Formalizing Properties of Agents

Richard Goodwin
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, U.S.A. 15213
rich@cs.cmu.edu

September 12, 1994

Abstract

There is a wide gulf between the formal logics used by logicians to describe agents and the informal vocabulary used by people who actually build robotic agents. In an effort to help bridge the gap, this paper applies techniques borrowed from the field of formal software methods to develop a common vocabulary. Terms useful for discussing agents are given formal definitions. A framework for describing agents, tasks and environments is developed using the Z specification language. The terms successful, capable, perceptive, predictive, interpretive, rational and sound are then defined relative to this framework. The aims of this paper are to develop a precise, common vocabulary for discussing agents and to provide a basis for rational design of agents.

Motivation

In the artificial intelligence community, it has become common to talk about agents that perform tasks and to describe such agents in terms of characteristics that would allow them to be successful. Common terms include; successful, capable, and perceptive. Each of these terms has an intuitive meaning that allows for informal discussion of the suitability of an agent for a task. The problem with these informal definitions is that they are often ambiguous and incomplete. What are needed are more precise definitions that will help to establish a common, uniform vocabulary for talking about agents, environments and tasks.

This paper formally defines some useful terms for discussing agents. First, a general framework for describing agents, tasks and environments is developed using the Z specification language [9]. This framework is useful for analyzing agents and communicating ideas, but is not intended to suggest representations or algorithms for implementing agents. Through developing the framework, a number of key issues are identified. Agent properties are then defined in relation to this general framework.

Overview

An agent is an entity that performs some task or set of tasks. Any property of an agent must therefore be defined in terms of the task and the environment in which the task is to be performed. The most basic question that can be asked is whether the agent achieves the task. Is the agent successful? Once there is a definition for success, it can be used to define other properties such as capable and perceptive.

The operational problem is how to determine if an agent is successful. The approach adopted here is to take the lead of the behavioural psychologists and make observable behaviour be the only criteria allowed for determining success [8]. Using a behaviourist approach limits the types of tasks that can be considered. Any task where success depends on inferring or interpreting the agent's internal state are ruled out. Tasks of this type would have to be modified so that any internal state of interest is manifested as external behaviour. For example, having a robot determine whether it is raining is not observable. However, having the robot open an umbrella if it is raining is observable.

Since success is the fundamental property for characterizing agents, some appropriate scale is needed for measuring success. The simplest scale that can be used is binary: An agent is either successful or not. A binary scale for success leads to binary definitions of other properties. The scale for measuring success can be refined to give a numeric value to the quality of task achievement. In economics, such a numeric measure of quality corresponds to a utility function defined over the possible task completions [7]. Further, the likelihood of possible initial conditions and external influences can be taken into account to give a measure of the expected success of the agent. Refining the measure of success allows corresponding refinements in agent properties defined in terms of success.

Although a behaviourist approach is being adopted for measuring success, it is sometimes useful to consider the internal organization of the agent and the extent to which that organization leads to behaviour that results in success. A particularly interesting class of agents are those that maintain some internal model of the world and reason about the results of their actions. Such *deliberative* agents choose actions that are more likely to lead to success based on their world models. Additional properties, for deliberative agents, relate to the accuracy of the agent's model and whether the agent behaves in a manner consistent with its model.

Informal Definitions

Successful :	An agent is successful to the extent that it accomplishes the specified task.
Capable :	An agent is capable if it possesses the effectors needed to accomplish the task.
Perceptive :	An agent is perceptive if it can distinguish salient characteristics of the world that would allow it to use its effectors to achieve the task.

Figure 1: **General Agent Properties**

The informal definitions given in figures 1 and 2 are derived from the set of characteristics

Predictive :	An agent is predictive if its model of how the world works is sufficiently accurate to allow it to correctly predict how it can and cannot achieve the task.
Interpretive :	An agent is interpretive if can correctly interpret its sensor readings.
Rational :	An agent is rational if it chooses to perform commands that it predicts will achieve its task.
Sound :	An agent is sound if it is predictive, interpretive and rational.

Figure 2: **Deliberative Agent Properties**

that Mitchell believes are necessary for a successful agent [6] [1]. These informal definitions provide the intuitive meaning for the agent properties that will be more precisely defined in the rest of this paper.

The formal property definitions presented in this paper are a subset of those defined in the original formal specification, which also includes definitions of reactive and reflexive. This subset was chosen since it forms an interesting and cohesive subset. There are also places in this text where details of the original specification have been suppressed to increase readability. The full text of the specification is presented in [4].

Issues

Before delving into the details of the definitions, we present a set of issues that any specification in this area must address. Then, with these issues in mind, we will present the approach taken here.

Any formal definition of agent properties must include a framework for describing an agent, a task and an environment. To say that an agent is successful at a particular task in a particular environment, some method is needed for specifying the agent, the task and the environment. The framework for these descriptions must be sufficiently detailed to allow the distinction between successful and unsuccessful agents to be made. At the same time, the framework must be general enough to permit a wide range of agents, tasks and environments to be specified. In designing the framework, the range of tasks and environments needs to be considered. The kinds of tasks can include tasks of achievement, tasks of maintenance and tasks with deadlines. The environment can be static or dynamic, can include other agents and can be non-deterministic. In addition, we also have to consider how to represent the interaction of the agent with the environment over time.

Clearly, in order to compare agents, we need to distinguish the agent from the environment. It is not obvious though, which sub-parts of the agent should be distinguished. One extreme possibility is to regard the agent as a black box. This is sufficient for defining some properties of agents, but not others. In particular, a black box view prevents describing the suitability of the agent's effectors or sensors for a task separately from the entire agent. The distinctions

[1]Contains references to *Reactive, perceptive and correct*. Some additional terms were described through personal communication.

that should be made are those that are useful for comparing and designing agents. Distinctions should also be sufficiently general to apply to a wide range of agent architectures.

Given distinctions between the environment and the agent, and between the sub-parts of the agent, a method is needed to describe how these components interact. The agent affects the environment through its actions and is affected by changes in the environment. Sub-parts of the agent interact either physically or by passing information. These interactions should be characterized to the level of detail necessary to make useful comparisons between behaviour.

