the planner we used supports them.

The goal of the compilation is to find individual plans which work alone, and thus fin_i is part of the goal for each agent i. As we will describe later, fin_i is achieved by two possible end_i actions, which have G_i as a precondition and

fin_i as an effect, thus ensuring each agent achieves its goal. Additionally, we want the joint schedule to fail, and thus $failure$ is also part of the goal.

Finally, we describe the initial state of the compilation, I'. First, each fact f is initialized with its value in I for all of its copies (f_i for all agents i and f_g). Additionally, act is initially true, enabling agents to execute "real" actions. Finally, we initialize inv_f to be 0.

$$F' = \{failure, act\} \cup \{f_1 ... f_n | f \in F\} \cup \{f_g | f \in F\} \cup$$
$$\{wt_{f,i} | f \in F, i = 1...n\} \cup \{fin_i | i = 1...n\} \cup \{waiting^i | i = 1...n\}$$
$$V' = \{inv_f | f \in F\}$$
$$G' = \{failure\} \cup \{fin_i | i = 1...n\}$$
$$I' = \{act\} \cup \{f_i | f \in I, i = 1...n\} \cup \{f_g | f \in I\} \cup \{inv_f = 0 | f \in F\}$$

Actions

We now turn our attention to describing the actions in the compilation, beginning with the success version of action a, denoted a_s. This version of a is only applicable when: (a) Both f_i and f_g satisfy all conditions of a (start, invariant, and end); (b) a does not delete any fact f that is an invariant of an action which is currently executing (this is checked both at start and at end); and (c) If $f \in add^{start} \cup add^{end}$ then no other agent is waiting for f forever ($wt_{f,j}$).

The effects of this version of the action affect both the local copy f_i and the global copy f_g of each affected fact f. In order to ensure no other agent deletes the invariant conditions of a, we also increase inv_f for each $f \in cond_i(a_i^s)$.

$$cond_s(a_i^s) = act \wedge (\wedge_{f \in cond_s^f(a)}(f_i \wedge f_g)) \wedge$$
$$\left(\wedge_{f \in cond_s^w(a)}(f_i \wedge f_g)\right) \wedge (inv_f = 0 | f \in del_s(a)) \wedge$$
$$(\neg wt_{f,i} | f \in add_s(a), i = 1...k) \wedge (\neg waiting_i)$$
$$cond_i(a_i^s) = act \wedge \left(\wedge_{f \in cond_i(a)}(f_i \wedge f_g)\right) \wedge$$
$$(\neg wt_{f,i} | f \in add_e(a), i = 1...k)$$
$$cond_e(a_i^s) = act \wedge \left(\wedge_{f \in cond_e(a)}(f_i \wedge f_g)\right) \wedge$$
$$(inv_f = 0 | f \in del_e(a)) \wedge (\neg wt_{f,i} | f \in add_e(a), i = 1...k)$$
$$add_s(a_i^s) = \{f_i, f_g | f \in add_s(a)\}$$
$$del_s(a_i^s) = \{f_i, f_g | f \in del_s(a)\}$$
$$num_s(a_i^s) = \{increase(inv_f, 1) | f \in cond_i(a)\}$$
$$add_e(a_i^s) = \{f_i, f_g | f \in add_e(a)\}$$
$$del_e(a_i^s) = \{f_i, f_g | f \in del_e(a)\} \cup$$
$$num_e(a_i^s) = \{decrease(inv_f, 1) | f \in cond_i(a)\}$$

Next, we describe the "simple" failure versions of each action a: a^{fstart}, in which one of the non wait-for start conditions does not hold when a starts, a^{finv}, in which the invariant condition does not hold immediately after a starts, and a_i^{fend}, in which one of the end conditions does not hold when a ends. To make sure the failure is a real one, the conditions of these actions check that one of the (non-wait) conditions indeed does not hold at the appropriate time (start or end). To check whether the invariant condition does not hold immediately after a starts, we consider only the facts in the invariant conditions which are *not* achieved by the start effect. Finally, all of these actions affect f_i as if a succeeded, but only affect f_g before the failure occurs. Additionally, these actions achieve $failure$.

