The Independent Choice Logic for modelling multiple agents under uncertainty*

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

April 18, 1997

Abstract

Inspired by game theory representations, Bayesian networks, influence diagrams, structured Markov decision process models, logic programming, and work in dynamical systems, the independent choice logic (ICL) is 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 specification for agents that act in a world, make observations of that world and have memory, as well as a modelling tool for dynamic environments with uncertainty. The rules specify the consequences of an action, what can be sensed and the utility of outcomes. This paper presents a possible-worlds semantics for ICL, and shows how to embed influence diagrams, structured Markov decision processes, and both the strategic (normal) form and extensive (game-tree) form

*Thanks to Craig Boutilier and Holger Hoos for detailed comments on this paper. 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 OGPOO44121.

†Scholar, Canadian Institute for Advanced Research

of games within the ICL. It's argued that the ICL provides a natural and concise representation for multi-agent decision-making under uncertainty that allows for the representation of structured probability tables, the dynamic construction of networks (through the use of logical variables) and a way to handle uncertainty and decisions in a logical representation.

1 Introduction

This paper presents the Independent Choice Logic (ICL), a logic for modelling multiple agents under uncertainty. It's inspired by game theory [53; 32; 17], Bayesian networks [35; 7], influence diagrams [23; 22], probabilistic Horn abduction [36], structured representations of Bayesian networks and Markov decision processes [5; 8; 7], agent modelling and dynamical systems [29; 57; 51; 48] and logical modelling of action and change [27; 50; 45].

First we motivate ICL from a number of different perspectives, then show how it fits within the paradigms of knowledge representation (Section 1.1). In separate subsections we present the foundations of ICL based on agents (Section 1.2), game theory (Section 1.3), influence diagrams (Section 1.4) and logic (Section 1.5). We then build the formal definition of the representation in Section 2. The majority of this paper presents examples of the use of the logic, including showing how influence diagrams, Markov decision processes, and the strategic and extensive forms of games can be represented.

Bayesian or belief networks [35] provide a useful representation for reasoning under uncertainty. Bayesian networks are a representation of independence amongst random variables. The Bayesian network model doesn't constrain how a variable depends on its parents, nor does it specify a representation for the conditional probability of a variable given its parents in the network. The conditional probabilities of variables given their parents are typically represented as tables, but can often be specified more compactly in terms of trees [7] or rules [36]. Rules are more compact than trees (unless the trees can have shared structure and redundant tests), in the sense that there are some functions where the tree representation is exponentially larger than the rule representation, but the converse doesn't hold[1]. Rules have the added advantage that there is a natural extension to

[1]As we allow negation as failure in the rules, the rules can be seen as a DNF definition of a concept (using Clark's completion [11]). It is known that DNF formulae sometimes entail an exponential blow up in size when converted to decision trees [46]. Decision trees can be converted simply to rules, with a rule for each leaf in the decision tree whose body corresponds to the path

the first-order case [36]. This paper builds on probabilistic Horn abduction [36], a first-order rule-based representation for Bayesian networks, allowing negation as failure and fewer restriction on the rules than in probabilistic Horn abduction. This paper extends the probabilistic framework to include utilities and decisions made by multiple agents, so that not only can the knowledge base be expressed compactly by rules, but agents' policies can also be expressed by rules.

Logic has become the primary focus of knowledge representation in AI. This is because it provides a way to give meaning to symbols and a way to specify what you want to compute independently of how it's computed [42]. It has often been argued (e.g., [33]) that any general representation scheme must be at least as rich as the first-order predicate calculus. One of the problems with the first order predicate calculus is the way it handles uncertainty; all it has available is disjunction. This is a rather blunt instrument and doesn't do justice to all of the subtleties involved in reasoning under uncertainty. Rather than adding uncertainty to the first-order predicate calculus [3; 24; 20; 19], which would entail having both disjunctive and probabilistic uncertainty, this paper proposes that we should use probability and decision theory, instead of disjunction, to handle uncertainty. In the ICL, we start with a logic doesn't include any uncertainty and is definitive on all propositions (every theory entails exactly one of p or $\neg p$ for all propositions p). Agents own *alternatives*, which are sets of propositions. An agent gets to choose one value from each alternative that it owns. Nature is a special agent; the alternatives owned by nature have a probability distribution over them. The logic gives the consequences of the choices made by nature and the agents. This allows us to have the advantages of logic, with symbols that can be given denotations, specifications of valid consequences and first-order representations, but also lets us use the normative tools of decision/game theory notions to determine what an agent should do.

Models of dynamical systems (see e.g., [29; 44]) have traditionally been described in terms of state spaces, for example treating state spaces in terms of vectors of states and state transition functions in terms of matrices. It's often much more convenient to describe a state space in terms of propositions, and describe the state transition function in terms of these propositions. The state transition function can be stated concisely in terms of Bayesian networks [12] or even more concisely by trees [5; 8] or rules [40], never referring to the explicit state. The number of variables is logarithmic in the size of the state space. The effects of actions are typically local, the value of a variable depending only on a few other

---

to the leaf.

variables. This provides the potential to take advantage of the compactness of the propositional representation. In the ICL we specify state transition functions in terms of rules. One advantage of rules is that they are closer to the traditional AI representations such as the situation calculus [31] (see [40]). The first-order nature of the rules, with explicit reference to the stage or situation make the rules perspicuous. The rule based representation also helps clarify the close relationship between regression planning and dynamic programming.

1.1 Knowledge Representation

There are two different views of what a knowledge representation should be.

- The first is that a knowledge representation should let users state whatever knowledge they have in a reasonably natural way. Under this view it isn't appropriate for the designer of a knowledge representation to specify how a piece of knowledge should be encoded. Reasoning can conclude what logically follows from the stated facts or can fill in missing facts in a common sense manner. An example of this view is in the use of the first-order predicate calculus for knowledge representation. It's a rich enough language to let us state many facts about the world, but with primitive means to handle uncertainty. Within this tradition, logics have been developed to handle uncertainty and multiple agents making decisions [3; 24; 20; 19; 15; 21]. Missing facts can be inferred using default reasoning [30], or by making maximum entropy or random worlds assumptions [4]. What's important is that the user can add whatever they like to the knowledge base, and the representation should be able to make appropriate inferences.

- The second view is that a knowledge representation should provide a high-level symbolic modelling language that makes some things easier to state. Under this view a knowledge representation should specify how to model a domain. It should guide users as to how they should think about the domain, what they should say; and once some choices have been made, it prescribes what information needs to be specified. An example is Bayesian networks [35], which provide a modelling tool for representing independence amongst random variables. The user needs to specify the random variables of interest, the values these variables can take, and the dependency amongst these variables. Once these are specified the Bayesian network model prescribes what probabilities need to be specified.

It is important not to confuse these, as judging a knowledge representation by the inappropriate criteria will lead to an unfair judgment. The knowledge representation in this paper should be seen as an instance of the second. We don't expect that people will be able to just throw any knowledge in. For example, missing rules have a particular meaning; if you want to assert ignorance there are specific ways to do it.

1.2 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, or anything else. **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.

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's received by its sensors. These sensors may or may not reflect what's true in the world;[2] sensors can be noisy, unreliable or broken; and even when sensors are reliable there is still ambiguity about the world from sensors' readings. An agent can only condition its actions on what it knows, even if it's very weak such as "sensor a appears to be outputting value v". Similarly actuators may be noisy, unreliable, slow or broken. What an agent can control is what message (command) it sends to its actuators.

An agent can be seen as an implementation of a transduction [57; 47; 48; 39], a function from input (sensor readings) history into outputs (action attempts or actuator settings) at each time point. These are causal in the sense that the output can only depend on current inputs and previous inputs and outputs; they can't be conditional on future inputs or outputs.

A **policy** or **strategy** is a specification of what an agent will do under various contingencies. That is, it's a representation of a transduction. A **plan** is a policy

[2] Of course if there is no correlation between what a sensor reading tells us and what's true in the world, and the preferences of the agent depend on what's true in the world (as they usually do), it may as well ignore the sensor.

that includes either time or the stage as part of the contingencies conditioned on.

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

We want to model agents and their environments with the same language. The language should provide a decision theoretic, or game theoretic (for more than one agent) framework that can be used to build agents that can be shown to be optimal (as in [48]) or at least to have a specification of the expected utility of an agent. A **planner** in this framework is a program that generates a (possibly stochastic) transduction for an agent to execute. The output of the planner should be suitable for actually controlling an agent. It has to be more than a sequence of steps that is the output of traditional planners. Here we consider reactive agents that have internal state. This paper doesn't consider the problem of building a planner. Even for the single-agent propositional case, the problem of finding an optimal policy is computationally prohibitive [25], but this is more a property of the problem than of the representation. By having a rich representation we can discuss the complexity of various restrictions and build approximation algorithms.

Under this view, beliefs, desires, intentions, and commitments [51] aren't essential to agenthood. It may, however, be the case that agents with beliefs, desires, intentions, and commitments that, for example, communicate by way of speech acts [51], 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.

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.

1.3 Game Theory

Game theory [53; 32; 17] is a 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 designed to be a general theory of economic behaviour [53] that is a generalization of decision theory. The use of the term "game" here is much richer than typically studied in AI text books 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-person games. Each of these assumptions can be lifted [53].

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 [17]) representations of games, namely the extensive form and the normalized [53] (or strategic [32; 17]) form of a game.

The **extensive form of a game** is specified 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 can't distinguish. The aim is for each agent to choose a move at each of the information sets.

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" [53, p. 79]. This is represented as a function from information available to the agent's move.

The initial framework developed here should be seen as a representation based on the normalized form of a game, with a possible world corresponding to a complete play of a game. In the ICL we add a logic program to give the consequences of the play. This allows us to use a logical representation for the world and for agents. Section 5.5 presents a representation that is closer to the extensive form of a game.

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

Example 1.1 Consider a problem in designing soccer playing robots. In particular we want to consider the problem of a penalty kick. Penalty kicks are used to decide a winner in some soccer games that are tied at the end of regulation time. In a penalty kick, there are two agents: a kicker who is trying to score a goal, and a goalie who is trying to prevent the goal. The goalie must commit to either jumping left or right before they know whether the kicker will kick right or left (of course, neither want to let the other know which direction they will go). Suppose for this example, that if the goalie jumps (to its) left and the kicker kicks

		goalie	
		left	right
kicker	left	$\langle 0.9, 0.1\rangle$	$\langle 0.1, 0.9\rangle$
	right	$\langle 0.2, 0.8\rangle$	$\langle 0.9, 0.1\rangle$

Figure 1: Expected pay-off matrix for the game of Example 1.1. Payoff $\langle u_1, u_2\rangle$ indicates that the kicker has an expected payoff of u_1 (where a goal is worth one, and a block is worth zero for the kicker) and that the goalie has an expected payoff of u_2 (where a goal is worth zero, and a block is worth one for the goalie).

