Opportunities and Challenges for Incorporating Runtime Ethical Constraints into a Learning Agent

Lauren Naylor
lmnaylor@umich.edu

John Laird
laird@umich.edu

December 14, 2017

Abstract

As the capabilities and performance of learning agents continue to improve, we must ensure that these agents behave ethically. Existing approaches introduce a separate component to filter agent behavior. We propose an alternative framework, where the processing uses fixed knowledge embedded in the agent. In this paper, we describe this approach and our implementation, contrasting it with using a separate component. The integration with the agent provides additional capabilities beyond previous approaches so that it can more efficiently evaluate actions and perform minor re-planning when all actions violate its constraints. We demonstrate an implementation of our approach in the cognitive architecture Soar for a simple agent and show that it effectively prevents the agent from executing actions that would violate its constraints. We also discuss challenges that arise from our implementation.

Keywords: trustable AI, cognitive architectures, ethical governor, machine ethics, autonomous learning agents, ethical constraints

Introduction

Autonomous agents are becoming increasingly prevalent (Anderson and Anderson 2007) in cognitive assistants, health-care settings (Shim and Arkin 2017), driver-less cars (Waldrop and others 2015), and household cleaning robots (Forlizzi and DiSalvo 2006). In addition to generating high-quality performance, it is imperative that they obey ethical norms. This is especially difficult to ensure in learning agents, where an agent's behaviors cannot be pre-analyzed through formal verification (Davis et al. 2016). As noted by Amodei et al. (2016), a less than perfect choice of an objective function in reinforcement learning can lead to unexpected and undesirable behavior. A recent and disturbing example was Microsoft's AI twitter bot that had to be shut down after learning to generate racist and hateful comments from interactions with human users (Kraft 2016).

In this paper, we focus on agents that learn completely new knowledge and tasks from interacting with a human instructor, an approach called Interactive Task Learning (ITL) (Laird et al. 2017). Existing ITL agents already learn games and puzzles (Kirk and Laird 2014; Kirk, Mininger, and Laird 2016; Hinrichs and Forbus 2014), delivery and navigation tasks (Mininger and Laird 2016; Talamadupula et al. 2013), and new procedures for personal assistants (Allen et al. 2007; Azaria, Krishnamurthy, and Mitchell 2016). As the space of tasks that are learned and performed expands, a critical issue is ensuring that ITL agents (and other learning agents) are not taught to violate their ethical norms. Off-line external verification is impossible, so instead an approach is needed that attempts to achieve the following, often conflicting goals:

1. Prohibit the agent from executing actions that will violate its ethical norms, directly or indirectly, while allowing actions that do not violate these norms;
2. Minimize the computational overhead associated with monitoring and preventing unethical behaviors; and
3. Enable ethical norms to be easily encoded.

In this paper, we describe our approach, which builds on Arkin's Ethical Governor (Arkin, Ulam, and Duncan 2009) and Winfield's Consequence Engine (Winfield, Blum, and Liu 2014). These approaches introduce components that attempt to ensure that the behavior produced by an agent is consistent with its norms. The Ethical Governor evaluates proposed actions against a set of constraints that are expressed in a formal logic. If an action violates any constraints, the agent does not execute it and instead resumes what it was previously doing. In the case where all proposed actions violate constraints, the governor requests replanning. It is assumed that the governor can accurately determine whether or not a constraint is violated using the information that is currently available to it.

The Consequence Engine simulates all possible actions of an agent, and generates a model of the expected outcome for each one. The safety outcome of each model is evaluated according to a pre-defined function, and the agent selects the action that leads to the highest safety outcome. The agent can also be programmed with safety preferences; in the example given, a robot is programmed to prefer actions in which a human is safest, regardless of the robot's safety.

Here we explore an alternative approach, where a single agent, created in a general cognitive architecture (in our case Soar (Laird 2012)), realizes the governor functionality as a set of primitive knowledge that enforces norms using the capabilities engendered by the cognitive architecture. A critical difference between our approach and the others is that Arkin and Winfield attempt to guarantee behavior of a black-

box agent, whose internal workings are suspect. In contrast, we assume that the initial agent can be trusted (possibly through some validation procedure), and what needs to be "governed" is the tasks taught by a human instructor.

The purpose of this paper is to explore the trade-offs of incorporating the governor functionality within an agent implemented in a cognitive architecture instead of as a separate component. Below we describe our implementation, evaluate its performance, and discuss the lessons we have learned from this research. The focus is *not* on which ethical norms are appropriate for such an agent (Allen, Wallach, and Smit 2006; Wallach, Allen, and Smit 2008; Powers 2006; Bello and Bringsjord 2013).

