Automated Verification of Social Law Robustness in STRIPS

Erez Karpas, Alexander Shleyfman, Moshe Tennenholtz
Faculty of Industrial Engineering and Management
Technion — Israel Institute of Technology

Abstract

Agents operating in a multi-agent environment must consider not just their own actions, but also those of the other agents in the system. Artificial social systems are a well known means for coordinating a set of agents, without requiring centralized planning or online negotiation between agents. Artificial social systems enact a social law which restricts the agents from performing some actions under some circumstances. A good social law prevents the agents from interfering with each other, but does not prevent them from achieving their goals. However, designing good social laws, or even checking whether a proposed social law is good, are hard questions. In this paper, we take a first step towards automating these processes, by formulating criteria for good social laws in a multi-agent planning framework. We then describe an automated technique for verifying if a proposed social law meets these criteria, based on a compilation to classical planning.

Introduction

The design of an agent which is about to operate in a multi-agent environment is quite different from the design of an agent which performs its activities in isolation from other agents. Typically, a plan that would have allowed an agent to obtain its goals had it operated in isolation might yield unexpected results as a consequence of other agents' activities. Various approaches to multi-agent coordination have been considered in the literature. We could, for instance, subordinate the agents to a central controller. This approach may be useful in various domains but might suffer from well-known limitations, such as bottlenecks at the central site or sensitivity to failure. Another approach is to design rules of encounter, that is, rules which determine the behavior of the agent, and in particular the structure of negotiation, when its activities interfere with those of another agent. Rules of encounter may be quite useful for conflict resolution, but might sometimes be inefficient, requiring repeated negotiations to solve on-line conflicts. In this paper we consider a canonical intermediate approach to coordination, referred to as *artificial social systems* (Tennenholtz 1991; Shoham and Tennenholtz 1992a; 1992b; 1995; Moses and Tennenholtz 1995).

An artificial social system institutes a social law that the agents shall obey. Intuitively, a social law restricts, off-line, the actions legally available to the agents, and thus minimizes the chances of an on-line conflict and the need to negotiate. Similarly to a code of laws in a human society (Rousseau 1762), an artificial social law regulates the individual behavior of the agents and benefits the community as a whole. Yet, the agents should still be able to achieve their goals, and restricting their legal actions to a too wide extent might leave them with no possible way to do so.

Consider for instance a domain involving a set of technicians, broken machines, and tools. The technicians can walk between locations, take and put down tools, and fix the machines using those tools. The goal consists of a set of machines that should be fixed. In order to guarantee that every machine gets fixed, we could set a law which forces the technicians to return every tool they use to the toolbox when they are finished with it. Although it may seem like this law is enough, a more subtle issue involves a possible deadlock, where technician 1 is holding tool A and waiting for tool B, while technician 2 is holding tool B and waiting for tool A. In order to avoid such deadlocks, we could modify the law to allow each technician to hold at most one tool at any given moment. This example illustrates the fact that we must be careful in designing social laws: Only useful social laws, i.e laws which guarantee that each agent achieves its goals, are to be considered.

The artificial social systems approach has become a canonical approach to the design of multi-agent systems (Woolridge 2001; Shoham and Leyton-Brown 2008; Horling and Lesser 2004; d'Inverno and Luck 2004; Klusch 1999). However, while the origins of the artificial social systems approach arise from a knowledge representation and planning perspective, and early work by the founders of that approach had advocated the use of planning paradigms, such as STRIPS-like representations for multi-agent planning (Tennenholtz and Moses 1989), the connection between artificial social systems to modern planning techniques has not been crystallized or exploited. The aim of our current line of research is to re-visit the artificial social systems approach in view of progress made in planning. Specifically, the contributions of this paper are threefold: First, we describe a formalism for representing and reasoning over social laws in a multi-agent planning framework. Second, we describe some robustness criteria we believe good social laws should satisfy. Third, we describe an algorithm for verifying if a given social law meets these criteria, which is based on a compilation to classical planning. An empirical evaluation shows this approach scales up very well.

Preliminaries

We consider multi-agent planning settings formulated in (a variation of) MA-STRIPS (Brafman and Domshlak 2008). Our focus in this paper is on problems which do not require cooperation, but do require coordination, and thus we modify MA-STRIPS to include a goal for each agent, rather than an overall goal. A multi-agent planning setting is defined by a tuple $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$, where: F is a set of facts, $I \subseteq F$ is the initial state, $G_i \subseteq F$ is the goal of agent i, and A_i is the set of actions of agent i, where agents are numbered $1 \ldots n$. Each action $a \in A_i$ is described by preconditions $pre(a) \subseteq F$, add effects $add(a) \subseteq F$, and delete effects $del(a) \subseteq F$. The result of applying action a in state s is $(s \setminus del(a)) \cup add(a)$.

The projection of Π for agent i is the (single agent) STRIPS (Fikes and Nilsson 1971) planning problem $\Pi_i = \langle F, A_i, I, G_i \rangle$. A sequence of actions π_i is a solution for Π_i if applying the actions in π_i from state I results in a state s that satisfies the goal, that is, a state such that $G_i \subseteq s$.

In the execution model we consider here, each agent plans offline, arriving at plan π_i. The agent then tries to follow π_i by executing it one action at a time. Thus, at any given moment there are several agents, each ready with its next action (the number could be less than n if some agents have already finished executing their plan). At each step, we assume that the choice of which agent acts next is made by some external scheduler. The chosen agent then executes its pending action, the state of the world changes accordingly, and the chosen agent then sets the next action in the plan as pending. Then, the next step starts, and the scheduler again chooses the agent to act next. Note that we do not make any assumptions about the fairness of the scheduler, and in fact, consider it to be adversarial.

