# Sensor Placement for Plan Monitoring using Genetic Programming

Felipe Meneguzzi[1], Ramon Fraga Pereira[1], and Nir Oren[2]

[1] Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil
felipe.meneguzzi@pucrs.br
ramon.pereira@acad.pucrs.br
[2] University of Aberdeen, Scotland, UK
n.oren@abdn.ac.uk

**Abstract.** Monitoring plan execution is useful in various multi-agent applications, from agent cooperation to norm enforcement. Realistic environments often impose constraints on the capabilities of such monitoring, limiting the amount and coverage of available sensors. In this paper, we consider the problem of sensor placement within an environment to determine whether some behaviour has occurred. Our model is based on the semantics of planning, and we provide a simple formalism for describing sensors and behaviours in such a model. Given the computational complexity of the sensor placement problem, we investigate heuristic techniques for performing sensor placement, demonstrating that such techniques perform well even in complex domains.

## 1 Introduction

Norms are commonly used to obtain desirable behaviour within an open multi-agent system. Such norms specify obligations, permissions, and prohibitions on individual behaviour, preventing actions or states of affairs that an agent might find beneficial, but which will have a negative effect on others or the system environment as a whole [7]. Given a normative multi-agent system, the question arises as to how to ensure that agents comply with the norms. While it is possible to sometimes design the system so that violating a norm is irrational [11], or design the agents so that they are incapable of violating norms [2], doing so within an open system is often difficult or impossible. Instead, sanctioning mechanisms are normally introduced to punish, and therefore disincentivise norm violation [7]. Recent work has developed an approach to define how norms should be modified to be monitorable given an available set of imperfect monitors [1], and the problem we address here is the dual of such work. In turn, we consider a further problem, namely how to combine a set of so-called *primitive sensors* — available within the environment — to form a new sensor that will be able to detect whether some state of affairs does, or does not hold.

We consider an abstract form of the problem, seeking to identify whether — with no prior knowledge on agent preferences — some behaviour can be detected by combining the primitive sensors. For example, consider two cameras overlooking different portions of a highway, and assume that each camera can uniquely identify individual cars. In such a system, the two cameras (i.e., the sensors) can be combined (synthesised) to form a new sensor which can detect a vehicle's average speed (i.e., a behaviour or state-of-affairs) over the stretch of highway. We refer to this problem as the *plan monitoring* problem.

We formally describe our primitive sensors and the behaviour we wish to detect as formulae within a simple logic. Synthesising a new sensor then involves creating a new formula by joining a subset of the primitive sensor formulae with operators from the logic. If this new formula is equivalent to the formula encoding the behaviour we wish to detect, then we are able to form a sensor for monitoring the behaviour. Since this is clearly a computationally hard problem, in this paper we consider a heuristic approach for the synthesis of our sensors using genetic programming.

## 2 Plan Monitoring

We formalise logic formulas over states in Definition 1 — these are basically propositional-logic formulas to be evaluated in individual states.

**Definition 1 (State Formula).** *Let $\mathcal{F}$ be a set of fluents*[3]*. If $\varphi \in \mathcal{F}$ then $\varphi$ is a state formula*[4]*. If $\varphi$ and $\psi$ are state formula, then $\varphi \wedge \psi$, $\varphi \vee \psi$, and $\neg\varphi$ are state formulas. Nothing else is a state formula. State formulas are evaluated according to a valuation function $V : S \rightarrow \mathcal{F}$, which returns a set of fluents that hold at state $s \in S$ (set of possible states). We write $s \models \phi$ ($s$ is a model of $\phi$) if $\phi \in V(s)$; $s \models \phi \wedge \psi$ iff $s \models \psi$ and $s \models \phi$; and $s \models \phi \vee \psi$ iff $s \models \psi$ or $s \models \phi$; and $s \models \neg\phi$ iff $s \not\models \phi$.*

To formalise plan monitoring tasks, we must express constraints over entire plans, made up of traces or sequences of states which occur (Definition 2).