The Soar Cognitive Architecture

We implement our embedded ethical governor in the Soar cognitive architecture (Laird 2012), which provides a framework within which to build an intelligent agent. The execution cycle in Soar begins by updating the current state with perception, and then elaborating perception with background knowledge. Next, relevant operators are proposed. These operators are evaluated, and a single operator is selected and then applied to modify the current state, retrieve knowledge from a long-term memory (semantic or episodic), and/or execute external motor actions. The knowledge to perform these steps is encoded in production rules that continually match against the current state to provide reactive, context-dependent behavior. The state is a relational graph structure that includes all the information relevant to the agent's current situation, including perception, elaborations of perception, goals, reasoning results, and retrievals from long-term memories.

If the agent's knowledge to select an operator is incomplete or in conflict, a substate is created automatically in order to attempt to resolve this impasse. In the substate, the execution cycle recurs, with operators being proposed and applied until the impasse is resolved.

Soar has multiple long-term memories, including procedural memory for storing production rules, episodic memory for storing the agent's past experiences, and semantic memory, which stores declarative knowledge. Knowledge stored in semantic memory is structured in the same relational graph structure that is used in working memory. The agent can store and retrieve information from semantic memory by creating queries in working memory.

Soar offers a wide array of problem-solving capabilities, making it a rich platform for designing agents. For the purpose of enforcing ethical constraints on behavior, Soar directly supports the encoding of constraints in operator selection rules that prohibit the selection of specific actions in certain situations. However, this is limited to constraints on the selection of operators independent of the consequences of those operators in the specific situation. To date, there is limited to no research on how to encode and enforce more complex ethical constraints in cognitive architectures such as Soar.

Possible Approaches

There are a few alternatives to implementing an ethical governor in a cognitive architecture such as Soar. One option (Figure 1) uses the approaches taken by Arkin and Winfield, in which there is a separate module that evaluates and potentially filters all motor actions. The disadvantage of that approach is that a completely new reasoning engine must be created to act as a governor, duplicating knowledge and capabilities already implemented for the agent. This structure implies that there is a very limited conduit for information to flow between those components, and in many cases, the governor will be starved for information - being forced to make a decision with only the information provided by the performance component.

A second alternative (Figure 2) is to modify the underlying cognitive architecture so that it is directly monitoring the reasoning and actions of an agent implemented in it. This would potentially give the governor access to more information about the reasoning of the agent, but it also requires once again the creation of another reasoning engine within the first to perform calculations that the cognitive architecture is already designed to do, and it raises many issues as to how it would efficiently access agent information and modify agent behavior. As will become evident as we describe the necessary processing below, the feasibility of this approach is questionable when one considers all of the capabilities we desire in an ethical governor. That said, one advantage of both of these approaches is that there is a strong boundary between the agent and the governor, so there can be assurances that changes in the agent do not in some way invalidate the governor.

We take a third approach (Figure 3), where the processing and knowledge for the governor is a part of the agent, implemented using unalterable knowledge encoded in the cognitive architecture framework (and replicated in all ethically-minded agents). This knowledge performs the processing of an ethical governor, invoked as needed to evaluate and possibly inhibit inappropriate agent behavior. One advantage is that by using Soar, governor developers can take advantage of Soar's computational capabilities in developing the governor functionality.

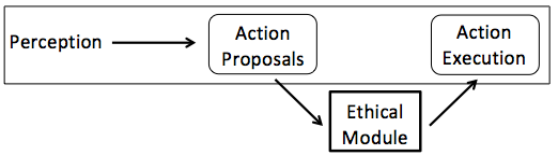

Figure 1: Ethical governor implemented as a separate module.

Implementation

The governor is implemented as a set of rules in Soar that interrupt the normal processing of an agent to evaluate proposed operators by matching the ethical constraints to the anticipated results of the operators' actions. These rules are fixed and not modifiable by any learning mechanism. The

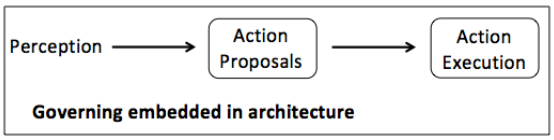

Figure 2: Ethical governor implemented as part of the cognitive architecture.