When performing a task, the interactions between the agent and the environment occur over a period of time. So, in addition to representing the form of the interaction, the evolution of the interaction over time must also be represented. Time can either be modeled as a continuous quantity or as discrete intervals. Of course, as the size of each time interval shrinks, a discrete representation better approximates continuous time.

Finally, we need to consider whether our model of the environment will include non-determinism. We could adopt the Newtonian view that the world is deterministic, if only we know its state in infinite detail. Any apparent non-determinism is the result of our limited information about the current state and our limited ability to compute future consequences. Neglecting these limits, the environment may still be non-deterministic if we allow for the free will of other agents. Either we can distinguish other agents from the environment and model them as non-deterministic entities or leave them as part of the environment and include non-determinism in the environment. If our aim were to define properties of co-operating agents, then the distinction between other agents and the environment is needed. Since this is not our objective, the distinction will not be made in order to simplify the framework.

Framework

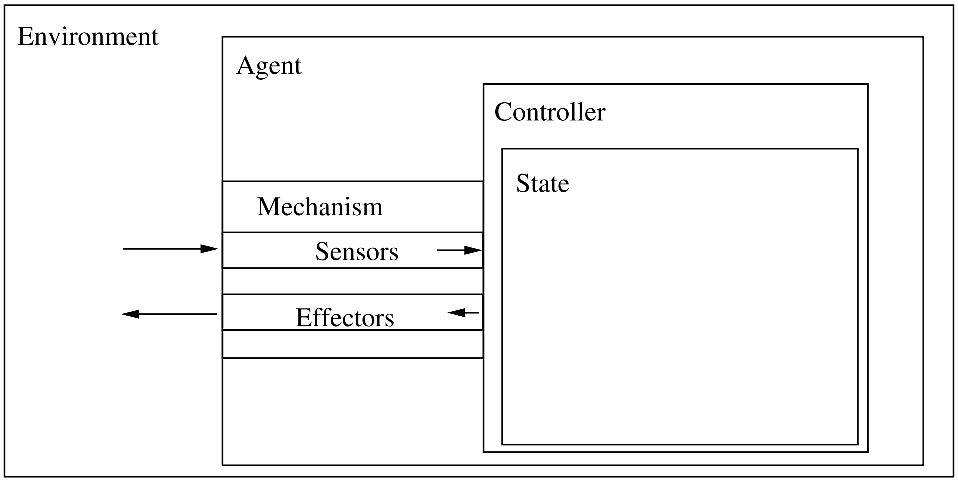

Figure 3: **Agent System Overview**

The general framework adopted is shown in figure 3. We make the necessary distinction between the agent and the environment in which it is situated. Within the agent, we further distinguish the mechanism from the controller. The mechanism consists of the sensors and effectors that allow the agent to interact with the environment. The controller is, conceptually at least, the component that accepts input from the sensors and controls the effectors. The controller may maintain some state (memory). The details of the nature of the environment, the structure and implementation of the agent and the agent's control have been abstracted away.

The model is intended to apply to systems as diverse as autonomous robots, knowbots[2] and thermostats. The agent may have a physical presence or just exist within the memory of a computer. The control component could be implemented in software or hardware. By refining this abstract model, particular classes of agents can be specified.

This formal specification first defines the basic data types, an environment, a task and an agent. These components are then used to define a general agent system. Binary definitions of agent properties are given in terms of this general agent system. Deliberative agents are then described and additional properties related to their world model are given. Following this, we add a preference function over possible task completions, and a probability distribution over possible initial states. This allows the expected performance of agents to be defined.

The formal aspects of our definitions are presented using the Z specification language. This language was chosen because it provides a rich mathematical syntax with well defined semantics.

Primitive Data Types

In this section, the primitive data types used to describe agents, environments and tasks are presented and discussed. These data types are used to describe the state of each component and to describe interactions between components. In addition, we define the term *Chronicle* which is used to represent how components of the agent and environment change over time.

External Descriptions

The externally observable state consists of the state of the environment and the externally observable state of the agent. Each is described by a primitive data type.

$[WORLD, CONFIGURATION]$

A WORLD is a complete, instantaneous description of the environment. It includes the state of the environment as well the instantaneous rates of change of that state. It is like a very detailed snapshot of the environment.

A CONFIGURATION refers both to the composition of the agent's "body" and its current position. For instance, a CONFIGURATION specifies how many arms the agent has and the position of each arm. Like a WORLD, a CONFIGURATION is an instantaneous description that includes the instantaneous rates of change of the state.

An ExternalState is a complete description of the external state of a system consisting of a WORLD and a CONFIGURATION.

[2] A knowbot is an agent that exists as a program running on a computer or computer network. It performs tasks such as information retrieval from databases.

ExternalState
world : *WORLD*
configuration : *CONFIGURATION*

It is impossible to completely specify the external state of the system in exact detail. It is also desirable to abstract away irrelevant details. An abstract description of the external state is represented by the set of all ExternalStates that are consistent with the partial description. A predicate such as *On(A,B)* is equivalent to the set of WORLDs where the predicate holds. An ExternalDescription is defined to be a set of ExternalStates that represents an abstract description of the external state.

$ExternalDescription == \mathbb{P}\ ExternalState$

Internal State

The controller within the agent may maintain some information in its memory. A STATE is defined to be a complete description of the internal state of the agent's controller. It represents the contents of the agent's memory. Note that since it is a primitive data type, the internal structure of the agent's state cannot be reasoned about.

$[STATE]$

Commands and Interactions

Interactions between the agent's controller and its effectors and between the agent's effectors and the environment are mediated by COMMANDs and INTERACTIONS, respectively.

$[COMMAND, INTERACTION]$

A COMMAND is a signal that the agent's control component can send to the agent's mechanism. The results of sending a particular COMMAND signal will depend on the CONFIGURATION of the Mechanism and WORLD in which it is sent. A COMMAND used in this sense is not the same as what is meant by an action in classical Artificial Intelligence planning. A COMMAND is not a high level action like *putOn(A,B)*. It is more like a setting of the switches that control the agent. A single configuration of the switches may cause the robot to begin moving forward and begin opening its gripper.

