Sensing and Acting in the Independent Choice Logic*

David Poole†
Department of Computer Science
University of British Columbia
2366 Main Mall
Vancouver, B.C., Canada V6T 1Z4
poole@cs.ubc.ca
http://www.cs.ubc.ca/spider/poole

Abstract
This paper shows how agents that sense and
act can be represented within the independent
choice logic, a semantic framework that
allows for independent choices (made by various
agents including nature) and a logic program
that gives the consequence of choices. This
representation can be used as a (runnable) speci-
ification for agents that observe the world and
have memory, as well as a modelling tool for
dynamic environments with uncertainty. The
general idea is that an agent is a (not neces-
sarily deterministic) function from sensor readings
(observations) and remembered values into ac-
tions. Actions and observations are both rep-
resented as propositions and a logic program
specifies how actions follow from experiences.
The state of an agent is what needs to be re-
membered about the past so that actions are a
function of current observations and the state.
There is a clean semantics, the overall frame-
work it is representationally powerful, and rea-
sonably efficient code can be generated from
the agent specifications, even if generating op-
timal agents (which is well defined for the case
of a single agent in an uncertain environment)
is computationally infeasible in general.

1 Introduction

This paper is part of a project to combine logic with
probability/decision/game theory to design agents that
can act effectively in a real world (whether it be a phys-
ical world, a diagnostic/treatment world or a softworld).
The design goals of this work are:
1. To provide a decision theoretic framework that can
   be used to build agents (robots) that can be shown
to be optimal (as in (Russell et al., 1993)) — or at
   least to have a specification of the expected utility
   of an agent.
2. The output of the ‘planner’ should be suitable for
   actually controlling a robot. It has to be more than
   a sequence of steps (or even an if-then-else program)
   that is the output of traditional planners. Here we
   consider reactive agents that have internal state.
3. It should have a clean semantics, both for the object
   level plan and for the representation of the problem
   (that includes uncertainty).
4. The same language should be able to be used for
   modelling the agent and for modelling the environ-
   ment. It should not be infeasible to run the agent
   specification to implement a situated agent (that
   gets sensor readings from the world and acts in the
   world). We should also be able to run the model of
   the agent in its environment to simulate a system.
5. The representation should not force too many un-
   realistic assumptions about the environment (e.g.,
   that effects can all be anticipated; that nature is
deterministic or that only a single action can occur
at any time).

In this paper we outline a semantic framework for deci-
sion theoretic planning, and a representation for sensors
and actions. The highlights of this approach are:
1. We have a representation for multi-agent reason-
ing under uncertainty that consists of independent
choices that are made by various agents, and an
acyclic logic program that gives the consequence
of the choices. This is an extension of the strategic
form of a game (Von Neumann and Morgenstern,
1953), and allows for conditional plans (strategies).
We can write logic programs to model the world.
This is an extension of probabilistic Horn abduction
(Poole, 1993) to include multiple agents and
negation as failure.
2. Within the logic, actions are represented as pro-
positions indexed by time that means that the agents are
doing that action at that time. Nature is regarded
as a special agent — this lets us model exogenous
random events, noisy sensors, etc.
3. Agents adopt strategies. A strategy is a function
from sensor values (observations) and remembered
values into actions. These functions are represented
as acyclic logic programs.

---

*This work was supported by Institute for Robotics and
Intelligent Systems, Project IC-7 and Natural Sciences and
Engineering Research Council of Canada Operating Grant
OGPO044121.
†Scholar, Canadian Institute for Advanced Research
4. We can write logic programs (with negation as failure) to model agents and the environments. It may seem as though this is too weak a logic, as we cannot represent disjunction. Disjunction is a form of uncertainty — we have a very powerful mechanism for modelling uncertainty using the independent choices that renders disjunction unnecessary (this is a hypothesis we are testing anyway).

5. This framework is a representation for decision theory that generalises Bayesian networks (Poole, 1993), influence diagrams and the strategic (normal) form of a game (and so also the extensive form of a game).

6. While we can use either discrete time or continuous time, in this paper we use a discrete representation of time. See (Poole, 1995b) for a description of continuous time in this framework where the output can be a continuous function of the input (and remembered events). (Poole, 1995b) shows how ‘events’ can be derived from continuous time, how remembered events can be used to record intentions, and how accumulation (integration over time) and differentiation over time can be modelled.

