Towards a Unified View of Social Laws with Instantaneous Actions

Alexander Tuisov[1], **Evgeny Mishlyakov**[1], **Alexander Shleyfman**[2] and **Erez Karpas**[1]
[1]Technion – Israel Institute of Technology
[2]Bar-Ilan University
queldelan@gmail.com, ym@campus.technion.ac.il, alexash@biu.ac.il, karpase@technion.ac.il

Abstract

Multiple agents operating in a shared environment can interfere with each other's ability to reach their goals. One of the approaches to address this issue is enacting a social law – a set of rules that restricts some possible behaviors of the agents. A social law is considered robust if it guarantees that each agent can achieve its goal independently of the actions of other agents. Recent work has shown how to verify that a given social law, encoded in an MA-STRIPS formalism, is robust by compilation to classical planning. Follow-up work presented an extended compilation which can handle numeric multi-agent planning. In this paper, we present a new compilation, which can handle both classical and numeric multi-agent planning formalisms, as well as any other multi-agent planning formalism with instantaneous actions, in which action preconditions can be negated using first-order logic with equality. This opens the door to using social laws in even richer planning formalisms. Our empirical evaluation shows that the added expressivity of the new compilation does not hurt its performance, and it achieves comparable performance to the previous state-of-the-art compilations.

1 Introduction

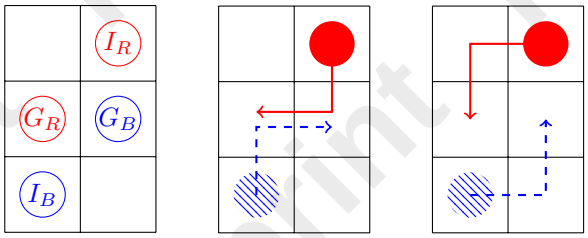
Autonomous agents and multi-agent systems have been extensively studied across different sub-fields of AI. A core challenge in complex, multi-agent environments is that in order for an agent to choose a course of action, it must account for both its own actions and those of the other agents. We consider a setting in which non-collaborative agents operate in a shared environment, particularly where self-interested agents risk unintentionally disrupting each other's activities. In such cases, a centralized planning algorithm that designs and coordinates individual plans is impractical.

The social law approach [Tenneholtz and Moses, 1989; Shoham and Tennenholtz, 1992] offers a means of imposing implicit coordination, and enables each agent to plan independently, while the system design aims to minimizes conflicts between the different plans. Social laws (SL) restrict permissible behaviors, akin to real-world traffic laws, where drivers follow rules to maintain collective safety. These laws, despite being restrictive, still permit every driver to reach their designated locations, i.e., these laws are robust. Generally, a SL is *robust* if it prevents conflicts among agents, ensuring each can reach its goal.

While the concept of SLs is general, it does not always specify how each agent chooses its actions. We are concerned with *planning-based* SLs – that is, applying SL to systems in which agents must choose a long-term course of action – agents must plan. Previous work on planning-based SLs [Karpas *et al.*, 2017] demonstrated that verifying the robustness of a SL in multi-agent (classical) planning can be reduced to a classical planning problem – the robustness verification problem. The objective of the robustness verification problem is to find a counterexample: if the compiled problem is unsolvable, then the SL is robust. Later work [Nir *et al.*, 2023] extended this approach to numeric planning, introducing a modified compilation to numeric planning that accounts for violations of numeric action preconditions.

In this paper, we propose a new compilation for verifying the robustness of a SL in a multi-agent planning system. Our approach is more general than previous methods, as it treats interference details between actions as independent of the compilation, determined by an external procedure. Thus, our new compilation can handle both numeric and classical planning formalisms, as well as **any new formalism** – provided (a) actions are instantaneous, and (b) we can efficiently compute the negation of the preconditions of every action. Note that our compilation uses the same features as the multi-agent (input) problem. That is, if the multi-agent problem has numeric variables, the compiled problem will have numeric variables as well. However, if the original problem only has classical preconditions/effects, the compiled problem will be classical.

A more general compilation might lead to a performance drop; however, our empirical evaluation demonstrates that solving the robustness verification problem from our new compilation is comparable to that of previous methods, maintaining state-of-the-art performance. Moreover, treating action preconditions as general logical formulas is a first step towards the integration of planning-based SLs with research on social norms [Ågotnes *et al.*, 2010; Ågotnes and Wooldridge, 2010; Ågotnes *et al.*, 2012], which explores a modal logic of norm compliance.



(a) Setting (b) Plan 1 (c) Plan 2

Figure 1: Illustrative Example

2 Preliminaries

Deterministic Multi-Agent Problem Formalism In this work we deal with problems represented in a formalism inspired by the work of Brafman and Domshlak ïs more general, than previous methodsbrafman-domshlak:icaps-2008, in which each agent has its own goal [Karpas *et al.*, 2017]. Such a problem is given by $\Pi = \langle \mathcal{V}, \{\mathcal{A}_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$, where $\mathcal{V}$ is the *set of variables* (boolean, finite-domain or numeric), $\{\mathcal{A}_i\}$ is the *set of actions* available to *agent* i (each agent is numbered from $[n] := \{1, \ldots, n\}$). The set of states $\mathcal{S}$ is a full assignment on the variables in $\mathcal{V}$. A computable boolean formula f over the variables in $\mathcal{V}$ is called a condition, and it can be either *true* or *false*. We say that a state $s \in \mathcal{S}$ satisfies the conditions f, $s \models f$, if the formula f is true under the assignment of the values of s to f. For a set of conditions $\mathcal{F}$, we say that $s \models \mathcal{F}$ if $s \models f$ for each $f \in \mathcal{F}$. Each *action* is a tuple $\langle \mathsf{pre}(a), \mathsf{eff}(a) \rangle$, where $\mathsf{pre}(a)$ is a set of computable boolean formulas over $\mathcal{V}$, and $\mathsf{eff}(a)$ is a set of changes (modeled as assignments to state variables) the action a applies to the values of s. We say that a is applicable in s if $s \models \mathsf{pre}(a)$. We also assume that the changes introduced by a to s are well-defined, i.e., $\mathsf{eff}(a)$ affects each variable at most once and the resulting value lies within the domain of the variable, while variables that are not affected by $\mathsf{eff}(a)$ keep their values. We assume that both preconditions and effects are computable in polynomial time. The result of applying an (applicable) action a to the state s is denoted by $s[\![a]\!]$. The *initial state* I is a state. Agent $i \in [n]$ has its own *goal* G_i, a set of conditions on $\mathcal{V}$.