To get a better idea of what a COMMAND is, imagine yourself as the controller inside the agent. In front of you is a control panel with some buttons. You control the agent by pushing combinations of these buttons. Each COMMAND corresponds to one combination of buttons you could push.

An INTERACTION is the influence the Mechanism exerts on the environment as the result of acting on a particular COMMAND signal. The exact INTERACTION depends on the Mechanism, the COMMAND signal and the CONFIGURATION of the Mechanism.

The advantage of having separate COMMANDs and INTERACTIONS is that it allows for the distinction between the signals sent to the mechanism and the effects the mechanism has on the environment. A COMMAND may turn on the arm motor. The resulting INTERACTION will depend on whether the motor fuse is burnt out. There can also be more than one way of achieving the same INTERACTION. I can push a block with my right hand or my left hand. The effect on the block is the same, as long as the force applied is the same.

Perception

An agent receives information about the state of the environment through its sensors. A PERCEPT is a signal that the agent's sensors pass to the agent's control component. It is the input counterpart to a COMMAND. The PERCEPT sent can be influenced both by the WORLD and the CONFIGURATION of the agent's mechanism. For instance, the output of a camera will depend on the state of the world and the direction the camera is pointed in. A PERCEPT can indicate both state and rates of change. For instance, a airplane's sensors can indicate both the altitude and the rate at which it is climbing.

Again imagine yourself as the controller inside the agent. The control panel has some lights on it indicating the current value of the sensor readings. These lights provide your only access to the information about the outside world. Each of the possible patterns of lights corresponds to a single PERCEPT.

$[\mathit{PERCEPT}]$

Time and Chronicles

In this specification, time is measured in discrete intervals. For convenience, we mapped each interval to an integer to keep track of the sequencing. A discrete time representation was chosen since it results in cleaner property definitions with little loss in generality. This specification does not depend on the size of the time interval. Nowhere is the time quanta specified. By making the time quanta sufficiently short, continuous time can be approximated to any desired granularity.

$\mathit{Time} == \mathbb{Z}$

The evolution of some entity, such as a component or an interaction, over time, can be represented using a Chronicle [5]. A Chronicle consists of the value of the entity at each discrete time interval. Chronicles are well suited for modeling things that change only at discrete times. A good example of something that changes at discrete times is a controller implemented using a micro-processor. The controller's outputs only change synchronously with the micro-processor's clock. For something that varies continuously, a Chronicle is only an approximate representation in the same way that the samples on an audio CD only approximate the original music. The critical factor in such an approximation is the sampling rate. The rate must be fast enough that any required information is not lost.

Formally, a Chronicle is defined to be a partial function that maps times to corresponding values. All Chronicles have definite start times. In addition, we define FiniteChroncles to have finite end times as well. The definition of Chronicle and FiniteChronicle are parameterized by the type of entity, Q. A particular kind of Chronicle is defined by specifying the type of entity. For example, a Chronicle of external states would be "Chronicle[ExternalState]".

$$\mathit{Chronicle}[Q] == \{c : \mathit{Time} \rightarrow\!\!\!\!\!+ \; Q \mid \exists\, t0 : \mathit{Time} \bullet \mathrm{dom}\, c = \{t1 : \mathit{Time} \mid t1 \geq t0\}\}$$

$$\mathit{FiniteChronicle}[Q] == \{c : \mathit{Time} \rightarrow\!\!\!\!\!+ \; Q \mid \exists\, t1, t2 : \mathit{Time} \bullet \mathrm{dom}\, c = t1 \mathrel{..} t2\}$$

Environment

Every agent operates in an environment. To describe an environment, we must characterize the valid states and how they change over time. In general, the state of an environment is dynamic, changing of its own accord and in response to interaction with the agent. In a deterministic environment, knowing the exact state, the interaction with the agent and the "laws of nature", allows the immediate future to be predicted exactly. In a non-deterministic environment, the best that can be done is to predict a set of possible future states. Any model of the environment must be able to account for both deterministic and non-deterministic state changes.

In this framework, changes in the state of the environment will be represented as a Chronicle of the external state. To characterize how the environment changes, we need only be able to predict its next state, given the current state and the current interaction with the agent. This can be done if the Markov property holds. Such is the case if the state description of the environment captures all relevant the features. For example, to predict the flight of a ball thrown into the air, the state description needs to include the velocity of the ball as well as its position. The definition of WORLD and CONFIGURATION include the state and instantaneous rates of change of state in order that the Markov property would hold for an ExternalState.

The approach we will take for non-deterministic environments is to model them as deterministic environments with hidden state variables. To get an idea of how this works, imagine that, at the beginning of time, you ask an oracle to write down the outcome of all future non-deterministic events. You then include this information in the state of the environment. Of course, there is no way that an agent can perceive this information. It remains hidden until each non-deterministic event takes place and the outcome is revealed. There are advantages to modeling non-determinism in this way. It allows agent to be compared under identical conditions. How does each agent do, if the coin flip comes up heads? It also allows a single probability distribution over initial states to account for the likelihood of each state and the probability of each non-deterministic event. In practical terms, being able to determine the next state rather than a set of possible next states, simplifies the specification without loss of generality.

The environment is represented by a schema that embodies the "laws of Nature". The schema includes a function for specifying how things change over time and a partitioning of external states for hiding state information. The consequence function maps the current ExternalState and the current INTERACTION with the agent to the next ExternalState. It is defined for all INTERACTIONs and ExternalStates.