Encoding Social Laws

We have described our multi-agent setting, but we have yet to describe how we represent social laws in this setting. To begin with, we will represent social laws as modifications to a MA-STRIPS problem. That is, a social law l takes an input MA-STRIPS problem Π, modifies it, and outputs a new MA-STRIPS problem Π^l. Such a law is described by:

1. The facts it adds or removes,
2. The actions it adds or removes,
3. The preconditions, add effects, or delete effects it adds or removes from each existing action,
4. The facts it adds or removes from the initial state
5. The facts it adds or removes from each agent's goal, and
6. The action preconditions which are denoted as *waitfor*

The first 5 items above are simply syntactic modifications of an MA-STRIPS setting. For example, the social law which states that technicians must return tools to the toolbox when they are done can be encoded by (a) removing all actions which put down tools in any other locations, and (b) adding the fact that the technician is not holding anything to each technician's goal. Note that while the definition allows a social law to modify all aspects of the MA-STRIPS problem, not all such modifications make sense as social laws (for example, making the goal $\emptyset$). For the purposes of this paper, we assume the social law is designed by a human to be reasonable, given an understanding of the problem at hand.

The *waitfor* precondition annotations, however, require some explanation. Waiting is one of the most basic forms of coordination, and can eliminate some failures. For example, waiting for a tool to be put back in the toolbox eliminates failures that result from not finding the tool. However, waiting also introduces the possibility of a *deadlock*, where two technicians are waiting for each other to finish and return their tools, as described previously.

The semantics of executing an action with a *waitfor* precondition in our model is as follows: When an agent's pending action a has a *waitfor* precondition p, the scheduler will only execute the action (that is, choose this agent to act next) if p holds in the current state s. We will denote the *waitfor* preconditions of action a by $pre_w(a)$ and the other preconditions of a by $pre_f(a)$. If an action a is executed in some state s which does not satisfy $pre_f(a)$ (i.e., $pre_f(a) \nsubseteq s$), we say that action a has *failed*. If at least one agent is waiting for some precondition, and all other agents are either waiting for some precondition or have already achieved their goal, then the system is in a *deadlock*.

We conclude this section with a brief discussion of when it does or does not make sense to wait for some precondition of an action. One of the original motivations for social laws comes from robotics, and in the real world, robots can not sense everything. Thus, it only makes sense to wait for something the agent can sense, as otherwise there is no way to implement the action on a robot. This is a subtle point with the assumptions underlying classical planning: assuming the actions are deterministic, do we sense at every state, or only at the initial state? For classical planning, the answer is irrelevant, but when there are multiple agents operating in the world, the answer is very important. *waitfor* precondition annotations answer this question by stating that we sense *before* we start executing an action.

Secondly, it does not make sense to wait for a precondition which the agent can achieve by itself — a private fact in multi-agent planning terms. This would automatically result in an agent entering a deadlock by itself. For example, consider the action $move(A, X, Y)$ which has a precondition $at(A, X)$, which the agent waits for. Unless some other agent can move A to X, and has a good reason to do so, A will be stuck waiting for itself to move to X.

Properties of Social Laws

Now that we have formalized the setting in which we consider social laws, we describe what are the criteria that define a *good* social law. We consider two different criteria for social laws, which we call *rational* and *adversarial* robustness. In rational robustness, we assume all agents are rational and want to achieve their goal, and ask whether there is any possible way for them to interfere with each other. In adversarial robustness, we assume all agents except for one specific agent (say agent i) are adversarial, and only want to prevent agent i from achieving its goal, without regard for achieving their own goal later. These criteria are formally stated in the following definition:

Definition 1. *A social law l for multi agent setting $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ is robust to:*

rational *iff for all agents $i = 1 \dots n$, for all individual solutions π_i for Π_i, for all possible action sequences π resulting from any arbitrary interleaving of $\{\pi_i\}_{i=1}^n$ which respects* waitfor *preconditions, π achieves $G_1 \cup \ldots \cup G_n$.*

adversarial against i *iff for all individual solutions π_i for Π_i, for all possible action sequences π resulting from an arbitrary interleaving of π_i which respects* waitfor *preconditions with any valid action sequence of all other agents, π achieves G_i.*

adversarial *iff it is robust to* adversarial against i *for all $i = 1 \dots n$.*

We remark that Definition 1 is somewhat restrictive. Specifically, it assumes each agent chooses a plan to execute *a-priori*, and then executes that plan. Without introducing the ability to wait, this is similar to conformant planning — whenever the scheduler tells an agent to act, it must execute its next action. If the preconditions of that action do not hold, the agent fails.

In general, we would like to be able to support more general policies for the agents. This would be similar to contingent planning, except that the non-determinism is really the result of other agents acting in the world. However, this makes the non-deterministic planning problem highly intractable, as any action can have many outcomes (Muise et al. 2015).

Therefore, in this paper we adopt a limited form of contingent planning — waiting until some condition holds. This means that each agent can treat its own individual planning problem as a classical planning problem, and does not need to reason about the possible changes introduced by the other agents. In fact, we argue that the purpose of a good social law is to allow agents to do just that — not have to reason about what the other agents are going to do, assuming they respect the social law. As our focus in this paper is on coordination problems, rather than on problems involving cooperation, we believe this is a good middle-ground.

To show that waiting is a natural way to specify social laws, consider our running example. If a technician executes the pickup tool action without waiting for the tool to be in the toolbox, the action will fail. However, if we denote the precondition of the tool being in the toolbox as *waitfor*, we obtain the desired behavior of waiting for the tool to be there.

One more point to note is that, in some cases, we might want to allow an agent's individual plan to explicitly include waiting for some external event. For example, we might want to wait for another agent to achieve its goal before executing some action. An action in which agent i waits for p can be easily incorporated into our framework, by adding an action to the MA-STRIPS problem Π with a *waitfor* precondition p, and adding an action which *achieves* p to Π_i — the single agent projection of Π. Thus, when agent i attempts to come up with its individual plan π_i it can be optimistic and assume that whatever it waits for will occur. In the next section, we describe how we can automatically verify whether such optimism is warranted.