(to its) left, there is a 90% chance of a goal. Similarly if they both go (to their own) right, there is a 90% chance of a goal. If the kicker kicks left and the goalie jumps right there is a 10% chance of a goal. If the kicker kicks right and the goalie jumps left, there is a 20% chance of a goal (this could happen if the goalie is right handed). See Figure 1 for a payoff matrix for this example. The goalie should not reason that "I am better when I jump right so I should jump right". For then the kicker (realizing this) will always kick right. The goalie could then think that "as the kicker will kick right I should jump left". In which case the kicker should kick left. This is a never ending regress. Such problems have been well studied in game theory [53]. It turns out that the best strategy for the goalie is to randomize its choice. Similarly the best strategy for the kicker is to randomize its choice. In this example it's best for the kicker to kick right with probability $\frac{8}{15}$, and best for the goalie to jump right with probability $\frac{7}{15}$ in the sense that if either deviate from this randomized strategy, the other can exploit the deviation to have a higher chance of either scoring a goal or stopping a goal (see Example 2.21 for a derivation of these numbers).

1.4 Influence Diagrams

Influence diagrams [23] (see the papers in [34]) are a graphical representation of decision problems that extend Bayesian networks to include decision nodes and value nodes. Influence diagrams provide a perspicuous representation for decision problems making explicit the probabilistic dependencies and the information available when a decision is made (see e.g., [22])

The propositional version of the logic presented here can be seen as a representation for influence diagrams (see Section 3.1), where we can use rules to specify conditional probabilities [36], rules to specify utility and the policies of agents are specified as logic programs that imply what an agent will do based on its observations. The ICL allows specification of the influence diagram in a logic that lets us axiomatise the dynamics of the world, derive implicit information from explicit knowledge, and has a formal and natural semantics.

The independent choice logic (ICL) preserves the representational clarity of influence diagrams and extends them in four ways:

- The first advance is for representation of structured probability tables. The use of rules allows for the compact representation of probability tables (similar to the use of decision trees for specifying probability and utility tables [52; 5]). For example, although some variable d may depend on variables a, b and c, it may only depend on b when a has one value and on c when a has another value. This asymmetric dependency can be easily expressed in rules, forming a much more compact representation than the traditional tables. The rule structure can be exploited for efficiency [38; 5; 41]. The same rule-based representation can be used to express the policies.

- The use of logical variables allows for a form of first-order influence diagrams. These form a method for the dynamic construction of influence diagrams [9; 36].

- The use of the rule base means we don't have to specify in one step how a variable depends on its parents; we can use arbitrary computation. For example, we can axiomatise the dynamics of a domain and use the axiomatisation to specify how the position at one time depends on the position at a previous time in a compact way. This features will be exploited for many of the examples.

- The ICL can also handle multiple agents making decisions, permitting a form of multiple-agent influence diagrams. We are thus importing the representational advantages of influence diagrams into game-theory representations.

Note that extending the representation to logical variables and multiple agents increases the worst case computational complexity of deriving optimal plans;[3] this is because the problems that can be represented are more complex. It is not a problem with the representation per se.

[3]The use of variables makes it undecidable, but even without variables, multi-agent reasoning is exponentially harder than modelling a single no-forgetting agent [25].

1.5 Logic

Our aim is to define a logic where all uncertainty is resolved by decision/game theory rather than using disjunction to encode uncertainty. We start with a logic that has no uncertainty; it's definitive on the value of every proposition. We then show how uncertainty can be modelled as alternatives that are chosen by agents or have a probability distribution over them. The logic tells us the consequences of the choices.

We are treating logic as the modelling language of the world. Rather than having logic at the meta-level describing an object level in another language such as GOLOG [28] or constraint nets [56], we are using the logic to represent the object level. There is no other representation apart from the logic. We use the logic to axiomatise the causal structure of the world and the causal structure of agents (all of which are defined in terms of propositions).[4]

Rather than using disjunction to handle uncertainty, as in the predicate calculus, we want to use probability and decision theory to handle the uncertainty. There are normative arguments as to why we should use probability and utilities for reasoning under uncertainty [49]. The aim here is to get as much as we can from logic, but using decision or game theory to handle all of the uncertainty.

Starting a logic without uncertainty potentially lets us sidestep many traditional problems, or at least adopt simple solutions. For example, it seems as though the frame problem in the situation calculus [31] is solved for the case with complete knowledge and deterministic actions [27; 50; 45]; when there is incomplete knowledge and nondeterministic actions, there are still many problems to be resolved (see e.g., [6]). This paper takes quite a different view to other proposals, where we resolve nondeterminism and uncertainty by considering "who gets to resolve the uncertainty". We consider all actions as deterministic but with hidden variables, and have probabilities over these variables or have them chosen by different agents. This become a very powerful and arguably natural way to model non-deterministic action (see Section 5.4 and [40]).

[4] All of the logical statements in this paper are at the object level (i.e., are about the domain being axiomatised rather than being axioms about the formalism). This was done in order to reduce confusion: we don't need two different languages and the problems of quoting one language. All of the meta-level statements here are given in English or normal mathematical notation.

2 Independent Choice Logic

In this section we formalize the independent choice logic. We first give a general abstract definition of how an independent choice logic can be constructed from a base logic. In order to make the paper and examples more concrete we adopt acyclic logic programs under the stable model semantics as the base logic.

An independent choice logic (ICL) is a logic built with a specific semantic construction. We assume that we are given a base logic that conforms to some restrictions. The construction below specifies how to build possible worlds. Possible worlds are built by choosing propositions from sets of independent choice alternatives. The base logic is used to determine truth in the possible worlds.

The base logic is defined on two languages, the language $\mathcal{L}_F$ of facts, and the language $\mathcal{L}_Q$ of queries, and a consequence relation $\mid\!\sim$ between elements of $\mathcal{L}_F$ and elements of $\mathcal{L}_Q$. That is, $\mid\!\sim$ is a relation on $\mathcal{L}_F \times \mathcal{L}_Q$. It's usually written in infix notation. We assume that languages $\mathcal{L}_F$ and $\mathcal{L}_Q$ are logical languages which share the same atomic formulae. After the definition of the semantic construction we discuss what properties we want of $\mathcal{L}_F$ and $\mathcal{L}_Q$.

Definition 2.1 A **base logic** is a triple $\langle \mathcal{L}_F, \mid\!\sim, \mathcal{L}_Q \rangle$ such that $\mathcal{L}_F$ and $\mathcal{L}_Q$ are languages and $\mid\!\sim$ is a consequence relation.

Definition 2.2 An **independent choice logic theory** on base $\langle \mathcal{L}_F, \mid\!\sim, \mathcal{L}_Q \rangle$ is a pair $\langle \mathcal{C}, \mathcal{F} \rangle$, where

$\mathcal{C}$, called the **choice space**, is a set of sets of ground atomic formulae from language $\mathcal{L}_F$, such that if $\chi_1 \in \mathcal{C}$, $\chi_2 \in \mathcal{C}$ and $\chi_1 \neq \chi_2$ then $\chi_1 \cap \chi_2 = \{\}$. An element of $\mathcal{C}$ is called an **alternative**. An element of an alternative is called an **atomic choice**.

$\mathcal{F}$, called the **facts** or the rule base, is a set of formulae in logic $\mathcal{L}_F$.

The base logic is often omitted when it can be understood from context.

The semantics of an ICL is defined in terms of possible worlds. There is a possible world for each selection of one element from each alternative. The atoms which follow using the consequence relation from these atoms together with $\mathcal{F}$ are true in this possible world.

Definition 2.3 Given independent choice logic theory $\langle \mathcal{C}, \mathcal{F} \rangle$, a **selector function** is a mapping $\tau : \mathcal{C} \to \cup\mathcal{C}$ such that $\tau(\chi) \in \chi$ for all $\chi \in \mathcal{C}$. The **range** of selector function τ, written $\mathcal{R}(\tau)$ is the set $\{\tau(\chi) : \chi \in \mathcal{C}\}$. The range of a selector function will be called a **total choice**.

The basic semantic construction we want for the ICL is that each selector function corresponds to one possible world, where every element of the range of the selector function is true. The facts $\mathcal{F}$ specify what else is true in the possible world.

First we define restrictions on the base logic to ensure that the semantic construction gives a well defined semantics:

Definition 2.4 Base logic $\langle \mathcal{L}_F, \mid\!\sim, \mathcal{L}_Q \rangle$ and ICL theory $\langle \mathcal{C}, \mathcal{F} \rangle$ are **definitive** if for every selector function τ,

- If $\neg a$ is the negation of a in language $\mathcal{L}_Q$[5] then for each ground atom a of $\mathcal{L}_Q$, either $\mathcal{F} \cup \mathcal{R}(\tau) \mid\!\sim a$ or $\mathcal{F} \cup \mathcal{R}(\tau) \mid\!\sim \neg a$, and it isn't the case that $\mathcal{F} \cup \mathcal{R}(\tau) \mid\!\sim a$ and $\mathcal{F} \cup \mathcal{R}(\tau) \mid\!\sim \neg a$, and
- if α is an atomic choice then $\mathcal{F} \cup \mathcal{R}(\tau) \mid\!\sim \alpha$ if and only if $\alpha \in \mathcal{R}(\tau)$.

Definition 2.5 Suppose we are given definitive base logic $\langle \mathcal{L}_F, \mid\!\sim, \mathcal{L}_Q \rangle$ and ICL theory $\langle \mathcal{C}, \mathcal{F} \rangle$. For each selector function τ there is a **possible world** w_τ. If f is a formula in language $\mathcal{L}_Q$, and w_τ is a possible world, we write $w_\tau \models_{\langle \mathcal{C}, \mathcal{F} \rangle} f$, read "$f$ is true in world w_τ based on $\langle \mathcal{C}, \mathcal{F} \rangle$", iff $\mathcal{F} \cup \mathcal{R}(\tau) \mid\!\sim f$. When understood from context, the $\langle \mathcal{C}, \mathcal{F} \rangle$ is omitted as a subscript of $\models$.

The fact that every proposition is either true or false in a possible world follows from the definitiveness of the base logic.

Note that, for each alternative $\chi \in \mathcal{C}$ and for each world w_τ, there is exactly one element of χ that's true in w_τ. In particular, $w_\tau \models \tau(\chi)$, and $w_\tau \not\models \alpha$ for all $\alpha \in \chi - \{\tau(\chi)\}$.

2.1 The Languages $\mathcal{L}_F$ and $\mathcal{L}_Q$

Languages $\mathcal{L}_F$ and $\mathcal{L}_Q$ are logical languages which share the same propositions. The reason they are different is that we want to impose restrictions on each so that they are appropriate for their task.

For the rest of this paper we assume that $\mathcal{L}_Q$ is the propositional logic with atoms (propositions) corresponding to the set of ground atoms of $\mathcal{L}_F$. In this

[5] If $\mathcal{L}_Q$ doesn't contain a negation then the property we need is that the set of atomic formulae that follow (using $\mid\!\sim$) from $\mathcal{F} \cup \mathcal{R}(\tau)$ completely determines the other formulae that follow from $\mathcal{F} \cup \mathcal{R}(\tau)$. This means, for example that if $\mathcal{F} \cup \mathcal{R}(\tau) \mid\!\sim a \vee b$ then $\mathcal{F} \cup \mathcal{R}(\tau) \mid\!\sim a$ or $\mathcal{F} \cup \mathcal{R}(\tau) \mid\!\sim b$.

paper we will ignore issues relating to variables in $\mathcal{L}_Q$. We will allow arbitrary logical connectives (e.g., conjunction, disjunction, negation, etc.) in $\mathcal{L}_Q$.