It is not possible for an agent or an observer to determine the value of the hidden state variables. If it were, then the environment would be deterministic. The environment schema partitions the set of external states into sets of indistinguishable states that differ only in the value of hidden state variables. This partitioning is used to enforce the restriction that hidden state remain unobservable.

$$
\begin{array}{l}
\textbf{Environment} \\
\hline
consequence : (ExternalState \times INTERACTION) \longrightarrow ExternalState \\
indistinguishable : \mathbb{P}\, ExternalDescription \\
\hline
\bigcup indistinguishable = \{e : ExternalState\} \\
\forall a, b : indistinguishable \bullet a \cap b \neq \varnothing \Leftrightarrow a = b
\end{array}
$$

To assist in enforcing the restriction on indistinguishable states, we define a function that takes an environment and returns the set of pairs of indistinguishable states. The returned set consists of all pairs of ExternalStates that differ only in their hidden state.

$$
\begin{array}{l}
sameExternalStates : Environment \longrightarrow \mathbb{P}(ExternalState \times ExternalState) \\
\hline
\forall env : Environment; e_1, e_2 : ExternalState \bullet \\
\quad (e_1, e_2) \in sameExternalStates(env) \\
\quad \Leftrightarrow (\exists ind : env.indistinguishable \bullet e_1 \in ind \wedge e_2 \in ind)
\end{array}
$$

External Chronicle

An ExternalChronicle is a Chronicle of the externalState. It describes the evolution of the environment and the behaviour of the agent.

$ExternalChronicle == FiniteChronicle[ExternalState]$

In a particular environment, only some Chronicles are possible. To be valid, the sequence of ExternalStates must follow the “laws of nature”, that is, they must result from repeated application of the consequence function. The Chronicles function returns the valid ExternalChronicles for a particular environment.

$$
\begin{array}{l}
chronicles : Environment \longrightarrow \mathbb{P}\, ExternalChronicle \\
\hline
\forall env : Environment \bullet \\
\quad chronicles(env) \\
\quad = \{chronicle : ExternalChronicle \mid \\
\qquad (\forall t : restTimesOf(chronicle) \bullet \exists interaction : INTERACTION \bullet \\
\qquad\quad (env.consequence(chronicle(t-1), interaction) = chronicle(t)))\}
\end{array}
$$

Three helper functions are used in defining and using chronicles. The timesOf function returns the set of times covered by a chronicle. The restTimesOf function returns the same set, minus the initial time in the chronicle. The third function, firstOf, returns the value at the first time in the chronicle.

Two Chronicles are indistinguishable if they cover the same time period and each of the external states in the Chronicle are indistinguishable. “sameChronicles” is a function that takes an environment as input and returns the set of pairs of indistinguishable Chronicles.

$$
\begin{array}{l}
sameChronicles : Environment \longrightarrow \mathbb{P}(ExternalChronicle \times ExternalChronicle) \\
\hline
\forall c_1, c_2 : ExternalChronicle; env : Environment \bullet \\
\quad (c_1, c_2) \in sameChronicles(env) \\
\quad \Leftrightarrow timesOf(c_1) = timesOf(c_2) \wedge \\
\qquad (\forall t : timesOf(c_1) \bullet (c_1(t), c_2(t)) \in sameExternalStates(env))
\end{array}
$$

Task

A task is a description of what the agent is to achieve in the environment. Possible tasks include classical Artificial Intelligence tasks like block world problems as well as tasks with time constraints and tasks of maintenance. The method of specifying tasks must be able to encode each of these types of tasks. In addition, the task specification should allow for different types of performance evaluation including simple binary evaluations and expected utility evaluations.

All task specifications include the environment, the set of possible initial conditions and a method for evaluating performance. The task schema provides the basic structure used to define all types of tasks. The method of evaluation is given as a parameter, "*E*" to allow the schema to be specialized for different types of task evaluation functions. Throughout this specification, the "*E*" parameter will refer to the method of task evaluation.

$$
\begin{array}{|l}
\hline
\textit{Task}[E] \\
\textit{environment} : \textit{Environment} \\
\textit{initialConditions} : \textit{ExternalDescription} \\
\textit{evaluation} : E \\
\hline
\end{array}
$$

The simplest form of task evaluation is binary. Either the task is accomplished or it is not. Such an evaluation can be encoded by listing all of the ways of accomplishing the task. An agent is successful if its behaviour, in response to conditions in the environment, corresponds to one of the methods of accomplishing the task.

A BinaryEvaluation is defined to be the set of desired ExternalChronicles corresponding to all the ways that the task can be accomplished. The BinaryEvaluation schema is used as a parameter to the task schema to define a BinaryTask. The BinaryTask schema imposes further restrictions on the desired Chronicles. The desired Chronicles must be valid Chronicles for the environment. In addition, each Chronicle must start in one of the given initial states. Finally, the task evaluation is not allowed to differentiate between Chronicles that differ only in hidden state.

$$\textit{BinaryEvaluation} == \mathbb{P}\ \textit{ExternalChronicle}$$

$$
\begin{array}{|l}
\hline
\textit{BinaryTask} \\
\textit{Task}[\textit{BinaryEvaluation}] \\
\hline
\textit{evaluation} \subseteq \textit{chronicles}(\textit{environment}) \\
\forall\, c : \textit{evaluation} \bullet \textit{firstOf}(c) \in \textit{initialConditions} \land \\
\quad (\forall\, c' : \textit{ExternalChronicle} \bullet \\
\qquad (c, c') \in \textit{sameChronicles}(\textit{environment}) \Rightarrow c' \in \textit{evaluation}) \\
\hline
\end{array}
$$

The following examples illustrate the generality of the task representation. The examples include tasks of achievement, tasks of maintenance and tasks of information gathering.

Tasks of achievement require the agent to achieve some condition in the world. Classical AI/STRIPS tasks are tasks of achievement where the conditions to be achieved are goals stated in predicate logic [2]. Such a task is represented by the set of Chronicles where the goals are eventually achieved. The desired set of Chronicles for the blocks world task *On(A,B)* would include all Chronicles that started in one of the initial states and end with block A on block B. Note that this type of task requires an infinite number of desired Chronicles since no time limit is specified. Any Chronicle where the goal is eventually achieved is allowed. Tasks that involve temporal constraints do restrict the number of desired Chronicles. The task On(A,B) by 5:00 pm today restricts the desired Chronicles to those that achieve the goal by the deadline.

