Norm Identification through Plan Recognition

Nir Oren[1] Felipe Meneguzzi[2]

[1]University of Aberdeen
n.oren@abdn.ac.uk

[2]Potifical Catholic University of Rio Grande do Sul
felipe.meneguzzi@pucrs.br

Overview

- When entering a normative society, an agent must become aware of any norms in order to ensure that it can act in a norm-complaint manner.
- When these norms are codified, they can be transmitted to the new agent.

Overview

- When entering a normative society, an agent must become aware of any norms in order to ensure that it can act in a norm-complaint manner.
- When these norms are codified, they can be transmitted to the new agent.
- But what if they are not?
- Or there's limited bandwidth?
- Or norms are in flux?
- Or there is no shared ontology?

Overview

- When entering a normative society, an agent must become aware of any norms in order to ensure that it can act in a norm-complaint manner.
- When these norms are codified, they can be transmitted to the new agent.
- But what if they are not?
- Or there's limited bandwidth?
- Or norms are in flux?
- Or there is no shared ontology?
- In such situations, the new agent must identify the norms, with little assistance from other agents in the system.

Existing Work

- Observation-based norm identification techniques track the behaviour of others to infer norms currently in force.
- Most common approach utilises the detection of a violation signal.
- The violation signal is raised when an agent violates a norm. By identifying the violating situation(s), the norm can be identified.
- This approach works well when norms are regularly violated and sanctions explicitly applied.
- But what if the system is in a steady state and very few violations occur? Or agents are mostly norm-abiding?

Plan Recognition Based Norm Identification

- We suggest that an agent can use a plan recogniser to identify the plans being executed by others from their actions.
- From these plans, goals can be identified.
- A planner is then used to generate alternative plans that achieve these goals.
- A comparison of the alternative plans and actual plans can, over time, identify avoided state/actions (corresponding to prohibitions) and repeatedly executed actions/states visited (corresponding to obligations).

Plans and the Environment

- As often done, we represent the environment as a state transition system.
- Each state contains a set of atoms.

$$\texttt{at}(\texttt{london}, \texttt{truck1}) \qquad \texttt{onTruck}(\texttt{cargo}, \texttt{truck}_1)$$

- Action execution causes a state transition. We utilise operators, or action templates, to generically specify the effects of actions.

$$o = (name(o), pre(o), post(o))$$

- Postconditions identify what atoms should be added to the state ($post^+(o)$), and removed from the state ($post^-(o)$) to obtain the new state.
- Actions are then grounding substitutions over all variables in the operator.
- A plan is a sequence of actions.

Plans

- Classical planning can be used to identify the plan required to achieve some goal.
- However, this is computationally expensive, and we therefore assume the existence of a plan library.
- A plan library contains plans generated offline, which can be composed to achieve high level goals.
- Some of the plans in the library specify their own goals, which require the execution of additional plans to satisfy.
- This plan decomposition continues until primitive tasks are identified which map directly onto actions.

Plans

- Note that multiple sub-plans could all be candidates in order to achieve a single step in a high-level plan.
- The planning problem then reduces to identify which sub-plans to compose in order to achieve high level goals, such that the entire overarching plan is consistent.
- This is a HTN planning problem.
- Note that HTN panning is analogous to AgentSpeak(L) planning.

HTN planning

- A HTN planner aims to decompose a set of high level tasks, encoded as a task network, into a set of primitive tasks.
- The task network is a directed graph, whose nodes are the tasks, and edges are temporal constraints.
- A task is an expression of the form $t(r_1, \ldots, r_n)$ where t is a task symbol, and $r_1, \ldots, r_n$ are terms.

$$\texttt{travel}(\texttt{S}, \texttt{D})$$

- Methods can be used to satisfy tasks — one could fly, or catch a train between source and destination.

$$m = (name(m), task(m), precond(m), network(m))$$

- $task(m)$ identifies the task the method can refine. $precond(m)$ are positive and negative preconditions that must be satisfied for the method to apply. $network(m)$ identifies the tasks that must be carried out in order to achieve the original task, represented as a task network.

