Automated Verification of Social Laws in Numeric Settings

Ronen Nir[1], **Alexander Shleyfman**[2], **Erez Karpas**[1]

[1] Technion – Israel Institute of Technology
[2] Bar-Ilan University
ronen.nx@gmail.com, alexash@biu.ac.il, karpase@technion.ac.il

Abstract

It is possible for agents operating in a shared environment to interfere with one another. One mechanism of coordination is called Social Law. Enacting such a law in a multi-agent setting restricts agents' behaviors. Robustness, in this case, ensures that the agents do not harmfully interfere with each other and that each agent achieves its goals regardless of what other agents do. Previous work on social law verification examined only the case of boolean state variables. However, many real-world problems require reasoning with numeric variables. Moreover, numeric fluents allow a more compact representation of multiple planning problems.

In this paper, we develop a method to verify whether a given social law is robust via compilation to numeric planning. A solution to this compilation constitutes a counterexample to the robustness of the problem, i.e., evidence of cross-agent conflict. Thus, the social law is robust if and only if the proposed compilation is unsolvable. We empirically verify robustness in multiple domains using state-of-the-art numeric planners. Additionally, this compilation raises a challenge by generating a set of non-trivial numeric domains where unsolvability should be either proved or disproved.

Introduction

During the past decade, the coordination of autonomous agents in shared environments has been one of the main challenges facing the AI community. Agents in this setting have a set of actions and goals they strive to achieve. Even with a single agent, planning or learning in such an environment is challenging. Managing interactions between multiple agents present an additional challenge, as the actions of one agent can negatively affect those of another. If coordination or communication is impossible, each agent must produce a contingent plan, which considers all detrimental courses of action of all other agents. This is not always possible, not only because computations do not scale, but also because even in the most basic settings, agents need to trust each other to behave reasonably. Imagine a pedestrian stopped at a red light at a crossroad; they still assume that cars will not drive on the sidewalk, even though they can.

Multiple approaches have been proposed to avoid destructive collisions between two or more agents. One may try to solve the coordination problem via centralized methods (e.g., Brafman and Domshlak 2008; Nissim, Brafman, and Domshlak 2010). This approach, unfortunately, has a major drawback – it produces a single plan for each agent, denying them the choice of plan, thus preventing adaptive behavior. This approach also requires the agents to trust and communicate with the central planner, which is often infeasible.

The social law (SL) paradigm aims at mitigating these drawbacks (Tennenholtz and Moses 1989). An SL is a set of restrictions imposed on a multi-agent system to prevent agents' undesirable behavior. Karpas et al. (2017) defined the concept of SL in the MA-STRIPS environment. An SL is robust if it allows agents to plan, act and achieve their goals without conflicts and without considering other agents' actions in the environment. Karpas proposed an algorithm that verifies whether an SL, given in a classical planning multi-agent environment with binary variables, is robust.

While classical multi-agent planning is quite popular, it falls short of faithfully representing real-world problems. Consider, for example, a robotic factory where planning involves reasoning about timing and numbers, which requires more expressive formalisms, such as temporal and numeric planning, respectively. Nir and Karpas (2019) partially addressed this by proposing an SL verification method for temporal planning. As in the classical case, they compiled the verification problem into a temporal planning problem.

Following this trail, we propose a method for solving the SL verification problem in the setting of simple numeric planning. Unsurprisingly, we do it via a compilation into a simple numeric planning problem. The proposed compilation constructs a counterexample to the robustness of a given SL. Thus to verify that the SL is indeed robust we need to prove that the numeric planning problem is unsolvable, which can be challenging, as numeric planning is undecidable even in the simplest constellations (Helmert 2002).

Nevertheless, although it is impossible to prove that an SL is robust in the numeric setting in the general case, our experimental evaluation shows that when the SL is not robust, state of the art numerical planners manage to find a counterexample rather quickly and that there are numeric domains where the existence of SL can be verified. Furthermore, there are several problems where we can manually prove that a social law is robust, yet numerical planners can not prove this as these problems have an infinite number of

reachable states. Proving unsolvability in such cases is an interesting challenge to the numerical planning community.

Another advantage of numerical SLs is their compactness. We demonstrate how numeric settings can yield exponentially more compact SLs than ones obtained using only classical representations. Finally, this paper brings us closer to designing SL in complex, meaningful environments.

Preliminaries

Numeric planning extends the definition of a classical planning problem by introducing problems that involve propositional facts, P, and numeric variables, V. A state in numeric planning can be represented by $s = \langle s^p, s^n \rangle$ where $s^p \subseteq P$ is a subset of propositions that are true in s and s^n is a numeric assignment for each numeric variable in V. In this paper, we consider *simple numeric planning* (SNP), given by the formalism of Numeric Restricted Tasks (RT), that was introduced by Hoffmann 2003a. In this formalism, a numeric condition ψ is given in the form $\psi : v \geq w_0$ with $v \in V$ and $w_0 \in \mathbb{Q}$, the value of v in s is denoted by $s[v]$ and, $s \models \psi$ if $s[v] \geq w_0$. The propositional conditions in this setting are the same as the ones used in classical STRIPS, i.e., a propositional condition ψ is a subset of P, and $s \models \psi$ iff $\psi \subseteq s^p$. For a set Ψ we say that $s \models \Psi$ if $s \models \psi$ for each $\psi \in \Psi$.