A task of maintenance is a task where the agent must maintain some condition in the environment. For example, a thermostat agent may have to maintain the temperature of the room between 20 and 22 degrees Celsius. The task is represented by mapping the initial state to the set of Chronicles where the temperature does not vary outside of the allowed limits. It is also possible to add temporal constraints such as "Don't let the temperature vary above 22 degrees for more than 3 minutes at a time."

Tasks of observation require the agent to determine some information about the state of the world and act on it. Since tasks are behavioural descriptions, tasks must be specified in terms of behaviour; they cannot specify anything about the mental state of the agent. For example, the task to "Go outside and see if it is raining" is not operational. There is no way to determine if the agent "knows" whether it is raining. We cannot infer that the agent would know if it was raining even if we observed the agent going outside. Such a task can be made operational by adding actions that depend on the observations. The example could be changed to : "Go outside and open your umbrella if it is raining."

When defining a task, one must be careful when dealing with tasks that don't have a deadline. Consider again the classic blocks world. In most formulations of planning problems for this domain, there is no time limit on how long the agent may take to complete the plan. As stated before, such a task results in an infinite set of desired Chronicles. An agent that does nothing will always be in a desired state, but will never achieve the goal. Even if the agent were to act, determining whether it will ever achieve the goal is an undecidable problem [1]. An analogy can be made with the difference between partial and total correctness for a computer program. A program is partially correct if it never outputs a bad result and is totally correct if it never outputs a bad result, but always outputs a result and stops. Similarly, an agent is partially successful if it never does anything wrong, but is only totally successful if it also eventually completes the task.

Agent

An agent is any entity created to accomplish a task. We distinguish the agent from the environment to enable us to substitute one agent for another to compare their performances. Within the agent, we distinguish the mechanism from the controller, again to allow comparisons between different mechanisms and controllers.

An agent's mechanism determines how the agent can perceive and affect its environment. It provides the agent's only means for interacting with the environment. The mechanism's sensors provide information about the external state and the mechanism's effectors provide a means for changing the external state.

A Mechanism is represented by a pair of functions that model its sensors and its effectors. The perceive function models sensors by mapping the current ExternalState to a PERCEPT. The effects function represents the effectors as a mapping from a COMMAND, issued by the agent's controller, and the agent's current CONFIGURATION to an INTERACTION with the environment. The perceive mapping allows for perceptual aliasing and sensor noise. Perceptual aliasing results when more than one distinguishable ExternalState maps onto the same PERCEPT. Sensor noise is the result of non-deterministic mapping of a single world onto multiple PERCEPTs. Noise is modeled by using the hidden world state to map indistinguishable ExternalStates to different PERCEPTs.

Mechanism

$$\begin{array}{l} \mathit{perceive} : \mathit{ExternalState} \rightarrowtail\!\!\!\!\!\!+\, \mathit{PERCEPT} \\ \mathit{effects} : (\mathit{COMMAND} \times \mathit{CONFIGURATION}) \rightarrowtail\!\!\!\!\!\!+\, \mathit{INTERACTION} \end{array}$$

In the abstract, the internals of an agent can be modeled as a finite state controller that consists of a control function and a memory. The control function is a mapping from a PERCEPT and current STATE to a COMMAND and next STATE. The exact nature of the control mapping and the information encoded in the state are determined by the agent's architecture and implementation.

The controller schema defines a controller to be a generic partial function where the type of the state is given as a parameter, "S". This allows controllers with different memory organizations to be specified as specializations of a generic controller. Throughout this specification, the "S" parameter will refer to the organization of the agent's memory.

$$\mathit{Controller}[S] == (\mathit{PERCEPT} \times S) \rightarrowtail\!\!\!\!\!\!+\, (\mathit{COMMAND} \times S)$$

A complete agent consists of a mechanism, a controller and an initial internal state. The controller and the mechanism must be matched in terms of PERCEPTs and COMMANDs; That is, the controller must be able to accept the PERCEPTs generated by the mechanism and the mechanism must be able to handle the COMMANDs issued by the controller. The functions ctrCommands and mechCommands extract the COMMANDs generated by the controller and handled by the mechanism respectively. Similarly, the functions mechPrecepts and ctrPercepts extract the PERCEPTs generated by the mechanism and handled by the controller respectively. The agent schema includes an initial state for the controller. This initial state encodes any explicit, a priori knowledge that the designer has given the agent. Different types of agents are defined as specializations of this general agent model.

Agent[S]

$$\begin{array}{l} \mathit{controller} : \mathit{Controller}[S] \\ \mathit{mechanism} : \mathit{Mechanism} \\ \mathit{initialInternalState} : S \\ \hline \mathit{ctrCommands}(\mathit{controller}) \subseteq \mathit{mechCommands}(\mathit{mechanism}) \\ \mathit{mechPercepts}(\mathit{mechanism}) \subseteq \mathit{ctrPercepts}(\mathit{controller}) \\ \mathit{initialInternalState} \in \mathit{ctrStates}(\mathit{controller}) \end{array}$$

Agent System

Putting the parts together, we get an agent system consisting of the agent and the task that the agent is to perform. The agent is parameterized by the organization of its internal state and the task is parameterized by the type of evaluation function.

$$\begin{array}{|l}
\hline
\textit{AgentSystem}[S, E] \\
\hline
\textit{Agent}[S] \\
\textit{Task}[E] \\
\hline
\end{array}$$

Now that the basic framework has been defined, we describe the machinery needed to characterize how the agent and the environment interact and evolve over time. The final result will be a function that takes an agent system and an initial state and returns the Chronicle of external state. This function will form the basis for defining *successful* and hence, the other properties.

The details of how the system changes with time are somewhat complex, but the basic idea is simple. Given a complete description of the AgentSystem and its state, the state at the next time interval can be determined. The future Chronicle of the system state is generated by repeatedly determining the next state. The part of the agent Chronicle that is of interest is the external state, since it is the observable behaviour of the agent. The ExternalChronicle can be obtained by projecting the external state out of the Chronicle of the system state.