**Definition 2 (Path Formula).** *If $\varphi$ is a state formula, then $\varphi$ is a path formula. If $\varphi$ and $\psi$ are path formulas, then $\varphi[Y]\psi$ and $\neg\varphi$ are path formulas. Let $t_\pi$ be a trace and $\varphi$ and $\varphi[Y]\psi$ be path formulas. $Y$ represents the number of steps between state formulas. We write $t_\pi \models \varphi$ ($t_\pi$ is model for $\varphi$) iff any state $s_i \in t_\pi \models \varphi$, and $t_\pi \models \varphi[Y]\psi$ iff $s_i, s_k \in t_\pi$; $s_i \models \varphi$; $s_{i+k} \models \psi$; and $Y \geq k$. We write $t_\pi \models \neg\varphi$ iff it is not the case that $t_\pi \models \varphi$.*

Path formulas can only be evaluated over plan traces (i.e., sequences of states), so a path formula $\varphi[Y]\psi$ is in a trace if $\varphi$ holds in any state of the trace, and $\psi$ holds in any state within $Y$ steps or less of when $\varphi$ held. It should be noted that conjunctions over path formulae can be captured using $\varphi[0]\psi$. Together with negation, this provides us with disjunctions over path formulae.

[3] Fluents are ground logical predicates, which can either be positive or negated, and include constants for truth ($\top$) and falsehood ($\bot$).

[4] A state formula is comprised of a finite set fluents that represent logical values according to some interpretation.

*Example 1. Consider the domain model illustrated in Figure 1, and a trace* $t_{b,a} = \langle [q], [p], [p, q] \rangle$ *for a plan* $\langle b, a \rangle$*. Formula* $q[2](p \wedge q)$ *is true for this trace, whereas formula* $q[1](p \wedge q)$ *is not.*

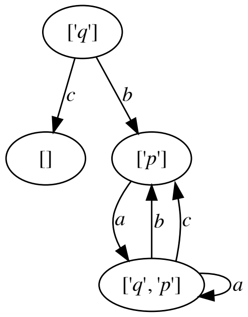

Fig. 1: Propositional domain example.

A sensor is a mechanism that evaluates path formulas over traces, and represents a concrete and indivisible (atomic) capability to evaluate path formulas on plan traces, following Definition 3.

**Definition 3 (Sensor).** *Let* $\varphi$ *be a path formula and* $t_\pi$ *be a trace, we say* $\varphi$ *is a sensor for* $t_\pi$ *iff* $t_\pi \models \varphi$*.*

Sensors can be aggregated to form monitors to detect specific desirable formulas, following Definition 4.

**Definition 4 (Monitor).** *Let* $\mathcal{S} = \{\varphi_1, \dots \varphi_n\}$ *be a set of sensors. A monitor* $M$ *is a sensor obtained by combining a subset of* $\mathcal{S}$ *using path operations.* $M$ *is a monitor for a trace* $t_\pi$ *iff* $t_\pi \models M$*.*

We note that this formalisation of sensors and monitors provides a simple mechanism to describe partially observable monitoring problems. For example, in Figure 1, if no available sensor has formulas referring to $p$, then a monitoring problem for this example is partially observable with respect to $p$. Algorithm 1 describes a simple function to compute whether a sensor is *sensitive* to a trace, returning a set of models for a given sensor, a trace, a state, and a set of actions.

## 2.1 Decision Problems

So far, we have defined individual sensors and described how these can be aggregated into more complex sensors, which we call monitors. We use these to represent imperfect sensing capabilities which, much like the real world, may not be capable of fully distinguishing the states and traces of interest. Thus, we need to be able to quantify the extent to which the resulting monitors can

**Algorithm 1** Computation of the $\models$ relation.

**Input:** A sensor $\sigma$, a trace $t_\pi$, a state $S$, and a set of actions $A$.
**Output:** A set of models ($\models$) relation.

```
 1: function MODELS($\sigma, t_\pi, S, A$)
 2:     if $\sigma$ is an atom then
 3:         return $\sigma = \top$ or $\sigma \in S$
 4:     else if $\sigma = \neg\varphi$ then
 5:         return $\neg$MODELS($\varphi, t_\pi, S, A$)
 6:     else                                          ▷ $\sigma$ is not an atom.
 7:         $(lhs, *, rhs) \leftarrow \sigma$
 8:         if $* = \wedge$ then
 9:             return MODELS($lhs, t_\pi, S, A$) and MODELS($rhs, t_\pi, S, A$)
10:         else if $* = \vee$ then
11:             return MODELS($lhs, t_\pi, S, A$) or MODELS($rhs, t_\pi, S, A$)
12:         else if $* = [k]$ then
13:             if $k = 0$ then
14:                 return MODELS($lhs, t_\pi, S, A$) and MODELS($rhs, t_\pi, S, A$)
15:             else
16:                 $a \leftarrow$ first action of $t_\pi$
17:                 $t'_\pi \leftarrow$ remainder $t_\pi$
18:                 $S' \leftarrow \gamma(S, a)$
19:                 if MODELS($lhs, t_\pi, S, A$) then
20:                     if not MODELS($rhs, t_\pi, S, A$) then
21:                         return MODELS($\top\ [k-1]\ rhs, t'_\pi, S', A$)
22:                     else
23:                         return $\top$
24:                 else              ▷ Did not apply to first state, need to check next.
25:                     return MODELS($\sigma, t'_\pi, S', A$)
```

capture such traces. Building on the definition of sensors and monitors, we can now formally define the notions of sensitivity and specificity of a sensor with regards to a set of traces. More specifically, a sensor is sensitive to a set of traces if the formula of the sensor is true for each trace (Definition 5).