$$cond_s(a_i^{fstart}) = act \wedge \left(\wedge_{f \in cond_s^w(a)}(f_i \wedge f_g)\right) \wedge$$
$$(\wedge_{f \in cond_s^f(a)}(f_i)) \wedge (\vee_{f \in cond_s^f(a)}(\neg f_g)) \wedge (\neg waiting_i)$$
$$cond_i(a_i^{fstart}) = act \wedge \left(\wedge_{f \in cond_i(a)}(f_i)\right)$$
$$cond_e(a_i^{fstart}) = act \wedge (\wedge_{f \in cond_e(a)}(f_i))$$
$$add_s(a_i^{fstart}) = \{failure\} \cup \{f_i | f \in add_s(a)\}$$
$$del_s(a_i^{fstart}) = \{f_i | f \in del_s(a)\}$$
$$add_e(a_i^{fstart}) = \{f_i | f \in add_e(a)\}$$
$$del_e(a_i^{fstart}) = \{f_i | f \in del_e(a)\}$$

$$cond_s(a_i^{finv}) = act \wedge \left(\wedge_{f \in cond_s^f(a)}(f_i \wedge f_g)\right) \wedge$$
$$\left(\wedge_{f \in cond_s^w(a)}(f_i \wedge f_g)\right) \wedge \left(\vee_{f \in cond_i(a) \setminus add_s(a)}(\neg f_g)\right)$$
$$(inv_f = 0 | f \in del_s(a)) \wedge$$
$$(\neg wt_{f,i} | f \in add_s(a), i = 1...k) \wedge (\neg waiting_i)$$
$$cond_i(a_i^{finv}) = act \wedge \left(\wedge_{f \in cond_i(a)}(f_i)\right)$$
$$cond_e(a_i^{finv}) = act \wedge (\wedge_{f \in cond_e(a)}(f_i))$$
$$add_s(a_i^{finv}) = \{f_i, f_g | f \in add_s(a)\} \cup \{failure\}$$
$$del_s(a_i^{finv}) = \{f_i, f_g | f \in del_s(a)\}$$
$$add_e(a_i^{finv}) = \{f_i | f \in add_e(a)\}$$
$$del_e(a_i^{finv}) = \{f_i | f \in del_e(a)\}$$

$$cond_s(a_i^{fend}) = cond_s(a^s)$$
$$cond_i(a_i^{fend}) = act \wedge \left(\wedge_{f \in cond_i(a)}(f_i \wedge f_g)\right) \wedge$$
$$(\neg wt_{f,i} | f \in add_e(a), i = 1...k)$$
$$cond_e(a_i^{fend}) = act \wedge (\wedge_{f \in cond_e(a)}(f_i)) \wedge$$
$$(\vee_{f \in cond_e(a)}(\neg f_g))$$
$$add_s(a_i^{fend}) = \{f_i, f_g | f \in add_s(a)\}$$
$$del_s(a_i^{fend}) = \{f_i, f_g | f \in del_s(a)\}$$
$$num_s(a_i^{fend}) = \{increase(inv_f, 1) | f \in cond_i(a)\}$$
$$add_e(a_i^{fend}) = \{failure\} \cup \{f_i | f \in add_e(a)\}$$
$$del_e(a_i^{fend}) = \{f_i | f \in del_e(a)\}$$

We now address the case where an action deletes an invariant condition of a currently executing action. $a_i^{finv_x^{start}}$ is a version of a which deletes x at the start, and $a_i^{finv_x^{end}}$ is a version of a which deletes x at the end. Both check that another action with invariant condition x is executing when x is deleted, using the condition $inv_x > 0$. These actions achieve $failure$, affect f_i as if they succeeded, and affect f_g normally until the moment of failure.