For a single agent i, a planning task is given as a tuple $\Pi_i := \langle \mathcal{V}, \mathcal{A}_i, I, G_i \rangle$. In the literature these are often called simply planning tasks. In what follows such tasks are represented with a single set of actions and goals. The result of sequentially applying (if possible) the sequence of actions π to the state s is denoted by $s[\![\pi]\!]$. The sequence of actions $\pi_i = \langle a_1, \ldots, a_m \rangle$ is called an *individual plan* if $I[\![\pi_i]\!] \models G_i$ and $a_j \in A_i$ for each $j \in [m]$. π_i is also a plan for the planning task $\langle \mathcal{V}, \mathcal{A}_i, I, G_i \rangle$. A *joint plan* is a consecutively applicable sequence of actions π such that $I[\![\pi]\!] \models \bigcup_{i=1}^n G_i$. Consequently, π is a plan for the task $\Pi_c := \langle \mathcal{V}, \bigcup_{i=1}^n \mathcal{A}_i, I, \bigcup_{i=1}^n G_i \rangle$.

Running Example Consider an example with 2 agents (solid red and dashed blue, denoted R and B, respectively) in a 2x3 grid (cells are denoted NW, NE, CW, CE, SW, SE for north-west, north-east, central-west, central-east, south-west, south-east, respectively). Initially R is at NE and B is at SW. Their goals are to reach CW and CE, respectively. This is depicted in Figure 1a, where initial positions are depicted with I_R, I_B and goals with G_R, G_B.

We encode this problem in our formalism using 2 finite domain variables: $\mathcal{V} = \{\mathrm{loc}(R), \mathrm{loc}(B)\}$, which denote the locations of each agent. Both variables share a domain, $dom(\mathrm{loc}(B)) = dom(\mathrm{loc}(R)) = \{\mathrm{NW}, \mathrm{NE}, \mathrm{CW}, \mathrm{CE}, \mathrm{SW}, \mathrm{SE}\}$. In the initial state I we have $\mathrm{loc}(B) = \mathrm{SW}$ and $\mathrm{loc}(R) = \mathrm{NE}$, while the goals are $G_B = \{\mathrm{loc}(B) == \mathrm{CE}\}$ and $G_R = \{\mathrm{loc}(R) == \mathrm{CW}\}$.

Each agent has 14 move actions corresponding to possible moves from each of the 6 different locations – 2 from each corner, 3 from the center positions. The action for moving agent i from location l_1 to adjacent location l_2 is denoted $\mathrm{move}(i, l_1, l_2)$ where $i \in \{R, B\}$, $l_1 \in \{\mathrm{NW}, \mathrm{NE}, \mathrm{CW}, \mathrm{CE}, \mathrm{SW}, \mathrm{SE}\}$, and l_2 iterates over the adjacent locations to l_1 (for example, if $l_1 = \mathrm{NW}$, l_2 iterates over NE and SW). The action preconditions check that agent i is indeed at location l_1, and that the other agent (denoted $-i$) is not at l_2, so as not to crash into it. This is given by $\mathsf{pre}(\mathrm{move}(i, l_1, l_2)) = \{\mathrm{loc}(i) == l_1, \mathrm{loc}(-i) \neq l_2\}$. The effects of $\mathrm{move}(i, l_1, l_2))$ change the location of agent i to l_2, that is, $\mathsf{eff}(\mathrm{move}(i, l_1, l_2)) = \{\mathrm{loc}(i) \leftarrow l_2\}$. Thus, the actions of agent i are $A_i = \{\mathrm{move}(i, l_1, l_2) \mid l_1 \in \{\mathrm{NW}, \mathrm{NE}, \mathrm{CW}, \mathrm{CE}, \mathrm{SW}, \mathrm{SE}\}, l_2 \text{ adjacent to } l_1\}$.

Social Laws In multi-agent systems, a common issue arises when agents' individual plans conflict. For instance, two agents may need the same unique tool for their tasks. If the first agent uses the tool and fails to return it to the toolbox, this prevents the second agent from completing its task.

Returning to our running example, the two individual plans depicted in Figure 1b (consisting of $\pi_B = \langle \mathrm{move}(B, \mathrm{SW}, \mathrm{CW}), \mathrm{move}(B, \mathrm{CW}, \mathrm{CE}) \rangle$ and $\pi_R = \langle \mathrm{move}(R, \mathrm{NE}, \mathrm{CE}), \mathrm{move}(R, \mathrm{CE}, \mathrm{CW}) \rangle$) conflict, as the two agents might collide, that is, one of the preconditions of one of the actions will be violated. The exact action that will fail depends on the timing – when each action is executed. For example, if the joint plan is $\langle \mathrm{move}(B, \mathrm{SW}, \mathrm{CW}), \mathrm{move}(B, \mathrm{CW}, \mathrm{CE}), \mathrm{move}(R, \mathrm{NE}, \mathrm{CE}), \mathrm{move}(R, \mathrm{CE}, \mathrm{CW}) \rangle$ then $\mathrm{move}(R, \mathrm{NE}, \mathrm{CE})$ will fail as its precondition $\mathrm{loc}(B) \neq CE$ is false when it is executed.

To address such conflicts, previous work [Tenneholtz and Moses, 1989] proposed the concept of SL – a set of restrictions designed to improve coordination among agents. SLs can be applied to multi-agent planning problem [Karpas *et al.*, 2017], where a SL modifies a multi-agent planning setting Π, yielding a new setting $\tilde{\Pi}$. This modification may involve removing certain actions [Nir *et al.*, 2020] or adding auxiliary facts that aid in managing agents' behaviors through bookkeeping [Tuisov and Karpas, 2020; Tuisov *et al.*, 2024]. SLs can also implement more complex restrictions using a *waitfor* mechanism, which holds an agent inactive until a condition f is satisfied before it executes its next action. We denote the *waitfor* condition of action a by $\mathsf{pre}_w(a)$, while the remainder of the condition of a is denoted by $\mathsf{pre}_f(a)$ (for failure).

To illustrate the concept of *waitfor*, consider our running example. It makes sense for an agent that is trying to move to some location l_2 to wait until l_2 is clear before it moves into it.