**Definition 5 (Sensitive Sensor).** *Let $\varphi$ be an arbitrary sensor and $\mathcal{T}_\Pi = \{t_{\pi_1}, \ldots t_{\pi_n}\}$ be a set of plan traces (i.e., the sequences of states induced by plans $\pi \in \Pi$) within a planning domain $\Pi$. $\varphi$ is sensitive for the traces in $\mathcal{T}_\Pi$ iff $\forall_{t_\pi \in \mathcal{T}_\Pi}(t_\pi \models \varphi)$, i.e., $\varphi$ is a sensor for all traces in $\mathcal{T}_\Pi$.*

Conversely, we want to be able to detect when specific plans *do not* trigger a sensor, leading to the notion of a specific sensor (Definition 6).

**Definition 6 (Specific Sensor).** *Let $\varphi$ be an arbitrary sensor and $\mathcal{T}_\Pi = \{t_{\pi_1}, \ldots t_{\pi_n}\}$ and $\mathcal{T}'_\Pi = \{t_{\pi_m}, \ldots t_{\pi_k}\}$ be two sets of plan traces (i.e., the sequences of states induced by plans $\pi \in \Pi$) within a planning domain $\Pi$ such that $\mathcal{T}_\Pi \cap \mathcal{T}'_\Pi = \emptyset$ and $\mathcal{T}_\Pi \cup \mathcal{T}'_\Pi = \Pi$ (i.e., $\mathcal{T}'_\Pi$ consists of all the plans not in $\mathcal{T}_\Pi$). $\varphi$ is specific for the traces in $\mathcal{T}_\Pi$ iff $\forall_{t_\pi \in \mathcal{T}'_\Pi}(t_\pi \not\models \varphi)$, i.e., that $\varphi$ is not a sensor for any of the traces not in $\mathcal{T}_\Pi$.*

Now that we can specify sets of traces for which a sensor is sensitive and specific to, we can proceed to defining the problem of generating a monitor that approximates the sensing capabilities of an intended sensor. That is, given a specific desired sensing capability, which we call an *intended sensor*, we want to be able to synthesise a sensor from a set of *actual* available sensors that covers as much of the model sensor's traces as possible. Thus, we define the problem of synthesising a sensor to agree with a model sensor as follows (Definition 7).

**Definition 7 (Monitor Synthesis).** *Let $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ be a set of* available sensors, *$\mathcal{T}_\Pi$ and $\mathcal{T}'_\Pi$ be two set of traces such that $\sigma$ is sensitive to $\mathcal{T}_\Pi$ and specific to $\mathcal{T}'_\Pi$, and $\sigma$ be an* intended sensor *formula such that no available sensor captures exactly the traces of the intended sensor, that is: $\forall_{\varphi \in \Phi} \exists_{t_\pi \in \mathcal{T}_\Pi}(t_\pi \models \sigma) \wedge (t_\pi \not\models \varphi)$, i.e., no sensor in $\Phi$ is sensitive to the same traces as $\sigma$; or $\forall_{\varphi \in \Phi} \forall_{t_\pi \in \mathcal{T}'_\Pi}(t_\pi \not\models \sigma) \wedge (t_\pi \models \varphi)$, i.e., no sensor in $\Phi$ is specific to the same traces as $\sigma$.*

The problem of synthesising a monitor for an *intended sensor $\sigma$* consists of creating a monitor $M_{\langle \varphi_j, \ldots, \varphi_j \rangle}$ such that $\{\varphi_j, \ldots, \varphi_j\} \subseteq \Phi$; $\forall_{t_\pi \in \mathcal{T}_\Pi}(t_\pi \models M_{\langle \varphi_j, \ldots, \varphi_j \rangle})$ *iff* $(t_\pi \models \sigma)$; and $\forall_{t_\pi \in \mathcal{T}'_\Pi}(t_\pi \not\models M_{\langle \varphi_j, \ldots, \varphi_j \rangle})$ *iff* $(t_\pi \not\models \sigma)$; i.e., the monitor agrees with the intended sensor for all traces. Synthesising an intended sensor may not be possible, given restrictions on the actual sensors available.