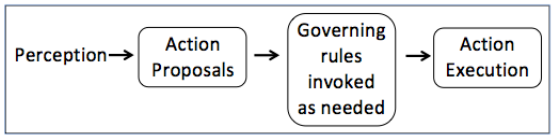

Figure 3: Ethical governor implemented as part of the agent (our approach).

ethical constraints are stored in Soar's long-term semantic memory, represented as relational symbolic structures, and retrieved into Soar's working memory as needed. The constraint structures are also not modifiable by the agent.

Soar provides the computational mechanisms for interrupting processing, evaluating actions, and so on, thereby providing a much simpler and more natural solution to implementing the governor than attempting to modify the inner workings of the architecture and building everything from scratch. This also gives us flexibility by allowing the agent programmer to modify the ethical constraints and governor rules to fit the needs of a specific application domain.

The basic steps of the governor are to

1. determine which operators must be evaluated;
2. evaluate those operators through internal simulation;
3. select an appropriate operator based on their evaluations and other task knowledge.

Below we describe the processing for each of these steps, but first describe how the ethical constraints are stored, represented and retrieved.

Constraint storage, representation and retrieval

The agent must have access to the constraints written by the agent programmer in order to determine if any of the proposed operators violate the norms. An important goal here is to formulate constraints in such a way so that it is easy for the programmer to write them. We also want constraints to be flexible enough to represent any desired forbidden state. Our approach uses declarative symbolic relation structures that are matched to the agent's state representations. Winfield's consequence engine requires a specified function to describe the safety value of each possible state, which is problematic given the fact that because of learning, the possible states that the agent might encounter is unknown ahead of time. In our system, we can specify constraints on a subset of the state structure while disregarding the rest; we may set a constraint that a fired weapon may not hit an ally, no matter what other state information is present. However, we can also create context-dependent constraints that take into account the complete state.

Specifically, we represent constraints in semantic memory in the same graph structure that is used to represent the agent's state in working memory (as shown in Figure 4). An object we name "Constraints" serves as the parent node to all of the present constraints. Each object connected to the parent Constraints then represents one of the constraints and serves as the analogous state parent node. The constraint is then specified by including attributes and objects to this analogous state node, and the constraint will be violated if an operator causes the agent to be in a simulated state that matches the state described by the constraint.

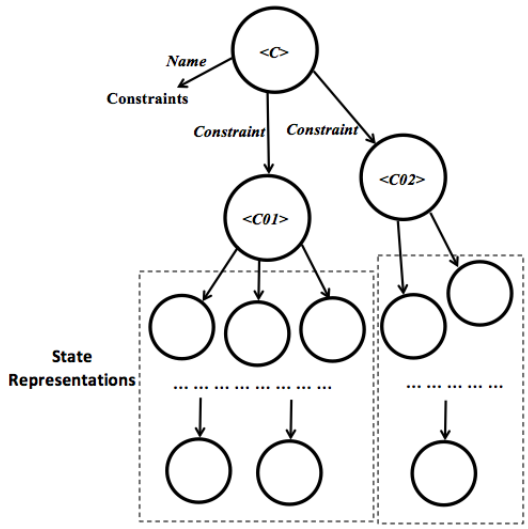

Figure 4: Our representation of constraints in a Soar agent.

Constraints can also specify that certain objects cannot be equal to each other; otherwise, a constraint is able to match a simulated state if it has multiple objects that all match against a single object in the state.

Because this is implemented in a single system, it is easy to add knowledge to compute state information that is only needed to test constraint violations. For example, if our agent should be constrained to remain a certain distance from an object (say, a dangerous explosive), and no representation of distances to other objects is initially included, this calculation is easily added to the agent. This gives us our desired flexibility. Additionally, because the constraint form is matched against the agent's state representation, constraints can be easily encoded by the programmer, who would already be familiar with how the state representation is structured.

The agent retrieves the ethical constraints that are relevant to the current situation from long-term memory, and uses these to evaluate operators (see below).

Determining which operators must be evaluated

When the agent proposes an operator, the governor must determine if any of the operator's external actions pose a risk

of violating the agent's constraints. If so, then the operator must be evaluated and its actions must be checked against the constraints before it can be executed. We must ensure that operators that might violate constraints are evaluated, but we also want to avoid wasting time evaluating operators that do not risk violating the constraints as this internal simulation can be computationally expensive, potentially impacting agent reactivity.