An RT is given by the 5-tuple $\Pi = \langle P, V, A, s_0, G \rangle$, where P is a set of propositional variables, V is the set of numeric variables, A is the set of available actions, I is the initial state which consists of $s_0^p \subseteq P$ propositions and, s_0^n is a full assignment on the numeric variables in V, and $G = G^p \cup G^n$ is the goal conditions which also have a propositional and a numeric part. Each action $a \in A$ is described by $a = \langle pre_p(a), pre_n(a), add(a), del(a), num(a) \rangle$, where $pre_p(a), add(a), del(a) \subseteq P$ are the propositional parts of the action, as in STRIPS (Fikes and Nilsson 1971), $pre_n(a)$ is a set of numeric conditions, and $num(a)$ is a set of numeric effects of the form $v \mathrel{+}= k_v$, where $v \in V$ and $k_v \in \mathbb{Q}$. We assume that each action affects each numeric variable in V at most once and in a meaningful way, i.e., there is at most one numeric effect for each $v \in V$, and $k_v \neq 0$.

An action a is said to be applicable in state s if $s \models pre_n(a) \cup pre_p(a)$. The resulting state, denoted by $s[\![a]\!]$, is given by $s[\![a]\!]^p := (s^p \backslash del(a)) \cup add(a)$ for its propositional part, and $s[\![a]\!][v] = s[v] + k_v$ for all $v \mathrel{+}= k_v \in num(a)$, for all $v \in V$ that are unaffected by a we have $s[\![a]\!][v] = s[v]$.

We say that s_* is a goal state if $s_* \models G$. A sequence of consecutively applicable actions $\pi = \langle a_1, \ldots, a_m \rangle$ leading from the initial state s_0 to a goal state s_* is called a plan.

Despite its perceived lack of expressive power, RT can account for linear numeric conditions of the form $\sum_{v \in V} w_v v \bowtie w_0$, where $w_v, w_0 \in \mathbb{Q}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Due to the space constraint, we omit here the compilation that replaces each linear formula with a single variable (cf. Hoffmann 2003a). We also note that all effects in RT add a finite set of constants, and each condition of the form $v > w_0$ can be replaced with $v \geq w_0 + \varepsilon$.

The **multi-agent planning setting** (MAP) considers multiple non-collaborative, non-communicating agents. In this work, we further modify the MA-STRIPS formalism (Brafman and Domshlak 2008) with individual agent goals (Karpas, Shleyfman, and Tennenholtz 2017).

A MAP setting is defined by a tuple $\Pi = \langle P, \{A_i\}_{i=1}^n, s_0, \{G_i\}_{i=1}^n \rangle$. Here $n \in \mathbb{N}$ represents the number of agents. Each agent i has an individual set of actions A_i, and an individual goal condition G_i. In everything else, MAP repeats, the standard STRIPS formalism. Each agent aims to construct and **execute** a plan that leads from the initial state to its goal state. The single-agent plans in the MAP context are named *individual plans*. The immediate problem in this setting is that the individual plans of the agents can contradict each other. Consider, for example, two agents that need to use the same unique tool for their tasks: say that the first agent took the tool and did not return it to the toolbox after completing its mission. This prevents the second agent from completing its task. To mitigate such conflicts, Karpas *et al.* introduced the notion of SL, a set of restrictions imposed on agents to promote coordination.

According to Karpas et al., an SL l is a modification of a MAP setting Π, resulting in a MAP setting Π^l. This modification may include, for example, removing a set of actions (Nir and Karpas 2019) or adding some auxiliary facts that manage the agents by clever bookkeeping (Tuisov and Karpas 2020). In addition, SLs can impose more complex restrictions based on the wait-for mechanism. This mechanism forces an agent to stay inactive rather than execute its next action until some condition ψ is met. Thus, it is possible to mark a certain fact $p \in P$ as a wait-for precondition for action a to be denoted as $p \in pre^w(a)$. In many cases, given an SL, one needs to check its **robustness**, i.e., if it prevents the possibility for a conflict between the agents, whatever the plans the agents choose to execute.

In the following section, we further describe the problem of SL robustness verification in the multi-agent simple numeric setting by giving a formal description of the system, the model of execution, and the possible conflicts.

Problem Statement

Multi-Agent Simple Numeric Planning. Following Brafman and Domshlak (2008), we extend simple numeric planning to the multi-agent setting (MA-SNP) by including individual action sets and goal specifications for each agent. The set of n agents is by denoted $[n] := \{1, \ldots, n\}$. The multi-agent restricted task (MART) then is given by the tuple $\Pi = \langle P, V, s_0, \{A_i\}_{i=1}^n, \{G_i\}_{i=1}^n \rangle$ where: P, V, s_0 stay the same, representing the facts, numeric variables, and initial state, A_i represents the actions available to agent $i \in [n]$ and, correspondingly, G_i represents the goal specification of agent i. The individual projection of Π on agent i, given by $\Pi = \langle P, V, s_0, A_i, G_i \rangle$, is an RT.

Execution Model. Our execution model involves n agents acting in the same environment. Each agent is ready to perform their next pending action according to their own predefined individual plans. The decision on which agent will act is arbitrary while respecting wait-for conditions. A joint execution is successful if every agent completes its plan without conflict and the final state meets its goal.

More formally, let $\pi^{\text{jnt}} = \{\pi_i\}_{i=1}^n$ be the set of individual plans for the n agents. A sequence of actions $r =$

$\langle a_1, \dots a_m \rangle$, which is an interleaving derived from π^{jnt} is a joint execution if the following holds: (1) all actions in the sequence except for, maybe, the last one are consecutively applicable, i.e.,

$$\forall j \in [m-1] : s_{j-1} \models pre(a_j) \wedge s_j := s_{j-1}[\![a_j]\!];$$

and (2) $s_{m-1} \models pre^w(a_m)$, i.e., the wait-for preconditions of the last action should be respected, while the action still can be inapplicable. This implies that all states along a joint execution respect the wait-for preconditions of the executed actions.