HTN Planning

- A HTN planning problem can be viewed as picking a set of leaf nodes from a AND/OR tree.
- All leaf nodes belonging to a AND node must be picked, and one node is picked for OR nodes.
- The temporal constraints limit legal sequences of leaf nodes.
- Given a plan, it is possible to identify both the set of tasks and specific task instances that form the plan.
- Doing the latter lies at the heart of plan recognition.

Plan Recognition

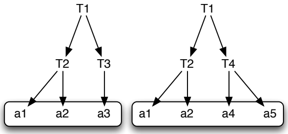
- A large body of work exists on plan recognition.
- In this work we utilise a simple NLP based plan recogniser, utilising a technique similar to parts of speech tagging.
- This makes use of the links between the structure of a HTN and of a context free grammar.
- In HTN planning, a task network is refined into primitive tasks.
- In parsing we transform an initial string containing non-terminal symbols into a string containing only terminal symbols via production rules.

Example

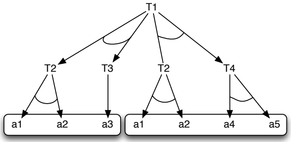
$$
\begin{aligned}
T_1 &\rightarrow T_2 T_3\\
T_1 &\rightarrow T_2 T_4\\
T_2 &\rightarrow a_1 a_2\\
T_3 &\rightarrow a_3\\
T_4 &\rightarrow a_4 a_5
\end{aligned}
$$

- Assume T_i are non-terminal symbols, a_i are terminals. We have a starting symbol T_1.
- We can generate $a_1 a_2 a_3$ or $a_1 a_2 a_4 a_5$

Example



- If we associate each rule $\mu \rightarrow \omega$ with a method of the form $(\textit{name}, \mu, \top, \omega)$ then we obtain an AND/OR tree of potential HTN expansions.



- For plan recognition we seek to identify non-terminal nodes from terminal nodes using this association.

Norms

- Norms are of the form $\mathbf{X}_y z$
  - $\mathbf{X}$ a deontic modality — Obligation (**O**) or prohibition (**F**)
  - y a context condition — when the norm is in effect
  - z a normative condition — what behaviour is expected
- The context condition is a task
- The normative condition is a task or state.

Norms

- A task z occurs in context y iff in the process of y's methods being executed, task z is executed.
- A state z occurs in context y if the state is entered while a method whose task is y is being executed.
- Violations occur if the norm condition must occur and does not, or must not occur and does.

Example

- We seek to travel from Aberdeen to Paris, and are prohibited from transiting via London.
- Our task instance is $\mathtt{travel}(\mathtt{aberdeen}, \mathtt{paris})$
- The prohibition $\mathbf{F}_{\mathtt{travel}(\mathtt{X},\mathtt{Y})}\mathtt{at}(\mathtt{london})$
- We could have the method

$$(\mathtt{fly}(\mathtt{X},\mathtt{Y}), \mathtt{travel}(\mathtt{X},\mathtt{Y}), \{\mathtt{at}(\mathtt{X}), \mathtt{connect}(\mathtt{X},\mathtt{Z}), \mathtt{connect}(\mathtt{Z},\mathtt{Y})\}, \{\mathtt{goto}(\mathtt{X},\mathtt{Z}) \prec \mathtt{goto}(\mathtt{Z},\mathtt{Y})\})$$

- The goto task is represented by the following operator

$$(\mathtt{goto}(\mathtt{X},\mathtt{Y}), \{\mathtt{at}(\mathtt{X})\}, \{\neg\mathtt{at}(\mathtt{X}), \mathtt{at}(\mathtt{Y})\})$$

- Preconditions $\mathtt{connect}(\mathtt{aberdeen}, \mathtt{london}), \mathtt{connect}(\mathtt{london}, \mathtt{paris})$ result in $\mathtt{at}(\mathtt{london})$ occurring in context $\mathtt{travel}(\mathtt{aberdeen}, \mathtt{paris})$), violating the norm.

Where are we?

- We have described HTN planning.
- We have described CFG based plan recognition.
- We have a description of norms in our system.
- We can now describe a basic norm identification mechanism.

