Automated Norm Synthesis in an Agent-based Planning Environment

George Christelis*
School of Informatics
The University of Edinburgh
Edinburgh EH8 9AB, United Kingdom
george.christelis@ed.ac.uk

Michael Rovatsos
School of Informatics
The University of Edinburgh
Edinburgh EH8 9AB, United Kingdom
michael.rovatsos@ed.ac.uk

ABSTRACT

Norms and social laws are one of the key mechanisms used to facilitate coordination in multiagent systems. In existing approaches the process of designing useful norms has to either be performed by a human expert, or requires a full enumeration of the state space which is bound to cause tractability problems in non-trivial domains. In this paper we propose a novel automated synthesis procedure for prohibitive norms in planning-based domains that disallow access to a set of predefined undesirable states. Our method performs local search around declarative specifications of states using AI planning methods. Using this approach, norms can be synthesised in a generalised way over incomplete state specifications to improve the efficiency of the process in many practical cases, while producing concise, generalised, social norms that are applicable to entire sets of system states. We present an algorithm that utilises traditional planning techniques to ensure continued accessibility under the prohibitions introduced by norms. An analysis of the computational properties of our algorithm is presented together with a discussion of possible heuristic improvements.

Categories and Subject Descriptors

1.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Design

Keywords

Norms, social laws, coordination, conflict resolution, automated planning

1. INTRODUCTION

In distributed systems in which agents act independently to achieve their objectives, it is common for the actions of an agent to have a negative impact on the outcome of the actions of another, or on the global performance of the system. The resulting conflicts might undermine the efficiency and stability of the system through resource contention, livelock and deadlock, and could impair agents' abilities to achieve their goals [10]. It is therefore in the best interest of the system designer to avoid or resolve such conflicts. A variety of coordination mechanisms for multiagent system have been proposed as a means of managing the interdependencies between actions of independent agents in a shared system. Via such mechanisms, agents deliberate and reason over their own actions with the additional knowledge of what actions other agents are likely to perform in particular situations.

One established coordination theory inspired by social science and philosophical research is that of social norms [4]. Social norms can be described as a set of established, expected patterns of behaviour that govern the agents in a system [1]. These patterns describe what behaviour is encouraged or discouraged in the system and can be used by agents as templates which regulate their own behaviour. In such systems, agents must all behave in a manner that is consistent with the expectations of the other agents within the system. This behavioural expectation is at the core of coordination via social norms.

Social norms play a key role in the social process of agents in a shared system and are key to achieving social objectives. They reduce the reasoning requirements of agents and reduce the amount of communication required in the system since conflict resolution techniques are no longer required (or are required less often) [2].

Shoham and Tennenholtz [8] show the complexity of norm synthesis to be NP-complete through a reduction from 3-SAT. Subsequent approaches to norm synthesis based on full state-space enumeration have been proposed yet these are often intractable in the general case or not easily applicable in real world systems.

In this paper we propose a novel norm synthesis mechanism that can be applied to existing declarative planning domains without any domain re-specification. We automatically synthesise norms over general state specifications, resulting in concise, generalised applicable norms. Furthermore, we guarantee accessibility to goal states under the new norms.

We begin by introducing the problem of norm synthesis in the context of a propositional planning formalism in the following section. Section 3 details our proposed conflict-rooted synthesis procedure. An example of the synthesis process is presented in Section 4. A discussion of the algorithm follows in Section 5, followed by a summary of related work in Section 6 and conclusions in Section 7.

*The author is a Commonwealth Scholar

Cite as: Automated Norm Synthesis in an Agent-based Planning Environment, George Christelis, Michael Rovatsos, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 161–168

2. NORM SYNTHESIS

To define the problem of norm synthesis in an abstract state transition system, we assume a finite set of discrete, instantaneous environment states. Agent actions transform the state of the environment and the execution of actions is assumed to be asynchronous.

The states are split into two disjoint sets: undesirable conflict states and conflict-free states. We make no further assumptions at this point about preferential ordering in the conflict-free states[1]. Prohibitionary norms in such a system define conditions under which agent actions are forbidden.

The problem of norm synthesis, restricted to prohibitive norms, can then be stated as follows: given a set of conflict states, what actions must be conditionally forbidden so as to prevent access to these conflict states while ensuring continued access to all conflict-free states. One simple solution to this problem is to deny all access to each conflict state by forbidding all actions leading to these states. Such prohibitions would be conditional on the entire state description of the precursor states. There are numerous downsides to this naive approach:

1. a complete state space enumeration is always required,
2. prohibitions are not general, but are conditional on complete states, which may result in a large number of prohibitive rules, and
3. enumerating all concrete conflict states could be avoided by using generalised state abstractions.

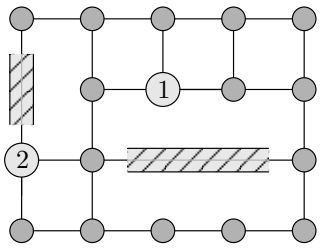
One abstraction over states commonly used in the planning literature is that of state specifications [5]. In this paper we show how a declarative planning domain specification can be utilised to synthesise norms at a state specification level, without necessarily considering a complete state enumeration. The following section introduces a running example, followed by our planning formalism.

2.1 Example Domain

In order to illustrate the process of norm synthesis, we present a simple rail network grid domain. Consider two agent-based trains, with goals to move from one grid location to another without collisions. Grid intersections represent junctions where an agent can choose to alter their direction, and to travel to any adjacent junction point. A collision in this domain occurs when an agent moves into a location occupied by another agent. Through the application of a single operator an agent will traverse to the next grid location. Furthermore, we assume the existence of train tunnels: specific grid locations that the agent is required to exit immediately after entering. Finally, for simplicity we assume the system to be turn-based.