7. In order to highlight conditional actions, and information producing actions (which are not different sorts of actions here), we show how to represent the widget example of (Draper et al., 1994).

1.1 Agents

An agent is something that acts in the world. An agent can, for example, be a person, a robot, a worm, the wind, gravity, a lamp, etc. Purposive agents have preferences, they prefer some states of the world to other states, and act in order to (try to) achieve worlds they prefer. The non-purposive agents are grouped together and called “nature”. Whether an agent is purposive or not is a modelling assumption that may or may not be appropriate. For example, for some applications it may be appropriate to model a dog as purposive, and for others it may suffice to model a dog as non-purposive.

A policy or strategy is a specification of what an agent will do under various contingencies. A plan is a policy that includes either time or the stage of the plan as part of the contingencies conditioned on.

Note that beliefs, desires, intentions, commitments etc., (Shoham, 1993) are not essential to agenthood. It may, however, be the case that agents with beliefs, desires, intentions etc (that, for example, communicate by way of speech acts (Shoham, 1993)) perform better (by some measure) than those that do not. We don’t want to define agenthood to exclude the possibility of formulating and testing this empirical claim.

Our aim is to provide a representation in which we can define perception, actions and preferences for agents. This can be used for to define a policy, the notion of when one policy is better (according to that agent’s preferences) and so what is an optimal policy for an agent. Once we have defined what an ‘optimal’ agent is, we can use exact and approximation algorithms to build policies for agents to use in the world.

Agents can have sensors, (possibly limited) memory, computational capabilities and effectors. Agents reason and act in time.

An agent should react to the world — it has to condition its actions on what is received by its sensors. These sensors may or may not reflect what is true in the world. We have to be able to consider sensors that may be noisy, unreliable or broken and we also need to consider ambiguity (about the world) from sensors. We condition on what we know, even if it is very weak such as “sensor a appears to be outputting value v”. Similarly actuators may be noisy, unreliable, slow or broken. What we can control is what message (command) we send to our actuators.

In this paper we provide a representation that can be used to model the world, agents (including available sensors and actuators) and goals (in terms of the agents utilities in different situations) that will allow us to design optimal (or approximately optimal) agents.

1.2 Game Theory

Game theory (von Neumann and Morgenstern, 1953; Fudenberg and Tirole, 1992) is a general theory of multi-agent reasoning under uncertainty. The general idea is that there is a set of players (agents) who make moves (take actions) based on what they observe. The agents each try to do as well as they can (maximize their utility).

Game theory is intended to be a general theory of economic behaviour (von Neumann and Morgenstern, 1953) that is a generalization of decision theory. The use of the term ‘game’ here is much richer than typically studied in AI textbooks for ‘parlour games’ such as chess. These could be described as deterministic (there are no chance moves by nature), perfect information (each player knows the previous moves of the other players), zero-sum (one player can only win by making the other player lose), two-player games. Each of these assumptions can be lifted (von Neumann and Morgenstern, 1953).

A game is a sequence of moves taken sequentially or concurrently by a finite set of agents. Nature is usually treated as a special agent. There are two main (essentially equivalent in power (Fudenberg and Tirole, 1992)) representations of games, namely the extensive form and the normalized (von Neumann and Morgenstern, 1953) (or strategic (Fudenberg and Tirole, 1992)) form of a game.

The extensive form of a game is in terms of a tree; each node belongs to an agent, and the arcs from a node correspond to all of the possible moves (actions) of that agent. A branch from the root to a leaf corresponds to a (possible) play of the game. Information availability is represented in terms of information sets which are sets of nodes that an agent cannot distinguish. The aim is for each agent to choose a move (action) at each of the information sets.

\(^\text{Footnote}\) Of course there is no correlation between what a sensor reading tells us and what is true in the world, and the utility depends on what is true in the world (as it usually does), then we may as well ignore the sensor.
In the strategic form of a game each player adopts a strategy, where a strategy is “a plan ... which specifies what choices [an agent] will make in every possible situation” (Von Neumann and Morgenstern, 1953, p. 79). This is represented as a function from information available to the agent’s move. The framework below should be seen as a representation based on the normalized (Von Neumann and Morgenstern, 1953) (or strategic (Fudenberg and Tirole, 1992)) form of a game, with a possible world corresponding to a complete play of a game. We have added a logic program to give the consequences of the play. This allows us to use a logical representation for the world and for agents.