$$cond_s(a_i^{finv_x^{start}}) = act \wedge \left(\wedge_{f \in cond_s^f(a)}(f_i \wedge f_g)\right)$$
$$\left(\wedge_{f \in cond_s^w(a)}(f_i \wedge f_g)\right) \wedge (\vee_{x \in del_s(a)} inv_x > 0) (\neg waiting_i)$$
$$cond_i(a_i^{finv_x^{start}}) = act \wedge \left(\wedge_{f \in cond_i(a)}(f_i)\right)$$
$$cond_e(a_i^{finv_x^{start}}) = act \wedge (\wedge_{f \in cond_e(a)}(f_i))$$
$$add_s(a_i^{finv_x^{start}}) = \{failure\} \cup \{f_i | f \in add_s(a)\}$$
$$del_s(a_i^{finv_x^{start}}) = \{f_i | f \in del_s(a)\}$$
$$add_e(a_i^{finv_x^{start}}) = \{f_i | f \in add_e(a)\}$$
$$del_e(a_i^{finv_x^{start}}) = \{f_i | f \in del_e(a)\}$$

$$cond_s(a_i^{finv_x^{end}}) = cond_s(a^s)$$
$$cond_i(a_i^{finv_x^{end}}) = act \wedge \left(\wedge_{f \in cond_i(a)}(f_i \wedge f_g)\right) \wedge$$
$$(\neg wt_{f,i} | f \in add_e(a), i = 1...n)$$
$$cond_e(a_i^{finv_x^{end}}) = act \wedge (\wedge_{f \in cond_e(a)}(f_i \wedge f_g)) \wedge$$
$$(\vee_{x \in del_e(a)} inv_x > 0)$$
$$add_s(a_i^{finv_x^{end}}) = \{f_i, f_g | f \in add_s(a)\}$$
$$del_s(a_i^{finv_x^{end}}) = \{f_i, f_g | f \in del_s(a)\}$$
$$num_s(a_i^{finv_x^{end}}) = \{increase(inv_f, 1) | f \in cond_i(a)\}$$

To conclude the versions of the "real" actions, we describe the deadlock version of a, $a_i^{w,x}$, which is used to indicate that agent i will wait for a certain wait-for condition x forever. This version of a adds $wt_{i,x}$ at the start, which forces all other agents to *not* achieve x_g without failing before x_g is achieved. This is done by including $\neg wt_{x,i}$ in the conditions of all events (start or end) of all other actions that would have achieved x_g otherwise. Other than that, $a_i^{w,x}$ affects f_i normally, and does not affect f_g at all (as a will not be executed in the "real world"), and achieves $failure$.

$$cond_s(a_i^{w,x}) = act \wedge \left(\wedge_{f \in cond_s^w(a)}(f_i)\right) \wedge \left(\wedge_{f \in cond_s^f(a)}(f_i)\right)$$
$$\wedge \left(\neg x_g | x \in cond_s^w(a)\right) \wedge (\neg waiting_i)$$
$$cond_i(a_i^{w,x}) = act \wedge \left(\wedge_{f \in cond_i(a)}(f_i)\right)$$
$$cond_e(a_i^{w,x}) = act \wedge (\wedge_{f \in cond_e(a)}(f_i))$$
$$add_s(a_i^{w,x}) = \{failure, wt_{x,i}, waiting^i\} \cup \{f_i | f \in add_s(a)\}$$
$$del_s(a_i^{w,x}) = \{f_i | f \in del_s(a)\}$$
$$add_e(a_i^{w,x}) = \{f_i | f \in add_e(a)\}$$
$$del_e(a_i^{w,x}) = \{f_i | f \in del_e(a)\}$$

To allow an agent that is waiting forever to complete its local plan, we also include $a_i^{waiting}$, which is a version of a that can only be applied if agent i is waiting forever (as indicated by $waiting^i$), and only checks and affects the local copy of the state, in the f_i facts. In order to make sure waiting agents do not execute any actions that affect the global state, all other "real" actions have a start condition that $waiting^i$ is false.

$$cond_s(a_i^{waiting}) = act \wedge (\wedge_{f \in cond_s(a)}(f_i)) \wedge \left(waiting^i\right)$$
$$cond_i(a_i^{waiting}) = act \wedge \left(\wedge_{f \in cond_i(a)}(f_i)\right)$$
$$cond_e(a_i^{waiting}) = act \wedge (\wedge_{f \in cond_e(a)}(f_i))$$
$$add_s(a_i^{waiting}) = \{f_i | f \in add_s(a)\}$$
$$del_s(a_i^{waiting}) = \{f_i | f \in del_s(a)\}$$
$$add_e(a_i^{waiting}) = \{f_i | f \in add_e(a)\}$$
$$del_e(a_i^{waiting}) = \{f_i | f \in del_e(a)\}$$