The joint execution can stop due to four possible reasons:

0. **Successful Termination** – all agents have executed their plans, and the last state is the goal state for each agent. In this case, we say that the execution was **successful**.
1. **Failure at Termination** – the same as before, but here at least one of the agents' goals is not achieved in the termination state.
2. **Action Conflict** – the agent tries to execute an action with *non-wait-for* preconditions that does not hold.
3. **Deadlock** – the *wait-for* conditions of all existing pending actions are not satisfied in the current state, i.e., one (or more) of the agents is waiting indefinitely for a condition that will never hold. Deadlock is a failure, even if the agent is in its goal state at the end of the execution.

Using this observation, we define the set of all joint executions of a task Π, denoted by $R^{\text{jnt}}(\Pi)$, as a subset of joint executions, where no execution is a prefix of another, longer execution. This is the same as saying that an execution is concluded due to one of the reasons listed above. The set $R^{\text{jnt}}_*(\Pi) \subseteq R^{\text{jnt}}(\Pi)$ denotes all successful executions.

Definition 1. *Let Π be a MAP. We say that Π has a **robust SL** if $R^{jnt}_*(\Pi) \neq \emptyset$ and $R^{jnt}_*(\Pi) = R^{jnt}(\Pi)$.*

Here, the condition $R^{\text{jnt}}_*(\Pi) \neq \emptyset$ implies that there is at least one successful joint execution, and thus all agents has at least one individual plan, and $R^{\text{jnt}}_*(\Pi) = R^{\text{jnt}}(\Pi)$ implies that any joint executions terminate successfully. We omit Π for brevity when it is clear from the context.

Social Law Verification

In this section, we present our compilation for verifying the robustness of an SL in a MART setting. This compilation builds on a similar compilation for the classical setting (Karpas, Shleyfman, and Tennenholtz 2017), with two contributions: (1) this compilation can handle numerical variables, and (2) this compilation fixes a mistake in the way the original compilation handled wait-for preconditions.

More specifically, to verify the robustness of the SL in the multi-agent setting Π, we construct a (single agent) numeric planning problem $\tilde{\Pi} = \langle \tilde{P}, \tilde{V}, \tilde{A}, \tilde{s}_0, \tilde{G} \rangle$) such that its plan $\tilde{\pi}$ constitutes a counterexample, i.e., it corresponds to a failed joint execution of the individual plans $\{\pi_i\}_{i=1}^n$ of the agents in Π. Following is an overview of how we design $\tilde{\Pi}$ to find an invalid joint plan execution.

As mentioned earlier, there are several possible reasons why a joint plan execution may fail: 1. failure at termination, 2. action conflict, or 3. deadlock. The goal of our compilation is to find a set of individual plans $\{\pi_i\}_{i=1}^n$ that results in a failed joint execution; thus, it has to simultaneously keep track of $n+1$ copies of the original state: n local copies for each agent $i \in [n]$ where each state i reflects the state of the system in the case that agent i acted alone, and a global copy (denoted by g) that captures the overall state of the system, taking into account all the actions taken by the agents. $\tilde{\Pi}$ has a special flag for reporting *failures* in execution. This flag is included in $\tilde{\Pi}$'s goal specification, and it is raised only when the invalid execution conditions are met.

The proposed compilation also creates multiple copies of each action, which can be divided into four types: (a) successful version of a_i that changes both the local copy of agent i and the global copy g; (b) a_i fails to meet its non-wait-for precondition; (c) executing a_i deadlocks the agent due to an unmet propositional/numeric wait-for condition which will never hold; (d) the agent i changes only its local copy. Type (d) actions can only be executed when a deadlock or action conflict has occurred.

Facts, Initial State and Goal. We start with compilation's fact and numeric variables sets, $\tilde{P}$ and $\tilde{V}$. The compilation tracks $n+1$ state copies: n copies to track the state of the environment as if the agent has acted alone with the local variables p_i and v_i, for $i \in [n]$, and one global copy that corresponds to the states of the original MA-STRIPS task, where all agents' actions are taken into account. The global copy is represented via variables p_g and v_g. We denote by P_i the set of atoms corresponding to copy $i \in [n] \cup \{g\}$, and use a similar notation V_i for the numeric variables. A restriction to a copy $i \in [n] \cup \{g\}$, reduces the atoms $\tilde{P}$ to P_i, and the variables $\tilde{V}$ to V_i. This will become handy in the following.

We define the following sets of atoms to account for the wait-for preconditions. Every agent $i \in [n]$ has a flag wt_i indicating it is currently waiting for some precondition to hold. In this compilation, the agent waits for something to hold indefinitely; thus, no action can delete wt_i. For every $f \in P$ that is a wait-for precondition of some action a_i, there is a flag wt^f that indicates an agent is waiting for an atom f. Similarly, $v \geq w_0$ is a wait-for precondition for $wt^{v \geq w_0}$ that indicates an agent is waiting for $v \geq w_0$. Thus, the number of waiting flags is the same as the size of all numeric and atomic conditions in Π. The waiting atoms are

$$Wt := \{wt_i, wt^\psi \mid \psi \in pre^w(a_i), a \in A_i, i \in [n]\},$$

where ψ can be either atomic $\psi : p$, or numeric $\psi : v \geq w_0$. Lastly, we introduce a set of atoms needed to finalize the task: Finding a set of individual plans, each is applicable separately, but when carried out in a particular order, it poses a conflict. The flag act indicates that the agents are still executing their actions. By construction, act can be removed only after the agents execute their individual plans and achieve their local projection goal.