Thus, we can write $\mathsf{pre}_w(\text{move}(i, l_1, l_2))) = \{\text{loc}(-i) \neq l_2\}$, meaning that the agent will not move into an occupied cell, but instead wait for the cell to become clear before moving. Note that the plans depicted in Figure 1b may now result in a *deadlock*, as each agent will wait for the other agent to move. Again, the exact position of the deadlock will depend on the timing of the actions.

A key property to evaluate for any SL is its **robustness**, which ensures it effectively prevents conflicts between agents, regardless of the plans they choose to execute.

Robustness We say that a planning task is *robust* if every agent can choose any individual plan, when it plans as if it was the only agent *acting* in the world, and be guaranteed it will achieve its goal. To formalize this notion we must first define the execution model of our multi-agent setting.

Let $\{\pi_i\}_{i=1}^n$ be a set of individual plans. The set of possible executions for these plans is defined by considering (a) the state of the world and (b) the position of each agent within its individual plan. The initial state of execution is $\langle I, 1, \dots 1\rangle$, indicating that the state of the world is I, and all of the agents are at the beginning of their individual plans. We define four special terminal states of execution: **Success**: all agents have executed their plans, ending in a goal state for each agent; **Failure** at execution: an agent tried to execute an action with non-*waitfor* preconditions that are not satisfied; **Deadlock**: none of the agents in the system can act; and **Goal miss**: all agents have executed their plans, ending in a non-goal state for some agent.

The possible transitions from a state of execution $\langle s, j_1, \dots, j_n\rangle$ are defined by the set of agents which can act at state s, by looking at their *waitfor* conditions. Denote the next action of agent i by $\pi_i[j_i]$ (that is, the action in position j_i in plan π_i, where we assume $\pi_i[j_i] = \bot$ if agent i has completed its plan), and denote the set of agents which can act by $C = \{i \mid \pi_i[j_i] \neq \bot, s \models \mathsf{pre}_w(\pi_i[j_i])\}$, that is, agents for which the current state of the world satisfies the *waitfor* condition of their next action. Note that the set of transitions only looks at *waitfor* conditions, and not at the full preconditions of the actions.

If there are no agents that can act, meaning $C = \emptyset$, we have two subcases. If $s \models \bigwedge_{i=1}^n G_i$, there is a single transition to **Success**, indicating that all agents have successfully completed their plans and achieved their goals. Otherwise, if $s \not\models \bigwedge_{i=1}^n G_i$, there is either a single transition to **Deadlock**, as no agents can act, or **Goal miss** as all agent finished executing their plans.

If agents can still act, there are several possible transitions, one for each agent $i \in C$. Again, we have two subcases. If $s \not\models \mathsf{pre}(\pi_i[j_i])$, meaning the current state of the world does not satisfy the preconditions of the next action of agent i, the resulting state from applying the action of agent i is **Failure**. Conversely, if $s \models \mathsf{pre}(\pi_i[j_i])$, the action is successfully applied, and the resulting state from applying the action of agent i is the execution state $\langle s[\![\pi_i[j_i]]\!], j_1, \dots, j_{i-1}, j_i + 1, j_{i+1}, \dots, j_n\rangle$, meaning the state of the world is updated, and the position in π_i is incremented by 1.

We can now define *robustness*: a SL is robust for Π iff for every set of individual plans $\{\pi_i\}_{i=1}^n$, every possible execution of these plans results in **Success**. Coming back to our running example, consider a social law in which (a) an agent does not move into an occupied location (formulated using a *waitfor* condition, as described above), and (b) agents are only allowed to move in a counter-clockwise direction, formulated by removing the actions which move clockwise from the action sets A_R and A_B. Figure 1c shows the resulting plans under this SL, and indeed it is possible to show that this SL is robust.

Previous work [Karpas *et al.*, 2017] has shown how to verify whether a given SL in an MA-STRIPS setting is robust. However, it cannot be directly applied to our running example, as (a) it uses finite domain variables, and (b) it employs negation in preconditions. In the next section, we describe a general computational technique for checking the robustness of any SL in any multi-agent setting, which can be directly applied to our running example, and many others.

3 Generalized Compilation

We now describe our contribution – a new technique for verifying the robustness of a given social law. Similarly to previous work [Karpas *et al.*, 2017] this technique relies on creating a planning problem whose objective is to find a counterexample to robustness (that is, a trajectory in the execution model which leads to **Failure**, **Deadlock**, or **Goal miss**). However, the previous work relied on the planning problem being formulated in MA-STRIPS or numeric MA-STRIPS [Nir *et al.*, 2023], while the compilation we present here is directly applicable with instantaneous actions, and for which we can compute the negation of a condition efficiently. Thus, the new compilation is directly applicable to our running example, without having to translate it to MA-STRIPS.

Similarly to the original compilation [Karpas *et al.*, 2017], the new compilation generates $n + 1$ copies of each variable from the original problem. We label copies 1 through n as *local copies* (one per agent), while the final copy, denoted by g, is the *global copy*. The global copy emulates joint execution, whereas the local copies assures that each agent indeed independently plans toward its own goal.

Our compilation operates in two stages. In stage 1, we simulate the execution process, where each agent applies its actions in an order chosen by the planner while respecting *waitfor* conditions. In other words, stage 1 accounts for the actions actually executed. Once stage 1 terminates, we move to stage 2, which handles bookkeeping – ensuring that the actions executed by each agent in stage 1 form a prefix of an individual plan for that agent (otherwise, agents could execute actions that do not lead to a goal).

In stage 1, an action a_i may succeed, fail, or result in a deadlock (waiting forever), which is represented by having 3 different versions of a_i applicable in stage 1: a_i^s (success), a_i^f (failure), and a_i^w (wait). Applying a_i^s is possible when the global state satisfies $\mathsf{pre}(a_i)$, updates the global state with the effects of a_i and keeps the compilation in stage 1. Applying a_i^f is possible when the global state does not satisfy $\mathsf{pre}(a_i)$ but does satisfy $\mathsf{pre}_w(a_i)$, raises the *precondition violation (PrV)* flag, and moves the compilation to stage 2. Following this, the compilation ensures that each agent can reach its goal

in its local copy of the state by applying a fourth version of each action, a_i^l, which only updates the local copy of agent i.