Through a single declarative conflict specification we illustrate how we create a set of norms that remove collisions from the system. The norms synthesised differ depending on the location of an agent, and on the sensing of other agents within the system. For instance, an agent will behave differently if another agent is encountered in a grid location, or in a tunnel. The domain is illustrated below.

[1]We make no differentiation between focal states (states that are essential to agents achieving their goals) and conflict-free states. All conflict-free states are assumed to be focal.



Agents in our example system have actions to `move` from one location to another, `enter` a tunnel, `exit` a tunnel and to remain `idle`. We introduce the following predicate knowledge in the domain: `conn(n1,n2)` symbolises that node `n1` is connected to `n2`, `at(n1)` symbolises that an agent is at location `n1` and `tunnel(n1)` to note that node `n1` is a tunnel. Agents can `move` between two locations if neither location is a tunnel and if the two locations are connected. An agent can `enter` a tunnel if they are at an adjacent location and must subsequently `exit` a tunnel once inside.

2.2 Propositional Planning Formalism

The domain and problem formalisms that follow are based upon the General Propositional Planning Formalism, an extension of traditional STRIPS with incomplete state specifications as described in [5]. Following their notation, we define Σ to be a finite set of propositional atoms used to uniquely represent each state within a system. The set $\hat{\Sigma}$ denotes the literals over Σ, the union of all atoms in Σ, their negations, and the constants $\top$, $\bot$ denoting truth and falsity. Propositional semantic entailment $\models$ is defined in the standard way.

We define $\neg L$ for some literal subset $L \subseteq \hat{\Sigma}$ as the element-wise negation of L, such that $L = \{p|\neg p \in L\} \cup \{\neg p|p \in L\}$. We write $L_1 \backslash L_2$ to denote set difference.

A state s is defined as a complete truth assignment for all atoms in Σ, which we represent by a subset of Σ to capture all those atoms in Σ that are considered to be true in s. However for the purposes of our formalism we consider state specifications used to describe a set of system states. A state specification S is therefore a truth assignment over a subset of $\hat{\Sigma}$. A state specification is:

- **consistent** if $\bot \notin S$, $\nexists l \in \Sigma$ such that $l \in S$ and $\neg l \in S$ (there are no complementary literals), and
- **complete** if for every $l \in \Sigma$, either $l \in S$ or $\neg l \in S$.

We write $s \models S$ to denote that state s is a model for state specification S, which semantically implies that the truth assignment to atoms in s satisfies the literals in S. For example, given the set of atoms $\{a, b, c\}$, a valid state is $s = \{b, c\}$ implying that b and c are true, and a is false in this assignment. If $S = \{\neg a, b\}$, then $s \models S$ (S models all states where a is false, and b true). The set of states that are modelled by a state specification S is written as $Mod(S)$ (if S is complete we have $||Mod(S)|| = 1$).

Operators are action schemata represented as named pairs of the form $o = \langle pre, post \rangle$. In this form $pre(o)$ is used to refer to the first element of the tuple and $post(o)$ to the second. In a propositional planning formalism, each operator consists of consistent precondition $pre \in 2^{\hat{\Sigma}}$ and postcondition $post \in 2^{\hat{\Sigma}}$. Instances of these action schemata are used as steps of a plan.

Operators intuitively represent transitions between states in the system. An operator o can be performed in state s if $s \models pre(o)$. The postconditions describe what changes will occur to s once o is performed. The application of an operator $o \in O$ to a state specification S is expressed as a function $R : 2^{\hat{\Sigma}} \times O \to 2^{\hat{\Sigma}}$ such that:

$$R(S,o) = \begin{cases} (S \backslash \neg post(o)) \cup post(o) & \text{if } S \not\models \bot \wedge S \models pre(o) \\ & \wedge\, post(o) \not\models \bot \\ \bot & \text{otherwise} \end{cases}$$

The planning problem can then be represented as a tuple

$$\Pi = \langle \Xi, I, G \rangle$$

where

- $\Xi = \langle \Sigma, O \rangle$ is the declarative domain structure consisting of a finite set of propositional atoms Σ and a set of operators O,
- $I \subseteq \hat{\Sigma}$ is the initial state specification,
- $G \subseteq \hat{\Sigma}$ is the goal state specification,

The result of applying a sequence of operators O^* to a state specification S is a recursive function $Res : 2^{\hat{\Sigma}} \times O^* \to 2^{\hat{\Sigma}}$ defined as follows:

$$\begin{aligned} Res(S, \langle\rangle) &= S, \\ Res(S, \langle o_1, o_2 ... o_n \rangle) &= Res(R(S, o_1), \langle o_2, ..., o_n \rangle) \end{aligned}$$

A *plan* $\Delta = \langle o_1, o_2, \ldots a_n \rangle$ is a *solution* for planning problem Π if and only if $Res(I, \Delta) \models G$.

In our example, if an agent at location `n3` wishes to traverse to location `n5`, then $I = \{\texttt{at(n3)}\}$, $G = \{\texttt{at(n5)}\}$ and a resulting plan might be $\Delta = \langle \texttt{move(n3,n4)}, \texttt{move(n4,n5)} \rangle$ if we assume the corresponding connectivity between nodes.

2.3 A Normative Planning Formalism

The general planning formalism presented above provides a declarative specification of a domain and the operators agents utilise within it. However it contains no explicit notion of social norms. In order to formalise the effect of a norm, a concrete norm representation and a corresponding normative planning formalism are required.