The complete, instantaneous description of the agent system consists of the AgentSystem augmented by the current internal and external states.

$$\begin{array}{|l}
\hline
\textit{AgentSystemState}[S, E] \\
\hline
\textit{AgentSystem}[S, E] \\
\textit{internalState} : S \\
\textit{externalState} : \textit{ExternalState} \\
\hline
\textit{internalState} \in \textit{ctrStates}(\textit{controller}) \\
\hline
\end{array}$$

Some of AgentSystemState bindings are possible start states. An AgentSystemStart is an AgentSystemState with the initial internal state and one of the possible initial external states.

$$\begin{array}{|l}
\hline
\textit{AgentSystemStart}[S, E] \\
\hline
\textit{AgentSystemState}[S, E] \\
\hline
\textit{internalState} = \textit{initialInternalState} \\
\textit{externalState} \in \textit{initialConditions} \\
\hline
\end{array}$$

Each transition of the system state must be a valid step which depends on the environment's consequence function and the agent. The AgentSystemStep transition schema defines a valid step. The agent's control function is used to update the internal state and issue a command. The command is passed to the agent's mechanism and interacts with the environment to produce the next external state. The agent and the task remain unchanged.

$$
\begin{array}{l}
\textit{AgentSystemStep}[S,E] \\
\hline
\Delta \textit{AgentSystemState}[S,E] \\
\hline
\exists\, \textit{cmd} : \textit{COMMAND} \bullet \\
\quad (\textit{cmd}, \textit{internalState}') = \\
\qquad \textit{controller}(\textit{mechanism}.\textit{perceive}(\textit{externalState}), \textit{internalState}) \land \\
\quad \textit{externalState}' = \\
\qquad \textit{environment}.\textit{consequence}(\textit{externalState}, \\
\qquad\quad \textit{mechanism}.\textit{effects}(\textit{cmd}, \textit{externalState}.\textit{configuration})) \\
\theta \textit{AgentSystem} = \theta \textit{AgentSystem}'
\end{array}
$$

As time proceeds, the external state and the agent's internal state are updated in lock step. This is not to say, however, that the controller must go through a decision cycle and generate a new COMMAND after each interval. Likewise, it does not mean that the environment only changes when the controller generates a new command. The controller may take any number of intervals to process the sensor output and calculate the next command to issue. In the mean time, the controller may continue to issue the same COMMAND (push the same buttons) or issue a null command (not push any buttons). What happens will depend on the implementation of the controller. The state of the controller is updated every interval to reflect any progress made in doing computation. Likewise, the state of the environment is updated to reflect any changes. The critical factor is the length of the time interval. It must be short enough that no significant information about the external state is lost.

An AgentChronicle is an infinite Chronicle of the evolution of the AgentSystemState. Each transition in the sequence is a valid step and the first state in the Chronicle must be a valid start state.

$$
\begin{array}{l}
\textit{AgentChronicle}[S,E] == \{\textit{agentChronicle} : \textit{Chronicle}[\textit{AgentSystemState}[S,E]] \mid \\
\quad \exists\, \textit{AgentSystemStart}[S,E] \bullet \theta \textit{AgentSystemStart} = \textit{firstOf}(\textit{agentChronicle}) \land \\
\quad (\forall\, t : \textit{restTimesOf}(\textit{agentChronicle}) \bullet \\
\qquad (\exists\, \textit{AgentSystemState}[S,E]; \textit{AgentSystemState}'[S,E] \bullet \\
\qquad\quad \theta \textit{AgentSystemState} = \textit{agentChronicle}(t-1) \land \\
\qquad\quad \theta \textit{AgentSystemState}' = \textit{agentChronicle}(t) \land \\
\qquad\quad \textit{AgentSystemStep}[S,E]))\}
\end{array}
$$

Finally, we create the chronicleFrom function that gives the behaviour of an agent in an environtment when started in a particular initial state. It completely characterizes the performance of the agent and forms the basis for defining *successful* and the other agent properties.

The chronicleFrom function generates the correct AgentChronicle and projects out the Chronicle of external state. Details of the definition were omitted since they are rather cumbersome and do not add much semantic value.

$$
\begin{array}{l}
[S,E] \\
\hline
\textit{chronicleFrom} : (\textit{AgentSystem}[S,E] \times \textit{ExternalState}) \longrightarrow \textit{ExternalChronicle} \\
\hline
\ldots \text{Details omitted} \ldots
\end{array}
$$

General Agent Properties

This section defines three general agent properties: successful, capable and perceptive. These properties are relevant for any type of agent performing a task with a binary evaluation. The relationship between these properties is shown in figure 4. An agent is capable if its effectors are able to accomplish the task. A perceptive agent also possesses the sensors needed to determine how to operate the effectors for achieving the task. A successful agent is both perceptive and capable.

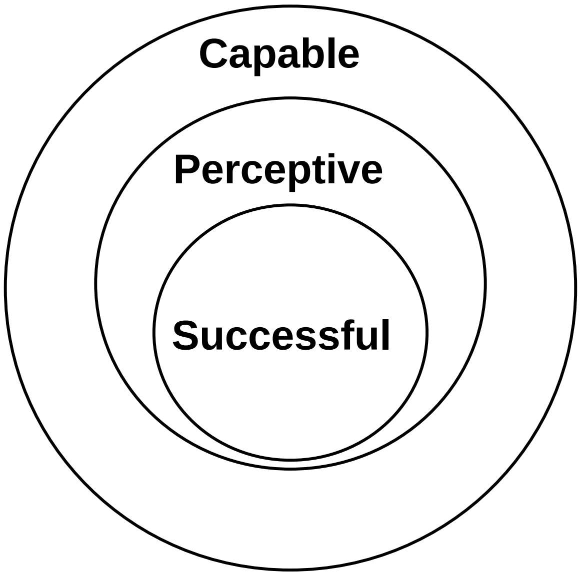

Figure 4: **General Agent Properties**

To assist in our definitions, we define the relation $=_{\text{Task}}$ that holds between two agent systems when the task, and hence, the environment are the same.

$$[S, E]$$

$$\_ =_{\text{Task}} \_ : AgentSystem[S, E] \longleftrightarrow AgentSystem[S, E]$$

$$\forall AgentSystem_1[S, E]; AgentSystem_2[S, E] \bullet$$
$$\theta AgentSystem_1 =_{\text{Task}} \theta AgentSystem_2 \Leftrightarrow (\theta Task_1 = \theta Task_2)$$