Applying the wait action a_i^w is only possible when the global state does not satisfy $\mathsf{pre}_w(a_i)$, signaling a potential **Deadlock**. This means that, from this point onward, agent i cannot execute any action except a_i until a_i is executed. Executing a_i^w sets a flag that allows agent i to execute only a_i, encoded by the condition $RA_i \wedge allow_{(a_i)}$, but does not update the global state with the effects of a_i. After this flag is raised, agent i has two options: either execute a_i in both its local and global copies or execute the deadlock action deadlock-a$_i$. If all agents are either in a deadlock state or have completed their plans, stage 1 ends, transitioning the compilation to stage 2. Then, as with a_i^f, the local copies are used for each agent to achieve its goal separately.

We also allow the search to continue to stage 2 once all agents have achieved their goals. The goal of the compilation is a fact *conflict* which signals that either a deadlock or a precondition violation occurred. This is achieved by declare-deadlock, which checks that a deadlock occurred or declare-fail, which checks that a precondition violation occurred by execution of some a_i^f. Finally, it may be the case that all agents have finished their respective individual plans with no interruption, but the goal of some of them is not achieved at the end of the execution. We check for this condition using the goals-not-achieved action, which performs a straightforward check. The goal flag *conflict* is achieved only by these three actions. We now describe the full details of our compilation, and then proceed with proving its correctness.

3.1 Full Compilation

We define $\Pi' = \langle \mathcal{V}', A', I', G' \rangle$ to be the compiled planning problem. A solution for Π' represents a conflict in the multi-agent problem Π. We start with defining $\mathcal{V}_i = \{v_i \mid v \in \mathcal{V}\}$ to be the variables of each local copy $i \in [n]$, and $\mathcal{V}_g = \{v_g \mid v \in \mathcal{V}\}$ to be the variables of the global copy, with $\mathsf{dom}(v_i) = \mathsf{dom}(v_g) = \mathsf{dom}(v)$. In addition, we define the following set of auxiliary flags, each with a Boolean domain:

$$\begin{aligned} \mathit{Aux} = &\{conflict, stage_1, stage_2, \mathit{PrV}, \mathit{PoD}\} \\ &\cup \{allow_{(a)} \mid a \in \cup_{i=1}^n A\} \cup \{\mathit{RA}_i, fin_i \mid i \in [n]\}. \end{aligned}$$

Using this notation we define the variables of Π' to be $\mathcal{V}' = Aux \cup \mathcal{V}_g \cup \bigcup_{i\in[n]} \mathcal{V}_i$. Also to ease the notation, for each flag $flag$ we use the following shorthand: $flag = \top$ is replaced with $flag$ and $flag = \bot$ with $\neg flag$, this holds for both assignments and equalities. Since all the conditions in the set must hold true, each set of conditions can be written as $X = \bigwedge_{f\in X} f$, using the same notation $\neg X = \bigvee_{f\in X} \neg f$.

We continue with the actions of Π'. For each set of formulas X over the variables in $\mathcal{V}$, we denote by X^i and X^g the same set of formulas but written over the variables in the copies $\mathcal{V}_i$ and $\mathcal{V}_g$, respectively. This applies to both sets of conditions such as $\mathsf{pre}(a_i)$ or G_i are mapped into conditions $\mathsf{pre}^i(a_i)$ and $\mathsf{pre}^g(a_i)$ and G_i^i and G_i^g, respectively. The goal of the global copy is denoted $G^g := \bigwedge_{i=1}^n G_i^g$. Note that each agent i affects only its copy i, the global copy g or both. The same notation also applies to the effects of each action a_i, which are applied to the the copy of the agent i and global copy g. For each action in A_i we define five distinct copies:

$$A_i' = \{a_i^s, a_i^f, a_i^w, a_i^l, \text{deadlock-a}_i \mid a_i \in A_i\}.$$

Of these, three change the local copy of the agent – a_i^s, a_i^f, a_i^l. These actions also do the following: a_i^s is the successful action that changes the global copy where all agents act in stage 1, a_i^f indicates failure due to precondition violation and stops the execution in the global copy ending stage 1 and indicating failure, and the action a_i^l that assures that each agent still achieves its goal in its local copy. Formally they are defined as follows:

- $\mathsf{pre}(a_i^s) = stage_1 \wedge (allow_{(a_i)} \vee \neg \mathit{RA}_i) \wedge \mathsf{pre}^i(a_i) \wedge \mathsf{pre}^g(a_i)$,
  $\mathsf{eff}(a_i^s) = \mathsf{eff}^i(a_i) \cup \mathsf{eff}^g(a_i) \cup \bigcup_{a_i \in A_i} \{allow_{(a_i)}\}$,
- $\mathsf{pre}(a_i^f) = stage_1 \wedge (allow_{(a_i)} \vee \neg \mathit{RA}_i) \wedge \mathsf{pre}^i(a_i) \wedge \mathsf{pre}_w^g(a) \wedge \neg \mathsf{pre}_f^g(a_i)$,
  $\mathsf{eff}(a_i^f) = \{\mathit{PrV}, stage_2, \neg stage_1\} \cup \mathsf{eff}_i(a_i)$,
- $\mathsf{pre}(a_i^l) = stage_2 \wedge (allow_{(a_i)} \vee \neg \mathit{RA}_i) \wedge \mathsf{pre}^i(a_i)$,
  $\mathsf{add}(a_i^l) = \mathsf{eff}^i(a_i) \cup \{allow_{a_i}, \neg \mathit{RA}_i\}$;

The actions a_i^w and deadlock-a$_i$ are actions that manage waiting, where the first action indicates that agent i is waiting to execute action a_i, and the second indicates that a potential deadlock has occurred. Formally:

- $\mathsf{pre}(a_i^w) = stage_1 \wedge allow_{(a_i)} \wedge \mathsf{pre}^i(a_i) \wedge \neg \mathsf{pre}_w^i(a_i)$,
  $\mathsf{eff}(a_i^w) = \bigcup_{a \in A_i \setminus \{a_i\}} \{\neg allow_{(a)}\} \cup \{\mathit{RA}_i\}$;
- $\mathsf{pre}(\text{deadlock-a}_i) = allow_{(a_i)} \wedge \neg \mathsf{pre}_w^g(a_i) \wedge \mathit{RA}_i$,
  $\mathsf{eff}(\text{deadlock-a}_i) = \{fin_i, \mathit{PoD}, \neg stage_1\}$;