If we want to use the independent choice framework we have to choose a logic (language $\mathcal{L}_F$ plus consequence relation $\mid\!\sim$) that has the property that it gives us a unique model for each total choice. This means two things:

- Each selection of an element from each alternative is consistent. This means that the logic can't allow a selection of choices from some alternatives to impose any restrictions on choices from other alternatives. This, for example, disallows the logic from being the arbitrary predicate calculus or even Horn clauses with integrity constraints [27].
- Each total choice can't be extended into more than one possible world. This excludes us from having explicit disjunctions in our logic[6]. It also means, for example, that we can't have logic programs under the stable model semantics which may have none or more than one stable model[7]. We are also excluding three valued models of logic programs (e.g., [43]) from consideration (whether extending our semantics in this way is useful or not is an open question).

2.2 Acyclic Logic Programs

In order to use an ICL we must commit to a base logic. In this paper, we consider the language $\mathcal{L}_F$ to consist of logic programs with a unique stable model [18], and the consequence relation to be truth in the stable model [18]. That is, logic program $\mathcal{P} \mid\!\sim q$ if q is true in the unique stable model of $\mathcal{P}$. One way to ensure there is a unique stable model is to restrict the programs to be acyclic [2].

In this section we give the language and the semantics of acyclic logic programs. The language follows Prolog's conventions.

Definition 2.6 A **variable** is an alphanumeric string (possibly including "_") starting with an upper case letter;

[6]Disjunction can be seen as a form of uncertainty. In some sense what we are pursuing here is that idea that all uncertainty can be relegated to the choice space, leaving the logic to give the consequences of the choices. This should be contrasted with other approaches (e.g., [3]) that allow both sorts of uncertainty. We end up with a much simpler language, but handle uncertainty by considering different agents getting to choose alternatives. Whether this is a good (both computationally and ergonomically) idea is an empirical question currently under study.

[7]The program $a \leftarrow \neg b, b \leftarrow \neg a$ has two stable models, one with a true and one with b true. The program $a \leftarrow \neg a$ has no stable models.

A **constant** or a **function symbol** or a **predicate symbol** is an alphanumeric string not starting with an uppercase letter;

A **term** is either a variable, a constant, or has the form $f(t_1, \ldots, t_m)$ where f is a function symbol and $t_1, \ldots, t_m$ are terms.

An **atom** is either a predicate symbol, or has the form $p(t_1, \ldots, t_m)$ where p is a predicate symbol and $t_1, \ldots, t_m$ are terms.

A **literal** is either an atom or has the form $\neg\alpha$ where α is an atom.

A **body** is either a literal or a conjunction of bodies (the conjunction of β_1 and β_2 is written as $\beta_1 \wedge \beta_2$).

A **clause** is either an atom or has the form $\alpha \leftarrow \beta$ where α is an atom (called the **head** of the clause) and β is a body. The latter form is called a **rule**.

A **program** is a set of clauses.

A **ground** term, atom or clause is one that doesn't contain any variables. A ground instance of a clause c is a clause obtained by uniformly replacing ground terms for the variables in c.

Definition 2.7 The **Herbrand base** of program $\mathcal{P}$ is the set of ground instances of the atoms formed from predicates, function symbols and constants in $\mathcal{P}$ (inventing a new constant if $\mathcal{P}$ does not contain any constants).

Definition 2.8 ([2]) A logic program $\mathcal{P}$ is **acyclic** if there is an assignment of a positive integer to each element of the Herbrand base of $\mathcal{P}$ such that, if $\mathcal{P}'$ is the set of ground instances of clauses in $\mathcal{P}$, then for every rule in $\mathcal{P}'$ the number assigned to the atom in the head of the rule is greater than the number assigned to each atom that appears in the body.

Acyclic programs are surprisingly general [2]. Note that acyclicity does not preclude recursive definitions. It just means that all such definitions have to be well founded.

Definition 2.9 An **interpretation** is an assignment of true or false to each member of the Herbrand base. Interpretation $\mathcal{M}$ is a **stable model** [18] of logic program $\mathcal{P}$ if for every ground atom h, h is true in $\mathcal{M}$ if and only if h is in $\mathcal{P}$ or there is a rule $h \leftarrow b$ in $\mathcal{P}'$ such that b is true in $\mathcal{M}$. Conjunction $a \wedge b$ is true in $\mathcal{M}$ if both a and b are true in $\mathcal{M}$. A negation $\neg a$ is true in $\mathcal{M}$ if and only if a isn't true in $\mathcal{M}$.

Note that the negation here is the so-called negation-as-failure [11]. We can use negation-as-failure in our knowledge base, although the standard procedural intuition doesn't necessarily hold [37].

Theorem 2.10 ([2]) An acyclic logic program has a unique stable model.

Acyclicity is also important for the physical realization of our game theory strategies; an agent can't condition on a value that depends on what it's going to do (see Section 5.5).

In some sense the possible world w_τ *is* the stable model of $\mathcal{F} \cup \mathcal{R}(\tau)$; they assign exactly the same truth values to propositions.

Example 2.11 Suppose we have ICL theory with $\mathcal{C} = \{\{a_1, a_2, a_3\}, \{b_1, b_2\}\}$, and with $\mathcal{F} = \{c \leftarrow a_1 \wedge b_1, c \leftarrow a_3 \wedge b_2, d \leftarrow a_1, d \leftarrow \neg a_2 \wedge b_1, e \leftarrow c, e \leftarrow \neg d\}$. There are 6 possible worlds with the following truth assignments:

$$
\begin{array}{lcccccccccc}
w_{\{a_1,b_1\}} & \models & a_1 & \neg a_2 & \neg a_3 & b_1 & \neg b_2 & c & d & e \\
w_{\{a_2,b_1\}} & \models & \neg a_1 & a_2 & \neg a_3 & b_1 & \neg b_2 & \neg c & \neg d & e \\
w_{\{a_3,b_1\}} & \models & \neg a_1 & \neg a_2 & a_3 & b_1 & \neg b_2 & \neg c & d & \neg e \\
w_{\{a_1,b_2\}} & \models & a_1 & \neg a_2 & \neg a_3 & \neg b_1 & b_2 & \neg c & d & \neg e \\
w_{\{a_2,b_2\}} & \models & \neg a_1 & a_2 & \neg a_3 & \neg b_1 & b_2 & \neg c & \neg d & e \\
w_{\{a_3,b_2\}} & \models & \neg a_1 & \neg a_2 & a_3 & \neg b_1 & b_2 & c & \neg d & e
\end{array}
$$

Note that there are two sorts of atoms; atomic choices (a_1, a_2, a_3, b_1, b_2) and derived atoms (c, d, e). The atomic choices that are true in the world are given by the selector function for the world (here we have subscripted the worlds with the range of the selector function), and there is a world for each selector function. The truth of the derived atoms is defined by the rules and the range of the selector function.

2.3 The Multi-agent Independent Choice Logic

The Independent Choice Logic (ICL) specifies a way to build possible worlds. In order to model multi-agent situations, we need to have more structure. In particular we need different agents to be able to control different choices.

Definition 2.12 A **multi-agent independent choice logic theory** is a tuple $\langle \mathcal{C}, \mathcal{F}, \mathcal{A}, controller, P_0 \rangle$ where

$\mathcal{C}$, the **choice space**, is as in Definition 2.2.

$\mathcal{F}$, 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 0 called "nature".

$controller$ is a function from $\mathcal{C} \to \mathcal{A}$. If $controller(\chi) = a$ then agent a is said to control alternative χ. If $a \in \mathcal{A}$ is an agent, the set of alternatives controlled by a is $\mathcal{C}_a = \{\chi \in \mathcal{C} : controller(\chi) = a\}$. Note that $\mathcal{C} = \bigcup_{a \in \mathcal{A}} \mathcal{C}_a$.

P_0 is a function $\cup\mathcal{C}_0 \to [0, 1]$ such that $\forall \chi \in \mathcal{C}_0$, $\sum_{\alpha \in \chi} P_0(\alpha) = 1$.[8] That is, for each alternative controlled by nature, P_0 is a probability measure over the atomic choices in the alternative.

Often, when the context is clear we refer to a multi-agent independent choice logic theory simply as an independent choice logic theory.

The idea is that an agent gets to choose one element from each of the alternatives it controls. The alternatives controlled by nature have a probability distribution over them. The facts give the consequences of the choices by the agents.

2.3.1 Rules for utility

Game theory and decision theory are based on the notion of *utility*, a cardinal value representing the worth to an agent of an outcome or possible world.[9] Higher utilities reflect preferred worlds. Agents act to increase their (expected) utility. Finding optimal strategies becomes trickier when there are multiple agents with competing objectives, but the idea of each agent trying to maximise its utility remains.

Utility is a function of both an agent and a world. Different agents have different preferences and so different utilities in the possible worlds. Note that nature (agent 0) doesn't have a utility.

The logic program can have rules for $utility(a, u)$, where $utility(a, u)$ is true in a possible world if u is the utility for agent $a \neq 0$ in that world.

[8]When χ isn't discrete, we may need to use an integration rather than summation. To avoid measurability and integrability issues, we assume in this paper that all sets are discrete and finite, although the framework isn't necessarily restricted to this case.

[9]The existence of a utility function, and the existence of a probability distribution is implied from a set of intuitive axioms about rational preferences, such that agents try to maximise expected utilities [53; 49]. Like most decision and game theory practitioners we take the notion of utility as something that we want to represent and use to derive optimal actions for agents. There is a large body of literature about how these utilities can be acquired (see e.g., papers in [34]).

Definition 2.13 ICL theory $\langle \mathcal{C}, \mathcal{F}, \mathcal{A}, controller, P_0 \rangle$ is

utility consistent for agent $a \in \mathcal{A}$ where $a \neq 0$ if, for each possible world w_τ, $w_\tau \models utility(a, u_1) \wedge utility(a, u_2)$ implies $u_1 = u_2$. The theory is **utility consistent** if it's utility consistent for all agents (other than agent 0).

utility complete for agent $a \in \mathcal{A}$ where $a \neq 0$ if, for each possible world w_τ, there is a unique number u such that $w_\tau \models utility(a, u)$. The theory is **utility complete** if it's utility complete for all agents (other than agent 0).

Thus an ICL theory is utility consistent and complete means that the $utility$ relation is a function for each possible world.

We assume that all of the theories are utility consistent and complete.

Example 2.14 Continuing example, 2.11 suppose the rules for utility are:

$$
\begin{aligned}
&utility(agent_1, 5) \leftarrow \neg e \\
&utility(agent_1, 0) \leftarrow e \wedge c \\
&utility(agent_1, 9) \leftarrow e \wedge \neg c \\
&utility(agent_2, 7) \leftarrow d \\
&utility(agent_2, 2) \leftarrow \neg d
\end{aligned}
$$

Note that, if these are all the rules for $utility$ then the ICL is utility consistent and complete for $agent_1$ and $agent_2$ independently of the choice space and the other rules.