Successful Agents

A successful agent always achieves its task. Given the concept of an AgentChronicle and the chronicleFrom function defined in the previous section, it is easy to define success for an agent performing a binary task. The chronicleFrom function is used to generate the infinite chronicle of the agent system when started in a particular state. The agent accomplishes the task from this state if some prefix of the infinite chronicle corresponds to one of the finite chronicles representing one way of accomplishing the task. The agent is successful if it accomplishes the task in all possible initial states.

$$
\begin{array}{|l}
\hline
\textit{SuccessfulAgentSystem}[S] \\
\textit{AgentSystem}[S, \textit{BinaryEvaluation}] \\
\hline
\forall\, w : \textit{initialConditions} \bullet \\
\qquad \textit{prefixesOf}(\textit{chronicleFrom}(\theta \textit{AgentSystem}, w)) \cap \textit{evaluation} \neq \emptyset \\
\hline
\end{array}
$$

Capable Agents

Capability reflects the ability of the agent's effectors to achieve a task. One way of demonstrating capability is to show that there is some agent with the same effectors that is successful, although it may not have the same perception or control function as the given agent. Capability ignores the fact that the given agent may not be able to perceive the relevant characteristics of the external state and may not choose the correct COMMANDs.

$$
\begin{array}{|l}
\hline
\textit{CapableAgentSystem}[S] \\
\textit{AgentSystem}[S, \textit{BinaryEvaluation}] \\
\hline
\exists\, \textit{SuccessfulAgentSystem}'[S] \bullet \\
\qquad \theta \textit{AgentSystem} =_{\text{Task}} \theta \textit{SuccessfulAgentSystem}' \wedge \\
\qquad \textit{mechanism.effects} = \textit{mechanism}'.\textit{effects} \\
\hline
\end{array}
$$

Perceptive Agent

Perceptiveness measures the agent's ability to distinguish salient features of the environment. What constitutes a salient feature is determined by the task and the agent's mechanism. Different tasks require the agent to respond to different features and events in the world. For example, if the task were to open the umbrella if and only if it is raining, then rain is a salient feature to which the agent must respond. The agent's mechanism is also important. Different mechanisms can achieve the same effect through different INTERACTIONS. Since an agent's PERCEPTs help determine its COMMANDs (and hence INTERACTIONs with the world), it is necessary to know the mechanism in order to determine which PERCEPTs are salient. For example, if the mechanism is a large tank and the task were to move from one location to another, then sensing bushes and rocks would not be required. If instead the mechanism were smaller and could get tangled in bushes, then sensing them would be salient.

An agent is perceptive if its sensors provide the information needed to select COMMANDs that accomplish the task. Perceptiveness ignores the fact that the agent's controller may not actually select COMMANDs that achieve the task. The definition of perceptive will again depend on the existence of a successful agent, this time one with the same mechanism. A PerceptiveAgentSystem is an AgentSystem which has the same sensors and effectors as a SuccessfulAgentSystem.

$$
\begin{array}{|l}
\hline
\textit{PerceptiveAgentSystem}[S] \\
\textit{CapableAgentSystem}[S] \\
\hline
\exists\, \textit{SuccessfulAgentSystem}'[S] \bullet \\
\qquad \theta \textit{AgentSystem} =_{\text{Task}} \theta \textit{SuccessfulAgentSystem}' \wedge \\
\qquad \textit{mechanism} = \textit{mechanism}' \\
\hline
\end{array}
$$

Deliberative Agent

The agent properties defined in the previous section apply to any type of agent. In this section we introduce a type of agent of particular interest to the AI community known as a *deliberative* agent. A deliberative agent has an internal model of the world and uses its model to reason about the effects of COMMANDs in order to select COMMANDs that it predicts will accomplish the task [3]. Figure 5 shows a conceptual model of the organization of a deliberative agent system. (Compare with figure 3)

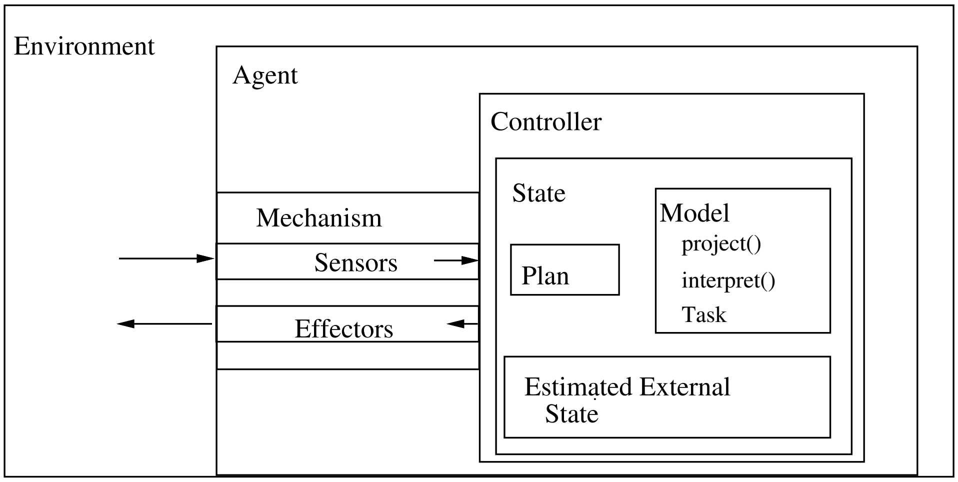

Figure 5: **Deliberative Agent System**

An agent's internal model of the environment must provide certain basic functionality. In order to reason about the consequences of COMMANDs, the model must predict how COMMANDs will affect the external state. The model must also be able to derive information about the external state from sensor output. The sensor model is also needed to predict which PERCEPTs to expect in predicted future external states. In addition to the model, the agent needs an estimate of the current external state. It is from this estimated external state that the agent does projections to infer the consequences of potential actions. The result of an agent's deliberation process is a plan to accomplish the task. The agent needs to maintain a representation of the plan to be able to issue the chosen COMMANDs at the correct time. The representation of the plan also allows the agent to further elaborate and revise the plan as new information is gathered and more computation is done.