Finally, we describe the end_i actions, which are non-durative actions used for book-keeping. For each agent i, we have two versions of the action end_i^s which corresponds to the agent achieving its goal in the real world, and end_i^f which corresponds to the agent failing to achieve its goal in the real world. Both of these actions have the agent's goal, G_i as a precondition in f_i, and the success version also checks if G_i holds in f_g. These actions are used to ensure that once an agent achieves its goal, it will not be deleted by another agent. Thus, the end_i actions also delete act, which allows only end_i actions to be executed afterwards.

$$pre(end_i^s) = \neg fin_i \wedge (\wedge_{f \in G_i} f_i) \wedge (\wedge_{f \in G_i} f_g)$$
$$add(end_i^s) = \{fin_i\}$$
$$del(end_i^s) = \{act\}$$
$$pre(end_i^f) = \neg fin_i \wedge (\wedge_{f \in G_i} f_i) \wedge (\vee_{f \in G_i} \neg f_g)$$
$$add(end_i^f) = \{fin_i\} \wedge \{failure\}$$
$$del(end_i^f) = \{act\}$$

It is easy to see that the compilation is a polynomial size blow-up and can be constructed in polynomial time. In fact, the process of compiling a domain is very fast. The proof of correctness of this compilation follows the structure of the compilation very closely.

Compilation Correctness

The algorithm we presented can compile the verification of robustness decision problem of Π, into a temporal planning problem Π'. Specifically, if Π is a multi-agent temporal planning problem with social law l; Π_i is the projection of Π on agent i; Π_i is solvable for any agent $i = 1 \ldots n$ and Π' is the compilation of Π. Then we argue that Π' is not solvable iff Π consists a robust social law.

As a first step we will show that if Π is robust then Π' is not solvable. The main steps in favor of this proof will be: (1) Let Π be robust and assume that Π' is solvable so that π' is its solution and τ' is a possible schedule for π'. (2) We will show that it is possible to: (2a) Split τ' into individual schedules τ'_i for every agent i. (2b) Reduce τ'_i to τ_i , an individual possible schedule to Π_i for agent i. (3) We will show that the joint schedule τ^u that is made out from $\{\tau_i\}_{i=1}^n$ is an illegal schedule and as such it will not satisfy $G_1 \cup ... \cup G_n$ making Π not robust. (4) However, Π is robust and therefore we conclude that π' does not exist, so the problem Π' is not solvable. Sections (1 - 2a) are trivial. To split τ' in the way described in (2a) one can simply pick the triplets that has attribution to agent i out of τ' and then construct τ_i. Let us proceed by proving (2b).

Theorem 1. *Assume, Π is a multi-agent temporal planning problem, solvable for all agents i; Π' is compiled from Π; τ' is a possible schedule for Π' and τ'_i is a possible schedule for Π'_i Then, τ'_i can be reduced to τ_i that is a possible individual schedule to Π_i.*

Proof. Let $\tau' = \{\tau'_1, \tau'_2 \ldots \tau'_k\}$ be a solution to $\Pi' = \{F', A', I', G'\}$. Every triples $\tau'_j = \langle a'_j, t'_j, d'_j \rangle$ in τ' has a certain attribution to some agent i. Let us denote the projection of Π' on $\{f_i | f \in F\} \cup \{fin_i\}$ by Π'_i . Notice that: (1) Since Π'_i is an abstraction of Π' any τ', a possible schedule for Π', is also a possible schedule for Π'_i; (2) Π'_i is only a simple extension of Π_i differing as follows: (2a) The goal of Π'_i is fin_i , the goal of Π_i is $\{f \in G_i\}$; (2b) There is an additional action in Π'_i , $end_i \in A'_i$. This action achieves fin_i (every other action is degenerated to the simple original action $a_i \in A_i$); (3) The action $end_i \in A'_i$ preconditions are $\bigwedge_{f \in G_i} f_i$ and (4) The only actions that modifies f_i are made by agent i. So, basically if τ' is a possible schedule to Π' then: τ'_i is a possible schedule for Π'_i and τ_i , a reduction

of τ'_i (without the end actions), is an individual possible schedule to Π_i. □

Now we can prove that the joint schedule τ^u that is made out of $\{\tau_i\}_{i=1}^n$ is an illegal joint schedule making Π not robust to rational.