The compilation finds a failing joint execution (which consists of n valid individual plans). Thus, we include the atom $fail$ in $\tilde{G}$ to represent an occurrence of some failure in the execution. Once an action conflict has occurred, the system enters a dead-end state. Thus, we add the flag cf to

mark this situation and act accordingly. To ensure the validity of the individual plans, every agent $i \in [n]$ has a finalization atom fin_i in the goal specification, $fin_i \in \tilde{G}$. fin_i indicates that the agent has finished acting and its individual plan is valid. The set of auxiliary atoms is denoted by:

$$Bk := \{act, fail, cf\} \cup \{fin_i \mid i \in [n]\}.$$

Thus, the sets of atoms and numeric variables of $\tilde{\Pi}$ are

$$\tilde{P} = Wt \cup Bk \cup \bigcup_{i \in [n] \cup \{g\}} P_i \text{ , and } \tilde{V} = \bigcup_{i \in [n] \cup \{g\}} V_i.$$

After describing the compilation's facts and variables, we can define its initial state and goal description. We set all $n+1$ copies of the initial state to represent the initial state of the original task and add the act atom, i.e.,

$$\tilde{s}_0^p = \{act\} \cup \{p_i \mid p \in s_0^p, i \in [n] \cup \{g\}\}$$
$$\tilde{s}_0^n = \{\tilde{s}_0[v_i] := s_0[v] \mid i \in [n] \cup \{g\}\}.$$

We also set the goal to be $\tilde{G} = \{fail\} \cup \{fin_i \mid i \in [n]\}$.

Actions. This subsection is dedicated to describing the actions of the proposed compilation. In the proposed compilation, we distinguish between four possible action versions for each original action $a \in \bigcup_{i \in [n]} A_i$: (1) action is applicable; (2) action is not applicable; (3) the agent goes into a deadlock – a phase of forever waiting; (4) a deadlock exists, or an action conflict has occurred.

For a condition $\psi \in pre(a)$, we denote its corresponding copy as ψ_i, for $i \in [n] \cup \{g\}$. Here if $\psi : p$ is an atom its i's copy is defined as $\psi_i : p_i$, similarly, if $\psi : v \geq w_0$ we have that $\psi_i : v_i \geq w_0$. Moreover, for each $v \in V$ we define the set of its constants to be $\mathcal{W}_v := \{w_0 \mid v \geq w_0 \in \bigcup_{i \in [n]} (G_i \cup \bigcup_{a \in A_i} pre_n(a_i))\}$.

Start with the successful version (a_i^s) of an action a_i. It represents the successful execution of a_i; thus, it changes both the individual copy of agent $i \in [n]$ and the global copy, marked g. This action copy is only applicable if: (a) the preconditions of a_i are satisfied in both the local and the global parts of the state; (b) agent i is not in a deadlock; (c) there was no actions conflict; and (d) it does not satisfy any condition ψ flagged by some agent in deadlock to be forever false. An important point required for dealing with numerical variables (but not propositional ones) is that when some other agent is waiting for a numerical condition $v \geq w_0$ (which corresponds to $wt^{v \geq w_0}$ being true in the compilation), then it is forbidden for a_i^s to increase the value of v past w_0, but it is allowed for a_i^s to increase v to some value lower than w_0. Thus, the last line of the precondition encodes a conditional statement that checks whether the value of v will increase past w_0 after the effects of a_i occur.

$$\begin{aligned} pre(a_i^s) = act \wedge \neg wt_i \wedge \neg cf \wedge \\ \bigwedge_{\psi \in pre(a)} (\psi_i \wedge \psi_g) \wedge \bigwedge_{p \in add(a)} \neg wt^p \wedge \\ \bigwedge_{\substack{v += k_v \\ \in num(a)}} \bigwedge_{w_0 \in \mathcal{W}_v} (\neg wt^{v \geq w_0} \vee (v < w_0 - k_v)); \end{aligned}$$

$$\begin{aligned} add(a_i^s) &= \{p_i, p_g \mid p \in add(a)\}; \\ del(a_i^s) &= \{p_i, p_g \mid p \in del(a)\}; \\ num(a_i^s) &= \{v_i \mathrel{+}= k, v_g \mathrel{+}= k \mid v \mathrel{+}= k_v \in num(a)\}. \end{aligned}$$

Here we note that the grounding of these actions heavily depends on how many numerical effects they have and, due to logical operator $\vee$ in its preconditions, may result in an exponential number of "grounded actions".

The actions of the type a_i^f are the actions that represent action conflict, i.e., action a was executed when one of its non-wait-for preconditions $\psi \in (pre^w(a))^c := pre(a) \setminus pre^w(a)$ is not satisfied by the state it was applied in. a_i^f is only applicable if: (a) the wait-for preconditions of a_i are satisfied in both the local state and the global state; (b) there was no action conflict yet; (c) all other conditions of a_i are satisfied in the local state (ensuring the individual plan is valid); however, there is a propositional precondition not satisfied in the global state; and (d) Agent i is not waiting forever (in deadlock). a_i^f represents an invalid action, and as such, it raises both $fail \wedge cf$ flags and affects only the local copy of the agent i, leaving the global copy g unchanged.

$$\begin{aligned} pre(a_i^f) = act \wedge \neg wt_i \wedge \neg cf \wedge \\ \bigwedge_{\psi \in pre(a)} \psi_i \wedge \bigwedge_{\psi \in pre^w(a)} \psi_g \wedge \bigvee_{\psi \in (pre^w(a))^c} \neg \psi_g; \\ add(a_i^f) = \{fail, cf\} \cup \{p_i \mid p \in add(a)\}; \\ del(a_i^f) = \{p_i \mid p \in del(a)\}; \\ num(a_i^f) = \{v_i \mathrel{+}= k \mid v \mathrel{+}= k_v \in num(a)\}. \end{aligned}$$