In our implementation, we assume that constraints pertain only to external actions, so the governor flags operators with external actions for evaluation, but does not flag operators that involve only internal processing. Additionally, we foresee situations in which the programmer has knowledge that certain actions will never violate constraints. Thus, the agent programmer can label specific external actions as always being acceptable, and the governor will skip the evaluation of operators with only those actions. To implement this logic in Soar, there are general rules that reject all proposed operators until all flagged operators are evaluated. If no operators are flagged, the rules do not apply and no evaluation is necessary. Rejecting all operators forces an impasse in Soar, which results in the creation of a substate in which the flagged operators can be evaluated. Once the operators are evaluated, a decision is made, informed by those evaluations.

Operator Evaluation

As described above, when there are operators to be evaluated, an impasse will be forced and a substate created. In the substate, the agent evaluates each proposed operator against the constraints, which describe states that are in violation of the ethical norms. To evaluate an operator, the agent simulates the operators' actions and compares the resulting state against the constraints. The simulation is performed internally, using knowledge (encoded as rules) that models the dynamics of the actions and the environment. If the state that results from the simulation matches any constraints, the operator is marked as a violation and the operator is rejected so it cannot be selected. This evaluation allows not only the testing of the ethical constraints but also the agent's task-specific preferences, avoiding duplicate simulations that might otherwise occur in other approaches.

If the operator is not found to violate any constraints, the governor assesses if the agent has all the relevant information to determine if a constraint is violated. If it does, the operator is marked as successful. If it does not, then the agent requests the necessary information. This is in contrast to other approaches, which assume that all relevant information is available to the governor for evaluation. Our alternative approach allows the agent to request more information when evaluating the constraints, which may require information above and beyond what the agent has available, both in terms of its perception of the environment and its overall situational awareness involving information derived through internal reasoning. For example, an agent might be instructed to fire a weapon without any of its sensors active. If there is an object or person in the line of fire that the agent is not allowed to hit, the governor will be unable to detect a constraint violation. Thus, the agent will make a decision based on a lack of knowledge, assuming that if something is unknown (the location of other objects) then it is not true (there are not objects to be hit), when instead, it should make its decisions based on positive knowledge that some fact is not true. In our approach, the governor detects when it needs more information to evaluate the outcome of an action, and can request additional information about the environment, which in this case is to activate its sensors.

Operator Selection

After all operators have been evaluated, the agent selects and executes the best operator that passed the ethical constraints based on its available domain knowledge. However, it may be that all operators failed the constraints, leaving the agent with no feasible actions. In this case, we want our agent to act sensibly, and select an acceptable operator as efficiently as possible. In order to avoid costly re-planning, our governor can alter and re-evaluate operators that initially violated constraints without rejecting all operators' actions out of hand. In addition to labeling actions as 'always acceptable', the programmer can label actions as 'potentially problematic'. If the agent evaluates all of its proposed operators as failures, it will remove any actions that have been labeled as 'potentially problematic' from those operators and re-evaluate them. If any of these newly modified operators do not violate the constraints, the agent is able to select one to apply.

It is still possible that these altered operators will violate the constraints and the agent will need to re-plan to generate new operators; however, giving the governor some ability in re-planning can save significant time in situations where small alterations in the actions can eliminate ethical issues. Our model of the governor gives flexibility to the programmers in deciding where re-planning should occur for each action. Continuing the previous example, the agent may have proposed only the operator to move forward and fire, but the governor finds that this will hit an unacceptable target. The governor can then modify the operator to only include the action to move forward, and evaluate this. (Or not, if moving forward has been labeled as safe).

Limitations

One limitation of our governor is that in some cases it cannot overcome constraint violations that occur due to external forces. While our agent is capable of modifying its proposed operators if none are acceptable, it may still find itself in situations where all operators are prohibited and it is unable to take actions that improve its situation. For instance, an agent may have a constraint that it cannot be within a certain distance of an object. If that object is moved to be within that distance of the agent by some other agent or force, the constraint is violated. If the agent cannot move far enough away from that object with a single action, then all its proposed operators will violate the constraint, and it cannot even select an action that takes it away from the object. Part of the problem is that our system only considers constraint failures that are all or none, whereas in many real-world cases there can be degrees of constraint violation.

Our governor is also limited in the scope of its operator evaluation. It is not able to detect if an action has any long-term ramifications that will lead to a constraint violation at some time in the future. Imagine an agent that has been taught to move toward other agents and then fire its weapon, but is constrained to not fire at allies. It will be allowed to move toward an ally, even though it is only doing so because it has the goal of firing at that agent. The governor will kick in when the agent does attempt to fire, but it would be preferable for our agent to not waste time taking actions to achieve a goal that will ultimately violate a constraint.