Theorem 2. *Assume, τ' is a possible schedule for Π' and τ_i , a reduction of τ', is a possible schedule to Π_i, then the schedule τ^u that is made out of $\{\tau_i\}_{i=1}^n$ is an illegal joint schedule for Π.*

Proof. let us define $A'_{failure}$ as a set of all actions $a' \in A'$, so that $failure \in add^{start}(a') \cup add^{end}(a')$.

Easy to see that there is at least one action $a' \in A'_{failure}$ in τ'. This is because $failure \in G'$. Let us denote the non-success action with the earliest failed happening as a'^{ns}_i and assume this action was committed by agent i.

We have already shown that we can construct τ_i out of τ'_i and hence if τ' is a possible schedule to Π' then there is τ^u a joint schedule made out of $\{\tau_i\}_{i=1}^n$, where every τ_i is a possible schedule for Π_i. τ^u contains at least one non-success action.

Consider a^{ns}_i, if it is originates from: (a) a_i^{fstart} then τ^u gives us a schedule where agent i does not respect the start conditions of action a_i; (b) a_i^{fend} then τ^u gives us a schedule where agent i does not respect the end conditions of action a_i; (c) a_i^{finv} then τ^u gives us a schedule where agent i does not respect the overall conditions of action a_i; (d) $a_i^{finv^x_{start}}$ then τ^u gives us a schedule where agent i deletes a fact x at start while another agent's action marked this fact as an invariant fact; (e) $a_i^{finv^x_{end}}$ then τ^u gives us a schedule where agent i deletes a fact x at end while another agent's action marked this fact as an invariant fact; (f) $a_i^{w,x}$ then τ^u gives us a schedule where agent i is in a deadlock state. In this state this agent waits for a certain $x \in pre_w^{start}(a)$ before performing action a. We can guarantee it will keep waiting because we do not allow any agent to modify x as soon as $wt_{x,i}$ is raised; (g) end_i^f then τ^u gives us a schedule where agent i do not meet its goal. All of the above will result in an illegal joint schedule. □

We are now ready to prove our main theorem regarding the connection between the our compilation and the social law robustness verification decision problem.

Theorem 3. *Assume, Π is a multi-agent planning problem, which is solvable for every agent $i = 1 \dots n$. Then Π' is not solvable iff Π is robust.*

Proof. Let Π be robust and assume that Π' is solvable. We denote τ' as a possible schedule for Π'. According to Theorem 1, it is possible to create a joint schedule τ^u out of τ' so that τ^u is made out of individual possible schedules τ_i for Π_i. According to Theorem 2 τ^u is an illegal joint schedule as opposed to the definition of social law robustness, therefore τ' does not exist, making Π' not solvable.

Now, Let Π' be unsolvable. Let τ_i be any possible schedule for Π_i. Let τ^u be the joint schedule made out of $\{\tau_i\}_{i=1}^n$. We argue that: (a) All actions in τ^u respect all conditions of any action a_i; (b) There is no action in τ^u that deletes a fact that is invariant by another action; (c) There is no agent in a deadlock state; (d) All agents can reach their goals by performing τ^u. As otherwise Π' would be solvable (by creating τ' out of τ^u with the appropriate $\pi'_{actions}$ and adding π'_{end} actions). If the above is correct then: for all agents, for all individual solutions π_i to Π_i, for all joint schedule τ^u, τ^u achieves $\{G_i\}_{i=1}^n$ and therefore the social law l in Π is robust to rational. □