The values for the possible worlds (omitting the false atomic choices) are:

$$
\begin{array}{llllllll}
w_{\{a_1,b_1\}} & \models & a_1 & b_1 & c & d & e & utility(agent_1, 0) & utility(agent_2, 7) \\
w_{\{a_2,b_1\}} & \models & a_2 & b_1 & \neg c & \neg d & e & utility(agent_1, 9) & utility(agent_2, 2) \\
w_{\{a_3,b_1\}} & \models & a_3 & b_1 & \neg c & d & \neg e & utility(agent_1, 5) & utility(agent_2, 7) \\
w_{\{a_1,b_2\}} & \models & a_1 & b_2 & \neg c & d & \neg e & utility(agent_1, 5) & utility(agent_2, 7) \\
w_{\{a_2,b_2\}} & \models & a_2 & b_2 & \neg c & \neg d & e & utility(agent_1, 9) & utility(agent_2, 2) \\
w_{\{a_3,b_2\}} & \models & a_3 & b_2 & c & \neg d & e & utility(agent_1, 0) & utility(agent_2, 2)
\end{array}
$$

2.3.2 Strategies

Given an ICL theory, agents adopt *strategies*. These are also often called *policies* for the single agent case. These strategies specify which atomic choices an agent chooses from the alternatives controlled by the agent. In general, a strategy can be stochastic where the agent adopts a probability distribution over the alternatives it controls.

Definition 2.15 If $\langle \mathcal{C}, \mathcal{F}, \mathcal{A}, controller, P_0 \rangle$ is a ICL theory and $a \in \mathcal{A}$, $a \neq 0$, then a **strategy for agent** a is a function $P_a : \cup\mathcal{C}_a \rightarrow [0, 1]$ such that

$$\forall \chi \in \mathcal{C}_a \sum_{\alpha \in \chi} P_a(\alpha) = 1.$$

In other words, for each alternative controlled by agent a, P_a is a probability measure over the atomic choices in the alternative.

Definition 2.16 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 $\mathcal{C}_a$ to have probability 1, and the other members thus have probability 0. A pure strategy for agent a thus corresponds to a selector function on $\mathcal{C}_a$.

Definition 2.17 A **strategy profile** is a function from agents (other than nature) into strategies for the agents. If σ is a strategy profile and $a \in \mathcal{A}$, $a \neq 0$ then $\sigma(a)$ is a strategy for agent a. We write $\sigma(a)$ as P_a^σ to emphasize that σ induces a probability over the alternatives controlled by agent a. (We also define $P_0^\sigma = P_0$.)

Thus a strategy profile specifies what each agent will do in the sense of specifying a probability distribution over their alternatives. Given the probability distribution over alternatives, we can derive the expected utility, which is the weighted sum of the utilities of the worlds (worlds weighted by their probability):

Definition 2.18 If ICL theory $\langle \mathcal{C}, \mathcal{F}, \mathcal{A}, controller, P_0 \rangle$ is utility consistent and complete, and σ is a strategy profile, then the **expected utility** for agent $a \neq 0$, under strategy profile σ is

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

(summing over all selector functions τ) where

$$u(\tau, a) = u \text{ iff } w_\tau \models utility(a, u)$$

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

$$p(\sigma, \tau) = \prod_{\chi \in \mathcal{C}} P^\sigma_{controller(\chi)}(\tau(\chi)).$$

$p(\sigma, \tau)$ is the probability of world τ under strategy profile σ, and $u(\tau, a)$ is the utility of world w_τ for agent a.

Note that the expected utility is undefined unless there is a probability distribution over every alternative. In particular, for the multi-agent case, there is no such thing as the expected utility for an agent of a strategy for that agent; the utility for that agent depends on the strategies of the other agents as well.

Each agent wants to choose a strategy that maximise its (expected) utility. For the single agent, finite choice (i.e., a finite number of finite alternatives) case, this definition is straightforward. Each of their (finite number of) strategies has an expected utility, and so they can choose a strategy with a maximal expected utility. For the multiple agent case, an agent has to consider what other agents will choose, and their choice depends on the first agent's choice. How to choose strategies has been well studied in game theory [53; 32; 17]. We can mirror the definitions of game theory; for example, we can define the Nash equilibrium and Pareto optimal (both of which reduce to maximum expected utility in the single agent case) as follows:

Definition 2.19 Given utility consistent and complete ICL theory $\langle \mathcal{C}, \mathcal{F}, \mathcal{A}, controller, P_0 \rangle$, strategy profile σ is a **Nash Equilibrium** if no agent can increase its utility by unilaterally deviating from σ. Formally, σ is a Nash equilibrium if for all agents $a \in \mathcal{A}$, if σ_a is a strategy profile such that $\sigma_a(a') = \sigma(a')$ for all $a' \neq a$ then $\varepsilon(a, \sigma_a) \leq \varepsilon(a, \sigma)$.

In other words, no strategy profile σ_a that's the same as strategy profile σ for all agents other than a is better for a than σ. That is, a cannot be better off by unilaterally deviating from σ.

One of the fundamental results of game theory is that every *finite* game has at least one Nash equilibrium [32; 17]. In general you need non-pure (randomised) strategies for the equilibrium to exist. For a single agent in an uncertain environment, a Nash equilibrium is an optimal decision theoretic strategy.

Definition 2.20 Given utility consistent and complete ICL theory $\langle \mathcal{C}, \mathcal{F}, \mathcal{A}, controller, P_0 \rangle$, strategy profile σ is **Pareto optimal** if no agent can do better without some other agents doing worse. Formally, σ is Pareto optimal if for all strategies σ', if there exists an agent $a \in \mathcal{A}$ such that $\varepsilon(a, \sigma') > \varepsilon(a, \sigma)$ then there exists an agent $a' \in \mathcal{A}$ 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 [36] can be used to represent the independence assumptions of Bayesian networks).

Example 2.21 Here we show how to represent Example 1.1. In the facts we axiomatise utility (this is a utility consistent and complete axiomatisation for both the kicker and the goalie):

$$utility(kicker, 1) \leftarrow goal.$$
$$utility(kicker, 0) \leftarrow \neg goal.$$
$$utility(goalie, 1) \leftarrow goal.$$
$$utility(goalie, 0) \leftarrow \neg goal.$$

In the facts we axiomatise when a goal is scored:[10]

$$goal \leftarrow kicks(D) \wedge jumps(D) \wedge goal\_if\_same\_dir.$$
$$goal \leftarrow kicks(left) \wedge jumps(right) \wedge goal\_if\_kl\_jr.$$
$$goal \leftarrow kicks(right) \wedge jumps(left) \wedge goal\_if\_kr\_jl.$$

In $\mathcal{C}$, we have one alternative owned by $kicker$, namely $\{kicks(right), kicks(left)\}$, one alternative owned by $goalie$, namely $\{jumps(right), jumps(left)\}$, and three alternatives owned by nature, namely: $\{goal\_if\_same\_dir, no\_goal\_if\_same\_dir\}$, $\{goal\_if\_kl\_jr$, $no\_goal\_if\_kl\_jr\}$, and $\{goal\_if\_kr\_jl$, $no\_goal\_if\_kr\_jl\}$ with $P_0(goal\_if\_same\_dir) = 0.9$, $P_0(goal\_if\_kl\_jr) = 0.1$ and $P_0(goal\_if\_kr\_jl) = 0.2$.

Suppose that the goalie is to choose a strategy with $p_g = P_{goalie}(jump(right))$ and the kicker is to choose a strategy with $p_k = P_{kicker}(kick(right))$. In this setup, there are four cases where $goal$ is true; these cases are exclusive, and so we can sum the probabilities. Thus,

$$P(goal) = p_k p_g 0.9 + (1 - p_k)(1 - p_g)0.9 + (1 - p_k)p_g 0.1 + p_k(1 - p_g)0.2$$

The problem for each agent is to choose their probability to maximise their expected utility. So the kicker has to choose p_k to maximise the probability of a goal and the goalie has to choose p_g to minimize the probability of a goal.

In a Nash equilibrium, neither agent can improve its expected utility by unilaterally changing its strategies. Take the kicker's point of view. If there is a randomised strategy, then, as the randomised strategy is a linear combination of the payoffs of the pure strategies, the pure strategies must have the same values (otherwise the kicker can improve its utility by choosing the pure strategy with

[10] The atoms $goal\_if\_same\_dir$, $goal\_if\_kl\_jr$ and $goal\_if\_kr\_jl$ are independent causal hypotheses [36]. These are introduced so that we can have normal logical rules, and independent alternatives.

the higher value). In a randomized equilibrium, the payoff for kicking right and kicking left must be equal. The payoff for kicking right is the above formula with $p_k = 1$, the payoff for kicking left is the formula with $p_k = 0$. These are equal when: $p_g 0.9 + (1 - p_g)0.2 = (1 - p_g)0.9 + p_g 0.1$. Solving for p_g we can derive $p_g = 7/15$. Thus the only time that the kicker would consider a mixed strategy is when the goalie jumps right with probability $\frac{7}{15}$. Using similar reasoning, we can show that the only randomised equilibrium for the goalie is when $p_k = 8/15$. It's easy to show there are no pure strategy equilibria. There is a unique Nash equilibrium with $p_g = \frac{7}{15}, p_k = \frac{8}{15}$. Under this equilibrium the probability of a goal is $\frac{79}{150} = 0.52666$; thus the kicker has a slight advantage (which should be expected, as the goalie is slightly worse when it jumps left).

3 Embedding other formalisms in the ICL

In this section we show how influence diagrams, Markov decision problems (MDPs) and the strategic form of games can be represented in the ICL. We will show rather direct embeddings of these formalisms.

Another embedding should be noted, and that is that probabilistic Horn abduction [36], a restriction of ICL (with only choices by nature, no negation as failure and more restrictions on the rules), can directly represent Bayesian networks [36]. The embedding of influence diagrams is based on this embedding.

3.1 Representing influence diagrams

An influence diagram or decision network [23] is a graphical representation of a decision problem. (See Section 1.4.) We show how to translate an influence diagram into a (single-agent) ICL theory such that there is an isomorphism between the policies of the influence diagram and the strategies of the ICL, with corresponding expected utilities equal. We only consider influence diagrams with a single value node (any other influence diagram can be mapped onto this representation).

Definition 3.1 An influence diagram is a tuple $\langle N, A, \Omega, P, U \rangle$ such that

N is a finite set of nodes, partitioned into the set R of random nodes, the set D of decision nodes and the singleton set $\{V\}$ containing the value node. Random nodes are drawn as ovals, decision nodes as rectangles and the value node as a diamond.

$A \subset N \times N$ is the set of arcs such that $\langle N, A\rangle$ forms an acyclic directed graph (DAG). If $\langle n_i, n_j\rangle \in A$ then n_i is said to be a parent of n_j and n_j is a child of n_i. Define $\pi(n) = \{m : \langle m, n\rangle \in A\}$. That is, $\pi(n)$ is the set of parents of node n. We assume that the value node doesn't have any children.

Ω is a function from $R \cup D$ into sets of variable values. $\Omega(n)$, called the **frame** of node n, is the set of values that the variable associated with node n can take. We extend Ω to cover sets of nodes by $\Omega(\{n_0, \ldots, n_m\}) = \Omega(n_0) \times \ldots \times \Omega(n_m)$.

P is a probability function over the random nodes given their parents. That is, for each $x \in R$, $P(x = v|\pi(x) = w)$ is a non-negative number such that

$$\forall w \sum_{v \in \Omega(x)} P(x = v|\pi(x) = w) = 1$$

The probability is often written simply as $P(x|\pi(x))$ where the values $v \in \Omega(x)$ and $w \in \Omega(\pi(x))$ are derived from context.