Where there are agents with competing interests, the best strategy is to often a randomized strategy. In these cases the agent decides to randomly choose actions based on some probability distribution.

2 The Independent Choice Logic

The independent Choice Logic (ICL) specifies a way to build possible worlds. Possible worlds are built from choosing propositions from sets of independent choice sets, and then extending these “total choices” with a logic program.

There are two languages we will use: \( \mathcal{L}_F \) of facts which for this paper we consider to be the language of acyclic logic programs that can include negation as failure (Apt and Bezem, 1991), and the language \( \mathcal{L}_Q \) of queries which we take to be arbitrary propositional formulae (the propositions corresponding to ground formulae of the language \( \mathcal{L}_F \)). We write \( f \models q \) where \( f \in \mathcal{L}_F \) and \( q \in \mathcal{L}_Q \) if \( q \) is true in the unique stable model of \( f \) or, equivalently, if \( q \) follows from Clark’s completion of \( q \) (the uniqueness of the stable model and the equivalence for acyclic programs are proved in (Apt and Bezem, 1991)). See (Poole, 1995a) for a detailed analysis of negation as failure in this framework, and for an inductive characterisation of the logic.

An independent choice logic theory is a tuple \( \langle \mathcal{C}, \mathcal{F}, \mathcal{A}, \text{controller}, P_0 \rangle \) where \( \mathcal{C} \) called the choice space, is a set of sets of ground atomic formulae, such that if \( \chi_1, \chi_2 \subseteq \mathcal{C} \) and \( \chi_1 \neq \chi_2 \) then \( \chi_1 \cap \chi_2 = \emptyset \). An element of \( \chi \) is called an alternative. An element of an alternative is a called a atomic choice. An atomic choice can appear in at most one alternative.

\( \mathcal{F} \) called the facts, is an acyclic logic program such that no atomic choice unifies with the head of any rule.

\( \mathcal{A} \) is a finite set of agents. There is a distinguished agent called ‘nature’.

controller is a function from \( \mathcal{C} \to \mathcal{A} \). If \( \text{controller}(\chi) = a \) then agent \( a \) is said to control alternative \( \chi \). If \( \chi \) is an agent the alternatives controlled by \( a \) is given by \( \text{controls}(a) = \{ \chi \in \mathcal{C} : \text{controller}(\chi) = a \} \).

\( P_0 \) is a function \( \mathcal{F} \to [0, 1] \) such that for \( \chi \in \mathcal{C} \) if \( \text{controller}(\chi) = 0 \) then \( \sum_{a \in \mathcal{A}} P_0(a) = 1 \). I.e., \( P_0 \) is a probability measure over the alternatives controlled by nature.

The independent choice logic specifies a particular semantic construction. The semantics is defined in terms of possible worlds. There is a possible world for each selection of one element from each alternative. What follows from these atoms together with \( \mathcal{F} \) are true in this possible world.

Definition 2.1 Given independent choice framework theory \( \langle \mathcal{C}, \mathcal{F} \rangle \), a selector function is a mapping \( \tau : \mathcal{C} \to \bigcup \mathcal{C} \) such that \( \tau(\chi) \in \chi \) for all \( \chi \in \mathcal{C} \). The range of selector function, written \( \mathcal{R}(\tau) \), is the set \( \{ \tau(\chi) : \chi \in \mathcal{C} \} \).

Definition 2.2 For each selector function \( \tau \) there is a possible world \( w_\tau \). If \( f \) is a formula in language \( \mathcal{L}_Q \), and \( w_\tau \) is a possible world, we write \( w_\tau \models f \) (read \( f \) is true in possible world \( w_\tau \)) if \( \mathcal{F} \cup \mathcal{R}(\tau) \models f \).

The uniqueness of the model follows from the acyclicity of the logic program (Apt and Bezem, 1991).

An independent choice logic theory is utility complete if for each agent \( a \in \mathcal{A} \) such that \( a \neq 0 \) and for each possible world \( w_\tau \) there is a unique number \( u \) such that \( w_\tau \models \text{utility}(a, u) \). The logic program will have rules for \( \text{utility}(a, u) \).