Verifying Social Laws

Now that we have a formal definition of what it means for a social law l to be robust to *adversarial* or to *rational*, we can easily formulate the corresponding decision problems: VERIFY-ADVERSARIAL and VERIFY-RATIONAL, which take a MA-STRIPS setting Π and a social law l, and return whether Π^l is robust to *adversarial* (respectively, *rational*). We begin by discussing the relation between these two verification problems.

Verifying Adversarial vs. Verifying Rational

It is easy to see that if a social law is robust to *adversarial*, then it is also robust to *rational*. Conversely, we now show that the *verification* problem for adversarial robustness VERIFY-ADVERSARIAL is reducible to the *verification* problem for rational robustness VERIFY-RATIONAL.

Theorem 1.
VERIFY-RATIONAL $\geq_p$ VERIFY-ADVERSARIAL.

Proof. Given a multi-agent setting Π and a social law l, we want to solve VERIFY-ADVERSARIAL(Π^l). From Definition 1, this is equivalent to verifying that Π^l is robust to adversarial against i for all $i = 1 \dots n$. To verify that Π^l is robust to adversarial against a given i, we will construct a 2-agent setting Π' and a social law l, such that Π'^l is robust to rational iff Π^l is robust to adversarial against i.

The facts and the initial state in Π' are the same facts as in Π: F and I, respectively. The first agent in Π' is agent i from Π, and its actions and goal are the same as in Π: A_i and G_i, respectively. The second agent in Π' is a single virtual agent, which controls all agents in Π except for i, that is, its action set is $\bigcup_{j \neq i} A_j$. The goal of this agent is always true ($\top$), that is, it can achieve its goal whenever it wants. The social law l is the same, except for the required modifications introduced by renaming the agents.

To see that Π'^l is robust to rational iff Π^l is robust to adversarial against i, note that for any solution π_i for Π_i, and for any sequence of actions π' of all other agents, the set of interleavings of π_i and π' which respects *waitfor* preconditions is the same in Π'^l and in Π^l. Furthermore, note that given any such interleaving π, it achieves G_i in Π'^l iff it achieves G_i in Π^l. Finally, note that π always achieves the (always true) goal of the second agent in Π'^l, and that the definition for adversarial against i does not require the agents other than i to achieve their goal. □

To complete the picture, we now describe how we can verify that a social law l is rationally robust in a multi-agent setting Π.

$\Pi' = \langle F', A', I', G' \rangle$, where:

- $F' = \{f_1 \dots f_n \mid f \in F\} \cup \{f_g, f_c \mid f \in F\} \cup \{wt_{f,i} \mid f \in F, i = 1 \dots n\} \cup \{fin_i \mid i = 1 \dots n\} \cup \{failure, act\}$
- $A' = \bigcup_{i=1}^{n} A'_i \cup \{\text{CHECK-NO-}f, \text{CHECK-NO-WAITING-}f \mid f \in F\}$, where: $A'_i = \{\text{END}_i^s, \text{END}_i^f, \text{END}_i^w\} \cup \{a_i^s, a_i^f \mid a_i \in A_i\} \cup \{a_i^{w,x} \mid a_i \in A_i, x \in pre_w(a_i)\}$, such that:

  $pre(a_i^s) = act \wedge (\bigwedge_{f \in pre(a_i)} (f_i \wedge f_g))$,
  $add(a_i^s) = \{f_i, f_g \mid f \in add(a_i)\}$,
  $del(a_i^s) = \{f_i, f_g \mid f \in del(a_i)\}$,

  $pre(a_i^f) = act \wedge (\bigwedge_{f \in pre(a_i)} f_i) \wedge (\bigwedge_{f \in pre_w(a)} f_g) \wedge (\bigvee_{f \in pre_f(a_i)} \neg f_g)$,
  $add(a_i^f) = \{failure\} \cup \{f_i \mid f \in add(a_i)\}$,
  $del(a_i^f) = \{f_i \mid f \in del(a_i)\}$,

  $pre(a_i^{w,x}) = act \wedge (\bigwedge_{f \in pre(a_i)} f_i) \wedge \neg x_g$,
  $add(a_i^{w,x}) = \{failure, wt_{x,i}\} \cup \{f_i \mid f \in add(a_i)\}$,
  $del(a_i^{w,x}) = \{f_i \mid f \in del(a_i)\}$,

  $pre(END_i^s) = \neg fin_i \wedge (\bigwedge_{f \in G_i} f_i) \wedge (\bigwedge_{f \in G_i} f_g)$,
  $add(END_i^s) = \{fin_i\}$,
  $del(END_i^s) = \{act\}$,

  $pre(END_i^f) = \neg fin_i \wedge (\bigwedge_{f \in G_i} f_i) \wedge (\bigvee_{f \in G_i} \neg f_g)$,
  $add(END_i^f) = \{fin_i, failure\}$,
  $del(END_i^f) = \{act\}$

  $pre(END_i^w) = \neg fin_i \wedge (\bigwedge_{f \in G_i} f_i) \wedge (\bigvee_{f \in F} wt_{f,i})$,
  $add(END_i^w) = \{fin_i, failure\}$,
  $del(END_i^w) = \{act\}$

  $pre(\text{CHECK-NO-}f) = (\bigwedge_{i=1 \dots n} fin_i) \wedge \neg f_g$,
  $add(\text{CHECK-NO-}f) = f_c$,
  $del(\text{CHECK-NO-}f) = \emptyset$

  $pre(\text{CHECK-NO-WAITING-}f) = (\bigwedge_{i=1 \dots n} fin_i) \wedge (\bigwedge_{i=1 \dots n} \neg wt_{f,i})$,
  $add(\text{CHECK-NO-WAITING-}f) = f_c$,
  $del(\text{CHECK-NO-WAITING-}f) = \emptyset$

- $I' = \{act\} \cup \{f_i \mid f \in I, i = 1 \dots n\} \cup \{f_g \mid f \in I\}$, and
- $G' = \{failure\} \cup \{f_c \mid f \in F\} \cup \{fin_i \mid i \in \{1 \dots n\}\}$

Figure 1: Formal Description of the Compilation. For ease of exposition, we use logic, rather than sets, to express preconditions.