Empirical Evaluation

In order to empirically evaluate our compilation, we implemented it in Python. Our compilation takes a PDDL 2.1 planning problem, as well as definitions of who the agents are, which goal fact is assigned to which agent, and which start conditions are wait-for conditions. It then generates a temporal planning problem in PDDL 2.1. To solve the generated problems, we used the OPTIC planner (Benton, Coles, and Coles 2012) with the CLP solver, because it was one of the only temporal planners to support required concurrency (Cushing et al. 2007), which is necessary to find conflicts involving one agent action interefering with another's invariant condition. We used a single Intel i7-7700K core on a computer with 32GB of RAM, and with a time limit of 30 minutes. We evaluated our compilation on one real world multi robot system and on 5 virtual PDDL domains: 3 IPC benchmark domains and 2 new domains [1] which illustrate interesting aspects of the compilation. For each domain, we evaluate the compilation with an empty social law, which allows all actions, and with a robust social law. When possible, we also evaluate with a non-trivial social law which is not robust.

Real World Multi Robot System

We have created a simple multi agent system consisting of 2 robots that move around on a grid. Each square is classified as either dirty or clean. The robots can either clean the square in which they are located or move to one of its neighboring squares. Once a robot enters a "clean" square, the square becomes dirty. The goal of each robot is to clean its set of squares.

The first possible conflict was found by our compilation in less than 3 seconds. This conflict was caused by one Turtlebot trying to enter an occupied square. In order to prevent this and make the Turtlebot wait for a square to be vacated we added a wait-for start condition to the move-to action. Then, the compilation found an additional form of failure in less than one second. This conflict was a deadlock situation where each Turtlebot is waiting for the other to vacate its current location. To deal with this conflict we divided the area into two disjoint territories, and assigned each territory to one of the Turtlebots. Each Turtlebot was only allowed to enter its own territory. Then, although we suspected that we had formulated a robust social law, OPTIC was not able to prove unsolvability within the time limit. Finally,we im-

[1] These new domains are available at https://tinyurl.com/y8cbgqbn

plemented [2] this domain on real Turtlebots using ROSPLAN (Cashmore et al. 2015).

New Domains

Intersection Domain The intersection domain describes a four-way intersection, which cars can cross in all four cardinal directions. Cars crossing from north to south do not interfere with cars crossing from south to north, but can collide with cars crossing from east to west or vice versa.

This domain is modeled using eight possible locations for each car: for each lane, there is a location before crossing the intersection and another after crossing the intersection. We use a durative action to describe crossing the intersection, with an invariant condition which ensures no interfering cars cross while the action is executing. Specifically, the drive(dir_1, dir_2) action (where dir_1 and dir_2 are directions) adds a proposition that indicates that a car is crossing the intersection in this direction, and deletes it at the end. It also has an invariant condition that no cars are crossing in the orthogonal direction.

Running our compilation on an instance of this domain with 4 cars, one in each direction, results in a failure, as cars collide with each other. We then added a wait-for start condition similar to the above invariant condition, resulting in a robust social law. In this case OPTIC was able to verify that this social law is robust within less than a second.

Drink Domain In this domain there are two agents, who are both thirsty at the beginning. There is a single cup on the table, and to quench their thirst they need to take the cup off the table, fill it and drink. Running our compilation on this problem with an empty social law finds an easy failure, as they both try to grab the cup at the same time, and one fails.

Marking the fact that the cup is on the table as a wait-for condition also results in a failure, this time a deadlock, as the first agent taking the cup does not have to return the cup to the table. To fix this, we added a goal to each agent stating that the cup must be on the table. Surprisingly, this is still not robust, as the first agent to take the cup can fill it, drink, and then fill it again — causing the fill action of the second agent to fail. Finally, adding another start condition to the action that returns the cup to the table, stating that the cup must be returned empty, yields a robust social law.

OPTIC was able to find counter examples for the non-robust social laws quickly (less than 0.15 seconds). However, it was not able to prove that the compilation of the robust social law is unsolvable. This is possibly due to the presence of reversible actions, which means there are infinitely long possible plans.

IPC Benchmarks

To conclude our empirical evaluation, and show the scalability of our compilation, we also used IPC Benchmark domains. We adapted each domain to our multi-agent setting by deciding who the agents were and assigning goal facts (from the original goal) to each agent. We remark that assigning each goal fact to one agent tends to improve heuristic guidance. We used 3 domains: DRIVERLOG (IPC 2002), ZENOTRAVEL (IPC 2002), and FLOORTILE (IPC 2008), and formulated a social law for each of them.