The action $a_i^{wt^\varphi}$ indicates that agent $i \in [n]$ has entered a deadlock, permanently waiting for a condition $\varphi \in pre_p^w(a_i)$ to execute action a_i. This action is only applicable if: (a) there is $\varphi \in pre^w(a)$ that is not satisfied in the global copy; (b) all preconditions are satisfied in the local copy i; (c) an action conflict has not already occurred; and (d) agent i is not already in a permanent waiting state. $a_i^{wt^\varphi}$ indicates that the agent is entering a deadlock; thus, it affects only the local state, leaving the global state unchanged. The rest of the actions of agent i do not affect the global copy. Raising the $fail$ flags and wt_i a potential deadlock. Since the compilation forbids making the condition φ true if someone waits for it, the existence of local plans for all agents indicates a deadlock in the joint plan execution.

$$\begin{aligned} pre(a_i^{wt^\varphi}) = act \wedge \neg wt_i \wedge \neg cf \wedge \neg \varphi_g \wedge \bigwedge_{\psi \in pre(a)} \psi_i; \\ add(a_i^{wt^\varphi}) = \{wt^\varphi, fail, wt_i\} \cup \{p_i \mid p \in add(a)\}; \\ del(a_i^{wt^\varphi}) = \{p_i \mid p \in del(a)\}; \\ num(a_i^{wt^\varphi}) = \{v_i \mathrel{+}= k \mathrel{+}= k_v \in num(a)\}. \end{aligned}$$

The action a_i^l is the version of a_i applicable after agent $i \in [n]$ has entered a deadlock or an action conflict has occurred. Actions of this type affect only the local copy of the agent,

ensuring that the rest of its individual plan is valid.

$$pre(a_i^l) = act \wedge (wt_i \vee cf) \wedge \bigwedge_{\psi \in pre(a)} \psi_i;$$
$$add(a_i^l) = \{p_i \mid p \in add(a)\};$$
$$del(a_i^l) = \{p_i \mid p \in del(a)\};$$
$$num(a_i^l) = \{v_i \mathrel{+}= k_v \in num(a)\}.$$

In total, there are four versions of actions for each original action $a \in A$: (1) a successful action version; (2) an inapplicable action version; (3) a deadlock action version, and, finally, (4) a local action version that keeps track of the agent's activity on its local variables. The rest of the compilation's actions are for bookkeeping purposes, ensuring that all agents have completed their valid individual plans.

For each agent i, we have two versions of the END action. END_i^s is the successful version of END, which is only applicable when the agent has achieved its goal specification G_i in both the local and global states. END_i^f is the version of END where the agent has only achieved its goal in the local copy. Once END is executed, we forbid executing any regular actions; hence $del(END_i^s) = del(END_i^f) = \{act\}$. Then,

$$pre(END_i^s) = \neg fin_i \wedge \bigwedge_{\psi \in G_i} (\psi_i \wedge \psi_g)$$
$$add(END_i^s) = \{fin_i\}, \text{ and}$$
$$pre(END_i^f) = \neg fin_i \wedge \bigwedge_{\psi \in G_i} \psi_i \wedge \bigvee_{\psi \in G_i} \neg \psi_g$$
$$add(END_i^f) = \{fin_i, fail\}.$$

The next section provides a proof sketch for the compilation's correctness. Full proof can be found in Supplementary Materials (SM).

Compilation Correctness

In the following, we assume a multi-agent RT setting Π, with n agents, each with its own solvable planning problem Π_i and its plan π_i, for $i \in [n]$. Also, let $\tilde{\Pi}$ be the result of applying the compilation on Π and let π' be its plan if one exists. The main theorem of this paper is as follows:

Theorem 1. *$\tilde{\Pi}$ is solvable $\iff$ Π is not robust*

Proof. ($\implies$) Let π' be a plan for $\tilde{\Pi}$. By Lemma 1, from π' it is possible to construct an individual plan π_i, for $i \in [n]$. Thus, it is possible to construct a set of n individual plans $\pi^{\text{jnt}} = \{\pi_i\}_{i=1}^n$. By Lemma 2, there exists a failed joint execution of π^{jnt}. Thus, by definition, Π is not robust.

($\impliedby$) Let Π be a not robust multi-agent planning problem. By definition, there exists a failed joint execution r of some set of individual plans π^{jnt}. By Lemma 3 it is possible to reconstruct a plan for $\tilde{\Pi}$ from π^{jnt} and its failed joint execution, making $\tilde{\Pi}$ solvable. □

Detailed descriptions and proofs of Lemmas 1-3 are provided in the following sections starting with Lemma 1.

Lemma 1. *Let Π be a MART with n agents, $\tilde{\Pi}$ be its compilation with its plan π'. There exists π_i a plan for Π_i the projection of Π on agent i, for $i \in [n]$.*

Proof. Consider the projection of $\tilde{\Pi}$ on agent i's facts and variables $\{p_i \mid p \in P\} \cup \{v_i \mid v \in V\} \cup \{fin_i\}$ and let us denote it as $\tilde{\Pi}_i$. Note that $\tilde{\Pi}_i$ is only a simple extension of Π_i with the following modifications: (a) one additional fact fin_i; (b) a goal specification that contains only one atom fin_i; (c) one additional action END_i with G_i as its precondition and fin_i as its add effect. A plan for $\tilde{\Pi}$ is a plan for any abstraction of $\tilde{\Pi}$, and specifically a plan for $\tilde{\Pi}_i$. Thus π_i exists, for $i \in [n]$. □

By Lemma 1, it is possible to construct a set of individual plans $\pi^{\text{jnt}} = \{\pi_i\}_{i=1}^n$ from π'. The next step is to show that there exists a failed joint execution of π^{jnt}.