Verifying Rational Robustness

Our algorithm compiles the VERIFY-RATIONAL problem for $\Pi^l = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ into a classical planning problem Π'. This compilation is described formally in full in Figure 1, but we will first provide some explanations of the compilation, and then prove its correctness.

The key idea behind this compilation is to create a planning problem Π' which attempts to find a set of individual plans for each agent, and an interleaving between these plans, which does *not* work — that is, which leads to either an action failing or to a deadlock. Thus, the social law is robust iff if the planning problem we create is *unsolvable*. On the other hand, a solution to this planning problem gives us a counter-example for the robustness of the social law.

The compilation creates $n + 1$ copies of each fact of the planning problem. We will refer to copies $1 \dots n$ as local copies (one for each agent), and the final copy as the global copy, which will be denoted by g. Each action of agent i affects both its local copy i and the global copy g. The difference is that the local copy i keeps track of the current state assuming agent i had acted alone, and the global copy g keeps track of the "real" current state, which accounts for the actions of all agents.

The objective is to find a plan for each agent which works alone (that is, achieves agent i's goal in local copy i), but when all plans are joined together (in an order chosen by the planner) in copy g, there is a failure. This is captured by the goal of this planning task, which is to achieve G_i in copy i, and have either a deadlock or some action fail in the global copy, as indicated by the flag $failure$. We remark that this duplication of facts is similar to the compilations for discovering worst case distinctiveness in goal recognition design (Keren, Gal, and Karpas 2014; 2015; 2016), although the rest of the compilation is very different.

In order to identify failures in the global copy, we create several versions of each action a_i for each of the possible outcomes of a_i: a_i succeeds, a_i fails due to a violated (non-wait) precondition, or a_i leads to a deadlock. Each of these affect copy i as if a_i succeeds, but has different effects on the global copy: the success outcome also succeeds in the global copy, the fail version requires one of the preconditions of a_i to be false in the global copy[1], and the deadlock version requires that one of the facts that a_i waits for is false, and remains false when agent i has a chance to act. This is achieved by raising a flag $wt_{f,i}$ that indicates that agent i is waiting for fact f. Since we assume the scheduler is adversarial to the agents, and thus under the control of the planner, the next opportunity when agent i is sure to be able to act is after all other agents are finished, i.e. have either achieved their goal or are also waiting.

In order to know when agents have finished, we also add an END action for each agent, whose preconditions are the goal facts of the agent. We create the success, fail, and deadlock versions of this action, and thus the only failures we need to consider are action preconditions not holding and deadlocks. However, in order to prevent a situation where an agent achieves its goal early, and then another agent invalidates it later, we do not allow any "regular" actions to occur after one agent executed an END action, which is controlled

[1] note that this requires disjunctive negative preconditions, which can be easily compiled away by adding more actions

by the act flag.

Finally, in order to make sure that deadlocks are true deadlocks (that is, that if agent i is waiting for fact f, then f will be false after all agents have finished), for each fact f we also add two actions which are meant to verify that no agent is waiting for f at the end, and f holds. These actions are called CHECK-NO-f and CHECK-NO-WAITING-f. The first checks that f does not hold, and the second checks that no agents are waiting for f. Both of these achieve a new fact, f_c, which is also included in the goal, and are only applicable after all agents have executed their END actions. Together, these actions verify that at the end, $wt_{f,i} \rightarrow \neg f$, that is, that if agent i is waiting for fact f, then f does not hold at the end. Note that, since we need this to hold for all agents, this is equivalent to $\neg f \vee (\bigwedge_{1=1\ldots n} \neg wt_{f,i})$, and each of these actions is responsible for checking one of the disjuncts.

It is easy to see that the compilation is polynomial, and that the description size of Π' is roughly n times more than that of Π^l. We now proceed to prove that the compilation is correct, through a series of lemmas. We begin by proving a lemma about the structure of any solution of Π'.

Lemma 1. *Any solution π of Π' can be divided into three sub-sequences, $\pi = \pi_a \cdot \pi_{END} \cdot \pi_{CHECK}$, such that π_a contains only "regular" actions (a_i^s, a_i^f or $a_i^{w,f}$ for some $a_i \in A_i$), π_{END} contains only END actions, and π_{CHECK} contains only CHECK-NO-f and CHECK-NO-WAITING-f actions.*

Proof. By construction of Π', as soon as one of the END actions is executed, act is deleted. As act is a precondition of all regular actions, they must all precede the first END action. Similarly, $\bigwedge_{i=1\ldots n} fin_i$ is a precondition of all CHECK actions, and since fin_i can only by achieved by one of the END$_i$ actions, all END actions must precede all CHECK actions. □

Next, we prove that any solution for Π' contains valid individual solutions for each of the agents:

Lemma 2. *Let $\pi = \pi_a \cdot \pi_{END} \cdot \pi_{CHECK}$ be an arbitrary solution of Π'. Define π_i to be the subsequence of π_a consisting only of actions of agent i. Then π_i is a solution for Π_i^l — the projection of Π^l for agent i.*