DRIVERLOG is a logistics problem involving drivers that can walk or drive trucks, and packages which can be loaded or unloaded. In order to turn this problem into a multi-agent one, the drivers were treated as the agents, and LOAD and UNLOAD actions were assigned to drivers, also adding the condition that the driver had to be present at the location. We also assigned each goal fact to a driver randomly.

ZENOTRAVEL is also a logistics problem, involving planes that fly or zoom between locations and pickup or drop off passengers, consuming fuel. Planes can also refuel. In this case, the agents were the planes, and each goal fact was randomly assigned to a plane.

FLOORTILE involves multiple robots moving around on a grid, painting tiles. In this case, each robot is an agent. Randomly assigning each goal fact (a specific tile painted a specific goal) to agents does not work, as this yielded unsolvable problems. Therefore we divided the floor into contiguous sections, and assigned each section to a specific robot, ensuring that each robot is able to achieve its goal individually, also leaving two rows of tiles unpainted. We also updated all action durations to one time unit, to allow OPTIC to solve more problems.

For each of the 20 instances in these 3 domains[3], we first ran OPTIC to solve the centralized planning problem. The solution time for each solved instance (within the given time and memory limits) is reported in Table 1, and serves as a baseline for comparison. We then ran our compilation with the empty social law and report (a) The time it took for OPTIC to solve the compiled instances and by that to prove that this social law is not robust. (b) The total number of instances that were solved. While the average solving time did increase from the baseline, these results show that our compilation alone does not introduce too much overhead.

We then formulated a social law for each domain. For DRIVERLOG, we began by formulating a non-robust social law, which assigned each truck to a specific driver, and prevented driver from interacting with trucks that were not assigned to them. This social law was not robust, because drivers could "steal" packages of other drivers. The time it took for OPTIC to solve these compiled instances is reported in Table 1. We then proceeded to formulate a robust social law, which also assigns each package to the driver for delivering it. OPTIC was unable to solve any of the robust instances within the time limit but was also unable prove unsolvability. For ZENOTRAVEL, we formulated a robust social law which assigns each passenger to a plane, and restricts planes from picking up passengers that were not assigned to them. Again, OPTIC was unable to solve or prove unsolvability for any instance within the time limit. Finally, for FLOORTILE, we formulated a social law which assigns each tile to a specific robot. Each goal tile was assigned to the robot re-

[2] A video illustrating both robots acting concurrently is available at https://youtu.be/oETH6wK9CwE

[3] The first two instances of FLOORTILE contained only a single robot, so were excluded.

	DRIVERLOG				ZENOTRAVEL			FLOORTILE		
	NA	CP	EMPTY	NR	NA	CP	EMPTY	NA	CP	EMPTY
1	2	0.01 (7)	0.07 (11)	0.07 (11)	1	NA	NA	2	0.22 (38)	15.64 (48)
2	2	0.09 (21)	1.49 (35)	TLE	1	NA	NA	2	0.17 (35)	33.85 (50)
3	2	0.06 (17)	1.08 (45)	6.72 (30)	2	0.01 (6)	0.18 (16)	2	0.13 (36)	34.32 (46)
4	3	0.13 (23)	35.72 (45)	303.7 (36)	2	0.02 (17)	0.18 (17)	2	0.16 (27)	33.86 (47)
5	3	0.12 (20)	41.43 (37)	5.95 (50)	2	0.03 (17)	2.25 (35)	2	0.11 (35)	2.86 (43)
6	3	0.09 (14)	0.44 (24)	79.09 (30)	2	0.13 (20)	4.26 (27)	3	0.15 (36)	22.9 (39)
7	3	0.31 (23)	0.57 (43)	24.02 (27)	2	0.25 (24)	15.09 (27)	3	0.1 (35)	20.43 (46)
8	3	0.49 (29)	TLE	545.9 (54)	3	0.27 (18)	6.37 (25)	3	0.17 (37)	28.96 (40)
9	2	0.25 (35)	52.77 (50)	84.68 (37)	3	0.86 (40)	55.76 (49)	3	0.17 (38)	TLE
10	2	0.14 (39)	1.73 (50)	41.89 (53)	3	0.47 (37)	83.71 (53)	3	0.37 (38)	46.28 (42)
11	2	0.14 (54)	5.99 (52)	109.9 (55)	3	0.36 (28)	65.36 (28)	2	2.69 (71)	TLE
12	2	4.63 (57)	79.79 (72)	TLE	3	0.49 (41)	468.8 (47)	2	701.8 (65)	TLE
13	3	6.47 (52)	1700.7 (77)	TLE	3	423 (37)	TLE	2	2.38 (67)	TLE
14	3	6.4 (55)	TLE	TLE	5	TLE	TLE	2	TLE	TLE
15	4	4.2 (80)	TLE	TLE	5	215.9 (69)	TLE	2	TLE	TLE
17	5	TLE	TLE	TLE	5	468.6 (112)	TLE	3	TLE	TLE
SOL		15	12	10		13	10		13	9