We also have auxiliary actions: end-success$_i$ is applicable if the agent achieves its goal in both the local and global copies. Another action initiates stage 2, where each agent pursues its goal in the local copy. Three additional actions conclude the plan with *failure*.

$$\begin{aligned} A' = \{\text{start-stage}_2\} \cup \bigcup_{i=1}^n (A_i' \cup \{\text{end-success}_i\}) \cup \\ \{\text{declare-deadlock}, \text{declare-fail}, \text{goals-not-achieved}\}. \end{aligned}$$

Below we present these actions in detail:

- $\mathsf{pre}(\text{end-success}_i) = G_i^g$,
  $\mathsf{eff}(\text{end-success}_i) = \{fin_i, \neg stage_1\}$;
- $\mathsf{pre}(\text{start-stage}_2) = \bigwedge_{i\in[n]} fin_i$,
  $\mathsf{eff}(\text{start-stage}_2) = \{stage_2, \neg stage_1\}$;
- $\mathsf{pre}(\text{goals-not-achieved}) = stage_2 \wedge \neg G^g \wedge \bigwedge_{i=1}^n G^i$,
  $\mathsf{eff}(\text{goals-not-achieved}) = \{conflict\}$;
- $\mathsf{pre}(\text{declare-deadlock}) = stage_2 \wedge \mathit{PoD} \wedge \bigwedge_{i=1}^n G^i$,
  $\mathsf{eff}(\text{declare-deadlock}) = \{conflict\}$; and
- $\mathsf{pre}(\text{declare-fail}) = stage_2 \wedge \mathit{PrV} \wedge \bigwedge_{i=1}^n G^i$,
  $\mathsf{eff}(\text{declare-fail}) = \{conflict\}$.

In the initial state we set all the copies to emulate the initial state of Π and declare that the agents currently in stage 1:

$$\begin{aligned} I' = &\{stage_1\} \cup \{I[v_i] = I[v] \mid v \in \mathcal{V}, i \in [n]\} \\ &\cup \{I[v_g] = I[v] \mid v \in \mathcal{V}\}, \end{aligned}$$

all flags other than $stage_1$ are set to *false*. The goal is to achieve *conflict*, i.e., $G' = \{\textit{conflict}\}$.

Compilation Example To better understand the compilation, we present the solution to the compilation in our running example under several different possible social laws. For the original setting without any social laws, recall the agents can crash into each other. One possible solution for the compilation (the same one explained above) is: $\langle \text{move}(B, \text{SW}, \text{CW})^s, \text{move}(B, \text{CW}, \text{CE})^s,$ $\text{move}(R, \text{NE}, \text{CE})^f, \text{move}(R, \text{CE}, \text{CW})^l, \text{declare-fail}\rangle$. Note that the third action in this solution is the one in which R moves into a cell already occupied by B, violating one of the preconditions, which is why we can use the the a^f version of the action.

Now consider the social law where we turn this violated precondition into a *waitfor* condition, meaning that an agent will wait rather than moving into an occupied cell. In this case, we can get a different solution to the compilation resulting in a deadlock: $\langle \text{move}(B, \text{SW}, \text{CW})^s, \text{move}(B, \text{CW}, \text{CE})^s,$ $\text{move}(R, \text{NE}, \text{CE})^w, \text{deadlock-move}(R, \text{NE}, \text{CE})^w,$ $\text{move}(R, \text{CE}, \text{CW})^l, \text{declare-deadlock}\rangle$. Finally, if we also add the restriction of only moving counter-clockwise (meaning the above solution is no longer possible, as B and R move clockwise in it), our compilation yields an unsolvable planning problem – proving that this social law is robust.

3.2 Proof of Correctness

We formally prove the correctness of our compilation in three steps. First, we establish a correspondence between executions in the original task and plans that solve the compiled task. Second, we demonstrate that if the compiled problem is solvable, the underlying problem is not robust. Finally, we prove the converse: if the underlying problem is not robust, then the compiled task is indeed solvable.

Let $\Pi = \langle \mathcal{V}, \{\mathcal{A}_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ denote the original problem, and let $\Pi' = \langle \mathcal{V}', \mathcal{A}', I', G' \rangle$ be the compiled problem as described in Section 3.1. We use a prime notation when referring to elements of Π' if the context might be ambiguous; for instance, π' denotes a plan that solves Π'.

Our goal is to provide the connection between the interleavings of agents' individual plans in Π and the plans of the compiled problem Π'.

Observation 1. *Every plan π' solving Π' must end with either goals-not-achieved, declare-deadlock or declare-fail actions. All of these have $stage_2$ as their preconditions and no action deletes $stage_2$. Moreover, the initial state I' includes $stage_1$, no action adds $stage_1$, and every action that adds $stage_2$ deletes $stage_1$ (but not vise versa, some actions delete $stage_1$ without adding $stage_2$). Thus any π' will include a unique action a'_i deleting $stage_1$, a unique a'_j that adds $stage_2$ and it is guaranteed that $i \leq j$. This means, it is always possible to split π' into a prefix $pref(\pi')$, a (possibly empty) infix $inf(\pi')$, and a suffix $suf(\pi')$, where the prefix ends with a'_i (ending $stage_1$), and the suffix starts with a'_{j+1} (starting $stage_2$).*

We establish a correspondence between executions in Π and plans for Π'.

Definition 1 (Equivalent sequence and execution)**.** *Let π' be a plan for the compiled problem Π'. Define $\pi^{[n]}$ to be an **equivalent execution of** π' as a restriction of $pref(\pi')$ to the variables in $\mathcal{V}_g = \{v_g \mid v \in \mathcal{V}\}$. Note that since a_i^w does not affect $\mathcal{V}_g$ it can be safely removed from consideration.*

*Let $\pi^{[n]} = (a_1, \ldots, a_{m-1}, a_m)$ be an execution of the individual plans $\{\pi_i\}_{i=1}^n$ in the multi-agent task Π. Define $eq(\pi^{[n]})$ an **equivalent sequence of** $\pi^{[n]}$ in the compilation Π' to be $(a_1^s, \ldots, a_{(m-1)}^s)$ if execution fails, otherwise $(a_1^s, \ldots, a_m^s)$.*

The equivalent sequence of a valid execution is always sequentially applicable in Π', since $\mathcal{V}_g$ and every $\mathcal{V}_i = \{v_i \mid v \in V\}$ are copies of $\mathcal{V}$, effects of any action a on $\mathcal{V}$ are identical to effects of a^s on $\mathcal{V}_g$, and the effects of any a_i are identical to the effects of a_i^s on $\mathcal{V}_i$. This means that the state of $\mathcal{V}$ and every Π_i after executing some $\pi^{[n]}$ is identical to the state $\mathcal{V}_g$ and every $\mathcal{V}_i$ respectively after executing $eq(\pi^{[n]})$.

