Automated Verification of Social Law Robustness in STRIPS

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

Abstract

Agents operating in a multi-agent system 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, which is based on a compilation to classical planning.

1 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 his activities in isolation from other agents. Typically, a plan that would have allowed an agent to obtain his goals had he 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 his 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 consisting of roads on which our agents travel. These roads cross one another at junctions where total freedom on the side of the agents makes an accident a likely event. In order to guarantee accident-free traffic we could set a law that allows an agent to enter an intersection only if the crossing road is free. This law certainly prevents accidents, but restricts the agents too much. Although we have guaranteed an accident-free environment, we have also introduced the possibility of a deadlock: when two agents reach the intersection via crossing roads, they might find themselves waiting indefinitely for the crossing road to get free before initiating their move. 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 his goals, are to be considered. In the example above, the law could be modified to give one direction the right-of-way, obliging cars coming from the crossing direction to yield.

The artificial social systems approach has become a canonical approach to the design of multi-agent systems [Wooldridge, 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 presentations 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.

2 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. 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 attempts to follow its own plan π_i. The plans interact with each other through a scheduler, which determines which agent acts next. We do not make any assumptions about the fairness of the scheduler, and in fact, consider it to be adversarial.

3 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 social law can be 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* preconditions

The first 5 items above are simply syntactic modifications of an MA-STRIPS setting. For example, the social law which says that everyone must drive on the right side of the road can be encoded by removing all actions which drive to the left side of the road.

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 an intersection to be clear before entering it eliminates the possibility of collision. However, waiting also introduces the possibility of a *deadlock* — if our social law states that to enter the intersection we wait until there are no cars to the right, then a deadlock occurs when there are four cars on the four sides of the intersection, as illustrated in Figure 2(c).

The semantics of executing an action with a *waitfor* precondition in our model is as follows: When an agent invokes an action a with a *waitfor* precondition p in state s, the scheduler will only execute the action if p holds in s. We will denote the *waitfor* preconditions of action a by $pre_w(a)$ and the other preconditions of a by $pre_f(a)$. Then the scheduler can only choose to execute an action a whose *waitfor* preconditions hold in the current state s, that is $pre_w(a) \subseteq s$. If all agents are currently either waiting for some precondition or finished (that is, 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.

4 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 $i = 1 \ldots n$, 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 $i = 1 \ldots n$.*

It is easy to see that if a social law is robust to *adversarial*, then it is also robust to *rational*. Conversely, we 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 $i = 1 \ldots n$. 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. $\square$

We conclude this section by noting 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.

To show that waiting is a natural way to specify social laws, consider a traffic light. Without waiting, if an agent chooses to execute the action which drives into the intersection, and the light happens to be red, the action will fail because one of its preconditions is violated. However, if we denote the precondition of having a green light as *waitfor*, we obtain the desired behavior of waiting for the light to turn green.

5 Verifying Social Laws

Now that we have formally stated the criteria we want in a social law, and showed that verifying adversarial robustness can be compiled into verifying rational robustness, we turn to describing how we can verify that a social law l applied in a multi-agent setting Π is rationally robust. 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. 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 idea behind this compilation is to create $n+1$ copies of each fact of the planning problem, and thus of the state. We will refer to copies $1 \ldots n$ 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. Thus, each agent i acts alone in copy i, and all agents act together in the global copy g. The goal is to find a plan for each agent which works alone (that is, in copy i), but when all plans are joined together (in an order chosen by the planner) in copy g, there is a failure. The goal of this planning task 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 *et al.*, 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 work in 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

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

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

- $F' = \{f_1 \ldots f_n \mid f \in F\} \cup \{f_g, f_c \mid f \in F\} \cup \{wt_{f,i} \mid f \in F, i = 1 \ldots n\} \cup \{fin_i \mid i = 1 \ldots n\} \cup \{failure, act\}$
- $A' = \bigcup_{i=1}^{n} A'_i \cup \{\text{CHECK-NO-}f, \text{CHECK-NO-WAITING-}f \mid f \in F\}$, 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:

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

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

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

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

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

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

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

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

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

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

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 have finished (either achieved their goal or are also waiting).

It is important to note here that we are attempting to find a sequence in which actions are *executed* which leads to a failure, not a sequence in which actions are started. Thus, if agent i has started action a_i, which is currently waiting for fact f, this could be reflected in the final plan in two different ways. If this is going to result in a deadlock, meaning that the wait precondition f must not hold at the end, then the plan will contain the deadlock version of a_i, at the point when agent i decides to apply a_i. However, if f is going to be achieved, and the scheduler is going to execute a_i when this happens, then the success version of a_i will appear in the plan *later*, when a_i is actually executed, and not when agent i decides to apply a_i.