Definition 2.3 If \( \langle \mathcal{C}, \mathcal{F}, \mathcal{A}, \text{controller}, P_0 \rangle \) is an ICL theory and \( a \in \mathcal{A}, a \neq 0 \), then a strategy for agent \( a \) is a function \( P_a : \mathcal{F} \to [0, 1] \) such that

\[ \forall \chi \text{ if } \text{controller}(\chi) = a \text{ then } \sum_{a \in \mathcal{A}} P_a(a) = 1. \]

In other words, strategy \( P_a \) is a probability measure over the alternatives controlled by agent \( a \).

Definition 2.4 A composite choice on \( \kappa \subseteq \mathcal{C} \) is a set consisting of exactly one element (atomic choice) from each \( \chi \in \kappa \).

Definition 2.5 A pure strategy for agent \( a \) is a strategy for agent \( a \) such that the range of \( P_a \) is \( \{ 0, 1 \} \). In other words, \( P_a \) selects a member of each element of \( \text{controls}(a) \) to have probability 1, and the other members thus have probability 0. A pure strategy for agent \( a \) thus corresponds to a composite choice on \( \text{controls}(a) \).

Definition 2.6 A strategy is a function from agents (other than nature) into strategies for the agents. If \( \sigma \) is a strategy and \( a \in \mathcal{A}, a \neq 0 \) then \( \sigma(a) \) is a strategy for agent \( a \). We write \( \sigma(a) \) as \( P_\sigma^a \) to emphasise that \( \sigma \) induces a probability over the alternatives controlled by agent \( a \). [We also define \( P_\sigma^a = P_0 \).]

Definition 2.7 If ICL theory \( \langle \mathcal{C}, \mathcal{F}, \mathcal{A}, \text{controller}, P_0 \rangle \) is utility complete, and \( \sigma \) is a strategy, then the expected utility for agent \( a \neq 0 \), under strategy \( \sigma \) is

\[ \varepsilon(a, \sigma) = \sum_{\tau} p(\sigma, \tau) \times u(\tau,a) \]

(summing over all selector functions \( \tau \) where

\[ u(\tau,a) = u \text{ if } w_\tau \models \text{utility}(a,u) \]

(this is well defined as the theory is utility complete), and

\[ p(\sigma, \tau) = \prod_{\chi \in \mathcal{C}} P_{\text{controller}(\chi)}(\tau(\chi)). \]

\( p(\sigma, \tau) \) is the probability of world \( \tau \) under strategy \( \sigma \), and \( u(\tau,a) \) is the utility of world \( w_\tau \) for agent \( a \).
Given this semantic structure we can mirror the definitions of game theory (Von Neumann and Morgenstern, 1953; Fudenberg and Tirole, 1992). For example, we can define the Nash equilibrium and Pareto optimal as follows:

**Definition 2.8** Given utility complete ICL theory \( (\mathcal{C}, \mathcal{F}, A, \text{controller}, P_0) \), strategy \( \sigma \) is a Nash Equilibrium if no agent can increase its utility by unilaterally deviating from \( \sigma \). Formally, \( \sigma \) is a Nash equilibrium if for all agents \( a \in A \), if \( \sigma_a \) is a strategy such that \( \varepsilon(a', \sigma_a) = \varepsilon(a, \sigma') \) for all \( a' \neq a \) then \( \varepsilon(a, \sigma_a) \leq \varepsilon(a', \sigma) \). \( \sigma \) here is a strategy that is the same as strategy \( \sigma \) for all agents other than \( a \).

One of the great results of game theory is that every finite game has at least one Nash equilibrium (we may need non-pure (randomised) strategies) (Fudenberg and Tirole, 1992). For a single agent in an uncertain environment, a Nash equilibrium is an optimal decision theoretic strategy.

**Definition 2.9** Given utility complete ICL theory \( (\mathcal{C}, \mathcal{F}, A, \text{controller}, P_0) \), strategy \( \sigma \) is Pareto optimal if no agent can do better without some other agents doing worse. Formally, \( \sigma \) is Pareto optimal if for all strategies \( \sigma' \), if there is some agent \( a \in A \) such that \( \varepsilon(a, \sigma') > \varepsilon(a, \sigma) \) there is some agent \( a' \in A \) such that \( \varepsilon(a', \sigma') < \varepsilon(a', \sigma) \).