Proof. Let us look at the projection of Π' on $\{f_i \mid f \in F\} \cup \{fin_i\}$, which we will denote by Π_i'. It is easy to see that Π_i' is simply Π_i^l with an END action that achieves fin_i added to it. As Π_i' is an abstraction of Π', any solution for Π' is also a solution for Π_i'. Since actions that do not belong to agent i do not affect $\{f_i \mid f \in F\}$ or fin_i, if π is a solution of Π', $\pi_i \cdot \langle \text{END}_i \rangle$ is a solution of Π_i', for any of the versions of END$_i$. Because Π_i' and Π_i^l are equivalent except for the addition of END and fin_i, π_i is a solution for Π_i^l. □

We now prove that any solution of Π' respects the *waitfor* preconditions:

Lemma 3. *Let $\pi = \pi_a \cdot \pi_{END} \cdot \pi_{CHECK}$ be an arbitrary solution of Π'. π respects the* waitfor *preconditions of all actions in π_a, that is, whenever one of the success (a_i^s) or fail (a_i^f) variants of action a_i is executed in π, all* waitfor *preconditions of a_i hold.*

Proof. $pre_w(a_i)$ is in the preconditions of both a_i^s and a_i^f, meaning that any action that is executed, is executed only when the agent could execute it (that is, would not have waited). □

We are now ready to prove our main theorem, about the correctness of the compilation:

Theorem 2. *Assume Π_i^l is solvable for all agents i. Then Π' is not solvable iff Π^l is rationally robust.*

Proof. Assume Π' is solvable, let $\pi = \pi_a \cdot \pi_{\text{END}} \cdot \pi_{\text{CHECK}}$ be a solution for Π', and denote by π_i the subsequence of π_a consisting only of actions of agent i. Let us denote the first non-success action in π_i (that is, a_i^f or $a_i^{w,f}$) by ns_i, where $ns_i = \bot$ if all actions in π_i are success actions (that is, a_i^s). From Lemma 2, each π_i is a solution for Π_i.

First, note that there must exist some j such that $ns_j \neq \bot$. Otherwise, none of the actions in the plan achieve $failure$, which is part of the goal. If there exists some j such that $ns_j = a_j^f$ then π_a gives us an interleaving of $\{\pi_i\}_{i=1}^n$ which violates one of the (non-wait) preconditions of a_j^f. From Lemma 3, π_a respects all *waitfor* preconditions. Thus, we have found an interleaving of valid individual plans, which respects *waitfor* preconditions, but leads to an illegal joint plan, and thus Π^l is not rationally robust.

If there does not exist some j such that $ns_j = a_j^f$ then there must exist some j such that $ns_j = a_j^{w,f}$. We can guarantee that f will not hold at the end, since the only way to achieve f_c, which is part of the goal, would be through CHECK-NO-f (as CHECK-NO-WAITING-f is not applicable after $a_j^{w,f}$ is executed). Thus, if the scheduler does not allow agent j to act until all other agents are done, we have an interleaving in which agent j is in a deadlock. Here again, we have found an interleaving of valid individual plans, which respects *waitfor* preconditions, but leads to an illegal joint plan, and thus Π^l is not rationally robust.

We have shown that if Π' has a solution then Π^l is not rationally robust. Now assume Π' does not have a solution, and let π_i be any solution for Π_i, for $i = 1 \ldots n$. Let π be any interleaving of these individual plans which respects *waitfor* preconditions. All preconditions of all actions in π hold when the action is executed, as otherwise Π' would have been solvable (taking π with the appropriate END and CHECK actions added at the end as a solution). Similarly, π achieves $G_1 \cup \ldots \cup G_n$, and none of the agents is stuck waiting for some fact (as again, either of these scenarios would have led to Π' being solvable). Thus, if Π' does not have a solution then Π^l is rationally robust. □

We have shown that verifying the rational robustness of a social law can be done via compilation to classical planning. One important point to note here is that if the social law is not robust, then the solution of the planning problem can provide a counter-example, and thus guide a human designer in fixing the social law. We discuss this point, and its relation to the choice of which planner to use, in the next section.

Empirical Evaluation

We implemented our compilation, based upon the script which was used to convert MA-PDDL to PDDL representing a centralized planning problem from the first Competition of Distributed and Multiagent Planners (Stolba, Komenda, and Kovacs 2015).[2] We remark that while we have described the compilation for a grounded MA-STRIPS setting, most of the compilation can actually be done on the lifted level of MA-PDDL, and our compilation performs partial grounding where it must.

In order to evaluate how our compilation scales for problems with increasing size, we used the benchmark domains from the first Competition of Distributed and Multiagent Planners (Stolba, Komenda, and Kovacs 2015). These benchmarks are for cooperative planning, and thus contain a single goal in each instance. We created an instance with a separate goal for each agent by allocating each fact in the goal to one of the agents, in a round-robin manner (except in cases where the first argument of the goal fact mentions a specific agent, in which case it was allocated to that agent). We excluded problem instances in which one of the agents was not able to achieve its goal alone (we checked this by solving the Π_i planning problem for each agent), which left us with only 3 domains (BLOCKSWORLD, DRIVERLOG, and ZENOTRAVEL). Additionally, we encoded the machine fixing domain introduced in this paper, which we call FIX.

The choice of planner to use here poses an interesting dilemma. If the social law we are trying to verify is robust, then the planning problem is going to be unsolvable. In such a case, planners for proving unsolvability (Bäckström, Jonsson, and Ståhlberg 2013; Hoffmann, Kissmann, and Torralba 2014) are a good choice, as our results will show. However, if we were sure that the social law is robust, we would not be verifying it, and thus using a "regular" planner tends to be faster, as our results will also show. Thus, we evaluate the performance of different planners on both the original formulation of the domains, and on the same domains containing robust social laws (which we formulated using our verification procedure).