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 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.

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 subsequences, $\pi = \pi_a \cdot \pi_{END} \cdot \pi_{CHECK}$, 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^l_i — 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^l_i 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^l_i are equivalent except for the addition of END and fin_i, π_i is a solution for Π^l_i. □

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^s_i) or fail (a^f_i) 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^s_i and a^f_i, meaning that any action that is executed, is executed only when the agent would not have waited to execute it. Recall that the meaning of $a^{w,f}_i$ is that agent i is attempting to execute a_i, and will now wait for f forever (that is, until the end of the plan). □

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

Theorem 2. *Assume Π^l_i 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^f_i or $a^{w,f}_i$) by ns_i, where $ns_i = \perp$ if all actions in π_i are success actions (that is, a^s_i). From Lemma 2, each π_i is a solution for Π_i.

First, note that there must exist some j such that $ns_j \neq \perp$. Otherwise, none of the actions in the plan achieve $failure$, which is part of the goal. If there exists some j such that $ns_j = a^f_j$ then π_a gives us an interleaving of $\{\pi_i\}^n_{i=1}$ which violates one of the (non-wait) preconditions of a^f_j. 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 $ns_j = a^f_j$ then there must exist some j such that $ns_j = a^{w,f}_j$. 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 $a^{w,f}_j$ 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.

□

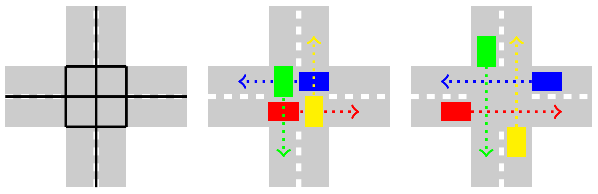
(a) (b) (c)

Figure 2: Intersection Example Illustration

6 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 *et al.*, 2015]. 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.

Our empirical evaluation is divided into two parts: First, to demonstrate the benefits of our approach, we describe a scenario in which a human designer uses it to create a useful social law. Second, we provide some empirical results on existing benchmarks, to demonstrate how our technique scales up with problem size.

6.1 Intersection Example

We demonstrate how one might use our proposed compilation on the classical example for deadlock — that of an intersection with entrances from the north, south, east, and west, where each car wants to go straight. This is illustrated in Figure 2.

This example can be modeled with facts: at(A, L) and clear(L) where A is an agent and L is one of the 12 locations illustrated in Figure 2(a): both lanes on the north, south, east,

BLOCKSWORLD	
Instance	Time (s)
9-0	0.1
9-1	0.09
9-2	0.11
10-0	0.1
10-1	0.08
10-2	0.09
11-0	0.17
11-1	0.18
11-2	0.11
12-0	0.18
12-1	0.26
13-0	0.32
13-1	0.48
14-0	0.44
14-1	0.19
15-0	4.17
15-1	1.83
16-1	1.83
16-2	0.43
17-0	8.23

DRIVERLOG	
Instance	Time (s)
pfile1	0
pfile2	0
pfile3	0
pfile4	0
pfile5	0
pfile6	0
pfile7	0
pfile8	0
pfile9	0
pfile10	0
pfile11	0.01
pfile12	0.01
pfile13	0.02
pfile14	0.04
pfile15	0.17
pfile16	1.25
pfile17	28.63
pfile18	23.53
pfile19	88.06
pfile20	111.54

ZENOTRAVEL	
Instance	Time (s)
pfile3	0
pfile4	0
pfile5	0
pfile6	0
pfile7	0
pfile8	0
pfile9	0
pfile10	0.01
pfile12	0
pfile13	0.04
pfile14	0.1
pfile15	0.22
pfile16	0.12
pfile17	0.61
pfile18	1.04
pfile19	5.26
pfile20	2.77
pfile21	3.85
pfile22	5.46
pfile23	15.06

SATELLITES	
Instance	Time (s)
p05	0.11
p07	0.24
p21	243.38
p24	—
p25	—

SOKOBAN	
Instance	Time (s)
p06-1	—

Table 1: Solution Times on Benchmark Domains (timeouts indicated by a dash)

or west of the intersection, as well as the southeast, southwest, northeast, or northwest corners of the intersection. The actions are:

- arrive(A, l), which models the arrival of agent A at location l, which must be one of the north, south, east, or west entrances to the intersection. This action adds at(A, l), and can only be applied once per agent[2].
- drive(A, l_1, l_2), which models agent A driving from location l_1 to location l_2. This requires at(A, l_1) and clear(l_2).