Other definitions from game theory can also be given in the logic of this paper. What we are adding to game theory is the use of a logic program to model the agents and the environment, and to provide a way to express independence (in the same way that probabilistic Horn abduction (Poole, 1993) can be used to represent the independence assumptions of Bayesian networks (Pearl, 1988)).

### 3 Agent Specification Module

So far we have modelled agents by naming them and specifying which choices they control. It helps to do more than this, we want to provide some structure that makes it easy to model actual agents. Not every logic program and set of assignments of agents to choices will make sense. Agents have inputs and outputs; they have some values that they cannot see, and some internal values that only they can see. We model them by giving a logic program that gives the relationship between the inputs and outputs. This logic program can use the internal values and sense values but cannot use these values the agent has no access to (i.e., cannot sense or otherwise determine).

Agents specification modules will not give any extra power to the formal framework set up. It will, however, allow us to modularise our knowledge, and use common computer science techniques like information hiding, abstract data types and modular program design.

A fluent is a function that depends on time. Each fluent has an associated set called the range of the fluent. A propositional fluent is a fluent with range \{true, false\}. Syntactically a fluent it a term in our language.

**Definition 3.1** An agent specification module for agent \( A \) is a tuple \( (\mathcal{C}_A, I_A, O_A, F_A) \) where \( \mathcal{C}_A \) is the set of alternatives controlled by \( A \). When \( A \) is nature we also include \( P_0 \) as part of the agent specification module.

\( I_A \) is a set of fluents, called the inputs, that the agent can sense. Atom sense \((\text{Fl}, \text{Val}, T)\) is true if input fluent \( \text{Fl} \) has value \( \text{Val} \) at time \( T \).

\( O \) is a set of propositional fluents called the outputs that specify actuator settings or action attempts at various times. Atom do(\( \text{Act}, T \)) is true if the agent is attempting to ‘do’ action \( \text{Act} \) at time \( T \).

\( F_A \) is an acyclic logic program. \( F_A \) specifies how the outputs are implied by the inputs, the local controllables \( (\mathcal{C}_A) \), and other (local) relations as interdependencies. Often it is useful to distinguish the propositions whose value will be referred to in the future (these form the ‘state’ of the agent).

Nature’s module will be the dual of other modules. The outputs of nature’s module will be the input of other agent’s module, and the input of nature’s module will be the output of other agents.

For each agent we axiomatise how it can “react” to the environment, perhaps depending on some remembered values.

The sensors that we consider are passive sensors that (at each time) receive a value from the environment. We also do not distinguish between information-producing actions and actions that ‘change the world’ — there is only one type of action. The nature module will specify the consequence of doing an action.

We can model ‘information-producing actions’ by having actions whose effect to make a sensor have a value that correlates with some value in the world. For example, the information producing action ‘look’ may affect what is sensed by the eyes; if the agent doesn’t ‘look’ they will sense the value ‘nothing’, if they do look (in a certain direction) they may sense what is in that direction. Of course the ‘look’ action may be unreliable (the lights may be out), and it may take an arbitrary amount of time to achieve its effect (as in a medical test).

What is also important is that the agent can only condition on its sense values or on values derived from these — the agent cannot condition on what it has no access to (e.g., the true state of the world). Similarly, the agent can only control what message is sent to its actuators — what it actually does may be quite different.

N.B. we do not distinguish between the ‘environment’ and the ‘plant’. These are grouped together as the ‘environment’ here.

### 4 The Widget Example

In this section we present the example of (Draper et al., 1994). The example is that of a robot that must process a widget. Its goal is to have the widget painted and processed and then to notify its supervisor that it is done. Processing consists of rejecting flawed widgets and shipping unflawed widgets. The robot can inspect the widget to see if it is blemished, which initially correlates with the widget being flawed. Painting the widget
usually results in the widget being painted but removes blemishes.

**AGENT MODULE** We first represent the agent. The agent has one sensor for detecting blemishes. It has 6 actions (one of which is possible at any time).