Lemma 2. *Let Π be a MART, and let $\tilde{\Pi}$ be its compilation with a plan π'. Then, there exist a set of individual plans π^{jnt}, and a failed joint execution r reconstructed from π'.*

Proof Sketch. Given a plan π' for $\tilde{\Pi}$, we can reconstruct a set of individual plans $\pi^{\text{jnt}} = \{\pi_i\}_{i=1}^n$ (Lem. 1). By construction of $\tilde{\Pi}$ the plan π' is a concatenation of π'_{actions} and π'_{end}, where π'_{actions} is some interleaving of the plans in π^{jnt} and π'_{end} is a sequence of END actions. Note that in π'_{end} there is exactly one finalizing action for each agent $i \in [n]$, and the order of these actions does not matter. The END actions check if the agents have achieved their goals in both global and local copies. The existence of π' implies that all agents achieved their goals in the local copies, and a failure occurred in the global one.

We aim at reconstructing r from π'. Let us look at the longest sub-sequence of successful actions[1] in π'_{actions} up until and including) the first failed action (if one exists) in π'_{actions}. The natural mapping of this sub-sequence on the original actions of Π constitutes r. Let us show that r is a joint execution that leads to a conflict, i.e., $r \in R^{\text{jnt}}(\Pi) \setminus R_*^{\text{jnt}}(\Pi)$. By construction of $\tilde{\Pi}$: (1) failed and waiting actions do not affect the global copy, and (2) an agent that executed a waiting action is in a deadlock (since once the agent has declared it waits for some condition, the compilation forbids all other agents from making it true). Thus, r is the longest successful prefix of some interleaving execution of π^{jnt}, hence $r \in R^{\text{jnt}}(\Pi)$.

To show that $r = \langle a_1, \dots a_m \rangle$ is a failed execution, i.e., $r \notin R_*^{\text{jnt}}(\Pi)$, we need to address three separate cases. If all actions in π'_{actions} are successful and π'_{end} still leads to failure, we have a case of **failure at termination**, i.e., all agent finish their plans in the local copies, but there is an agent whose goal was sabotaged by some other agent. The case when r ends with a failed action clearly leads to an **action conflict**. Since the first $m-1$ actions in r where successful we have that $j \in [m-1] : s_{j-1} \models pre(a_j)$ where $s_j := s_{j-1}[\![a_j]\!]$, and by construction of the failed actions

[1] Successful action belongs to the a^s type.

$s_{m-1} \not\models pre(a_m) \setminus pre^w(a_m)$. Lastly, if there is no action-conflict and still $|r| \neq |\pi^{\text{jnt}}|$, i.e., there are agents with unfinished plans, some of the agents are in a **deadlock**. In our compilation, if an agent declares that it is waiting for some condition (numeric or otherwise), no other agent is allowed to make this condition true. Since π' is a plan, each agent achieves its goal in its local copy. Thus there exists a joint execution, where an agent is forever waiting for some condition that will never hold. Hence, it is a deadlock. □

Lemma 2 proves that there is a failed joint execution of π^{jnt}. Next, we show that if the SL is not robust, i.e., there is a possible failed joint execution, the compilation is solvable.

Lemma 3. *Let* Π *be a non-robust MART with* n *agents,* $\tilde{\Pi}$ *the compilation of* Π *is solvable.*

Proof Sketch. Given that Π is not robust, there must be at least one set of individual joint plans π^{jnt} that has at least one failed joint execution $r \in R^{\text{jnt}}(\Pi)$. Our goal is to reconstruct π' that solves $\tilde{\Pi}$, given π^{jnt} and r. By construction, all possible plans of $\tilde{\Pi}$ are of the form $\pi'_{\text{actions}} \circ \pi'_{\text{end}}$, where π'_{actions} is a sequence of actions in $\bigcup_{i\in[n]} A_i$ of one of the four possible types, and π'_{end} is a sequence of the END_i actions, and can be seen as a set. There is exactly one END_i action in π'_{end} for each agent i, and each agent achieves its goal in the local copy if it acts according to π_i.

Once again, we iterate over the three possible reasons for failure. 1. Say that r has **failed at termination**. Then, all actions of π^{jnt} were applied successfully in r. Thus, r contains as a sub-sequence any $\pi_i \in \pi^{\text{jnt}}$, and no other actions. Let r^s be the r sequence, where each action a is replaced by its counterpart of the successful type a^s. Then, $\pi' = r^s \circ \pi'_{\text{end}}$. Since we know that at least one agent did not achieved its goal, we know that $\tilde{s}_0[\![r]\!] \not\models (G_i)_g$ for some $i \in [n]$. Which means that $END_i^f \in \pi'_{\text{end}}$. Thus, π' is indeed a plan.

2. Assume that r has terminated with **action conflict**. Then, all actions in r but the last one are of the successful type and the last one of the failed type. This happens since action failure trumps deadlock, in the sense that deadlock actions are not executed in the global copy, and we can apply all the actions in r to get straight to the failure. Denote by $r^{s,f}$ the sequence of actions that corresponds to r in $\tilde{\Pi}$. We have that $\tilde{s}_0[\![r]\!] \models cf \wedge fail$, thus $\pi'_{\text{actions}} = r^{s,f} \circ \sigma^l$, where σ^l is the rest of the execution of π^{jnt} where all actions are of the a_i^l type, i.e., executed in the local copies. Since all agents achieve their goals in the local copy, π'_{end}, and thus π' are executable. Since no action removes $fail$, π' is a plan.