Since many plan monitoring applications rely on the ability to detect the execution of specific actions in an environment, we need to define sensors capable of detecting them. Definition 8 formally describes how such intended sensors can be built from the action specification.

**Definition 8 (Sensor for Action).** *Let $a$ be an action of the form $\langle pre(a),$ $eff^+(a), eff^-(a)\rangle$. We say a sensor built to detect formula:*

$$\left(\left(\bigwedge_{\phi \in pre(a)} \phi\right)[1]\left(\bigwedge_{\psi \in eff^+(a)} \psi \wedge \bigwedge_{\psi \in eff^-(a)} \neg\psi\right)\right)$$

*is a sensor for $a$.*

## 3 Synthesising Monitors using Genetic Programming

We are now in a position to develop our approach to synthesising monitors using genetic programming. Consider a desired sensor $F$, and a set of traces $T$. We can partition this set of traces into two mutually exclusive sets, namely $T^t$ where for any $t \in T^t$, $t \models F$, and $T^f$ where for any $t \in T^f$, $t \not\models F$. Now given some other set of primitive sensors $\{k_1, \ldots, k_n\}$, we seek to find a formula containing these primitive sensors which partitions the traces in the same way. To generate such a formula — a candidate sensor — we must perform a search over the space of all possible sensors that can be constructed from our primitive sensors. One approach that has proven successful for performing a search over such a symbolic space is genetic programming [6], a form of evolutionary computing.

To describe the space of possible individuals within a genetic program, we must identify the terminal (leaf) nodes, as well as the form that non-terminal nodes can take. Now within the plan monitoring domain, we consider a set of *primitive sensors* consisting of formulae in the language described in Section 2. Our goal is to combine these primitive sensors in such a way so as to obtain the same inferences as some other formula, the *goal sensor*. The primitive sensors thus comprise one class of terminal nodes.

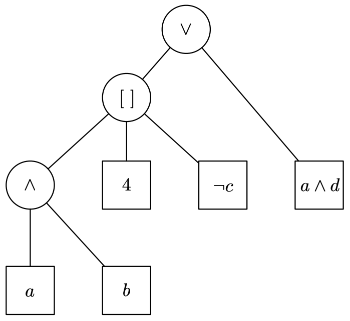

Fig. 2: An individual genetic program for the formula $((a \wedge b)[4]\neg c) \vee (a \wedge d)$ with primitive sensors $a$, $b$, $\neg c$ and $a \wedge d$.

Our logic consists of four operators — negation, conjunction, disjunction, and the path operator. Each of these forms a potential non-terminal node. We note that the path operator is a ternary operator which takes in two formulae as well as an integer. Therefore, the set of integers consists of another class of terminal nodes available within the genetic program, though this latter class of terminal nodes can only be used within a path operator. Figure 2 illustrates how the formula $((a \wedge b)[4]\neg c) \vee (a \wedge d)$ is represented as a genetic program, where $a$, $b$, $\neg c$ and $a \wedge d$ are primitive sensors.

Given an individual sensor, a set of traces, and a formula representing a target sensor, we can specify the fitness of the individual by evaluating the traces over the individual, and the target sensor, summing up the number of true positive and negative classifications of traces, and subtracting the false positive and negative trace classifications. For example, if the target sensor returns *true* for traces $t_1, t_3$ and $t_4$, and $false$ for traces $t_2$ and $t_5$, while the individual returns true for $t_1, t_2, t_3$ and $t_4$ (and false for $t_5$), the individual's fitness would be 1. We then select fit individuals reproduce them (using copy, mutation and cross-over operations [6]) to create a new generation of individuals. This process repeats until a sufficiently fit individual is found encoding the synthesised sensor.

To define the quality of a synthesised sensor, we formally define a *monitor fitness* as an *F1-Score*[5] between the traces of the intended sensor and the invisible traces, following Definition 9.