Input: $s\{blemished, Val, T\}$

Output: $do(\text{reject}, T), do(\text{ship}, T), do(\text{notify}, T), do(\text{paint}, T), do(\text{inspect}, T), do(\text{nothing}, T)$. To handle this example, the agent needs to be able to remember whether the widget is ok or bad. The simplest way to do this is to let it believe the widget is OK until it senses that it is bad:

\[
\begin{align*}
\text{bel}(ok, 0) & \leftarrow \text{true}. \\
\text{bel}(ok, T + 1) & \leftarrow \\
\text{bel}(ok, T) & \wedge \\
\sim \text{sense}(\text{blemish, bad}, T + 1).
\end{align*}
\]

For many of the actions the agent can just choose to do them. The agent can also choose to reject or ship depending on the sensor value: for each time $T$, \{$do(\text{notify}, T), do(\text{paint}, T), do(\text{inspect}, T), do(\text{nothing}, T), rejectORship(T)\} \in C_A$.

The way to have actions depend on sense values is to write rules that imply what the agent will do under various contingencies. The agent can decide whether to reject or ship a widget (or do one of the other actions) depending on its belief about the widget:

\[
\begin{align*}
do(\text{reject}, T) & \leftarrow \\
\text{rejectORship}(T) & \wedge \\
\text{rejectifOK}(T) & \wedge \\
\text{bel}(ok, T).
\end{align*}
\]

\[
\begin{align*}
do(\text{reject}, T) & \leftarrow \\
\text{rejectORship}(T) & \wedge \\
\text{rejectifBAD}(T) & \wedge \\
\sim \text{bel}(ok, T).
\end{align*}
\]

\[
\begin{align*}
do(\text{ship}, T) & \leftarrow \\
\text{rejectORship}(T) & \wedge \\
\text{shipifOK}(T) & \wedge \\
\text{bel}(ok, T).
\end{align*}
\]

\[
\begin{align*}
do(\text{ship}, T) & \leftarrow \\
\text{rejectORship}(T) & \wedge \\
\text{shipifBAD}(T) & \wedge \\
\sim \text{bel}(ok, T).
\end{align*}
\]

What to do under the various sensing situations is represented as alternatives controlled by the robot: $\forall T \\{\text{rejectifOK}(T), \text{shipifOK}(T)\} \in C_A$, and $\{\text{rejectifBAD}(T), \text{shipifBAD}(T)\} \in C_A$.

**NATURE MODULE**: To represent nature’s module, we axiomatise how the agents actions affect the world, how the world affects the senses of the agent.

The widget being painted persists in the world. Painting the widget can result in the widget being painted (with probability 0.95). We assume that whether painting works does not depend on the time (a second painting will not make the widget more likely to be painted). Painting only works if it has not already been shipped or rejected — this disallows the plan to ship or reject then paint, which is a simpler plan as the agent doesn’t need to remember anything to execute it.

\[
\begin{align*}
painted(T + 1) & \leftarrow \\
\text{do(paint, T)} & \wedge \\
paint\text{works} & \wedge \\
\sim \text{shipped}(T) & \wedge \\
\sim \text{rejected}(T).
\end{align*}
\]

\[
\begin{align*}
painted(T + 1) & \leftarrow \\
painted(T).
\end{align*}
\]

Painting succeeds 95% of the time when it can:

\[
\{\text{paint\_works, paint\_fails}\} \in C_0
\]

$P_0(\text{paint\_works}) = 0.95$, $P_0(\text{paint\_fails}) = 0.05$

The widget is blemished if and only if it is flawed and not painted:

\[
\begin{align*}
\text{blemished}(T) & \leftarrow \\
\text{flawed}(T) & \wedge \\
\sim \text{painted}(T).
\end{align*}
\]

Whether the widget is flawed or not persists:

\[
\text{flawed}(T + 1) \leftarrow \text{flawed}(T).
\]

The widget is processed if it is rejected and flawed or shipped and not flawed:

\[
\begin{align*}
\text{processed}(T) & \leftarrow \\
\sim \text{rejected}(T) & \wedge \\
\text{flawed}(T).
\end{align*}
\]

\[
\begin{align*}
\text{processed}(T) & \leftarrow \\
\text{shipped}(T) & \wedge \\
\sim \text{flawed}(T).
\end{align*}
\]

The widget is shipped if the robot ships it, and being shipped persists:

\[
\begin{align*}
\text{shipped}(T) & \leftarrow \text{do(ship, T)}.
\end{align*}
\]

\[
\begin{align*}
\text{shipped}(T + 1) & \leftarrow \text{shipped}(T).
\end{align*}
\]