$U : \Omega(\pi(V)) \to \Re$. U is the utility function that gives the utility for different values of the parents of V.

The parents of a random node represent probabilistic dependence (as is a Bayesian network [35]). The parents of the value node represent functional dependence; the utility only depends on the values of the parents of the value node. The parents of a decision node represent information available; one value for each parent of the decision node will be known when the decision is made.

If $d_i \in D$, a **decision function** for d_i is a function $\delta_i : \Omega(\pi(d_i)) \to \Omega(d_i)$. If the decision nodes are $\langle d_1, \ldots, d_k\rangle$, a **policy** is a tuple $\langle \delta_1, \ldots, \delta_k\rangle$ where δ_i is a decision function for d_i.

Policy δ induces a conditional probability P_δ on the decision variables defined by

$$P_\delta(d_i|\pi(d_i)) = \begin{cases} 1 & \text{if } \delta_i(\pi(d_i)) = d_i \\ 0 & \text{otherwise} \end{cases}$$

Suppose $R \cup D = \{x_1, \ldots, x_n\}$. The joint distribution given policy δ is:

$$P_\delta(x_1, \ldots, x_n) = \prod_{x_i \in R} P(x_i|\pi(x_i)) \times \prod_{x_j \in D} P_\delta(x_j|\pi(x_j))$$

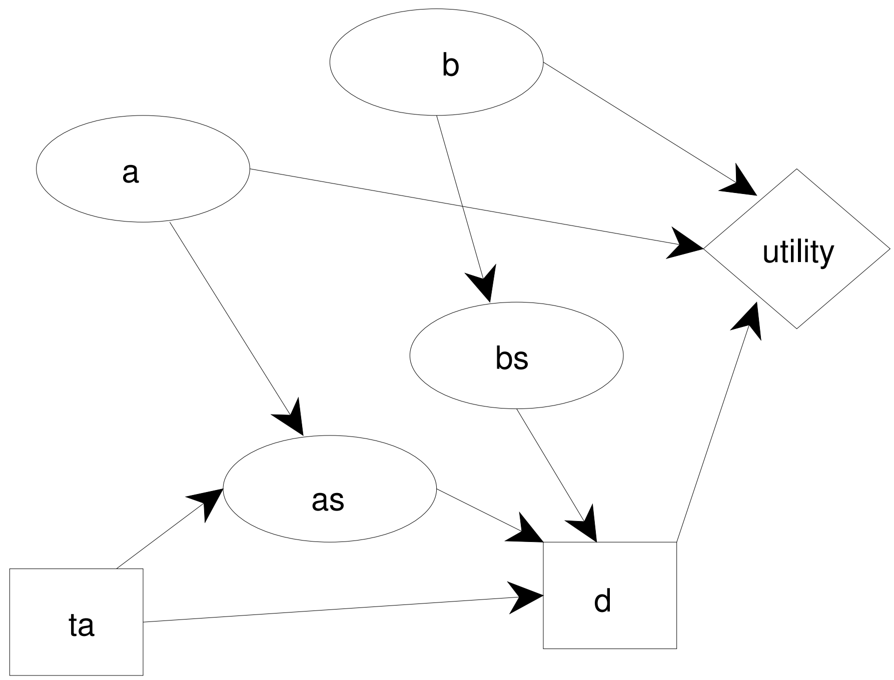

Figure 2: An influence diagram

(what is meant by P_δ should be clear from context).

The expected utility of policy δ is given by

$$\begin{aligned}\varepsilon(\delta) &= \sum_{x_1,\ldots,x_n} P_\delta(x_1,\cdots,x_n) \times U(\pi(V)) \\ &= \sum_{x_1,\ldots,x_n} \prod_{x_i \in R} P(x_i|\pi(x_i)) \times \prod_{x_j \in D} P_\delta(x_j|\pi(x_j)) \times U(\pi(V))\end{aligned}$$

where we are summing over all of the values of variables $x_1,\ldots,x_n$.

Example 3.2 Figure 2 shows an influence diagram with two decision nodes ta and d, four random nodes a, as, b, bs, and one value node $utility$. The intuition for this diagram is that there is one decision d to be made that depends on a and b. bs is a noisy sensor for b and as is a sensor for a that can be controlled by ta.

Associated with the influence diagram (not shown in the diagram) is a frame for each variable, and the conditional probability table for each random variable given its parents. These probability tables for the random nodes are a major source of complexity as their size is exponential in the number of parents of the node.

Suppose the frames are as follows:

$$\begin{aligned}\Omega(ta) &= \{high, low\}\\ \Omega(a) &= \{low, medium, high\}\\ \Omega(as) &= \{pos, neg\}\\ \Omega(b) &= \{pos, neg\}\\ \Omega(bs) &= \{pos, neg\}\\ \Omega(d) &= \{d_1, d_2, d_3\}\end{aligned}$$

There are ten probability distributions to be assigned; one for a, six for as (one for each assignment of values to a and ta, such as $P(as = pos|a = low \wedge ta = high)$), one for b and two for bs.

The mapping of an influence diagram into a (single agent) ICL theory is as follows:[11]

- Random variable x_i has value v_i is represented as the proposition $x_i(v_i)$.[12]
- Random variable x_i with k_i parents $x_{i_1} \ldots x_{i_{k_i}}$ is represented as a rule and exponentially (in k_i) many alternatives. There is one rule:

  $$x_i(V_i) \leftarrow x_{i_1}(V_{i_1}) \wedge \ldots \wedge x_{i_{k_i}}(V_{i_{k_i}}) \wedge c_i(V_i, V_{i_1}, \ldots, V_{i_{k_i}})$$

  For each assignment of values to the x_{i_j}, that is for each $\left\langle v_{i_1}, \ldots, v_{i_{k_i}} \right\rangle \in \Omega(x_{i_1}) \times \cdots \times \Omega(x_{i_{k_i}})$ there is an alternative controlled by nature:

  $$\{c_i(v_1, v_{i_1}, \ldots, v_{i_{k_i}}), \ldots, c_i(v_r, v_{i_1}, \ldots, v_{i_{k_i}})\}$$

  where $\Omega(x_i) = \{v_1, \ldots, v_r\}$.

  The probability of each atomic choice is the value of the corresponding conditional probability:

  $$P_0(c_i(v_1, v_{i_1}, \ldots, v_{i_{k_i}})) = P(x_i = v_1 | x_{i_1} = v_{i_1}, \ldots, x_{i_{k_i}} = v_{i_{k_i}})$$

[11] The mapping for random nodes is the same as the representation of Bayesian networks in probabilistic Horn abduction [36].

[12] We have not used the standard probabilistic notation of $x_i = v_i$ because logicians usually mean something different by equality, namely that two terms denote the same object.

The conditional probabilities on the right hand side are provided as part of the influence diagram. Note that under this mapping there are the same number of alternatives as there are rows in the probability tables for x_i, and the same number of probabilities are provided.

In many cases the probability can be represented more compactly. In particular this occurs when some parents are irrelevant in the context of values to other variables [7].

- Value node with parents $x_{i_1} \ldots x_{i_{k_i}}$ is represented as a rule of the form:

$$utility(agent, v) \leftarrow x_{i_1}(v_{i_1}) \wedge \ldots \wedge x_{i_{k_i}}(v_{i_{k_i}})$$

for each $\left\langle v_{i_1}, \ldots, v_{i_{k_i}} \right\rangle \in \Omega(x_{i_1}) \times \cdots \times \Omega(x_{i_{k_i}})$, where $v = U(v_{i_1}, \ldots, v_{i_{k_i}})$. As with chance nodes, in many cases the value function can be represented more compactly than this.

- Decision variable x_i with k_i parents $x_{i_1} \ldots x_{i_{k_i}}$ is represented as a rule and exponentially (in k_i) many alternatives. There is one rule:

$$x_i(V_i) \leftarrow x_{i_1}(V_{i_1}) \wedge \ldots \wedge x_{i_{k_i}}(V_{i_{k_i}}) \wedge c_i(V_i, V_{i_1}, \ldots, V_{i_{k_i}}).$$

For each $\left\langle v_{i_1}, \ldots, v_{i_{k_i}} \right\rangle \in \Omega(x_{i_1}) \times \cdots \times \Omega(x_{i_{k_i}})$ there is an alternative controlled by the agent:

$$\{c_i(v_1, v_{i_1}, \ldots, v_{i_{k_i}}), \ldots, c_i(v_r, v_{i_1}, \ldots, v_{i_{k_i}})\}$$

where $\Omega(x_i) = \{v_1, \ldots, v_r\}$. Just as the influence diagram policy has to choose a value for each value of the parents we have to choose a value for each alternative. There is a one to one mapping between the alternatives and the values of the parents of a node.

Example 3.3 Continuing Example 3.2, with the influence diagram of Figure 2, variable ta has no parents, therefore there is one value to be chosen. This can be represented as having $\{ta(hi), ta(low)\} \in \mathcal{C}_1$. There are 8 independent choices to be made for d (one for each assignment of values to its parents). This can be represented as the rule:

$$d(DV) \leftarrow ta(TV) \wedge as(AV) \wedge bs(BV) \wedge d\_does(DV, TV, AV, BV)$$

with

$$\{d\_does(d_1, TV, AV, BV), d\_does(d_2, TV, AV, BV), d\_does(d_3, TV, AV, BV)\} \in \mathcal{C}_1$$

for each value of TV, AV, BV.

Theorem 3.4 Given an influence diagram ID and the corresponding ICL theory, defined by the mapping above, there is a correspondence between the policies of the influence diagram and the pure strategies of the ICL theory. The corresponding policies and strategies have the same expected utility.

Proof: A policy of an influence diagram specifies a decision function for each decision node. Each decision function is a function from the values of the parent to the values of the nodes. A decision function, δ_i corresponds to the selection of

$$c_i(\delta_i(v_{i_1}, \ldots, v_{i_{k_i}}), v_{i_1}, \ldots, v_{i_{k_i}})$$

from the corresponding alternative. It is easy to see that different policies correspond to different selections, and that different selections correspond to different policies.

The expected utility of the influence diagram policy δ is:

$$\begin{aligned}\varepsilon(\delta) &= \sum_{x_1,\ldots,x_n} \prod_{x_i \in R} P(x_i|\pi(x_i)) \times \prod_{x_j \in D} P_\delta(x_j|\pi(x_j)) \times U(\pi(V)) \\ &= \sum_{x_1,\ldots,x_n} \prod_{x_i \in R} P_0(c_i(x_i, \pi(x_i))) \times \prod_{x_j \in D} P_1(c_j(x_j, \pi(x_j))) \times U(\pi(V))\end{aligned}$$

where $c_i(x_i, \pi(x_i))$ has the obvious meaning, and P_1 is the probability induced by the policy. This is the expected utility of the PHA theory for the same policy.
$\square$

3.2 Markov Decision Processes

Markov decision processes [44] are models of single-agent stochastic sequential decision problems where a notion of *state* conveys all of the information about the past history.

A **Markov decision process** is defined in terms of a set S of states, a set A of actions, a state transition function $Pr(s_1|s_0, a)$ which specifies the probability that s_1 is the state resulting from carrying out action a in state s, and a reward function $R(s_o, a, s_1)$ that specifies the reward obtained when action a is carried out in s_0 and the resulting state is s_1.