2.3.1 Prohibitionary Norm Representation

We consider a prohibitionary social norm model, where norms act as behavioural constraints over operators of the planning formalism. A set $N = \{n_1, n_2 ...\}$ of norms contains pairs of the format $n_i = \langle pre, op \rangle$ where $pre \in 2^{\hat{\Sigma}}$ and $op \in O$, denoting that if a state specification S satisfies the precondition pre, then the operator op is forbidden. As with operator preconditions we restrict pre to a set of literals; $pre(n_i)$ is used to refer to the first element in the tuple and $op(n_i)$ to the second.

It should be noted that the set of operators O is assumed to contain operator specifications for all agents.

2.3.2 A Normative Planning Extension

We now extend the general planning formalism presented above to incorporate an explicit representation of social norms. A normative planning problem (NPP) is an extension to the general planning instance represented as a tuple

$$\Pi_N = \langle \Xi, I, G, N \rangle$$

where Ξ is the domain structure, I is the initial state specification, G is the goal specification and N is a prohibitionary set of social norms. Any application of operators in this formalism is conditional on the set of prohibitions. We can define a prohibition function over a state specification as $F : 2^{\hat{\Sigma}} \times O \times 2^N \to \{\top, \bot\}$ such that:

$$F(S,o,N) = \begin{cases} \top & \text{if } \exists n \in N : S \models pre(n) \wedge op(n) = o \\ \bot & \text{otherwise.} \end{cases}$$

An operator o is forbidden for a state specification S under norms N if there exists a norm prohibiting o with precondition satisfied by S. This prohibition function can now be used to extend the state transition function as follows:

$$R(S,o,N) = \begin{cases} R(S,o) & \text{if } F(S,o,N) = \bot \\ \bot & \text{otherwise} \end{cases}$$

A solution to the NPP $\Delta_N = \langle o_1, o_2, \ldots \rangle$ is a set of operators that, if applied to the initial state specification I, will result in a state that satisfies the goal state specification G without violating any of the social norms in N.

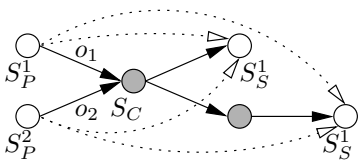
A static implementation of the prohibitionary norm set can be accomplished through an operator specification rewrite procedure. For each operator $o \in O$, the set of applicable norms are those $N_o \subseteq N$ where $\forall n \in N_o, op(n) = o$. It is then possible to rewrite the precondition for o as $pre'(o) = pre(o) \wedge \left(\bigwedge_{n \in N_o} \neg pre(n) \right)$.

It is important to note that the rewriting of operators is not conditional on the initial or final state specifications of the planning instance. This complies with the notion of social norms governing the behaviour of agents independently of what the agents are attempting to achieve at any point: norms are persistent and applicable for the long term.

3. CONFLICT-ROOTED SYNTHESIS

In the context of a propositional planning domain specification and a provided specification of undesirable conflict states, norm synthesis can be defined as a procedure that creates a set of norms for the provided domain. These norms must prohibit access to all conflict states but ensure access to conflict-free states. Our novel approach to this problem is called *conflict-rooted synthesis*, and is based on localised state specification traversal around conflict specifications.

Intuitively, the procedure operates as outlined in the example below. In the figure, shaded states represent conflict states, and white states are conflict-free.



Given some conflict specification S_C and domain Ξ, it is possible to infer the set of all precursor specifications (states labelled S_P) that, through the application of some operator (o_1, o_2), lead to S_C. Similarly, it is possible to find all successor states from S_C that, through the application of a run lead to a non-conflict successor state (the white states labelled S_S). Once this has been established a synthesis procedure will guarantee accessibility by showing that it is

possible to traverse from each conflict-free precursor state specification to each conflict-free successor, without traversing to any intermediate conflict specifications.

At this point it is useful to note that state specification size is inversely related to the expressiveness of the specification. Using our example domain, the specification $\{at_1(n1)\}$ models all states where agent 1 is at n1, whereas $\{at_1(n1), at_2(n1)\}$ models a subset of those states where both agents are at n1.

3.1 Definitions

Consider a transition system with states $E \subseteq 2^{\Sigma}$ and binary relation $\theta : 2^{\Sigma} \times O \rightarrow 2^{\Sigma}$. We assume the specification relation $R : 2^{\hat{\Sigma}} \times O \rightarrow 2^{\hat{\Sigma}}$ as defined previously. We write $s \xrightarrow{o} s'$ to denote that a transition between s and s' for operator o exists. If we consider a state specification S and an operator o, then we define $S \models pre(o)$ to denote $pre(o) \subseteq S$.

Definition 1. We define the following relationships between state, state specifications and operators:

1. For a state $s \in E$ and operator $o \in O$, o *contributes* to s if $\exists s' \in E$ such that $s' \xrightarrow{o} s \in \theta$.
2. For state specification $S \in 2^{\hat{\Sigma}}$ and operator $o \in O$, o *contributes* to S if $\forall s \models S$, o contributes to s.
3. An operator $o \in O$ is *applicable* from a state $s \in E$ if $\exists s' \in E$ such that $s \xrightarrow{o} s' \in \theta$.
4. For any state specification $S \in 2^{\hat{\Sigma}}$, an operator o is *partially applicable* from S if it is neither applicable nor forbidden, such that $\exists E' \subset Mod(S)$ where o is applicable from all states in E'.