Programmer Responsibilities

While we have tried to ensure that our ethical governor is applicable to any agent, there are some specific details that must be enacted by the agent's programmer. Note that the main use case we are considering is where the programmer can be trusted, but through instruction by other users, the agent may acquire questionable knowledge.

One important responsibility is that the agent can accurately simulate the results of its actions; otherwise, the governor's evaluations will not be correct. As mentioned above, the governor may find a constraint to not be violated because it does not have enough information, and is capable of flagging a need for more information. The programmer must implement rules that determine how the agent acts in this situation. It may benefit the agent to create specific rules of how to gather more information for different actions.

Evaluation

Although our goal is to implement this within an ITL agent, we first evaluate a non-learning agent where we can deliberately manipulate its knowledge (simulating learning) and then evaluate the ability of our approach to ensure normative behavior. We then demonstrate our ethical governor integrated with a learning agent called Rosie.

Tanksoar

To demonstrate some of the characteristics of our ethical governor, we use a simple simulated domain, called Tanksoar. Tanksoar is a game with discrete time that takes place on a two-dimensional grid filled with various objects. A Tanksoar agent controls a tank object, which can move around the grid, use radar to detect objects, and shoot missiles to destroy other tanks (Figure 5).

Figure 5: A simple Tanksoar state.

Setup

We compare five different Tanksoar agents: one which has no ethical governor, one with an ethical governor but no action labels, one with the governor and only safe action labels, one with the governor and only problematic action labels, and one with the governor and both types of action labels. In each scenario, all agents propose an operator to move forward and fire, and an operator to turn left and fire, with the exception of the agent with only safe labels. This agent proposes an operator to move forward and fire and an operator to turn left only. The operator to move and fire is set to be preferred over all other operators. All agents (except the one with no governor) are given the constraint that they *cannot* hit a missile-pack object with a fired missile. The agents with safe labels have the turn action labeled as safe, and the agents with problematic labels have the fire action labeled as potentially problematic. We put our agents in three different scenarios: one in which there is a missile-pack three squares in front and a missile-pack one square to the left, one in which there is a missile-pack three squares in front, and one in which there are no missile-packs surrounding the agent.

To measure the overhead incurred by the ethical governor, we examine the number of decision cycles it takes the agent to apply one of its two proposed operators. This is a straightforward test that the governor does what it is designed to do and that there are no unexpected interactions between the different labels and constraints.

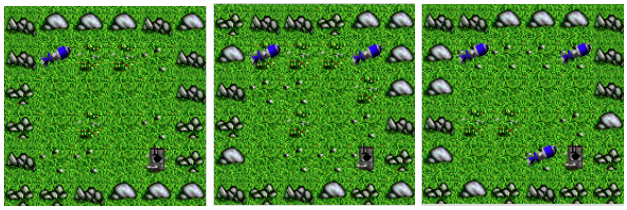

Figure 6: Three Tanksoar scenarios; from left to right: No visible missile-packs, one visible missile-pack, and two visible missile-packs.

Results

Figure 7 shows the number of decision cycles used in each agent and the actions taken by each agent are shown in Table 1. Without any ethical governor, the agent can perform actions that violate its constraints. We see that in all three situations, the Tanksoar agent without a governor moves forward and fires, even when there is a missile-pack in front of it. The different versions of ethical governors all require more decision cycles to select an operator, but all prevent the agent from violating its constraint and firing at the missile-pack.

The agent with safe action labels and both types of labels avoids unnecessary action evaluations. In our evaluation, we have the Tanksoar agent with safe action labels propose to turn left only, rather than turn and fire, in order to demonstrate that it is capable of selecting an operator with only labeled safe actions without evaluating it. When its two opera-