Norm Identification - Initialisation

- We consider a set of runs, which could originate from watching an agent multiple times, or watching multiple agents execute actions, or a combination of the two.
- We assume non-interference between individual agents actions.
- We keep track of the set of all possible obligations in the system (*potO*), all possible prohibitions (*potF*) and all impossible prohibitions (*notF*).
- Our algorithm will monotonically reduce the set of potential obligations, meaning that it must be initialised as all possible obligations of our system.
- Given that we assume a finite number of predicates and constant symbols, *potO* is finite (but large).

Norm Identification - Overview

- We operate over a set of runs.
- For each run in the set, we utilise plan recognition to identify a <u>single</u> plan being executed.
- This plan defines high level tasks and their decompositions all the way down to primitive tasks.
- Our algorithm operates in two phases.
  1. Process the plan that was actually executed to identify potential obligations pO in the context of the run and impossible prohibitions.
  2. Process alternative plans which active the same goals to identify potential prohibitions pF.
- We then integrate these into the globally recognised potential obligations and potential and impossible prohibitions.

Norm Identification - Phase 1

- For every task t in the plan, every subtask t' is a potential obligation in the context of the task.

$$pO \leftarrow pO \cup \{\mathbf{O}_t t'\}$$

- Similarly, since it was executed, there is no way it can be a potential prohibition, so we add it to the global set of impossible prohibitions.

$$notF \leftarrow notF \cup \{\mathbf{F}_t t'\}$$

- We do the same for states

$$pO \leftarrow pO \cup \{\mathbf{O}_t s\}$$

$$notF \leftarrow notF \cup \{\mathbf{F}_t s\}$$

Norm Identification - Phase 2

- We consider all possible alternative plans with the same start and end states as the actually recognised plans.
- Any subtask τ' of an alternative plan which are not subtasks of the executed plan are potentially prohibited in their parent context.

$$pF \rightarrow pF \cup \{\mathbf{F}_{\tau}\tau'\}$$

- Ditto for states.

Norm Identification - Merging

- At the end of the run, the potential prohibitions are those potential prohibitions we already had, together with the new potential prohibitions, less those elements which are definitely not prohibited.

$$potF \leftarrow (potF \cup pF) \setminus notF$$

- The new obligations are those old obligations that have still been executed:

$$potO \leftarrow potO \cap_t pO$$

- $\cap_t$ is a context sensitive intersection, preserving any element of *potO* which does not share the same context, and performing a normal intersection where context is shared.
- Repeatedly executing the algorithm over a large number of runs will slowly remove from *potO* those contexts, tasks and states which are not obliged but were often executed.
- It will non-monotonically alter the set of possible prohibitions.

Norm Identification

- This algorithm works well when no agent ever violates norms.
- However, a single violation of an obligation or prohibition will cause this norm to never be learned.
- We can utilise a simple heuristic to overcome this problem.
- We use a counter to count how many times some situation potentially is, or is not an obligation or prohibition.
- A threshold over the ratio is then used to bin the situation into as one of an obligation, prohibition or neither.
- Refer to the paper for further details.

Discussion

- We are in the process of evaluating our approach — an earlier evaluation using classical planning shows promise, but is not totally applicable to this work.
- While we utilised a simple plan recogniser, a more complex one could simply be “slotted in”.
- Obligations in our system can come about when agents have no possible alternative plan to achieve some goal. It should be possible to detect such situations, and filter them out.
- False positive prohibitions can also be generated when a prohibition “blocks” future evolutions of the system. If we assume norm compliance, this is not actually a problem.

Future Work

- How do utilities, and norm-aware agents (and norm/utility aware norm identifiers) affect the norm identification process?
- How can we integrate contrary-to-duties into the system?
- What about interference between agent actions?
- How do we deal with large domains efficiently, given that we need to start by considering all possible obligations in the system?
- How can we merge our plan recognition based approach with a violation signal based technique to get the best of both approaches?

Conclusions

- We described norm identification algorithms based on plan recognition rather than violation signal detection.
- These approaches are aimed at working in systems where violations rarely occur.
- While our initial approach could not handle any violations, an extension of the basic approach can.
- To our knowledge, no one has attempted an approach similar to the one we describe, and there are several exciting avenues of future work which we are actively pursuing.