Additionally, given two state specifications S_1 and S_2, we write $S_1 \models S_2$ if $S_2 \subseteq S_1$. That is, every state that satisfies the specification S_2 will also satisfy S_1. We define a *run* as a sequence of state specifications/operators of the form $R = S_0 \xrightarrow{o_1} S_1 \xrightarrow{o_2} \ldots \xrightarrow{o_n} S_n$where $S_i \xrightarrow{o_{i+1}} S_{i+1}$ represents the transition from S_i to S_{i+1} via operator o_{i+1}. We use the notation $first(R)$ and $last(R)$ to refer to the first and last state specifications in R respectively.

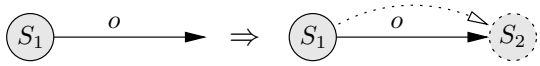
Plan projection follows directly from the application of applicable operators to state specifications. If o is applicable for some state specification S and $S' = R(S, o)$ then $\forall s \models S$, there exists an s' such that $s \xrightarrow{o} s' \in \theta$ and $s' \models S'$.

The conflict-rooted synthesis procedure presented below differs from traditional plan projection, since it considers not only applicable operators, but also partially applicable ones. If we prohibit access to conflict states, to ensure accessibility between all conflict-free states we must ensure accessibility between all conflict-free precursor specifications, and *every* resulting conflict-free successor state. Therefore we are not only interested in the conflict-free specifications accessible commonly from every state modelled by S_C, but also in the conflict-free specifications accessible from *any* state modelled by S_C.

3.2 Inference and Refinement

Two central notions of the algorithm are state inference and state refinement. Two procedures are detailed below that form the building blocks of the synthesis mechanism, with key functions defined as appropriate.

Inference First, given a state specification S_1 and operator o, we present a forward inference mechanism to infer the state specification S_2, as depicted below.



Operator o is deemed *partially applicable* if the operator's precondition does not conflict with S_1:

$$\forall l \in S_1 : \nexists l' \in pre(o).(l = \neg l' \vee l' = \neg l)$$

We define the function $Applicable : 2^{\hat{\Sigma}} \rightarrow 2^{O}$ to return all applicable operators for the provided specification.

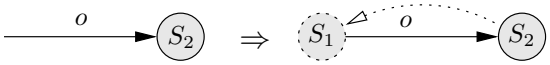
This differs from classical planning approaches where all literals in the precondition of o must be satisfied by S_1. Under the assumption that o can be performed from S_1, a refinement of the state specification of S_1 can be inferred via the function:

$$\overrightarrow{Ref}(S, o) = S \cup pre(o)$$

and $S_1' = \overrightarrow{Ref}(S_1, o)$. We infer S_2, the resulting state specification attained by performing operator o in S_1', via the function:

$$\overrightarrow{Inf}(o, S) = (S \backslash \neg post(o)) \cup post(o)$$

and $S_2 = \overrightarrow{Inf}(o, S_1')$. Similarly, we can reverse infer a precursor state S_1 given an operator o and successor state S_2, as depicted below.



An operator o *contributes* to S_2 if:

$$\exists l \in post(o).(l \in S_2 \wedge l \notin pre(o))$$

and

$$\nexists l \in ((pre(o) \backslash \neg post(o)) \cup post(o)).\neg l \in S_2.$$

That is, o is contributing if the application of o results in at least one literal that occurs in S_2, so long as none of the effects of o contradict S_2, and none of the preconditions of o that are not removed by $\neg post(o)$ are inconsistent with S_2. We define the function $Contributes : 2^{\hat{\Sigma}} \rightarrow 2^{O}$ to return all contributing operators for a provided specification.

We refine S_2 via the function:

$$\overleftarrow{Ref}(S, o) = S \cup (pre(o) \backslash \neg post(o)) \cup post(o)$$

and $S_2' = \overleftarrow{Ref}(S_2, o)$. The inference of the precursor state S_1 follows via the function:

$$\overleftarrow{Inf}(S, o) = (S \backslash post(o)) \cup pre(o)$$

where $S_1 = \overleftarrow{Inf}(S_2', o)$.

State refinement This occurs when additional literals are introduced into some state specification (i.e. the state is refined). If such a state specification forms part of a run, then the introduction of an additional literal has a potential impact on the specifications of all other states in the sequence. We now define how refinement can be consistently applied to runs of state specifications.

Consider a run $R = S_0 \xrightarrow{o_1} S_1 \ldots \xrightarrow{o_k} S_k$. Suppose state specification S_k is being refined, such that $S_k' = S_k \cup L$

where L is a set of new literals not already present in S_k. L is consistent with the existing state S_k if $\forall l \in L : \neg l \notin S_k$. L is consistent with R if it is consistent with the state specifications $S_0 \dots S_k$ and

$$\forall l \in L, \forall i \leq k : \neg l \notin post(o_i) \quad .$$

Consequently a new literal l is consistent with the prefix run if every preceding operator does not remove l. The case where l is removed and added via a subsequent operator is not considered since l would already be part of the state specification for S_k. The refined reverse run is then defined via the function

$$RunRefine(R, L) = S'_0 \stackrel{o_1}{\rightarrow} \dots \stackrel{o_k}{\rightarrow} S'_k$$

where $S_i \in R$ and $S'_i = S_i \cup L$.

Consider a run in our example domain of the form $S_1 \stackrel{o_1}{\rightarrow} S_2 \stackrel{o_2}{\rightarrow} S_3$ where $o_1 =$ `move(n1,n2)` and o_2 is unspecified. We know $o_2 \neq$ `exit(n2,n3)` since this introduces an inconsistency (the agent cannot exit a tunnel that has not been entered). However, $o_2 =$ `move(n2,n3)` is consistent with the current run and in turn introduces additional knowledge (the literal `conn(n2,n3)` for instance). A refinement of the run implies that this additional knowledge must now form part of every preceding specification in the run, since no preceding operator has added it as a postcondition.