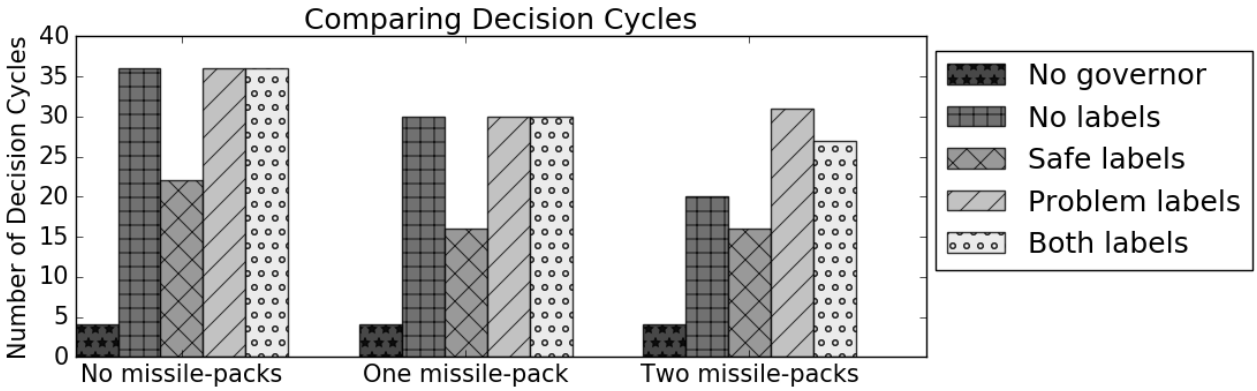

Figure 7: The number of decision cycles required for our Tanksoar agents to select an action.

	No missile-packs	One missile-pack	Two missile-packs
No governor	Move forward and fire	*Move forward and fire*	*Move forward and fire*
No labels	Move forward and fire	Turn left and fire	Cannot act
Safe labels	Move forward and fire	Turn left	Turn left
Problem labels	Move forward and fire	Turn left and fire	Move forward
Both labels	Move forward and fire	Turn left and fire	Move forward

Table 1: The actions taken by each Tanksoar agent in each scenario. Actions that are italicized violate the agent's constraint.

tors are proposed, it does not flag turn left as needing evaluation, and automatically labels it as a success. In the one- and two-missile-pack scenarios, the agent evaluates move forward + fire to violate its constraint, so it instead turns left. In the two-missile-pack scenario, the agent with both types of labels finds that all of its proposed operators evaluate as failures, and removes the 'fire' action from each. Because turning has been labeled as safe, it only evaluates the 'move forward' operator, reducing the number of needed decision cycles.

When evaluating the proposed operators, all versions of the governor initially find that none of the actions violate the constraints. However, instead of labeling all operators as a success, the governor flags a need for more information, because its radar is turned off. This results in the agent turning on its radar. Now the agent detects the surrounding missile-packs, and accurately evaluates its proposed operators against its constraint.

In the two-missile-pack scenario, the governor with no labels cannot act because both of its proposed actions will fire a missile at a missile-pack. (The governor with only safe action labels would also be unable to act, if we had specified that to should propose turn left + fire, instead of turn left only.) The governors with problematic action labels are able to modify the failed proposed operators. The 'fire' action is labeled problematic, so the proposed operators are each modified to 'turn left' and 'move forward'. These operators are then re-evaluated (although the governor with 'turn' labeled as safe does not evaluate 'turn left'), and both are found to be successes. Because the original operator to move forward and fire was preferred by the agent, it selects the operator to move forward only. Despite its initial operators violating the constraint, the Tanksoar agent can act sensibly without requesting complete re-planning.

Rosie

To demonstrate how our ethical governor can be combined with an ITL agent, we have integrated it with a learning agent called Rosie. For this work, Rosie operates in a simulated domain which contains four rooms and three hallways, shown in figure 8. Rosie can be given commands to pick up or put down an object or to go to a certain room, and it is also capable of being taught to perform new tasks, such as to deliver an object to a room. In this work, we focus on using the ethical governor to constrain Rosie when given a specific command, such as "Pick up the red box" and "Go to the kitchen".

When Rosie is given the command "Go to the kitchen" (Location 3 in figure 8), it creates an operator to execute this task. While applying this operator, Rosie enters different substates to determine how to execute the task "Go to the kitchen" - first it must go to Hallway 5, and then it can

go to the kitchen. Our ethical governor evaluates each of these subactions, rather than the main action of going to the kitchen. This allows it to determine if any constraints will be violated during execution of the main action, instead of merely checking the constraints against the final state. For example, if Rosie is constrained to not enter Hallway 5, evaluating the subactions will allow the ethical governor to realize that going to the kitchen will in fact cause a violation.

Our ethical governor is successful in prohibiting Rosie from picking up objects, as well as entering specific locations. Given constraints that it should not hold the red box and it should not enter Hallway 5, Rosie did not execute commands to pick up the red box or go to the kitchen, but was still able to execute a command to pick up the blue box. By altering the second constraint to forbid entering Hallway 7, Rosie was then able to obey the command to go to the kitchen.