The widget is rejected if the robot rejects it, and being rejected persists:

\[
\begin{align*}
\text{rejected}(T) & \leftarrow \text{do(reject, T)}.
\end{align*}
\]

\[
\begin{align*}
\text{rejected}(T + 1) & \leftarrow \text{rejected}(T).
\end{align*}
\]

We axiomatise how what the robot senses is affected by the robot’s actions and the world:

\[
\begin{align*}
\text{sense}(\text{blemish, bad}, T + 1) & \leftarrow \\
\text{do(inspect, T)} & \wedge \\
\text{blemished}(T) & \wedge \\
\sim \text{falsepos}(T).
\end{align*}
\]

\footnote{Many other representations are possible. What is important is that the agent must actually remember some proposition to be able to use it in the future (it must be able to recall whether it thinks the widget is OK or bad, so this information can be used after it has painted the widget). There are other axiomatisations where what it remembers is a decision to be made (and so can be optimised over).}
The sensor gives a false positive with probability 0.1. Unlike whether painting succeeds, we specify here that the probability of a false positive at each time is independent of what happens at other times:

\[
\{\text{falsepos}(T), \text{notfalsepos}(T)\} \in C_0
\]

\[
P_0(\text{falsepos}(T)) = 0.1, P_0(\text{notfalsepos}(T)) = 0.9
\]

30% of widgets are initially flawed:

\[
\{[\text{flawed}(0), \text{unflawed}(0)] \in C_0
\]

\[
P_0(\text{flawed}(0)) = 0.3, P_0(\text{unflawed}(0)) = 0.7
\]

Finally, we specify how the utility is dependent on the world and actions of the agent. The utility is one if the widget is painted and processed the first time the agent notifies, and is zero otherwise.

\[
\text{utility}(\text{robot}, 1) \leftarrow
\]

\[
do(\text{notify}, T) \land
\]

\[
\neg \text{notified before}(T) \land
\]

\[
\text{paint}(T) \land
\]

\[
\text{processed}(T).
\]

\[
\text{utility}(\text{robot}, 0) \leftarrow \neg \text{utility}(\text{robot}, 1).
\]

\[
\neg \text{notified before}(T) \land T < T \land \neg \text{do(notify), T}.
\]

One policy for our agent is: \{do(inspect, 0), do(paint, 1), rejectORship(2), shiipOK(2), rejectIFBAD(2), do(notify, 3)\}. This has expected utility 0.925.

This policy is not optimal. Policy: \{do(inspect, 0), do(paint, 1), rejectORship(3), shiipOK(3), rejectIFBAD(3), do(notify, 4)\} has expected utility 0.94715. There is no optimal policy for this example (it is not a finite game so Nash’s theorem does not apply here), we can add more inspectors to keep raising the expected utility.

The policy without inspecting, \{do(paint, 0), rejectORship(1), shiipOK(1), do(notify, 2)\} has expected utility 0.665.

Of course we can always define the utility so that the agent is penalised for taking too much time, e.g., by making the head of the first utility clause:

\[
\text{utility}(\text{robot}, 1 - T/10) \rightarrow \ldots
\]

and something appropriate for the second clause.

Under the revised utility, the first policy above is optimal, with expected utility 0.625.

5 Conclusion

This paper has only scratched the surface of the issues.

The current action representation (and its continuous counterpart (Poole, 1995b)) is simple yet surprisingly general. For example, it can represent concurrent actions (see (Poole and Kanazawa, 1994)), and can represent examples in the range from traditional planning domains such as the block worlds (Poole and Kanazawa, 1994) to continuous domains like controlling a non-constrictive maze travelling vehicle (Poole, 1995b).

We are developing this framework to include discovering what to remember (making remembering a proposition a choice that has a cost associated with it), and discovering what to condition on (not hard wiring the last two as was done here). Allowing what to condition on as a choice means expanding the presentation slightly to let a policy for an agent be an implication from sensor values and remembered values to actions.

Conspicuous by its absence in this paper is a discussion on computation. This can mean three things: (1) building a situated agent that embodies a policy (2) simulating a policy and environment or (3) finding an optimal policy. Only the second has been implemented for the example here. In implementing an agent, we can exploit the fact that all of the queries will refer to a progression of this type (see (Poole, 1995b)). There is much more to be done here. Various parts of this project have been implemented. See my WWW site for details.

References