3. If r reaches a **deadlock**, once again let r^s be the successful copy of r. Note that since by definition r is not a sub-sequence of any other joint execution, thus no action the suffixes of π^{jnt}can be executed after r. Let I be the set of agents that have not completed their individual plans in r. Then, $\pi'_{\text{actions}} = r^s \circ \sigma^{wt,I} \circ \sigma^l$, where r^s is a successful action execution, $\sigma^{wt,I}$ is an independent execution of actions of the type a_i^{wt} for all agents that are in a deadlock (one per agent), and σ^l is the rest of the actions that are executed in the local copy. Since all local goals are achieved, and each a_i^{wt} adds the $fail$ atom, π' is a plan. □

Empirical Evaluation

We implemented the compilation in Python[2] and tested it on domains from previous work on numeric planning along with the BRIDGE domain that we have formulated to illustrate an interesting view of how numerical SLs are more compact than SLs in classical planning settings. Our compilation takes a numeric multi-agent problem defined in PDDL2.1 (Fox and Long 2003) and an additional JSON file with information about the agents' goal affiliation and wait-for conditions. A numerical planning problem is then generated. To solve the generated problems, we used the metric-fast forward planner (Hoffmann 2003b). Its performance has been demonstrated to be better than that of other planners, such as NFD (Aldinger and Nebel 2017) and ENHSP-20 (Scala 2020). We used a single Intel i7-7700K core on a computer with 32GB of RAM. The time limit is 30 minutes.

SL Compactness Example. The bridge domain consists of a bridge and several weighted agents that can cross it to reach the other bank. Domain facts describe the current positions of each agent (left, right, and bridge), and the only numeric variable represents how much weight the bridge can support. Each agent has four actions (GET_ON(a_1, $right$), GET_OFF(a_1, $left$), etc.).

We ran our compilation on an instance with two weighted agents, T_1 and T_2, initially positioned on the right bank, with their weights, 50 and 60 units, respectively, while the bridge can hold only 100 units of weight. The first agent's goal is to be on the left bank, and the second agent has an empty goal set. Our compilation found a possible conflict existed when both of the agents tried to board the bridge at the same time.

One possible SL makes agents wait before they get on the bridge if their weight is greater than the remaining weight capacity of the bridge. Our compilation shows that this SL is not robust because there is a possible conflict where the second agent climbs on the bridge first and does not go down, making the first agent stay in a forever waiting mode.

An improved SL may give the second agent a specific goal position (right or left) and make the agents wait before they climb on the bridge. Testing this SL with our compilation proves that this SL is robust.

This domain can be represented using STRIPS or FDR by keeping track of which agents are currently on the bridge and maintaining a list of allowed sets of agents which can be on the bridge at the same time, i.e., those whose combined weights do not exceed the limit. However, the representation size of this SL is exponential in the number of agents, while numeric SLs can represent the same restriction in constant size. Thus, we have shown that numerical SLs are more compact than SLs in classical planning settings.

Domains From Previous Works. The domains DEPOTS, SAILING, PLANT-WATERING (Scala et al. 2016) and DELIVERY (Shleyfman et al. 2022) were used to demonstrate the scalability of the compilation. These were slightly modified to fit the MART setting.

Trucks and hoists transport crates in DEPOTSẆe adapted this domain to the multi-agent setting by creating a meta-

[2]https://github.com/ronen85/numeric-slv

	DEPOTS			SAILING			WATERING			DELIVERY		
#	|A|	RSLT	T[s]	|A|	RSLT	T[s]	|A|	RSLT	T[s]	|A|	RSLT	T[s]
1	156	CE	0.1	16	CE	0.0	20	CE	0.1	136	CE	0.1
2	372	CE	0.1	18	CE	21.2	20	CE	0.2	200	CE	0.1
3	684	CE	1.0	20	CE	0.1	20	CE	0.1	264	CE	0.1
4	1092	CE	95.9	22	CE	0.1	22	CE	0.1	412	CE	0.3
5	1596	TO	-	24	CE	211.0	22	CE	0.1	392	CE	0.2
6		PU	-	26	CE	0.1	22	CE	0.2	576	CE	0.6
7	828	CE	0.2	28	CE	374.9	24	CE	0.3	656	CE	1.0
8	1836	CE	8.0	30	CE	0.1	24	CE	0.3	736	CE	1.6
9	3636	TO	-	32	CE	0.6	24	CE	1.2	980	CE	9.3
10	1656	CE	49.7	34	CE	382.0	20	CE	0.1	1614	ERR	-
11	3624	TO	-	36	TO	-	20	CE	0.3	1154	TO	-
12		PU	-	38	CE	1388.5	20	CE	0.7	1887	ERR	-
13	1992	CE	1.2	40	CE	0.2	22	CE	0.5	2031	ERR	-
14	4184	CE	339.3	42	CE	0.2	22	CE	0.7	2541	ERR	-
15	8004	ERR	-	44	CE	0.2	22	CE	0.3	2709	ERR	-
16	2840	CE	1.3	46	CE	0.2	24	CE	3.6	4305	ERR	-
17	5908	ERR	-	48	CE	0.3	24	CE	0.3	4557	ERR	-
18	11168	ERR	-	50	TO	-	24	CE	0.8	4809	ERR	-
19	4768	ERR	-	52	CE	0.2	20	CE	0.3	5061	ERR	-
20		PU	-	54	CE	0.3	20	CE	1.0	5313	ERR	-
TOTAL	13			18			20			9		

Table 1: Results on benchmarks and planning time: |A|= number of grounded actions, RSLT = result, CE = not robust, found counterexample, PU = not robust, individual projection unsolvable, TO = timeout, ERR = parser error (problem too long).

agent that controls every hoist and every truck to meet a specific crate position goal. In addition, the agents were randomly assigned goals.