Our goal it to show that for every failed execution $\pi^{[n]}$ in Π there is a plan π' for Π', and vice versa. More formally:

Theorem 1 (The correctness of the compilation)**.** *Given a multi-agent task $\Pi = \langle \mathcal{V}, \{\mathcal{A}_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ and the complied single agent task $\Pi' = \langle \mathcal{V}', \mathcal{A}', I', G' \rangle$, Π' is solvable if and only if Π is not robust.*

Proof is by the combining the following two lemmas.

Lemma 1. *$\pi^{[n]}$ is a failed execution $\implies$ Π' is solvable.*

Proof. Assume $\pi^{[n]}$ is a failed execution. We want to construct a corresponding plan π' for Π'. This means that π is applicable in the initial state I' and achieves $G' = \{conflict\}$. To construct π', it is sufficient to specify $pref(\pi'), inf(\pi')$, and $suf(\pi')$.

By construction of $\mathcal{V}'$, $eq(\pi^{[n]})$ is consecutively applicable in Π'. Since $pref(\pi')$ includes all actions in π' executed in $stage_1$, we have that $eq(\pi^{[n]})$ is a prefix of $pref(\pi')$. Failed execution $\pi^{[n]}$ implies that individual plans $\{\pi_i\}_{i=1}^n$ ended either with **Missing the goal** $\bigwedge_{i=1}^n G_i$, in a **Failure**, or in a **Deadlock**. For each agent i, either π_i is a subsequence of $\pi^{[n]}$, which means i had finished its execution by the end of $\pi^{[n]}$, or π_i is not a subsequence of $\pi^{[n]}$.

Denote the set of all agents that finished their execution as $\text{FINISHED}(\pi^{[n]})$ and define $\text{NOT-FINISHED}(\pi^{[n]}) := [n] \setminus \text{FINISHED}(\pi^{[n]})$. In a case when $\text{NOT-FINISHED}(\pi^{[n]}) \neq \emptyset$, we call the suffix of π_i that does not appear in $\pi^{[n]}$ an unexecuted suffix of i. By γ^l we denote some arbitrary interleaving of unfinished suffixes for all $i \in \text{NOT-FINISHED}(\pi^{[n]})$. Note that γ^l are executed in local copies of the appropriate agents and therefore the order of agents is not important. The sequence of actions γ^l is the part of $suf(\pi')$

We aim at reconstructing a plan π', such that $eq(\pi^{[n]})$ is the prefix of π'. We also introduce the following notation for plan reconstruction purposes: $x \parallel y$ stands for *concatenation* of action sequences x and y. Consider three cases:

Case 1 (Goal miss): $\bigwedge_{i=1}^n G_i$ is not achieved after executing $\pi^{[n]}$ and neither failure nor deadlock occurred. Since $eq(\pi^{[n]})$ is the equivalent sequence of $\pi^{[n]}$ the global copy variables in the states in Π' correspond to the states of Π, we know that after the execution of $\pi^{[n]}$, the condition $\bigwedge_{i \in [n]} G_i^g =$

$\bigwedge_{i\in[n]}\bigwedge_{f\in G_i} f_g$ was not achieved. And since all agents have finished their executions the local copy goals $G^i_i := \bigwedge_{f\in G_i} f_i$ were achieved for each i.

Hence, the plan π' looks as follows: $pref(\pi') = eq(\pi^{[n]})$, $inf(\pi') = \left(\|_{i\in[n]}\text{end-success}_i\right) \|$ start-phase-2, $suf(\pi') =$ goals-not-achieved. Note that the application of the actions end-success$_i$ can be done in any order, thus any predefined order will do. π' is valid since the actions are consecutively applicable, and *conflict* is true in the final state.

Case 2 (Failure): $\pi^{[n]}$ ends in a failure. Denote the action failing as a^α. The full plan π' will look as follows: $pref(\pi') = eq(\pi^{[n]}) \| a^\alpha$, $inf(\pi') = \emptyset$, $suf(\pi') = \gamma^l \|$ declare-fail. This plan is valid because after applying $eq(\pi^{[n]})$, global preconditions of a^α are not satisfied by equivalence of $eq(\pi^{[n]})$. Local preconditions of a^α, however, are satisfied because otherwise a^α could not be picked to be the next action performed in Π. At the end of $eq(\pi^{[n]})$ the local executions continue. The local actions are executable in arbitrary order since the local copies have no effect on each other, and after their execution it necessarily holds that $G^i_i := \bigwedge_{f\in G_i} f_i$ is satisfied for each i, otherwise π_i would not be a plan in Π_i.

Case 3 (Deadlock): $\pi^{[n]}$ ends in a deadlock. This means, every agent had either finished its execution or waiting for some $waitfor$ precondition to become true. Then π' will look as follows: $pref(\pi') = eq(\pi^{[n]}) \| \left(\|_{i\in\text{NOT-FINISHED}(\pi^{[n]})} a^w_i\right)$ where a_i is the first action in the unexecuted suffix for each i;

$$inf(\pi') = \left(\|_{i\in\text{NOT-FINISHED}(\pi^{[n]})}\text{deadlock-a}_i\right) \| \left(\|_{i\in\text{FINISHED}(\pi^{[n]})}\text{end-success}_i\right) \| \text{start-phase-2};$$

and $suf(\pi') = \gamma^l \|$ declare-deadlock, where all the actions end-success$_i$ and deadlock-a$_i$ are executed in arbitrary order.