Table 1: Planning Time on IPC Benchmarks and (Plan Lengths) (CP= centralized planning, NR = non robust, TLE = time limit exceeded, NA = number of agents)

sponsible for it. Each non-goal tile (the two unpainted rows), was assigned in a way that ensures each robot can access the tiles assigned to it. Here as well, OPTIC was not able to solve or prove unsolvability for any instance within the time limit.

Conclusion

In this paper, we have described a technique for verifying whether a social law is robust in a continuous time setting in which agents act concurrently. Our technique is based on a compilation to temporal planning, which attempts to find a set of plans which work for each agent by itself, and some schedule that combines them in a way that leads to failure. Our empirical evaluation shows that our technique scales fairly well when the social law is not robust. However, except for a single example in the intersection domain, the temporal planner we used, OPTIC, was not able to prove unsolvability. Thus, we claim that further work on proving unsolvability for temporal planning would improve the usefulness of this technique. It is also important to note that others have defined different notions of robustness in the context of social laws. For example, a notion of robustness relating to how many agents must obey the social law before the multi-agent system fails was proposed (Ågotnes, van der Hoek, and Wooldridge 2009). However, if a social law is robust according to our definition, and agents utility is defined by whether they achieve their goal or not (ignoring costs), then it is always in their best interest to obey.

References

Ågotnes, T.; van der Hoek, W.; and Wooldridge, M. 2009. Robust normative systems. In Boella, G.; Noriega, P.; Pigozzi, G.; and Verhagen, H., eds., *Normative Multi-Agent Systems, 15.03. - 20.03.2009*, volume 09121 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS 2008*, 28–35. AAAI Press.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In Brafman, R. I.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, 333–341. AAAI Press.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 1852–1859.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence* 20(1):61–124.

Karpas, E.; Shleyfman, A.; and Tennenholtz, M. 2017. Automated verification of social law robustness in STRIPS. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017.*, 163–171. AAAI Press.

Morris, P. 2006. A structural characterization of temporal dynamic controllability. In *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, 375–389.

Moses, Y., and Tennenholtz, M. 1995. Artificial social systems. *Computers and Artificial Intelligence* 14(6).

Muise, C. J.; Felli, P.; Miller, T.; Pearce, A. R.; and Sonenberg, L. 2016. Planning for a single agent in a multi-agent environment using FOND. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 3206–3212.

Shoham, Y., and Tennenholtz, M. 1992a. On the synthesis of useful social laws for artificial agent societies (preliminary report). In *AAAI 1992*, 276–281.

Shoham, Y., and Tennenholtz, M. 1992b. On traffic laws for mobile robots (extended abstract). In *AIPS 1992*. San Mateo, CA: Kaufmann. 309–310.

Shoham, Y., and Tennenholtz, M. 1995. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence* 73(1-2):231–252.

Tennenholtz, M. 1991. *Efficient Representation and Reasoning in Multi-Agent Systems*. Ph.D. Dissertation, Weizmann Institute, Israel.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. Exp. Theor. Artif. Intell.* 11(1):23–45.