Another important consideration is that planners which are geared towards proving unsolvability typically do not return a plan, even if the problem turns out to be solvable. While this allows these planners to make more efficient use of memory, it means that if the social law we are trying to verify is not robust, we do not have a counter-example showing us what needs to be fixed.

While in general it is probably a good idea to combine several planners in a portfolio, possibly also using some prior estimate on the probability of whether the social law is robust or not, in our empirical evaluation, we simply used the FF planner (Hoffmann and Nebel 2001), as well as the top 3 planners from the Unsolvability International Planning Competition (Muise and Lipovetzky 2015): Aidos 1 (Seipp et al. 2016), M&S (Torralba, Hoffmann, and Kissmann 2016), and SymPA (IRR) (Torralba 2016). We remark that initial experiments with other planners from the unsolvability competition showed the same trend as the competition, and the top 3 planners are also the best on our benchmarks. All planners had a time limit of 30 minutes and a memory limit of 8GB on an Intel i7-6700k CPU runnning at 4GHz.

	FIX				FIX with social law			
prob	FF	SymPA (irr)	M&S	Aidos 1	FF	SymPA (irr)	M&S	Aidos 1
probFIX-2-2	0.0	1394.17	-	855.89	0.0	304.78	1.32	-
probFIX-2-3	0.0	-	-	855.93	0.0	304.72	1.32	-
probFIX-2-4	0.0	-	-	-	0.02	43.45	1.71	-
probFIX-3-5	0.0	-	-	-	0.12	313.01	6.96	-
probFIX-3-7	0.0	-	-	-	0.25	353.62	33.3	-
probFIX-4-9	0.0	-	-	-	2.28	0.44	0.4	0.47
probFIX-4-11	0.0	-	-	-	3.21	0.51	0.47	0.53
probFIX-5-11	0.01	-	-	-	34.05	0.95	0.88	0.99
probFIX-5-13	0.02	-	-	-	124.46	-	-	-
probFIX-6-15	0.02	-	-	-	679.14	-	2.65	3.0
probFIX-6-17	0.02	-	-	-	1659.24	-	-	-
SOLVED	**11**	1	0	2	**11**	8	9	4

Table 1: Results of Different Planners on FIX

Tables 1,2,3, and 4 show the results of running these planners on each instance of our compilation. Each table shows the runtime in seconds on each problem, as well as the total number of problems solved for each domain. Note that we consider a planner to solve a problem if it either finds a plan or proves that one does not exist.

The left-hand side of each sub-table shows the results of verifying whether the "empty" social law encoded in the original domain is robust. The results clearly show that FF beats all three planners from the unsolvability track on these problems, which are all, indeed, solvable.

However, we also wanted to evaluate our approach on robust social laws. For each of the four domains above, we attempted to create such a robust social law, and succeeded in doing so on three of them. We used our compilation to guide this process, using the FF planner to obtain counterexamples. The results for verifying these social laws appear on the right-hand side of the sub-tables in these tables. Here, indeed, the planners from the unsolvability track perform much better than FF. Thus, we conclude that a useful tool for verifying social law robustness should combine between these approaches.

We now describe the process of creating these social laws for each domain.

FIX

We have encoded our running example into a MA-PDDL domain called FIX. The FIX domain involves a set of technicians, broken machines, and tools. The technicians can walk between locations, take and put down tools, and fix the machines using those tools. The goal consists of a set of machines that should be fixed. In the multi-agent version of this problem, the agents are the technicians, each of whom is assigned a set of machines to fix. At the beginning all tools are stored in a special location called "toolbox".

[2] The compilation script and our new FIX domain are available at https://github.com/karpase/social_laws

	BLOCKSWORLD			
prob	FF	SymPA (irr)	M&S	Aidos 1
9-0	0.1	-	-	-
9-1	0.1	-	-	-
9-2	0.1	-	-	-
10-0	0.1	-	-	-
10-1	0.14	-	-	-
10-2	0.12	-	-	-
11-0	0.43	-	-	-
11-1	0.33	-	-	-
11-2	0.27	-	-	-
12-0	0.75	-	-	-
12-1	0.32	-	-	-
13-0	0.29	-	-	-
13-1	0.57	-	-	-
14-0	0.93	-	-	-
14-1	0.96	-	-	-
15-0	1.46	-	-	-
15-1	2.31	-	-	-
16-1	1.94	-	-	-
16-2	0.6	-	-	-
17-0	19.72	-	-	1308.03
SOLVED	**20**	0	0	1

Table 2: Results of Different Planners on BLOCKSWORLD

We propose the next set of social laws for this domain:

- "Put it back where you took it" – each technician can put down the tool only at the "toolbox"
- "Leave work at work" – the agent should have no tool on him at the goal state
- "Don't be greedy" – to avoid deadlocks each technician can take one tool at a time

Results for this domain appear in Table 1. The results indicate that the unsatisfiability solvers do a much better job at proving no plan exists than they do in proving that one does exist, while FF performs better on both sides of the table.

BLOCKSWORLD

The multi-agent BLOCKSWORLD domain involves a set of robotic arms, each of which constitutes an agent. Each arm can pick up and stack blocks, as in the single-agent version of BLOCKSWORLD. Each agent is assigned a goal which consists of a set of ON(x, y) propositions.

Unfortunately, we could not find any reasonable social law for this domain. This is because, although each agent can perform all actions necessary to achieve its goal by itself, when agents are operating together they must cooperate in order to achieve all of their goals. For example, if the goal of agent 1 is ON(A, B) and the goal of agent 2 is ON(B, C), then each agent can easily achieve its goal by building a tower of 2 blocks, but achieving both goals requires a tower of height 3. This requires cooperation between the agents, while our focus here is on simple coordination.

Results for this domain appear in Table 2. The results indicate that the unsatisfiability solvers can barely prove solvability here, while FF solves all 20 instances in seconds.