This plan is valid because after executing $\pi^{[n]}$, we know that every $i \in$ NOT-FINISHED$(\pi^{[n]})$ was waiting for some $waitfor$ precondition to become true. Thus, after applying $eq(\pi^{[n]})$, for each $i \in$ NOT-FINISHED$(\pi^{[n]})$ there will be such an a_i that its regular preconditions are met, but *wait-for* preconditions are not. Thus, a^w_i are applicable, and can be applied in arbitrary order since they have no effect on each others preconditions. After that, deadlock-a$_i$ actions become applicable. As soon as one is applied, $stage_1$ is deleted, and no actions but the deadlock-a$_i$, end-success$_i$, and start-stage$_2$ are applicable. To achieve fin_i for all $i \in [n]$, deadlock-a$_i$ is performed for waiting agents, and end-success$_i$ is performed for finished agents, again in arbitrary order. After that, start-stage$_2$, and, subsequently local actions become applicable. The local actions are executable in arbitrary order since they have no effect on each other, and after their execution it is necessary that $\bigwedge_{f\in G_i} f_i$ is satisfied for each i, otherwise, π_i would not be a plan in Π_i. □

Lemma 2. *Π' is solvable $\implies$ there is failed execution $\pi^{[n]}$.*

Proof. Let π' be a plan for Π'. We need to show that there is a set of individual plans $\{\pi_i\}_{i=1}^n$ such that their interleaving execution $\pi^{[n]}$ is not a joint plan. In other words, $\pi^{[n]}$ ends in either **Goal miss**, **Failure**, or **Deadlock**.

The plan π' restricted to local copies of each agent variables $\mathcal{V}_i = \{v_i \mid v \in \mathcal{V}\}$. The only actions that have effect on local copy i are a^s_i, a^f_i and a^l_i, thus to reconstruct individual plan for agent i every other action can be safely removed from consideration. The remaining actions corresponds to some individual plan π_i, since the plan π' should achieve local goals of each agent to achieve the $conflict$ goal in Π'.

Note that π' defines the waitfor-respecting order of execution of the individual plans $\{\pi_i\}_{i=1}^n$. Define $\pi^{[n]}$ to be the $stage_1$ subsequence of actions in π' restricted to the original actions that correspond to a^s_i and a^f_i. Note that we ignore the waiting actions of the form a^w_i since these actions change only the flags, but not the copies of the original variables.

From Observation 1 we know that π' includes either goals-not-achieved, declare-fail or declare-deadlock actions. We distinguish between these cases, since they define the nature of the executions failure:

Case 1 (Goal miss): π' includes goals-not-achieved. By construction, this means that a goals-not-achieved flag was raised. This is only possible if every fin_i was raised, but some goal was not achieved in the global copy. By equivalence of π' and $\pi^{[n]}$, at the end of $\pi^{[n]}$ some global goal will not be achieved as well.

Case 2 (Failure): π' includes declare-fail. By construction, the precondition-violation flag was raised, thus some action a^f_i was applied, and it was the last action in $pref(\pi')$ since every a^f_i deletes $stage_1$ flag. By equivalence of π' and $\pi^{[n]}$, at the last step of the execution action a_i was not applicable, thus its application would result in a failure. Moreover, every a_i is performed in an individual plan π_i. Such π_i is a subsequence of π' restricted to agent i, where each $a_i \in \pi_i$ has its source in a^s_i, a^f_i or a^l_i, respectively. It is guaranteed that such π_i exists because π' has to achieve $\bigwedge_{i=1}^n G_i$ in the local copies to apply declare-fail.

Case 3 (Deadlock): π' includes declare-deadlock. By construction, the possible-deadlock flag was raised, thus at least one action deadlock-a$_i$ was applied. This means that action start-stage$_2$ was applied, because deadlock-a$_i$ and a^f_j for any agent i, j are mutually exclusive, since both require and remove the stage$_1$ flag. The start-stage$_2$ action requires performing either end-success$_i$ or deadlock-a$_i$ for each i. Since there are no failures, either end-success$_i$ or deadlock-a$_i$ were used. This means no agent i can apply a successful action, because it either has achieved its goal or is in a deadlock. The action end-success$_i$ requires $\bigwedge_{f\in G_i} f_i$, which means in the equivalent $\pi^{[n]}$ agent i has achieved its goal and stopped the execution. The action deadlock-a$_i$ requires both RA_i and $\wedge \bigvee_{f\in \mathsf{pre}_w(a_i)} \neg f_g$. The first requirement means that the previous action of agent i was a^w_i since otherwise actions in $A_i \setminus \{a_i\}$ would not be disallowed. The second requirement ensures that no action of the type a^s, not even a^s_i can be performed. Hence, during $\pi^{[n]}$ agent i waits for some precondition to become true. Thus, every agent i is either finished or waiting, meaning that $\pi^{[n]}$ results in a deadlock. □

Domain	Old Coverage	Old Time	New Coverage	New Time
CLASSICAL WITH SL				
GRID	6	5.04	6	6.23
ZENOTRAVEL	8	268.16	10	272.93
CLASSICAL NO SL				
BLOCKSWORLD	20	3.74	20	19.76
DRIVERLOG	20	3.94	19	7.2
GRID	20	6.09	20	20.58
ZENOTRAVEL	20	9.46	20	15.26
NUMERIC WITH SL				
GRID	-	-	4	18.21
EXPEDITION	-	-	1	65.89
ZENOTRAVEL	-	-	9	4.52
NUMERIC NO SL				
GRID	-	-	20	9.56
MARKET	-	-	4	82.2
EXPEDITION	-	-	4	10.89
ZENOTRAVEL	-	-	4	85.2

Table 1: Coverage (number of problems solved) and geometric mean of runtime for domains with and without SL.

4 Experimental Evaluation

The compilation presented above is more general than the earlier approaches [Karpas *et al.*, 2017; Nir *et al.*, 2023], but this increased generality might lead to more challenging planning problems. To evaluate this trade-off, we conducted an empirical study, comparing the solving times for problems generated by our compilation (**new**) and the previous one (**old**). We do not employ the numeric compilation by Nir et al. [2023], since it requires manual problem adjustment.

Both compilations were implemented[1] on top of the Unified Planning Framework [upf, 2025], and all of our benchmarks were encoded in this framework. Classical problems were grounded using the Fast-Downward grounder [Helmert, 2006]; the planner used for both compilations was the LAMA-first planner since it is the baseline/winner of the last IPC 2023 Agile Track [Taitler *et al.*, 2024]. Numeric problems were solved by ENHSP-20 [Scala *et al.*, 2020]. The planners had a time limit of 30 minutes on an Intel i7-6700k CPU, and a memory limit of 8GB for Fast Downward and 16GB for ENHSP. We measure the time it takes to solve the resulting planning problems from both compilations.