**Definition 9 (Monitor Fitness).** *Let $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ be a set of* available sensors, *$\mathcal{T}_\Pi$ and $\mathcal{T}'_\Pi$ be two set of traces, and $\sigma$ be an* intended sensor *formula such that $\mathcal{T}_\Pi \models \sigma$, $\mathcal{T}'_\Pi \not\models \sigma$. We define quality in terms of the Precision and Recall of a monitor, where Precision is $Pr = \frac{|\{t_\pi \in \mathcal{T}_\Pi | \mathcal{M}_{\Phi'} \models t_\pi\}|}{|\mathcal{T}_\Pi|+|\mathcal{T}'_\Pi|}$, and Recall is $Re = \frac{|\{t_\pi \in \mathcal{T}_\Pi | \mathcal{M}_{\Phi'} \models t_\pi\}|}{|\mathcal{T}_\Pi|}$. The* quality *of an arbitrary monitor $\mathcal{M}_{\Phi'}$ such that $\Phi' \in \mathcal{P}(\Phi)$ (i.e., $\Phi'$ is an element of the power set of the available sensors) is the harmonic mean between Precision and Recall, F1-Score, which is quality $Q(\mathcal{M}_{\Phi'}, \mathcal{T}_\Pi, \mathcal{T}'_\Pi) = 2 \cdot \frac{Pr+Re}{Pr*Re}$.*

## 4 Related Work

A related approach to ours is the work of Keren et al. [4]. In this work, the authors introduce the problem of re-designing a domain model in order to facilitate (or improve) the process of goal and plan recognition, and such problem is called goal recognition *design* [4]. Goal recognition design aims to optimize the domain design so that goal and plan recognition approaches can provide inferences with as few observations as possible [5].

Alechina et al. [1] developed an approach that considers how norms should be modified to be monitorable given an available set of imperfect monitors. In this work, the authors define that a monitor is imperfect for a norm if it does not have sufficient observational capabilities to determine if an execution trace of a multi-agent system complies with or violates a given norm.

[5] *F1-Score* is the harmonic mean between *Precision* (i.e., positive predictive value) and *Recall* (i.e., true positive rate).

## 5 Conclusions and Future Work

In this work, we have demonstrated that a genetic programming based approach to sensor synthesis can create useful sensors for detecting the execution of actions and the occurrence of specific states within planning domains.

We are currently investigating several applications and future extensions of our work. First, the output of our approach can serve as input to Bayesian goal and plan recognition algorithms [10], with the quality of the synthesised sensor serving as a prior probability for the action having taken place. Second, we can apply our work to normative domains, determining the likelihood that some obliged or prohibited state of affairs did, or did not take place. Apart from these applications, we are also investigating more complex forms of the sensor synthesis problem including creating sensors given some fixed budget. We also aim to use the the notion of planning landmarks [3] (fluents or actions that cannot be avoided to achieve a goal from an initial state) in our approach for monitoring particular states (landmarks), since it has been done successfully for recognizing goals and detecting commitment abandonment [8,9].

## References

1. Alechina, N., Dastani, M., Logan, B.: Norm approximation for imperfect monitors. In: Proc. of the 13th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 117–124 (2014)
2. Grossi, D., Aldewereld, H., Dignum, F.: Coordination, Organizations, Institutions, and Norms in Agent Systems II. chap. Ubi Lex, Ibi Poena: Designing Norm Enforcement in E-Institutions, pp. 101–114. Springer-Verlag, Berlin, Heidelberg (2007)
3. Hoffmann, J., Porteous, J., Sebastia, L.: Landmarks in Planning. Journal of Artificial Intelligence Research **22**(1), 215–278 (2004)
4. Keren, S., Gal, A., Karpas, E.: Goal Recognition Design. In: International Conference on Automated Planning and Scheduling (ICAPS) (2014)
5. Keren, S., Gal, A., Karpas, E.: Goal Recognition Design with Non-Observable Actions. In: Proc. of the 31st AAAI Conference on Artificial Intelligence (2016)
6. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992)
7. Luck, M., Mahmoud, S., Meneguzzi, F., Kollingbaum, M., Norman, T., Criado, N., Fagundes, M.: Normative Agents. In: Ossowski, S. (ed.) Agreement Technologies, pp. 209–220. Springer Netherlands (2013)
8. Pereira, R.F., Oren, N., Meneguzzi, F.: Detecting Commitment Abandonment by Monitoring Sub-Optimal Steps during Plan Execution. In: Proc. of the 16th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 1685–1687 (2017)
9. Pereira, R.F., Oren, N., Meneguzzi, F.: Landmark-Based Heuristics for Goal Recognition. In: Proc. of the 32nd AAAI Conference on Artificial Intelligence (2017)
10. Ramírez, M., Geffner, H.: Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners. In: Proc. of the 24th AAAI Conference on Artificial Intelligence (2010)
11. Shoham, Y., Tennenholtz, M.: On Social Laws for Artificial Agent Societies: Off-Line Design. Artificial Intelligence **73**(1-2), 231–252 (1995)