	DRIVERLOG				DRIVERLOG with social law			
prob	FF	SymPA (irr)	M&S	Aidos 1	FF	SymPA (irr)	M&S	Aidos 1
pfile1	0.0	1133.14	-	0.32	0.0	0.29	0.27	0.37
pfile2	0.0	-	-	1071.08	0.08	0.55	0.56	0.66
pfile3	0.0	-	-	931.22	0.67	-	60.41	-
pfile4	0.01	-	-	-	31.23	-	156.61	-
pfile5	0.02	-	-	-	-	0.25	0.24	1.45
pfile6	0.0	-	-	-	-	1.31	1.34	2.26
pfile7	0.02	-	-	-	-	389.12	-	-
pfile8	0.03	-	-	-	-	1.61	1.25	2.89
pfile9	383.84	-	-	-	-	1.91	1.33	3.79
pfile10	0.0	-	-	-	-	0.28	0.18	0.25
pfile11	-	-	-	-	-	0.44	0.31	3.24
pfile12	-	-	-	-	-	13.91	11.05	23.94
pfile13	0.08	-	-	-	-	1.0	0.73	1.93
pfile14	0.31	-	-	-	-	0.79	0.59	13.02
pfile15	-	-	-	-	-	0.86	0.62	17.29
pfile16	1.49	-	-	-	-	-	-	-
pfile17	-	-	-	-	-	-	57.37	-
pfile18	-	-	-	-	-	6.77	6.91	-
pfile19	-	-	-	-	-	-	-	-
pfile20	-	-	-	-	-	-	-	-
SOLVED	**13**	1	0	3	4	14	**16**	12

Table 3: Results of Different Planners on DRIVERLOG

DRIVERLOG

The DRIVERLOG domain involves a set of drivers, trucks, and packages. Drivers can walk between locations, or enter trucks and drive them. Packages can be loaded onto trucks and unloaded. The goal consists of a set of target locations for each truck and packages. In the multi-agent version of this problem, the agents are the drivers, each of whom is assigned a set of packages and a truck, as its individual goal.

The "empty" social law (that is, the original formulation of this domain) is not robust, because any driver can load a package which is part of another driver's goal, and never unload it. Below, we describe 10 different social laws we tried with this domain. While the details of these are not important, this does show that (a) coming up with robust social laws is non-trivial, (b) our compilation can help a user find problems with candidate social laws, and (c) counterexamples are helpful in the process of designing social laws.

First, we tried a social law which states that a driver can only drive trucks and handle packages if they are assigned to its goal. However, because some drivers might not have a truck in their goal, some of the individual projections are unsolvable.

In order to fix this, we tried a social law which forces trucks to be empty of drivers and packages in the end (that is, the social law modifies the goal). However, this social law is still not robust, as driver i can move a package which is in driver j's goal, after driver j is finished.

We then extended the above social law by adding a restriction that a driver is only allowed to handle a package assigned to its own goal. Our compilation then found another type of error, where a driver boards a truck that already has another driver in it.

We then further extended the above social law by making the precondition of board-truck which states that the truck has no driver a *waitfor* precondition. However, the board-truck action fails if the truck becomes empty of its driver at a different location than where there is a driver waiting to board it.

Naturally, we also made the truck being in the same location as the driver waiting to board it a *waitfor* precondition. However, this leads to a deadlock when a driver never returns a truck to where another driver is waiting to board it.

We then tried to fix this issue by forcing drivers to return a truck they boarded to the original location they boarded it. This had the side effect of some of the individual projections not being solvable, because the truck's goal might be different than its original location.

We then relaxed the above social law, by only requiring drivers who do not have a goal on a truck's location to return it to where they originally boarded it. This leads to a deadlock when some driver is waiting for a truck in its original location, but another driver moved the truck to its goal location.

To address this, we modified the social law to force drivers who do not have a goal regarding a truck's location to board the truck only at its goal location. This leads to a new kind of failure, where a driver can not disembark from a truck that another driver has loaded packages on, because our social law requires the truck to be empty of packages for the disembark action to be applied.

We then modified this social law again, to only allow drivers to board trucks that have no packages in them, and changed the truck being empty of packages precondition of disembark to be a *waitfor* precondition. However, this leads to a deadlock where a driver is waiting forever to disembark from a truck.

Finally, we added a restriction that each driver can only board one truck, and arrived at the following social law:

- Trucks must be empty of both packages and drivers at the end
- Drivers can only handle packages assigned to their own goal
- Drivers wait for the truck to be empty and at the right location before boarding it
- Drivers who do not have a goal on a truck's location can only pick it up at its goal location, and must return it there
- Drivers wait for the truck to be empty before boarding and disembarking from it
- Drivers can only board one truck

We were then able to show this social law is robust.

Results for this domain appear in Table 3. The results indicate that the unsatisfiability solvers are much better at proving unsolvability, while FF is much better at finding plans.

ZENOTRAVEL

The ZENOTRAVEL domain involves a set of aircraft and people. Aircraft fly between different cities, using actions fly and zoom, which consume different amounts of fuel, and can also refuel. People can board and disembark aircraft, to arrive at their goal location. The agents in the multi-agent version of the domain are the aircraft.