3.3 Synthesis Procedure

A conflict state specification S_C models a set of system states $s_1, \dots, s_n$. Any synthesised norm must prohibit an operator that, as an effect, results in a state modelled by S_C. We begin by considering the set of precursor operators *contributing* to S_C, $O_P \subseteq O$. Via the inference and refinement process defined above, we can create an initial run for each inferred precursor state, contributing operator and refined conflict state.

For each run we now consider the set of consistent successor operators, the operators that are *applicable* in S_C. Each operator might potentially lead to a unique conflict-free successor state. We infer the new successor state for each operator, append the new operator/state transition to the run and subsequently refine all specifications in the run. The process repeats for each run instance until the inferred successor state is conflict-free or a loop is detected.

A *complete* run is of the form $S_P \stackrel{o_1}{\rightarrow} S_C \stackrel{o_2}{\rightarrow} \dots \stackrel{o_k}{\rightarrow} S_S$ where the inferred precursor and successor states (S_P and S_S) are conflict-free, and all intermediate states are modelled by S_C. An *incomplete* run does not terminate in a conflict-free state.

Loop detection ensures that the algorithm terminates. Since we consider all operators from a state specification there is no benefit to examining such a specification again. Every specification is analysed once and since we assume a finite set of state literals the algorithm terminates.

Algorithm 1 details the state specification based traversal of runs, leading from conflict-free precursor state specifications to conflict-free successors, through conflict states specified in S_C.

3.4 Accessibility

Our algorithm guarantees accessibility to all conflict-free states without entering the specified conflict states. The set of all complete runs provided by Algorithm 1 is subsequently used to check accessibility. For each run, we attempt to find an alternate path from the conflict-free first and last state specifications of the run.

Algorithm 1: Conflict-Free Run Traversal

Input: A conflict specification S_C, and list of operators O
Result: The set complete, the runs from conflict-free precursor to conflict-free successor states

begin
  $O_P \leftarrow$ *Contribute* (S_C)
  for *each precursor operator* $o_i \in O_P$ **do**
    $S^i_C \leftarrow \overleftarrow{Ref}(S_C, o_i)$
    $S^i_P \leftarrow \overleftarrow{Inf}(o_i, S^i_C)$
    (Initialise run for this specification, operator pair)
    $R_i \leftarrow S^i_P \stackrel{o_i}{\rightarrow} S^i_C$
    complete $\longleftarrow \{\}$
    unsafe $\longleftarrow \{R_i\}$
    while |unsafe | > 0 **do**
      $R_i \leftarrow$ *first*(unsafe)
      $O_S \leftarrow$ *Applicable*(*last*(R_i))
      for *each successor operator* $o_j \in O_P$ **do**
        $R_{i,j} \leftarrow R_i$
        last($R_{i,j}$) $\leftarrow \overrightarrow{Ref}$(*last*($R_{i,j}$), o_j)
        $S^{i,j}_S \leftarrow \overrightarrow{Inf}(o_j,$ *last*($R_{i,j}$))
        if *Consistent*($S^{i,j}_S, R_{i,j}$) **then**
          (Loop detection)
          if $S^{i,j}_S \notin R_{i,j}$ **then**
            $R_{i,j} \leftarrow$ *RunRefine*($R_{i,j}$, $S^{i,j}_S$)
            $R_{i,j} \leftarrow R_{i,j} \stackrel{o_j}{\rightarrow} S^{i,j}_S$
            if $S^{i,j}_S$ *is not a conflict state* **then**
              complete $\leftarrow$ complete $\cup R_{i,j}$
            else
              unsafe $\leftarrow$ unsafe $\cup R_{i,j}$
  return complete
end

The necessity of full accessibility checking may be domain dependent[2]. In general, we propose that accessibility is guaranteed for a run R if a solution to the planning instance problem Π_N exists for $I = first(R)$ and $G = last(R)$, under the set of synthesised prohibitions. Algorithm 2 details the norm synthesis and accessibility verification process.

With each successor operator appended to a run, the specifications of the run are further refined since the final specification of the run must satisfy the precondition of the operator. It is probable that for any run the state specifications within the run will not be complete. Such a specification run therefore applies to multiple state/operator transitions in the system, and as the states are refined more, so is the set of states modelled reduced. If an accessibility plan exists for such a specification run, then it exists for each of the states modelled by such a run.

Finally, should accessibility not be achievable from some specification S, then we iterate the process again and include S as a conflict state. Consider an agent in a state modelled by S. The agent can perform only a single operator o in S, and the application of that operator leads to a conflict

[2]For example, certain domains may define accessibility to be resource constrained, while in others it could be sufficient to guarantee that any alternative action is possible, as opposed to a complete plan.

Algorithm 2: Norm Synthesis and Accessibility Check

Input: The set of complete produced by Algorithm 2
Result: The set of synthesised norms N, or `false`
begin
  (Create the set of prohibitionary norms)
  N ← {}
  for *each run* $R \in$ complete **do**
    pre ← *first*(R)
    op ← *first_op*(R)
    N ← N ∪⟨pre, op ⟩
  (Ensure accessibility for all runs)
  for *each run* $R \in$complete **do**
    $I \leftarrow$ *first*(R)
    $G \leftarrow$ *last*(R)
    if *no solution exists for* $\Pi_N = \langle \Xi, I, G, N \rangle$ **then**
      return `false`
  return N
end

state. By prohibiting o, accessibility is not achievable since the agent has no alternative action. Here we consider S as a conflict state as well, since if the agent enters this state, then subsequent conflict is unavoidable.