A stationary policy is a selection of an action for each state; what the agent does at any time depends on the state.

We can represent the choice function for an agent, assuming we want stationary policies, as:

$$\forall S \; \{do(a_1, S), \ldots, do(a_m, S)\} \in \mathcal{C}_1$$

where $do(A, S)$ is true if the agent will do action A in state S and $A = \{a_1, \ldots, a_m\}$ is the set of available actions. The agent gets to choose what it does for each state. If we want a non-stationary policy (i.e., the policy depends on the time or stage), we add a time parameter to do.

We also axiomatise the state transition function, which specifies how states transform under actions:

$$state(S', s(T)) \leftarrow state(S, T) \wedge do(A, S) \wedge st\_trans(S, A, S')$$

where $state(S, T)$ is true if the system is in state S at time T, and $st\_trans(S, A, S')$ is true if action A transforms state S into state S'. This is a stochastic transition:

$$\forall S \, \forall A \; \{st\_trans(S, A, s_0), \ldots, st\_trans(S, A, s_n)\} \in \mathcal{C}_0$$

where $\{s_0, \ldots, s_n\}$ is the set of all states. Note that these are alternatives controlled by nature. $P_0(st\_trans(S, A, S'))$ is the probability that state S' will be the result of carrying out action A in state S.

A reward function can be defined in terms of rules of the form:

$$reward(r_i, T) \leftarrow state(s_i, T)$$

for each state s_i and for some number r_i.

We typically don't want to write Markov decision processes by explicitly referring to the states, but instead want to divide the state into propositions (or random variables) [12]. This can reduce the size of the probabilistic assessment necessary. This can be reduced further by the use of rules; these allows us to express structured probability distributions concisely. This concise specification can be exploited for computational gain; Boutilier et. al. [5] exploit the rule (or tree) structure of probability tables for computational gain for MDPs. The ICL representation also allows for the concise axiomatisation in logic, with a well defined semantics, of the dynamics of the system. This is similar to Kanazawa [24], but incorporates a particular, and we claim useful, probability independence.

Example 3.5 Let's axiomatise the structured MDP example of Boutilier et. al.[5] in the independent choice logic. Essentially we can convert the trees into rules, but we don't need separate rules for each action (which is exactly the frame problem [31]).

In this example, there are six state propositions: $loc\_off(T)$, the location of the robot is at the office (as opposed to being at the café) at time T; $wet(T)$, the robot is wet; $umbrella(T)$, the robot is carrying an umbrella; $raining(T)$, it is raining; $rhc(T)$, the robot has coffee; and $uhc(T)$, the user has coffee.

There are four actions: $go(T)$, go to opposite location; $buyC(T)$, buy coffee; $delC(T)$, deliver coffee; and $getU(T)$, get coffee.

We can specify the dynamics using logic, for example, the following clauses define wet and hcu:

$$
\begin{aligned}
&wet(T+1) \leftarrow wet(T) \\
&wet(T+1) \leftarrow go(T) \wedge raining(T) \wedge \neg umbrella(T) \\
&hcu(T+1) \leftarrow hcu(T) \\
&hcu(T+1) \leftarrow delC(T) \wedge \neg hcu(T) \wedge loc\_off(T) \wedge hcr(T) \wedge delCsucceeds(T)
\end{aligned}
$$

where $\forall T\ \{delCsucceeds(T), delCfails(T)\} \in \mathcal{C}_0$, and $\forall T\ P_0(delCsucceeds(T)) = 0.8$. We can also define the reward function using rules:

$$
\begin{aligned}
&reward(1.0, T) \leftarrow hcu(T) \wedge \neg wet(T) \\
&reward(0.9, T) \leftarrow hcu(T) \wedge wet(T) \\
&reward(0.1, T) \leftarrow \neg hcu(T) \wedge \neg wet(T) \\
&reward(0.0, T) \leftarrow \neg hcu(T) \wedge wet(T)
\end{aligned}
$$

For finite horizon problems, the value can be specified in terms of rules. For example,

$$
\begin{aligned}
&valueto(R+U, T+1) \leftarrow reward(R, T) \wedge valueto(U, T). \\
&valueto(0, 0).
\end{aligned}
$$

where $valueto(V, T)$ is true if V is the sum of the rewards up to time T.

For infinite horizon problems, it is not so simple. You could imagine writing, for the discounted reward function [44]:

$$
value(R + U \times \gamma, T) \leftarrow reward(R, T) \wedge value(U, T+1)
$$

where γ is the discount factor. However, such rules are problematic as the recursion doesn't terminate. It is probably better to define the value external to the logic. Having the specification of the value function separate from the other parts of the problem specification, as is traditionally done in MDPs, doesn't seem to be too problematic.

3.3 The strategic form of a game

The most direct connection of the ICL is to the strategic form of a game (see Section 5.5 for a comparison to the extensive form of a game).

The strategic form of a game [32; 17] is a tuple $\langle \mathcal{A}, \Sigma, u \rangle$ where

$\mathcal{A}$ is a non-empty set of players (agents),

Σ is a function from agents into non-empty sets (of pure strategies). Thus $\Sigma(a)$ is the set of all pure strategies for agent a.

u is a function

$$u : \mathcal{A} \rightarrow \left(\bigtimes_{a \in \mathcal{A}} \Sigma(a) \rightarrow \Re \right)$$

Suppose $\mathcal{A} = \{a_1, \ldots a_n\}$ and $a \in \mathcal{A}$. $u(a)$ is a function that given an n-tuple of strategies, one for each agent in $\mathcal{A}$, returns the utility for agent a under this strategy profile. Thus $u(a)(\langle \sigma_{a_1}, \ldots, \sigma_{a_n} \rangle)$ where $\sigma_{a_i} \in \Sigma(a_i)$ is the von Neumann–Morgenstern utility for agent a when each player a_i chooses strategy σ_{a_i}.

The general idea is that each player chooses a strategy which specifies what it will do under all contingencies. Following a complete play (specified by each player's strategy) each player receives a utility.

Note that there are two different forms which we treat as the same here. One is where nature isn't an agent, and all of the payoffs are expectations (averaging over nature's choices). The second is where nature is a player, and has a probability distribution over its strategies; this is called the Bayesian form of a game [17], where the players have partial information about nature's choice. The private information about nature's move is called the player's *type*. The Bayesian form of a game assumes that each player chooses its strategy after it learns its type. Such a distinction is beyond the scope of this paper, as we don't consider how or when strategies are computed (for example, whether they are computed online or offline).

The ICL can be seen as a particular representation for the strategic or the Bayesian form of a game. The set of agents is the same. We divide the space of strategies for players into independent choices (i.e, we allow more structure in the strategy space) and use a logic program to axiomatise the u function.

There is a direct mapping of the strategic or Bayesian form of a game into an ICL theory: strategic game $\langle \mathcal{A}, \Sigma, u\rangle$ is mapped into the multi-agent ICL theory $\langle \mathcal{C}, \mathcal{F}, \mathcal{A}, controller, P_0\rangle$, where A is the same set, $\mathcal{C}$ is the set

$$\{\{do(a_i, \sigma_{a_i}) : \sigma_{a_i} \text{ is a strategy for } a_i\} : a_i \in \mathcal{A}\}$$

where $do(a_i, \sigma_{a_i})$ is an atom that says that agent a_i is adopting strategy σ_{a_i} (all we need is a name for each strategy), $controller$ is the function $\{do(a_i, \sigma_{a_i}), \ldots\} \mapsto a_i$, P_0 is the probability distribution over types (for the Bayesian model of a game) and $\mathcal{F}$ is is set of rules of the form

$$utility(a, util) \leftarrow do(a_1, \sigma_{a_1}) \wedge \ldots \wedge do(a_n, \sigma_{a_n})$$

where $util = u(a)(\langle \sigma_{a_1}, \ldots, \sigma_{a_n}\rangle)$.

This mapping has trivialised the fact that there is a lot hidden in the structure of the strategies. We have assumed we can name the strategies and say what follows from them. For simple games we may be able to, but for most realistic situations we want to be able to specify the choices at a lower level of detail, and be able to control the selection of components of strategies. The consequence of a number of different agents choosing strategies should involve reasoning about the building blocks of the strategies (what is done under what circumstances), and reasoning about the dynamics of the domain to determine the consequences of the actions.

4 The Dynamic ICL

The ICL presented so far is only good at representing problems where the the decision problem can be statically expressed (even if the problem to be solved involves dynamics and change). Like the strategic form of a game, the building blocks of the strategies have to be constructed ahead of time. For example, for the influence diagram mapping we had to create an alternative corresponding to each assignment of values to parents, rather than creating the appropriate rules for defining the policy on the fly.

There are a number of problems with this:

- What the agent will do (or attempt to do) is buried within the representation. The alternatives are at a lower level than the choices faced by the agent; they specify what the agent will do under each contingency. While we can represent the dynamic structure structure of reasoning and acting, the formalism presented so far gives us no help in doing so.

- We have to create an alternative for each independent choice that the agent could make; that is, we have to a priori divide up information states for the decision. The problem is that the a priori division needs to be at the finest level of detail. For example, although some decision d may have much information available when the decision is made (in the influence diagram d may have many parents), the specification of what the agent should do may not require all of the distinctions of the information state. There may be a more concise encoding of the policy. Just as we have used rules as a concise specification of probability distributions, we may like to express the policy for an agent as rules.

- We may want to create alternatives on the fly; what options are available to an agent and what information it knows may depend on the context, and it may be more economical to create alternatives as needed, rather than having to anticipate all of them as part of a strategy.

- We want to reason about the program the agent used to compute an action rather than just the action itself [48].

We want to build a representation upon a more natural specification of dynamic systems. We will extend the ICL to the *dynamic* ICL logic that is slightly more complicated, but arguably more natural. We model the dynamics of the world rather than the structure of the choices.

The dynamic ICL is more like a representation for the extensive form of a game than a representation for the strategic form of a game; it will tell us how to construct the appropriate game/decision tree (see Section 5.5). This will be done without losing the advantages of the ICL, namely, the embedding of the independence of Bayesian networks, the ability to represent structured decision tables, and the use of logical variables.

We build the theory upon a general model of agents interacting in an environment. This is important as it places the ICL within a wider theoretical context, and introduces the notions of traces, transductions, state and sensing.

4.1 Dynamical Systems

Modelling dynamical systems [29; 13] is common in many areas of science, from mechanical engineering to economics to ecology.

We assume a time structure $\mathcal{T}$, that is totally ordered and has a metric over intervals. $\mathcal{T}$ can either be continuous or discrete; for this paper we will consider discrete time. (See [39] for a development of continuous time in this framework.) A **trace** is a function from $\mathcal{T}$ into some domain $\mathcal{A}$.

A **transduction** is a function from (input) traces into (output) traces that is "causal" in the sense that the output at time t can only depend in inputs at times t' where $t' \leq t$ (i.e., the output at some time is a function of the input history up to that time). An agent will be a specification of a transduction. Transductions form a general abstraction of dynamic agents and dynamic systems in general [57; 47; 39], although they don't adequately handle the case of nondeterministic agents[13].

The **state** of an agent is that information that needs to be remembered in order for the output to be a function of the state and the current inputs. At one extreme a state can contain the entire history of the agent. At the other extreme an agent can have no state and just react to its current inputs.