When we run our compilation on this problem, we of course obtain a failure, as agents can crash into each other in the intersection, which is represented by violating the clear preconditions of drive.

In an effort to correct this, we add to drive a wait annotation which avoids driving into occupied locations. Running our compilation on this yields a deadlock, as illustrated in Figure 2(b).

Attempting to correct this, we add to drive a precondition stating that a car that is about to enter the intersection must yield to a car which is about to enter the intersection from its right. Running our compilation on this yields a deadlock, as illustrated in Figure 2(c).

Finally, we arrive at a deadlock free solution, by dropping the yield preconditions we added for cars entering from the east and the west, while still making cars entering from the north and south yield, and also adding preconditions which ensure that a car does not enter the intersection when it is blocked. Our compilation then verifies that this social law is rationally robust. All planner runs in this example terminate within fractions of a second.

6.2 Benchmarks

In order to evaluate the effectiveness of our compilation on problems with increasing size, we used the benchmark domains from the first Competition of Distributed and Multiagent Planners [Stolba *et al.*, 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 full domains (BLOCKSWORLD, DRIVERLOG, and ZENOTRAVEL), as well as 5 instances from SATELLITE and one instance of SOKOBAN.

The choice of planner to use here poses an interesting question. On the one hand, 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 *et al.*, 2013; Hoffmann *et al.*, 2014] might be a good choice. On the other hand, if we were sure that the social law is robust, we would not be verifying it, and thus using a planner that is geared towards finding solutions might be better.

While in general it is probably a good idea to combine several planners in a portfolio, we have no reason to assume that the multi-agent planning benchmarks already contain a robust social law. In fact, if they did, they would have been fairly easy planning tasks. Therefore, we used the FF planner [Hoffmann and Nebel, 2001] on the compilation for each of these instances, with a timeout of 5 minutes. Table 1 shows how much time it took to solve each of these instances. As these results show, we are able to solve almost all of them very quickly.

7 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.

We remark that the principles behind the compilation we present can be easily 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.

[2]we keep track of this through another fact, which we omit for the sake of brevity

References

[Bäckström *et al.*, 2013] Christer Bäckström, Peter Jonsson, and Simon Ståhlberg. 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, 2013.

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

[d'Inverno and Luck, 2004] Mark d'Inverno and Michael Luck. *Understanding agent systems*. Springer, 2004.

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

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

[Hoffmann *et al.*, 2014] Jörg Hoffmann, Peter Kissmann, and Álvaro Torralba. 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)*, pages 441–446. IOS Press, 2014.

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

[Keren *et al.*, 2014] Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design. In *ICAPS Conference Proceedings*, June 2014.

[Keren *et al.*, 2015] Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design for non optimal agents. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2015)*, January 2015.

[Keren *et al.*, 2016] Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design with non oservable actions. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2016) - to appear*, February 2016.

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

[Moses and Tennenholtz, 1995] Yoram Moses and Moshe Tennenholtz. Artificial social systems. *Computers and Artificial Intelligence*, 14(6), 1995.

[Muise *et al.*, 2015] Christian Muise, Paolo Felli, Tim Miller, Adrian R. Pearce, and Liz Sonenberg. Leveraging fond planning technology to solve multi-agent planning problems. In *Workshop on Distributed and Multi-Agent Planning (DMAP'15)*, 2015.

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

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

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

[Shoham and Tennenholtz, 1992b] Yoav Shoham and Moshe Tennenholtz. On traffic laws for mobile robots (extended abstract). In *AIPS 1992*, pages 309–310. Kaufmann, San Mateo, CA, 1992.

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

[Stolba *et al.*, 2015] Michal Stolba, Antonín Komenda, and Daniel Laszlo Kovacs. Competition of distributed and multiagent planners (CoDMAP). http://agents.fel.cvut.cz/codmap/, 2015.

[Tennenholtz and Moses, 1989] Moshe Tennenholtz and Yoram Moses. On cooperation in a multi-entity model. In *IJCAI 1989*, pages 918–923. Morgan Kaufmann, 1989.

[Tennenholtz, 1991] M. Tennenholtz. *Efficient Representation and Reasoning in Multi-Agent Systems*. PhD thesis, Weizmann Institute, Israel, 1991.

[Woolridge, 2001] Michael Woolridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.