Many propositional planning formalisms allow for variables in the operator descriptions. We accommodate this throughout the conflict-rooted search process through typical logic programming variable unifications. The process of unification is further highlighted in the following example.

4. EXAMPLE SYNTHESIS

The planning specification of the operators `move`, `enter`, `exit` and `idle` are presented below. We use atoms with capital letters to denote variables in the operator specifications, and the subscript index $i = \{1, 2\}$ to identify each agent. From the operator preconditions it should be clear that once an agent has entered a tunnel, the only option available to them in their subsequent turn is to exit the tunnel they entered.

move$_i$(L1, L2)
  pre : {at$_i$(L1),conn(L1,L2),¬tunnel(L1),¬tunnel(L2)}
  post : {¬at$_i$(L1),at$_i$(L2)}
enter$_i$(L1, T1)
  pre : {at$_i$(L1),conn(L1,T1),¬tunnel(L1),tunnel(T1)}
  post : {¬at$_i$(L1),at$_i$(T1)}
exit$_i$(T1, L1)
  pre : {at$_i$(T1),tunnel(T1),conn(T1,L1),¬tunnel(L1)}
  post : {¬at$_i$(T1),at$_i$(L1)}
idle$_i$(L)
  pre : {¬tunnel(L),at$_i$(L)}
  post : {}

Agent indices and variables are used purely for notation. We emphasise that such a representation can easily be mapped to a propositional formalism by assigning a single propositional atom to each possible action occurrence.

We assume variables with the same name to be unified, and to avoid confusion, we use `?` to symbolise an ununified variable. The set of contributing precursor operators (O_P) for both agents ($i \in 1, 2$) are therefore `move`$_i$`(?,N)`, `enter`$_i$`(?,N)` and `exit`$_i$`(?,N)`.

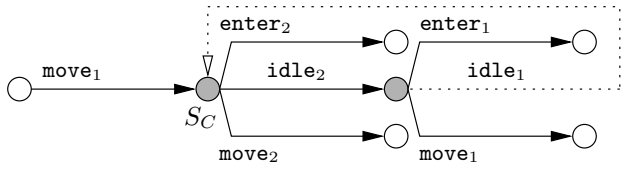
First, we consider the action o_i =(`move`$_1$`(?,N)`) where agent 1 moves into a conflict state. The refined conflict state and precursor specifications resulting in the initial incomplete run ($S^i_P \xrightarrow{o_i} S^i_C$) are:

$$S^i_C = \{\texttt{at}_1\texttt{(N)}, \texttt{at}_2\texttt{(N)}, \neg\texttt{at}_1\texttt{(L)}, \texttt{conn(L, N)}, \neg\texttt{tunnel(L)}, \neg\texttt{tunnel(N)}\}$$
$$S^i_P = \{\texttt{at}_1\texttt{(L)}, \texttt{at}_2\texttt{(N)}, \texttt{conn(L, N)}, \neg\texttt{tunnel(L)}, \neg\texttt{tunnel(N)}\}$$

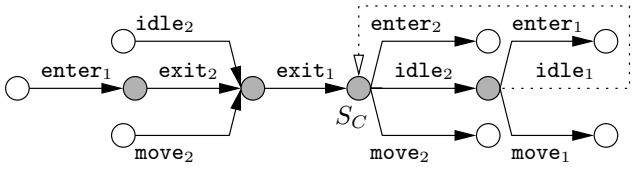
Due to agents taking alternating turns in the system, the subsequent set of applicable actions for agent 2 is the following for this partial run:

$$O_S = \{\texttt{idle}_2\texttt{(N)}, \texttt{move}_2\texttt{(N,?)}, \texttt{enter}_2\texttt{(N,?)}\}$$

The successor actions `move`$_2$ and `enter`$_2$ both lead to conflict-free successor states and these runs are complete. The idle operator leads to a state specification that still satisfies the collision state specification. Hence the resulting incomplete run is iterated again. The successor operators O_S for the second iterations show identical actions for agent 1. We do not allow the invocation of the `idle` action again since this qualifies as a loop. The resulting complete runs are depicted below (the state specifications are omitted). Specifications matching the conflict state are shaded nodes.



The synthesised norm for the `move` operator therefore states that if an agent is in a location that is adjacent to some location occupied by another agent, then the agent is forbidden to move to the occupied location. The norm is detailed as N_1 in Table 4. A key benefit of this synthesis procedure is that from a single state specification we are able to synthesise a set of norms, each of which applies to specific actions that lead to conflict. Continuing the application of Algorithm 1 for the precursor operator `exit`$_1$`(?,N)` the following transition graph is inferred.



The synthesised norms for the `exit` precursor action are itemised below, with a single norm per unique run, and are presented in detail in Table 4:

- N_2: An agent should not choose to `enter` a tunnel, if the exit of the tunnel and the exit of another tunnel in which the other agent is in are common.
- N_3: An agent should not choose to `idle` if the other agent is in a tunnel, and the tunnel is connected to the agents location.
- N_4: An agent should not choose to `move` to a location, if that location is the exit of a tunnel in which the other agent is in.