We performed the comparison on a set of benchmarks adapted from the first Competition of Distributed and Multiagent Planners [Stolba *et al.*, 2015] and from the Numeric track of IPC 2023 [Taitler *et al.*, 2024] described below:

GRID: each agent starts at a randomly assigned point and navigates toward a designated finish point to exit the map. The experiments examine movement with and without a social law. Under the social law, agents follow directional rules: they move in a circular pattern along grid borders and alternate upward and downward in inner columns, similar to a warehouse layout. The law prevents collisions by requiring agents to wait for spaces to be clear before entering them. There are both classical and numeric versions of this domain.

[1]Code available at https://github.com/TechnionCognitiveRoboticsLab/up_social_laws_ijcai2025

ZENOTRAVEL: aircraft transport individuals between cities, consuming varying fuel amounts and with the option to refuel, where each aircraft acts as an agent. Individuals board and disembark from aircraft to reach their destinations. A robust social law assigns individuals to the aircraft that is responsible for delivering them to their destinations. here is both classical and numeric version of this domain.

BLOCKSWORLD: robotic arms act as individual agents that pick up and stack blocks. While agents can independently reach their own goals, cooperation is needed for collective objectives. For instance, if agent 1's goal is $ON(A, B)$ and agent 2's goal is $ON(B, C)$, they need a three-block tower to satisfy both goals, requiring coordination beyond individual efforts. An effective social law to ensure such cooperation is yet to be identified.

DRIVERLOG: Drivers travel on foot or by truck, loading or unloading packages to reach targets. Each driver manages specific packages and, at most, one truck. A robust SL is unworkable because truckless drivers depend on borrowing and returning trucks, creating dependencies that prevent fully independent problem-solving.

MARKET TRADER: In this trading scenario, agents (camels) travel between markets to buy and sell goods, balancing costs, availability, and capacity. Actions include traveling, purchasing, upgrading capacity, and selling for profit. The goal is to maximize cash through strategic resource allocation. Instances were adjusted by reducing the number of goods to ensure solvability by existing planners.

EXPEDITION: In this domain, agents (sleds) navigate between waypoints in a network. Sleds consume supplies to travel between adjacent waypoints. Supplies can be stored at or retrieved from waypoints, constrained by sled capacity. The objective is to manage supplies efficiently while traversing the network.

Table 1 shows results for each domain without the SL, and with the SL when one is available. **new** denotes the compilation in this paper, while **old** is a compilation for MA-STRIPS presented by [Karpas *et al.*, 2017]. The table presents coverage and the geometric mean of running time (in seconds) for the problems that were solved both by **old** and **new** compilations. For numeric multiagent domains we present the (geometric) mean time of the problems solved by the **new** compilation. The tables also show that while the new compilation is slightly slower, it is able to verify two more instances with SL in the classical ZENOTRAVEL, and finds one less counter example in DRIVERLOG. Exact runtimes for each problem are available in the supplementary material.

5 Conclusions

We introduced a novel compilation for verifying the robustness of social laws in multi-agent planning scenarios, which is more general than previous methods, accommodating both classical and numeric planning formalisms. Despite its broader applicability, the proposed method demonstrates state-of-the-art performance.This advancement not only enhances the verification process but also lays the groundwork for integrating social laws into richer planning frameworks, opening new possibilities for research in multi-agent systems.

Acknowledgements

Alexander Shleyfman's work was partially supported by ISF grant 2443/23. Work also supported by a grant from the Trudy Lewis Foundation and the Israeli Planning and Budgeting Committee of the Council for Higher Education.

References

[Ågotnes and Wooldridge, 2010] Thomas Ågotnes and Michael J. Wooldridge. Optimal social laws. In *AAMAS*, pages 667–674. IFAAMAS, 2010.

[Ågotnes *et al.*, 2010] Thomas Ågotnes, Wiebe van der Hoek, and Michael J. Wooldridge. Robust normative systems and a logic of norm compliance. *Log. J. IGPL*, 18(1):4–30, 2010.

[Ågotnes *et al.*, 2012] Thomas Ågotnes, Wiebe van der Hoek, and Michael J. Wooldridge. Conservative social laws. In *ECAI*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 49–54. IOS Press, 2012.

[Helmert, 2006] Malte Helmert. New complexity results for classical planning benchmarks. In *ICAPS*, pages 52–61, 2006.

[Karpas *et al.*, 2017] Erez Karpas, Alexander Shleyfman, and Moshe Tennenholtz. Automated verification of social law robustness in strips. In *ICAPS*, pages 163–171, 2017.

[Nir *et al.*, 2020] Ronen Nir, Alexander Shleyfman, and Erez Karpas. Automated synthesis of social laws in STRIPS. In *AAAI*, pages 9941–9948. AAAI Press, 2020.

[Nir *et al.*, 2023] Ronen Nir, Alexander Shleyfman, and Erez Karpas. Automated verification of social laws in numeric settings. In *AAAI*, pages 12087–12094, 2023.

[Scala *et al.*, 2020] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramírez. Subgoaling techniques for satisficing and optimal numeric planning. *JAIR*, 68:691–752, 2020.

[Shoham and Tennenholtz, 1992] Yoav Shoham and Moshe Tennenholtz. On the synthesis of useful social laws for artificial agent societies (preliminary report). In *AAAI*, pages 276–281. AAAI Press / The MIT Press, 1992.

[Stolba *et al.*, 2015] Michal Stolba, Daniel Fiser, and Antonín Komenda. Admissible landmark heuristic for multi-agent planning. In *ICAPS*, pages 211–219, 2015.

[Taitler *et al.*, 2024] Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fiser, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 international planning competition. *AI Mag.*, 45(2):280–296, 2024.

[Tenneholtz and Moses, 1989] Noshe Tenneholtz and Yoram Moses. On Cooperation in a Multi-Entity Model. In *IJCAI*, pages 918–936, 1989.

[Tuisov and Karpas, 2020] Alexander Tuisov and Erez Karpas. Automated verification of social law robustness for reactive agents. In *ECAI*, volume 325, pages 2386–2393, 2020.

[Tuisov *et al.*, 2024] Alexander Tuisov, Alexander Shleyfman, and Erez Karpas. Good things come to those who wait: The power of sensing in social laws. In *ECAI*, volume 392, pages 4328–4335, 2024.

[upf, 2025] Unified planning: Modeling, manipulating and solving ai planning problems in python. *SoftwareX*, 29:102012, 2025.