A deliberative agent depends on its model of the world to enable it to accomplish its task. The properties defined for deliberative agents characterize the accuracy and suitability of the model for the task and how well the agent uses its model. Predictiveness characterizes the model's ability to make predictions in the environment. Likewise, interpretiveness characterizes the model's ability to infer information about the state of the world from the output of the agent's sensors. Independent of the accuracy of the model, an agent is rational if it behaves in accordance with its model of the world. A rational agent with a correct model is sound.

Details of the definitions of properties defined for deliberative agents can be found in the full specification. This brief description is to motivate these definitions and to illustrate how

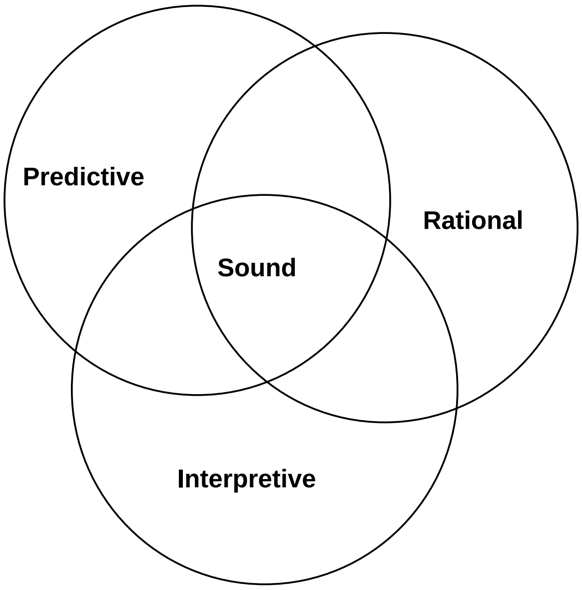

Figure 6: **Deliberative Agent Properties**

new agent properties can be defined for particular classes of agents.

Expected Task Achievement

The general agent properties presented thus far use a simplistic measure of task accomplishment. We recognize that some ways of achieving a task are better than others. There are tradeoffs that must be made in terms of efficiency, resource use and time. When making these tradeoffs, the probability of possible events should be taken into account. One method for doing this is to define a utility function over possible task completions and to weight the utility achieved in each situation by the probability of the situation. Doing this gives the expected utility for the agent.

To use expected utility as a measure of success, we define an expected performance task evaluation schema. The schema consists of a utility function and an initial distribution of ExternalStates. The expected performance is calculated by weighting the utility of the performance in initial ExternalState by the probability of that initial ExternalState. The expected performance evaluation schema can be used to derive expected value definitions of successful, capable and preceptive.

ExpectedEvaluation

$$\begin{array}{l} \mathit{utility} : \mathit{ExternalChronicle} \longrightarrow \mathbb{Z} \\ \mathit{initialDistribution} : \mathit{Distribution}[\mathit{ExternalState}] \end{array}$$

Future Work

The artificial intelligence community is particularly interested in agents that learn from their experience. The agent framework presented in this paper does not include a model of learning. It is suggested that a deliberative agent could learn by becoming more predictive, interpretive and rational. While this is probably correct, the issues in learning have more to do with how well the learner generalizes and how performance of one task can improve performance of related tasks. To do this, some method is needed for talking about related tasks.

A second direction for future work is to define the deliberative properties; predictive, interpretive and rational, in terms of the expected distribution of external states. It is more important to make correct predictions in situations that are more likely to occur. Such definitions would result in a finer scale for measuring these properties. This would be useful when defining learning agents. The problem in doing this is that the agent selects its actions based on its models. Changing the model could change the actions selected and thus change the expected distribution of external states. Any definition must take this shifting of external state distributions into account.

Another direction for future work would involve applying the framework developed in this specification to an actual agent system. This would involve tailoring the representations of the agent, the task and the environment for the particular system. Abstract definitions would have to be made operational. For instance, the evaluation function for a binary task could not be represented as an infinite set of infinitely detailed Chronicles. Extending this idea, multiple agents could be formalized and compared.

Conclusions

This paper has presented formal definitions for some properties of agents that perform tasks. For a general class of agents we have defined successful, capable and perceptive. In addition, definitions of predictive, interpretive, rational and sound were described for deliberative agents. These definitions have been given in terms of a framework for discussing agents, tasks and environments. Through developing this framework, we have identified some important issues that any work in this area should address. These issues include how to represent time, non-determinism and interaction between the agent and the environment.

It is intended that the definitions and framework presented in this paper be useful for analyzing and comparing agents. It is also hoped that this work will help foster discussion by providing a common and well defined vocabulary for talking about environments, tasks and agents.

Acknowledgments

I wish to thank David Garlan and Reid Simmons who were instrumental in guiding and encouraging the work reported in this paper.

References

[1] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32, 1987.

[2] Richard E. Fikes and Nils Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2), 1971.

[3] M. R. Genesereth. Deliberate agents. Technical Report Logic-87-2, Stanford University, Logic Group, Stanford CA. USA, 1987.

[4] Richard T. Goodwin. A formal specification of agent properties. Technical Report CMU-CS-93-159, Carnegie Mellon University, Pittsburgh PA. USA, May 1993.

[5] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[6] Tom M. Mitchell. Becoming increasingly reactive. In *Proceedings, Eight National Conference on Artificial Intelligence*, volume 2, pages 1051–1058. AAAI, The MIT Press, July 1990.

[7] Howard Raiffa. *Decision Analysis: Introductory Lectures on Choices under Uncertainty*. Addison-Wesley, Reading Mass., 1968.

[8] B. F. Skinner. *About Behaviorism*. Random House Inc, New York, 1974.

[9] J. M. Spivey. *The Z Notation, A Reference Manual*. Prentice Hall International Series in Computer Science. Prentice Hall, 66 Wood Lane End, Hamel Hemstead, Hertfordshire HP2 4RG, Endland, 1989.