4.2 Agent Structure

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 input and outputs; there are some values that they have no access to, and some internal values that only they can access.

We will model agents as a logic program that specifies how the outputs are entailed by certain inputs [39]. This logic program can use the internal values and sense values but can't use those values the agent has no access to (i.e., can't sense or otherwise determine).

In modelling agents we have to be careful about a number of things:

[13]With deterministic agents, only the input history is needed. Nondeterministic agents (agents need to be nondeterministic if they inhabit an environment with other (competing) agents, or if they have limited memory), need to be able to recall their inputs as well as choice commitments made. For example, for an agent with no inputs to implement $(a; b)|(c; d)$ where "|" is nondeterministic choice and ";" is sequential composition, at the second time step the agent has to be able to recall what it chose in the first time step.

- What are the inputs and what are the outputs? When we have noisy sensors and actuators with slop and failure, we can't condition on the values in the world, but only on what our sensors tell us. We have to be able to model what an agent can observe (and how it relates to the world), and what an agent controls (and how it relates to what the agent actually does).

- We have to make sure that the agent can actually carry out the policy specified for the agent. An executable policy can't depend on events the agent can't observe, or aren't under the agent's control.

- The logic programs are *models* of the agents. They aren't the agents themselves. We want to be able to model many different sorts of agents, both natural and artificial. We want to be able to model agents that are implemented as simple logic circuits or in some traditional programming language, for example. We want to use the same language to model agents we design, agents that our agent may encounter, and the environment. We also want to be able to model how long the agent will take to execute an action (including the time to execute the program to choose the action) [48]. This doesn't mean we couldn't run this specification to make an agent (but it will be a different agent, for example, if we forward chain on the axioms than if we backward chain on the axioms; they will have very different timing properties).

We distinguish the "controller" and the "plant" of an agent [13]. (See Figure 3). The **controller** is that part that we have to optimize; it receives digital signals ("observations") and outputs digital "controls" or "actuator commands". The **plant** or **body** is the physical embodiment of the agent that includes input devices such as cameras, microphones, radio receivers as well as wheels, limbs and transmitters. The plant receives "percepts" from the environment (e.g., sound, light, radio signals), and sends observations to the controller. The observations are usually correlated with the percepts received, but are typically not identical as sensors are noisy. The plant also receives controls from the controller and makes actions in the environment (e.g., actually moving, sending messages).

Multiple agents will all interact through the environment; the only way for agents to communicate is through the environment[14] and they all act in the en-

[14]If there is some form of direct communication channel, then this is also modelled as part of the environment. This makes modelling more uniform and allows us to model noise and failure in communication.

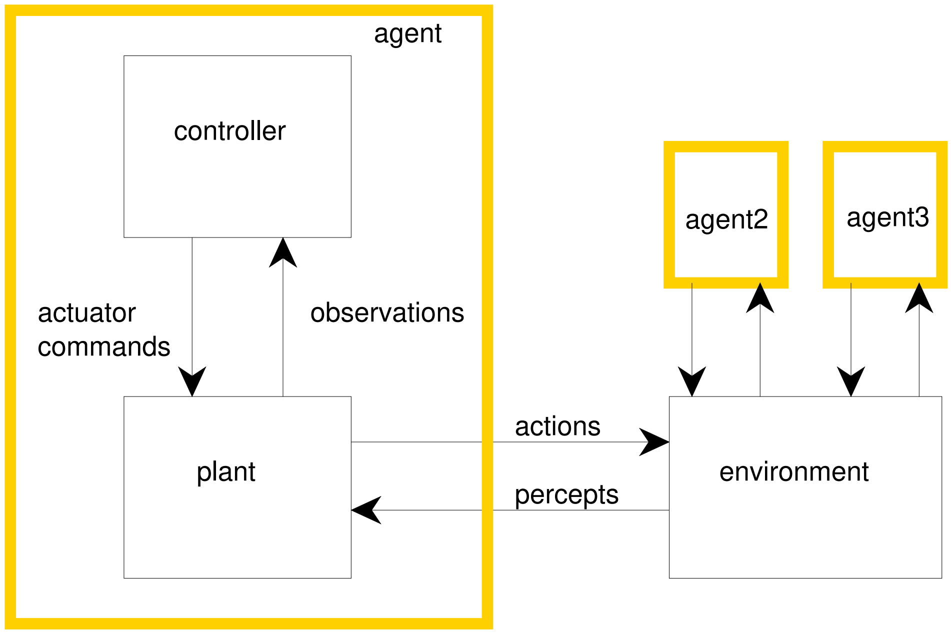

Figure 3: An agent acting in an environment

vironment (as in Figure 3). There may be many agents, all of which carry out actions in the environment and receive percepts from the environment.

As far as the outside world (including other agents) is concerned, an agent receives *percepts* such as messages, light, sounds, etc., and performs *actions* such as moving limbs, sending messages. Thus other agents will tend to group the controller and the plant together as “the agent” (or “the robot” for physical implementations).

As far as the controller is concerned, it can group the plant and the environment together. It receives *observations*, and outputs *controls*. The distinction between the plant and the environment is essentially arbitrary; we usually make the distinction because we often build controllers for particular plants, but for more general environments. While the distinction between the controller and the plant may seem to be arbitrary to an outside observer, when building an agent we have to commit to a particular division in order to construct a controller. (Typically we want a hierarchy of controllers and plants [1; 10; 55], but this is beyond the scope of this paper.)

For the rest of this paper we take the controller's point of view; we assume a controller receives definitive observations and can issue controls to the plant. The plant will be modelled as part of the environment. In particular, uncertainty in observations (observations may not always reflect what's true in the environment) will be modelled in terms of rules that depend on the percepts as well as nature's choices.

4.3 Agent Specification Module

The agent specification module takes the controller's point of view; the inputs will be observations and the outputs will be controls. The agent will be able to condition on the observations and will need to choose values for the controls.

Agent specification modules will allow us to modularise our knowledge, and use common computer science techniques like information hiding, abstract data types and modular program design. We will generalise the ICL to the "dynamic ICL" through the use of agent specification modules; this will allow a more concise representation of decision problems.

Definition 4.1 An **agent specification module** for agent $a \neq 0$, written ASM_a, is a tuple $\langle \mathcal{C}_a, \mathcal{O}_a, \pi \rangle$ where

$\mathcal{C}_a$ is a set of alternatives controlled by a. This will be the set of possible actuator commands for the agent; the agent can attempt to "do" one of the actuator commands in each element of $\mathcal{C}_a$.

$\mathcal{O}_a$, the **observables**, is a set of sets of ground atomic formulae. Elements of $\mathcal{O}_a$ are called **observation alternatives**; elements of observation alternatives are called **atomic observations**.

π, the **observable function**, is a function $\pi : \mathcal{C}_a \rightarrow 2^{\mathcal{O}_a}$. The idea is that when the agent decides which element of alternative $\chi \in \mathcal{C}_a$ to choose, it will have observed one atomic observation from each observation alternative in $\pi(\chi)$. Elements of $\pi(\chi)$ are the information available to the agent when it has to choose an element of χ. $\pi(\chi)$ corresponds to the parents of a decision node χ in an influence diagram.

The following definition mirrors the analogous definition from game theory:

Definition 4.2 Agent a has **perfect recall** (or is **no-forgetting**) if it remembers all of its previous observations and previous actions. Formally this means that the element of $\mathcal{C}_a$ are totally ordered and if $\chi_1 \in \mathcal{C}_a$, $\chi_2 \in \mathcal{C}_a$, $\chi_1 < \chi_2$ then $\chi_1 \in \pi(\chi_2)$ and $\pi(\chi_1) \subset \pi(\chi_2)$.

A dynamic ICL theory consists of an agent specification module for each agent, and a logic program, plus stochastic choices to axiomatise what follows from the agent's choices:

Definition 4.3 A **dynamic independent choice logic theory** is a tuple $\langle \mathcal{A}, \mathcal{C}_0, \mathcal{F}_0, P_0, ASM \rangle$ where

$\mathcal{A}$ is a finite set of agents containing a distinguished agent 0 called "nature"

$\mathcal{C}_0$, **nature's choice space**, is a choice space with alternatives controlled by nature

$\mathcal{F}_0$, nature's **facts**, is a logic program such that no atomic choice unifies with the head of any rule

P_0 is a function $\cup\mathcal{C}_0 \rightarrow [0,1]$ such that $\forall \chi \in \mathcal{C}_0 \ \sum_{\alpha \in \chi} P_0(\alpha) = 1$

ASM is a function on $A - \{0\}$ such that ASM_a is an agent specification module for agent a

such that $\mathcal{F}_0$ is acyclic with an acyclic ordering where $\forall a \in \mathcal{A}$, $\forall \chi \in \mathcal{C}_a$, $\forall \alpha \in \chi$, $\forall O \in \pi(\chi)$, $\forall \alpha' \in O$, $\alpha' < \alpha$ in the acyclic ordering. That is, there is an acyclic ordering where the actions are after their corresponding observables.

Note that $\langle \mathcal{F}_0, \mathcal{C}_0, P_0 \rangle$ will correspond to a particular (stochastic) strategy for nature (see Definition 4.10). It's a specification of what choices nature will make. This specifies the stochastic dynamics of the system.

We have to make sure that the observables for each agent really cover the possibilities and really are alternatives.

Definition 4.4 Given dynamic ICL theory $\langle \mathcal{A}, \mathcal{C}_0, \mathcal{F}_0, P_0, ASM \rangle$, set Γ of ground formulae is **non-exclusive** if there exists a choice function τ on $\bigcup_{a \in \mathcal{A}} \mathcal{C}_a$ and there exists $\alpha_1 \in \Gamma$, $\alpha_2 \in \Gamma$, $\alpha_1 \neq \alpha_2$ such that $\mathcal{R}(\tau) \cup \mathcal{F}_0 \mid\!\sim \alpha_1 \wedge \alpha_2$. Otherwise Γ is **exclusive**. Γ is **covering** if for every choice function τ on $\bigcup_{a \in \mathcal{A}} \mathcal{C}_a$, there is $\alpha \in \Gamma$ such that $\mathcal{R}(\tau) \cup \mathcal{F}_0 \mid\!\sim \alpha$.

For example, every choice alternative is exclusive and covering. Every set of the form $\{a, \neg a\}$ is exclusive and covering. A less trivial example is: given $\mathcal{C} = \{\{a, b\}, \{c, d\}, \{e, f\}\}$, and $\mathcal{F} = \{g \leftarrow a, h \leftarrow b \wedge c, i \leftarrow b \wedge d\}$, $\{g, h, i\}$ is an exclusive and covering set. If $g \leftarrow e$ were added to $\mathcal{F}$ then $\{g, h, i\}$ would no longer be exclusive.

Definition 4.5 A dynamic ICL theory is **observation consistent** if for every $a \in \mathcal{A}$, every element of $\mathcal{O}_a$ is exclusive with respect to the dynamic ICL theory. A dynamic ICL theory is **observation complete** if for every $a \in \mathcal{A}$, every element of $\mathcal{O}_a$ is covering.

The above definitions are to make sure that we can treat the elements of $\mathcal{O}$ as random variables. Unlike elements of $\mathcal{C}$, they aren't exclusive and covering by definition. We will always require a theory to be observation consistent, but, when we have negation as failure in the logic [37], we will not require the theory to be observation complete (there may be an extra, unnamed element of each element of $\mathcal{O}$). Note that observation consistency isn't a severe restriction; we can always make $\mathcal{O}$ a set of singleton sets, but then we can't exploit the structure of observations.