Table 1: The Synthesised Norms For The Conflict Specification $\{\mathbf{at_1(N)}, \mathbf{at_2(N)}\}$

$N_1 = \langle \{\text{at}_1(\text{L}), \text{conn}(\text{L,N}), \neg\text{tunnel}(\text{L}), \neg\text{tunnel}(\text{N}), \text{at}_2(\text{N})\}, \mathbf{move_1(L, N)} \rangle$
$N_2 = \langle \{\text{at}_1(\text{L}), \text{at}_2(\text{T2}), \text{tunnel}(\text{T2}), \text{tunnel}(\text{T1}), \neg\text{tunnel}(\text{N}), \neg\text{tunnel}(\text{L}), \text{conn}(\text{T2,N}), \text{conn}(\text{T1,N}), \text{conn}(\text{L,T1})\}, \mathbf{enter_1(L,T1)} \rangle$
$N_3 = \langle \{\text{at}_1(\text{L}), \text{at}_2(\text{N}), \text{conn}(\text{L,N}), \neg\text{tunnel}(\text{N}), \text{tunnel}(\text{L})\}, \mathbf{idle_2(N)} \rangle$
$N_4 = \langle \{\text{at}_1(\text{T}), \text{at}_2(\text{L}), \text{conn}(\text{T,N}), \text{conn}(\text{L,N}), \text{tunnel}(\text{T}), \neg\text{tunnel}(\text{N}), \neg\text{tunnel}(\text{L})\}, \mathbf{move_2(L, N)} \rangle$
$N_5 = \langle \{\text{at}_1(\text{L}), \text{at}_2(\text{N}), \text{conn}(\text{L,N}), \neg\text{tunnel}(\text{L}), \text{tunnel}(\text{N})\}, \mathbf{enter_2(L, N)} \rangle$

The third, and simplest complete run for precursor operator $\texttt{enter}_1$ is $S_P \overset{\texttt{enter}_1}{\rightarrow} S_C \overset{\texttt{exit}_2}{\rightarrow} S_S$, and is concerned with collisions that occur within a tunnel. The final norm synthesised for this run, detailed by N_5 in Table 4, details that an agent should not choose to `enter` a tunnel if the other agent is already in that tunnel.

It is clear that the synthesised norms are applicable to multiple states in the system. The synthesised norms are not only general across system states but also across instances of the domain itself.

5. DISCUSSION

The novel norm synthesis procedure presented in this paper produces prohibitionary norms over state specifications, performing a traversal of the specification space only as required. While such synthesis is typically performed offline, it is possible to apply our procedure online. Agents in conflict states might employ the synthesis procedure to propose sequences of actions to resolve conflict. Additionally, they may propose new norms to ensure it does not happen again. In such an online application, a computationally bound synthesis would be employed to restrict the number of runs considered. For instance, agents might only be interested in a single complete run for proposal and evaluation in a normative society. Furthermore, proposals for norms themselves can be evaluated simply by adopting the proposed norms and checking accessibility to the targeted conflict states.

One of the benefits of the proposed knowledge based approach is that domain knowledge is used to direct the synthesis traversal. Therefore only valid and viable runs are considered in each traversal. While in the general case it is possible for many of the defined operators to be applicable in some specification, it is more common that a smaller subset of these would be applicable. Additionally, the use of variables in operator specifications allows for the synthesis of general norms over state specifications, that are in fact applicable to multiple states in the system. Such generally applicable norms are simpler for the agent to apply since they do not rely on full conflict state enumeration.

We now present a discussion on the termination and complexity of the procedure, followed by an argument for completeness and correctness.

Termination and Complexity Termination of the algorithm is guaranteed due to the monotonicity of the state specification refinement function, under the assumption that we have a finite set of literals. For each successor operator appended to a run, the resulting refinement of the run can have one of two effects: the additional constraints imposed by the new operator are not consistent with the existing run, or the run is refined according to the new constraints imposed by the operator. Refinement never removes literals. Each refinement results in a new run that is at least as specific as the previous one. Furthermore, no specification is considered more than once per run since this would yield a loop. Under these conditions the algorithm will always terminate, as our refinement process will eventually result in complete state specifications, or repeated states.

Norm synthesis has been shown to be NP-complete [8]. The decision problem belonging to the run traversal in Algorithm 1 is NP-complete in the worst case, where the set of partially applicable operators for any run includes all operators (excluding the latest operator in the run). For each incomplete run processed, m refined runs will result, one for each of the m operators applicable. Again, in the worst case, each of these runs is deemed incomplete and the process continues. Therefore, from an initial run an exponential number of runs will be inferred through multiple iterations of the traversal loop.

Correctness and Completeness We begin by assuming a complete run $R^* = S_1 \overset{o_1}{\rightarrow} S_2 \overset{o_2}{\rightarrow} \ldots \overset{o_n}{\rightarrow} S_n$. We show that we always consider the operator o_1 since $(S_2 \backslash S_1) \cap S_C \neq \emptyset$ and $post(o_1) \cap S_C \neq \emptyset$. All *contributing* actions satisfy this requirement by definition, and hence o_1 is considered.

Next we show that the precursor state inference is sound and that the condition on the synthesised norm is correct. Assume the inferred partial run $S_P \overset{o_1}{\rightarrow} S_C$. It is sufficient to show that $S_P \subseteq S_1$. We write $S_1' \subseteq S_2 \backslash post(o) \cup pre(o)$, and by $\overleftarrow{Inf}$ we define $S_P = S_C \backslash post(o) \cup pre(o)$. But since $S_C \subseteq S_2$ then $S_P \subseteq S_1$ holds. Furthermore, each successor operator might introduce new literals resulting in S_P being continually refined. However, since these literals have not been added by any precursor operator in the synthesised run, we infer that they are part of the initial state S_P. Similarly, such literals must be part of S_1, and therefore any new literal added to S_P must already be present in S_1. Therefore any norm conditional on S_P will include S_1.