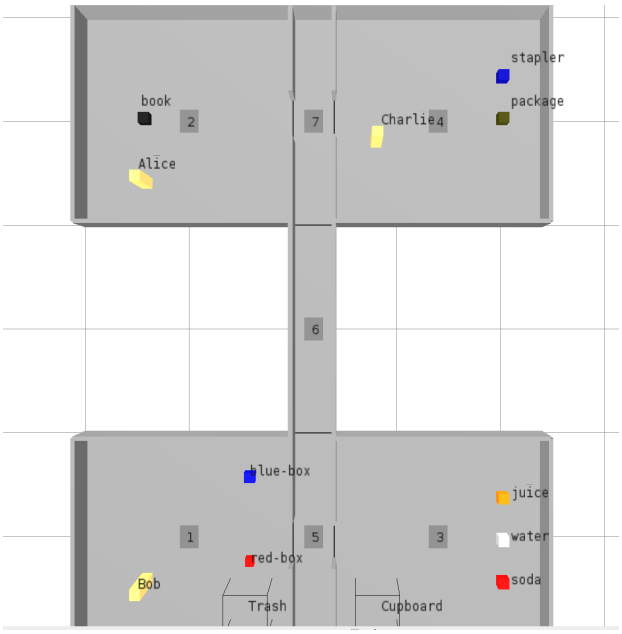

Figure 8: The simulated environment used with Rosie.

Discussion

Our ethical governor extends the capabilities of Arkin's Ethical Governor and Winfield's Consequence Engine in several ways. Our constraints are easily written, taking the same structure used to describe the agent's state. Rather than assigning a function to describe each possible state's safety value, our constraints are directly expressible in terms of states that should be avoided by the agent. Unlike Arkin's governor, our governor is capable of flagging a need for more information, allowing it to more accurately determine if a constraint is violated. It can also better handle situations in which all constraints are violated by modifying its existing proposed operators to attempt to find one that is acceptable. While not guaranteed to provide a solution in all cases, this provides the chance to select an acceptable operator without needing to request re-planning from a separate architecture component. Our safe action labels save the governor time by skipping the evaluation of operators containing only actions known by the programmer to always be acceptable.

It should be noted that one decision cycle in Soar takes approximately 0.3 milliseconds.If we want our agent to have ten interactions with the environment per second, then we can allow it about 300 decision cycles for each external action. As seen in Figure 7, even the slowest governor is roughly ten times faster than this requirement. So although our ethical governor significantly increases the number of decision cycles to select an operator, the agent is still able to maintain reactivity.

There still remain many areas for improvement in our embedded ethical governor. Our governor's capability to flag a need for more information can be expanded and implemented in a more sophisticated way. Currently, the governor sets this flag if it finds a constraint is not matched, because it may find that constraint to be matched if it had more information. (For example, our Tanksoar agent needed to turn on its radar to see that firing a missile would hit a missile-pack in its path). However, the agent cannot determine what specific information would allow it to confirm a constraint to be (un)violated. While it is currently left to the programmer to add specific rules for how to gather more information when the flag is set, we hope to eventually shift more of this responsibility to the ethical governor.

Our governor is incapable of acting if all proposed operators violate constraints (including any modified operators). A future goal is to enable the governor in this situation to propose operators that will put the agent in a state that is 'less' of a violation than its current state. The governor may be able to give the agent the goal of achieving a state in which the constraint is not violated, and temporarily allow the agent to violate the constraint until it reaches its goal. This could, however, introduce more issues: should the agent be allowed to violate other constraints, if doing so would allow it to reach its goal much faster than not doing so? (Or what if it cannot reach its goal without violating other constraints?) This suggests it may be necessary to include some kind of ranking system that details which constraints may be violated to avoid others, and which may not be violated under any circumstances.

Although one of the advantages of our approach is that the governor can actively gather more information in knowledge-poor situations, such as by turning on the radar in Tanksoar, it is possible for these actions to negatively interact with other goals and desires of the agent. One reason for having the radar off is to stay concealed (other tanks have radar detectors), so turning the radar on can make the tank more vulnerable. Thus, we need to expand that agent's capabilities to also include reasoning about the ramifications of its sensor actions, so it avoids interference with the agent's goals.

While we aim to give the ethical governor as much responsibility as possible in evaluating constraints and determining alternative actions when necessary, our implementation allows the programmer to add their own specific rules.

By using the existing capabilities of the cognitive architecture to create our governor, it is easy for the programmer to make changes as they see fit. However, this also means that a malicious programmer could alter the agent's code and render the ethical governor ineffective. As we improve the ethical governor's capabilities, it may be necessary to explore methods of directly altering the architecture in order to prevent explicit attacks against the governor.