	ZENOTRAVEL				ZENOTRAVEL with social law			
prob	FF	SymPA (irr)	M&S	Aidos 1	FF	SymPA (irr)	M&S	Aidos 1
pfile3	0.0	2.08	-	3.09	0.0	0.34	0.3	0.33
pfile4	0.0	-	-	3.18	0.0	0.28	0.28	0.3
pfile5	0.0	-	-	1218.04	0.01	0.44	0.44	0.48
pfile6	0.01	-	-	-	0.0	0.46	0.45	0.5
pfile7	0.0	-	-	-	0.0	0.44	0.45	0.51
pfile8	0.0	-	-	-	0.0	0.98	0.89	1.23
pfile9	0.01	-	-	-	0.01	0.93	0.88	1.19
pfile10	0.01	-	-	-	0.0	0.93	0.88	1.17
pfile12	0.01	-	-	-	0.01	1.4	1.33	1.74
pfile13	0.02	-	-	-	0.02	1.52	1.35	1.82
pfile14	0.14	-	-	-	0.14	7.63	6.86	10.15
pfile15	0.28	-	-	-	0.17	11.57	11.11	15.4
pfile16	0.15	-	-	-	-	2.92	2.84	3.33
pfile17	0.76	-	-	-	0.6	24.46	23.23	31.51
pfile18	0.91	-	-	-	0.6	31.93	30.18	43.25
pfile19	4.34	-	-	-	2.66	41.95	40.96	56.14
pfile20	4.02	-	-	-	2.64	52.65	51.78	69.47
pfile21	5.05	-	-	-	-	53.52	52.11	72.64
pfile22	7.5	-	-	-	5.02	75.39	74.11	103.09
pfile23	12.02	-	-	-	-	11.83	12.21	13.06
SOLVED	**20**	1	0	3	17	**20**	**20**	**20**

Table 4: Results of Different Planners on ZENOTRAVEL

The original version of this domain is not robust because two aircraft can attempt to board the same person. We quickly arrived at a robust social law for this domain, which arbitrarily allocates people to aircraft. People can only board the aircraft they are assigned to, meaning that they can not interfere with each other.

Results for this domain appear in Table 4. In the results here, the unsatisfiability solvers prove all 20 problems on the right side of the table unsolvable, while performing very badly on the left. On the other hand, FF finds plans for all 20 instances on the left side, but fails to prove no plans exist for 3 instances on the right side.

Discussion

In this paper, we have connected social laws to model-based planning, and formalized some criteria which we believe "good" social laws should exhibit under this framework. We have also described a compilation to classical planning for verifying whether an artificial social system meets these criteria, and provided an empirical demonstration that this compilation is feasible in practice.

As our empirical results have shown, planners that are geared towards proving unsolvability do perform better at verifying that a given robust social law is indeed robust. However, if the social law is not robust, then "regular" planners often end up being more helpul, as they provide a counter-example for why the social law is not robust, and are often faster in practice. This might indicate the existence of a certain bias, where unsatisfiability solvers assume they need to prove unsolvability, while classical planners to assume a plan exists.

We remark that the principles behind the compilation we present can be extended to more realistic settings. First, agents might not know what the goals of other agents are, and only have some idea of what the possible goals are. A simple compilation which eliminates disjunctive goals can solve this problem. Second, agents might enter the system at different places and at different times. It is very easy to define actions for "adding" agents at legal locations, and our compilation will take care of making sure the social law criteria are not violated by this.

We conclude by noting that, in this paper, we focus on the problem of *verifying* whether a social law (encoded in a multi-agent planning framework) meets some desired criterion. However, our ultimate goal is to automatically synthesize such social laws, rather than just verifying them. Having efficient verification techniques is a first step in this direction. If these verification techniques can yield counterexamples, those could be used to guide a process of automatically refining a social law, until it becomes robust.

Acknowledgements

The work of Alexander Shleyfman was supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

References

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast detection of unsolvable planning instances using local consistency. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013, Leavenworth, Washington, USA, July 11-13, 2013*. AAAI Press.

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

d'Inverno, M., and Luck, M. 2004. *Understanding agent systems*. Springer.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3/4):189–208.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence* 14:253–302.

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. Distance? who cares? tailoring merge-and-shrink heuristics to detect unsolvability. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, 441–446. IOS Press.

Horling, B., and Lesser, V. 2004. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review* 19(04):281–316.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *ICAPS 2014*.

Keren, S.; Gal, A.; and Karpas, E. 2015. Goal recognition design for non optimal agents. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2015)*.

Keren, S.; Gal, A.; and Karpas, E. 2016. Goal recognition design with non oservable actions. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2016)*.

Klusch, M. 1999. *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1st edition.

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

Muise, C., and Lipovetzky, N. 2015. Unplannability IPC track. In *Workshop on the International Planning Competition (WIPC'15)*.

Muise, C.; Felli, P.; Miller, T.; Pearce, A. R.; and Sonenberg, L. 2015. Leveraging fond planning technology to solve multi-agent planning problems. In *Workshop on Distributed and Multi-Agent Planning (DMAP'15)*.

Rousseau, J. 1762. *Du Contrat Social*.

Seipp, J.; Pommerening, F.; Sievers, S.; and Wehrle, M. 2016. Fast downward aidos. In *UIPC 2016 planner abstracts*, 28–38.

Shoham, Y., and Leyton-Brown, K. 2008. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. New York, NY, USA: Cambridge University Press.

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

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

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

Stolba, M.; Komenda, A.; and Kovacs, D. L. 2015. Competition of distributed and multiagent planners (CoDMAP). http://agents.fel.cvut.cz/codmap/.

Tennenholtz, M., and Moses, Y. 1989. On cooperation in a multi-entity model. In *IJCAI 1989*, 918–923. Morgan Kaufmann.

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

Torralba, Á.; Hoffmann, J.; and Kissmann, P. 2016. MS-Unsat and SimulationDominance: Merge-and-shrink and dominance pruning for proving unsolvability. In *UIPC 2016 planner abstracts*, 12–15.

Torralba, Á. 2016. Sympa: Symbolic perimeter abstractions for proving unsolvability. In *UIPC 2016 planner abstracts*, 8–11.

Woolridge, M. 2001. *Introduction to Multiagent Systems*. New York, NY, USA: John Wiley & Sons, Inc.