Finally, we argue that for our algorithm to not find the proposed run, then some successor operator must exist that was overlooked. We disregard operators in two situations: if a successor operator is inconsistent with the synthesised run, or if the current state specification is conflict-free. Neither hold, since if the first is true, then the operator is inconsistent with R^*, and if the second is true then R^* contains an intermediate conflict-free state specification.

It follows that no such run exists and from this it follows that the algorithm is correct. If a run exists, it will be found and if no complete runs are produced by the algorithm then all incomplete runs loop or terminate in conflict states and no complete runs exist.

5.1 Improving Synthesis Performance

While the conflict-rooted traversal is required in the gen-

eral case to ensure accessibility under synthesised norms, there are likely to be special cases where assumptions can be made to improve the efficiency of the process. In certain domains (such as our example domain) it is sufficient to ensure accessibility to *any* conflict-free state. In this case, the traversal should check accessibility once any complete run has been found, rather than obtaining all such runs first. This approach holds in systems where the actions of one agent are not dependent on another, as an agent is able to achieve a goal independently of whether they are in a conflict-state or not.

Alternatively, planning heuristics can be used to improve the process. For instance, by assuming no negated literals, an approximate plan length can be computed and used as a heuristic to evaluate which runs to further traverse. Furthermore, if a system designer provides a set of focal states then the process is further simplified. If each of these specifications is accessible from all precursor states under the synthesised prohibitions, then there is no need for an explicit full traversal procedure.

Via static analysis of the operators it may be possible to reduce the number of runs traversed during synthesis. For some $o \in O$, if $post(o) \models S_C$ then the operator should always be prohibited via the norm precondition $\top$. If none of the operators traverse from conflict to conflict-free specifications, then there is no need to consider accessibility, and conflict-states must be prohibited.

6. RELATED WORK

The first formal notion of behavioural constraints as social norms was presented by Shoham and Tennenholtz [7, 8]. The NP-complete complexity result for automatically synthesising norms is one of the seminal results of this work. In this work the problem of norm synthesis is presented as the process of finding useful norms given a state transition system. While a theoretical complexity result is provided, no concrete synthesis procedure is listed.

The synthesis problem is further investigated in [6], where norm synthesis in a robot mobilisation domain is reduced to a graph routing problem and computed efficiently using existing graph theoretic methods. However, this solution is very domain-specific as it can only operate on a very specific graph-based structure.

In [3], the synthesis of minimal and simple social laws is presented as an extension of Tennenholtz' artificial social systems work. This synthesis does not produce a useful law according to some social objective, but rather takes a social law as input and provides a minimal or simple social law as output. In this sense, this work is concerned with norm refinement rather than norm synthesis since a norm is assumed as provided by the designer.

In [9] the authors show that Alternating-time Temporal Logic can be used to express and understand social laws for multiagent systems. They illustrate how the synthesis of social laws can be framed as a model checking problem that is shown to be NP-complete in general. However, this work assumes a complete action-based alternative transition system representation which is often infeasible to develop and maintain in practical systems.

7. CONCLUSIONS

The problem of norm synthesis has been shown to be NP-complete [8], a complexity result born primarily from the state enumeration required in abstract systems. In this paper we argue that a knowledge based approach to the problem of norm synthesis can lead to a procedure that exploits additional domain information to synthesise norms more efficiently. Planning domains commonly expose such structure and information through their domain specifications, and we have illustrated a novel method whereby norms are synthesised independently of the specific instances of the planning domain. Since the synthesis is a localised traversal of state specifications the resulting norms are generally applicable and apply to groups of states.

It is noteworthy that our method makes no assumptions regarding possible agent goals as it is requires that conflict-free paths are found among all conflict-free states. The complexity problems that arise due to this requirement can of course be avoided if such additional knowledge about agent goals is available. We have also not explicitly addressed the problem of escaping conflict states when they occur, but this can be easily done using our methods by starting the traversal procedure at the conflict state rather than its predecessor and imposing norms that force the agents to leave the conflict state set as quickly as possible.

In the future, we would like to investigate more complex planning languages (e.g. including boolean preconditions rather than just sets of literals), to explore the use of "suggested" actions rather than prohibitions, and to introduce global preferences over states in order to extend the applicability of our method beyond domains with a binary conflict/non-conflict distinction.

8. REFERENCES

[1] G. Boella, L. van der Torre, and H. Verhagen. A normative framework for agent-based systems. *Computational and Mathematical Organization Theory*, 12(2-3):71–79, Oct. 2006.

[2] W. Briggs and D. Cook. Flexible social laws. In *In Proceedings 14th International Joint Conference on Artificial Intelligence*, pages 688–693, 1995.

[3] D. Fitoussi and M. Tennenholtz. Choosing social laws for multi-agent systems: Minimality and simplicity. *Artificial Intelligence*, 119:61–101, 2000.

[4] D. Lewis. *Convention: A Philosophical Study*. Harvard University Press, first edition, 1969.

[5] B. Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence*, 12:271–315, May 2000.

[6] S. Onn and M. Tennenholtz. Determination of social laws for multi-agent mobilization. *Artificial Intelligence*, 95:155–167, Jun 1997.

[7] Y. Shoham and M. Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 276–281, 1992.

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

[9] W. van der Hoek, M. Roberts, and M. Wooldridge. Social laws in alternating time: Effectiveness, feasibility, and synthesis. *Synthese*, 156(1), May 2007.

[10] G. Weiß, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1999.