Despite these challenges, we have proposed a framework for an ethical governor embedded in an agent's knowledge that can evaluate that agent's actions at runtime, and demonstrated a working implementation of this governor in the Soar cognitive architecture. In the future, we aim to further test our ethical governor in an ITL agent. Our goal is that the governor would prevent the agent from executing actions that violate its constraints, even if it has been taught to perform these actions to complete a task. Our approach offers improvement over previous work and provides a method by which an ethical governor can be embedded in an agent, harnessing the power of its cognitive architecture.

References

Allen, J.; Chambers, N.; Ferguson, G.; Galescu, L.; Jung, H.; Swift, M.; and Taysom, W. 2007. Plow: A Collaborative Task Learning Agent. In *AAAI*, 1514–1519.

Allen, C.; Wallach, W.; and Smit, I. 2006. Why Machine Ethics? *IEEE Intelligent Systems* 21(4):12–17.

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete Problems in AI Safety. *arXiv preprint arXiv:1606.06565*.

Anderson, M., and Anderson, S. L. 2007. Machine Ethics: Creating an Ethical Intelligent Agent. *AI Magazine* 28(4):15.

Arkin, R. C.; Ulam, P.; and Duncan, B. 2009. An Ethical Governor for Constraining Lethal Action in an Autonomous System. Technical report, Georgia Inst Of Tech Atlanta Mobile Robot Lab.

Azaria, A.; Krishnamurthy, J.; and Mitchell, T. M. 2016. Instructable Intelligent Personal Agent. In *AAAI*, 2681–2689.

Bello, P., and Bringsjord, S. 2013. On How to Build a Moral Machine. *Topoi* 32(2):251–266.

Davis, D.; Brutzman, D.; Blais, C.; and McGhee, R. 2016. Ethical Mission Definition and Execution for Maritime Robotic Vehicles: A Practical Approach. In *OCEANS 2016 MTS/IEEE Monterey*, 1–10. IEEE.

Forlizzi, J., and DiSalvo, C. 2006. Service Robots in the Domestic Environment: A Study of the Roomba Vacuum in the Home. In *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction*, 258–265. ACM.

Hinrichs, T. R., and Forbus, K. D. 2014. X Goes First: Teaching Simple Games Through Multimodal Interaction. *Advances in Cognitive Systems* 3:31–46.

Kirk, J. R., and Laird, J. 2014. Interactive Task Learning for Simple Games. *Advances in Cognitive Systems* 3:13–30.

Kirk, J.; Mininger, A.; and Laird, J. 2016. Learning Task Goals Interactively With Visual Demonstrations. *Biologically Inspired Cognitive Architectures* 18:1–8.

Kraft, A. 2016. Microsoft Shuts Down AI Chatbot After it Turned into a Nazi. *CBS News*.

Laird, J. E.; Gluck, K.; Anderson, J.; Forbus, K. D.; Jenkins, O. C.; Lebiere, C.; Salvucci, D.; Scheutz, M.; Thomaz, A.; Trafton, G.; et al. 2017. Interactive Task Learning. *IEEE Intelligent Systems* 32(4):6–21.

Laird, J. E. 2012. *The Soar Cognitive Architecture*. MIT press.

Mininger, A., and Laird, J. 2016. Interactively Learning Strategies for Handling References to Unseen or Unknown Objects. *Adv. Cogn. Syst* 5.

Powers, T. M. 2006. Prospects for a Kantian Machine. *IEEE Intelligent Systems* 21(4):46–51.

Shim, J., and Arkin, R. C. 2017. An Intervening Ethical Governor for a Robot Mediator in Patient-Caregiver Relationships. In *A World with Robots*. Springer. 77–91.

Talamadupula, K.; Briggs, G.; Scheutz, M.; and Kambhampati, S. 2013. Architectural Mechanisms for Handling Human Instructions in Open-World Mixed-Initiative Team Tasks. *Advances in Cognitive Systems (ACS)* 6.

Waldrop, M. M., et al. 2015. No Drivers Required. *Nature* 518(7537):20–20.

Wallach, W.; Allen, C.; and Smit, I. 2008. Machine Morality: Bottom-Up and Top-Down Approaches for Modelling Human Moral Faculties. *AI & Society* 22(4):565–582.

Winfield, A. F.; Blum, C.; and Liu, W. 2014. Towards an Ethical Robot: Internal Models, Consequences and Ethical Action Selection. In *Conference Towards Autonomous Robotic Systems*, 85–96. Springer.