DELIVERY involves multi-gripper robots with trays and weight limits that move weighted objects on a map of locations. A random assignment is made here as well.

In the PLANT-WATERING domain, the agents move in a grid that contains plants and taps, then they load water from the taps and pour it on the plants. Numeric variables are used to describe the locations of the agents, plants, and taps.

In the SAILING domain, boats (agents) move around and rescue persons if they meet a linear constraint on their x, y location. The only change made to this domain is the addition of $\neg saved$ to the save-person action's precondition.

In this domain, an SL assigns each person to a specific boat agent. It is easy to see that this SL prevents all conflicts because the only source of contention between the agents is the save-person action. However, since each boat agent can reach infinitely many locations, the metric-FF planner fails to prove SL robustness in all instances. Proving the unsolvability of the compilation in these cases is a challenge to the numerical planning community.

On each of the 20 instances in the four domains, the metric-FF planner was used to solve the compiled problem, i.e., determine whether the SL is robust. Table 1 reports (1) the number of grounded actions for comparison; (2) the decision result (whether the SL is robust or not); and (3) for each solved instance, the solution time. Based on the results, the compilation alone does not add too much overhead: 60 of 80 problems were solved. Let us note that, 14 times, we encountered a parsing problem – a known problem related to the length of the problem files, which include thousands of grounded actions. These instances could be parsed but not solved by other planners (e.g., NFD 2017).

Conclusions and Future Work

In this paper, we have shown how to verify the robustness of an SL in an SNP setting. In the process, we have fixed some mistakes in previous work on SL robustness verification in classical planning settings (Karpas, Shleyfman, and Tennenholtz 2017). We have also shown that numerical SLs can be much more compact than SLs in classical settings.

This compilation also presents a new challenge to the numerical planning community. Specifically, verifying that a numerical planning problem is unsolvable is usually done by exhausting the reachable states. Unfortunately, in many numeric planning problems, there are infinitely many reachable states. Nevertheless, in some of these problems, including some that are the result of the compilation presented here, it is possible to prove that the problem is unsolvable manually. Thus, further developments in proving unsolvability (e.g., Christen et al. 2022) for numerical planning would also benefit this work.

Dealing with numbers in planning brings us one step closer to handling real-world multi-robot problems. Combining numbers with SLs in a continuous time setting (Nir and Karpas 2019) is the next logical step. This would require dealing with continuous change, which presents a new set of challenges. Furthermore, robustness verification constitutes the backbone of the SL synthesis as a search procedure (Nir, Shleyfman, and Karpas 2020, 2021), which uses it as its goal test. Thus, it is straightforward to synthesize robust SLs in a numerical setting by combining the compilation presented in this paper with the abovementioned search procedure.

Acknowledgements

This work was partially supported by the Peter Monk Research Institute (PMRI). The work of Alexander Shleyfman was partially supported by the Israel Academy of Sciences and Humanities program for Israeli postdoctoral researchers.

References

Aldinger, J.; and Nebel, B. 2017. Interval Based Relaxation Heuristics for Numeric Planning with Action Costs. In *Proc. SOCS*, 155–156.

Brafman, R. I.; and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *ICAPS*, volume 8, 28–35.

Christen, R.; Eriksson, S.; Pommerening, F.; and Helmert, M. 2022. Detecting Unsolvability Based on Separating Functions. In *ICAPS*, 44–52.

Fikes, R.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.*, 2(3/4): 189–208.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *JAIR*, 20(1).

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *AIPS*, 303–312.

Hoffmann, J. 2003a. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *J. Artif. Intell. Res.*, 20: 291–341.

Hoffmann, J. 2003b. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *J. Artif. Intell. Res.*, 20: 291–341.

Karpas, E.; Shleyfman, A.; and Tennenholtz, M. 2017. Automated Verification of Social Law Robustness in STRIPS. In *ICAPS*, 163–171.

Nir, R.; and Karpas, E. 2019. Automated Verification of Social Laws for Continuous Time Multi-Robot Systems. In *AAAI*, 7683–7690.

Nir, R.; Shleyfman, A.; and Karpas, E. 2020. Automated Synthesis of Social Laws in STRIPS. In *AAAI*, 9941–9948.

Nir, R.; Shleyfman, A.; and Karpas, E. 2021. Learning-Based Synthesis of Social Laws in STRIPS. In *SOCS*, 88–96.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *AAMAS*, 1323–1330.

Scala, E. 2020. The ENHSP Planning System. https://sites.google.com/view/enhsp/. Accessed: 2022-08-15.

Scala, E.; Ramírez, M.; Haslum, P.; and Thiébaux, S. 2016. Numeric Planning with Disjunctive Global Constraints via SMT. In *Proc. ICAPS*, 276–284.

Shleyfman, A.; Kuroiwa, R.; ; and Beck, J. C. 2022. Symmetry Detection and Breaking in Cost-Optimal Numeric Planning. In *Proc. HSDIP*.

Tennenholtz, M.; and Moses, Y. 1989. On Cooperation in a Multi-Entity Model. In *IJCAI*, 918–923.

Tuisov, A.; and Karpas, E. 2020. Automated Verification of Social Law Robustness for Reactive Agents. In *ECAI*, volume 325, 2386–2393.