Example 4.6 *For example, suppose we have $\{high, medium, low\}$ as an observation alternative. We want the theory to never allow choices of the agents to entail both $high$ and $medium$. This means that a strategy can be specified as a function from this set into actuator settings. If these values were not exclusive, we could make sets $\{high, \neg high\}$, $\{medium, \neg medium\}$, and $\{low, \neg low\}$ into observation alternatives (which are each exclusive and covering), but this would mean a strategy would be a function from the cross product of these into actuator settings. If $\{high, medium, low\}$ isn't covering, we also have to cover the case $\neg high \wedge \neg medium \wedge \neg low$ in defining a strategy.*

The general idea is that the agent will always observe one element of each member of $\pi(\chi)$ *before* choosing one element of χ. The acyclicity restrictions and the observation completeness and consistency requirements above ensure this temporal ordering is possible.

In this paper we assume all our theories are observation consistent and complete.

4.4 Pure Strategies

A pure strategy is a specification of what an agent will do based on what it observes. This strategy is represented as a logic program. There are restrictions on the logic program to ensure an agent can't condition on values to which it doesn't have access.

Definition 4.7 If $\langle \mathcal{C}_a, \mathcal{O}_a, \pi_a \rangle$ is an agent specification module for agent $a \in \mathcal{A}$, then a **pure strategy** for agent a is a logic program $\mathcal{F}_a$ such that

- $\mathcal{F}_a$ is acyclic with an acyclic ordering such that, for every $\chi \in \mathcal{C}_a$, every element of each element of $\pi_a(\chi)$ is before every element of χ. That is, the agent can observe *before* making the decision.

- For every $\chi \in \mathcal{C}_a$, and for every selection function $\tau_{\pi(\chi)}$ on $\pi(\chi)$, there is a unique $\alpha \in \chi$ that is true in the unique stable model of $\mathcal{F}_a \cup \mathcal{R}(\tau_{\pi(\chi)})$. That is, whatever is observed, the logic program specifies what the agent will do.

- The heads of rules in $\mathcal{F}_a$ only unify with either local atoms (that don't unify with atoms in the agent specification module of any other agent or in $\mathcal{F}_0$) or members of choice alternatives in χ_a. Thus $\mathcal{F}_a$ can only be used to imply alternatives owned by agent a, perhaps with some local atoms as intermediaries.

- For each $\chi \in \mathcal{C}_a$, the only formulae that can appear in the bodies of rules in $\mathcal{F}_a$ to prove an element of χ are: elements of members of $\pi(\chi)$, local atoms and atoms whose definition doesn't depend on the choices of other agents (and formulae built from these). While we don't want to restrict the complexity of programs to compute the choice from an element of χ, the choice can't depend on values that the agent can't observe or otherwise compute.

Thus a strategy for an agent is just a program to specify what the agent will do based on what information it receives.

Definition 4.8 Given a dynamic ICL theory, a **(pure) strategy profile** is a selection of one (pure) strategy for each agent (other than nature). Thus a strategy profile is a logic program $\mathcal{F} = \bigcup_{a \in \mathcal{A}} \mathcal{F}_a$ that specifies what each agent will attempt to do.

There are two (equivalent) ways to define the semantics. One it to have a possible world for each selection of an element from each alternative controlled by nature, and to have $\mathcal{F}$ specify what's true in each world. In this case, the probability of a world is independent of the strategy, but a strategy profile specifies what's true in each world. The second is to have a possible world for each selection of one element from each alternative. In this case, what's true in a world doesn't depend on the strategy profile chosen, but the probability of a world does. The second has many possible worlds with zero probability that were not created in the first scenario. We will define the first method formally here.

Definition 4.9 If dynamic ICL theory $\langle \mathcal{A}, \mathcal{C}_0, \mathcal{F}_0, P_0, ASM \rangle$ is utility complete, and $\mathcal{F}$ is a pure strategy profile, then the **expected utility** of strategy profile $\mathcal{F}$ for agent a is

$$\varepsilon(a, \mathcal{F}) = \sum_{\tau} p(\tau) \times u(\tau, a, \mathcal{F})$$

(summing over all selector functions τ on $\mathcal{C}_0$) where

$$u(\tau, a, \mathcal{F}) = u \quad \text{if} \quad \mathcal{R}(\tau) \cup \mathcal{F} \mid\!\sim utility(a, u)$$

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

$$p(\tau) = \prod_{\chi \in \mathcal{C}_0} P_0(\tau(\chi))$$

$u(\tau, a, \mathcal{F})$ is the utility of world w_τ for agent a under strategy profile $\mathcal{F}$. $p(\tau)$ is the probability of world τ.

4.5 Stochastic Strategies

As in the goal kick example above, it's often desirable for agents to adopt random strategies. In this section we define random (stochastic) strategies. The general idea is that a random strategy is a probability distribution over pure strategies.

Definition 4.10 If $\langle \mathcal{C}_a, \mathcal{O}_a, \pi_a \rangle$ is an agent specification module for agent $a \in \mathcal{A}$, then a **(stochastic) strategy** for agent a is a tuple $\langle \mathcal{F}_a, \mathcal{C}'_a, P_a \rangle$ where $\mathcal{C}'_a$ is a choice space whose atomic choices do not appear outside of this strategy, P_a is a probability distribution over each element of $\mathcal{C}'_a$ (i.e., $\forall \chi \in \mathcal{C}'_a,\ \forall \alpha \in \chi,\ 0 \leq P_a(\alpha) \leq 1$ and $\sum_{\alpha \in \chi} P(\alpha) = 1$), and $\mathcal{F}$ is a logic program such that for all selector functions τ_a on $\mathcal{C}'_a$, $\mathcal{F} \cup \mathcal{R}(\tau_a)$ is a pure strategy (the element of $\mathcal{R}(\tau_a)$ will be local atoms in the strategy).

Example 4.11 Suppose $\langle \{\{up, down, left\}\}, \{\{high, medium, low\}\}, \pi \rangle$, where $\pi(\{up, down, left\}) = \{\{high, medium, low\}\}$ is an agent specification module for agent a. That is, a can do one of $\{up, down, left\}$, and when it has to act, it will know which of $\{high, medium, low\}$ is true. One stochastic strategy could have facts:

$$
\begin{aligned}
&up \leftarrow high \wedge uh,\\
&down \leftarrow high \wedge dh,\\
&left \leftarrow high \wedge lh,\\
&up \leftarrow medium \wedge um,\\
&down \leftarrow medium \wedge dm,\\
&left \leftarrow low,
\end{aligned}
$$

choice space $\mathcal{C}'_a = \{\{uh, dh, lh\}, \{um, dm\}\}$, and P_a given by $P_a(uh) = 0.7$, $P_a(dh) = 0.2$, $P_a(lh) = 0.1$, $P_a(um) = 0.6$, and $P_a(mh) = 0.4$.

Definition 4.12 A strategy profile σ is an assignment of a stochastic strategy for each agent (i.e., for each $a \in \mathcal{A}$, $\sigma(a)$ is a stochastic strategy). If σ is a strategy profile, define $\mathcal{F}_a^\sigma$ to be the first component of $\sigma(a)$, $\mathcal{C}_a^\sigma$ to be the second component of $\sigma(a)$, and P_a^σ to be the third component of $\sigma(a)$.

It remains to define the expected value of a strategy profile.

Definition 4.13 If dynamic ICL theory $\langle \mathcal{A}, \mathcal{C}_0, \mathcal{F}_0, P_0, ASM \rangle$ is utility consistent and complete, and σ is a stochastic strategy profile, then the **expected utility** σ for agent a is

$$\varepsilon(a, \sigma) = \sum_\tau p(\tau, \sigma) \times u(\tau, a, \sigma)$$

(summing over all selector functions τ on $\mathcal{C}_0 \cup \bigcup_{a \in \mathcal{A}} \mathcal{C}_a^\sigma$) where

$$u(\tau, a, \sigma) = u \quad \text{if} \quad \mathcal{R}(\tau) \cup \mathcal{F}^\sigma \mid\!\sim utility(a, u)$$

where $\mathcal{F}^\sigma = \bigcup_{a \in \mathcal{A}} \mathcal{F}_a^\sigma$ (u is well defined as the theory is utility consistent and complete), and

$$p(\tau, \sigma) = \prod_{a \in \mathcal{A}} \prod_{\chi \in \mathcal{C}_a^\sigma} P_a^\sigma(\tau(\chi))$$

$u(\tau, a, \mathcal{F})$ is the utility of world w_τ for agent a under strategy profile $\mathcal{F}$. $p(\tau, \sigma)$ is the probability of world τ under strategy profile σ.

5 Discussion

In this section we discuss some modelling issues for the dynamic ICL. We first discuss how to model information-producing actions, how to model noisy sensors and actuators, what it means to execute a stochastic strategy, and finally the relationship to the extensive form of a game. Section 6 presents some detailed examples.

5.1 Passive Sensors and Information Seeking Actions

The observations represent passive sensors that (at each time) receive values from the environment (one value from each observation alternative). We also don't distinguish between information-producing actions and actions that "change the world"; there is only one type of action. The nature module will specify the consequences 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's 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's 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's also important is that the agent can only condition on its sense values or on values derived from these. The agent can't 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.

Section 6.1 gives a detailed example and discussion of modelling passive sensors and information seeking actions.

5.2 Noisy Sensors and Actuators

There is a straightforward way to model noisy sensors and actuators. This follows the distinction between the plant and the environment depicted in Figure 3. The general idea is to axiomatise how the observations are a function of the percepts plus noise. Similarly we can axiomatise how the actions of the agent are a function of the controls plus noise (for slop, errors, slippage, etc).

We can divide up the noise into systematic errors (e.g., that the sensor is actually broken, and always makes the same error) and intermittent noise (the independent error for each reading), and a continuum in between. For example, consider a sensor for checking road speeds on a highway [16]:

Example 5.1 In this example we show how to axiomatise a noisy sensor that can break down. The sensor is either working or not. If it's working there is some "normal" error from the true reading. If the sensor is not working it produces some reading at random (independent of the actual velocity).

$$
\begin{aligned}
&sense\_velocity(V + DV, T) \leftarrow \\
&\quad velocity\_sensor\_OK(T) \wedge \\
&\quad velocity(V, T) \wedge \\
&\quad normal\_error(DV, T). \\
&sense\_velocity(EV, T) \leftarrow \\
&\quad \neg velocity\_sensor\_OK(T) \wedge \\
&\quad error\_reading(DV, T).
\end{aligned}
$$

We need to have a probability distribution over the normal errors. For example if the errors are discrete in 10km/h steps, we can specify something like:

$$
\begin{aligned}
&\forall T \{normal\_error(DV, T) : DV \in \{-30, -20, -10, 0, 10, 20, 30\}\} \in \mathcal{C}_0 \\
&P_0(normal\_error(-30, T)) = 0.01 \\
&P_0(normal\_error(-20, T)) = 0.03 \\
&P_0(normal\_error(-10, T)) = 0.06 \\
&P_0(normal\_error(0, T)) = 0.8 \\
&P_0(normal\_error(10, T)) = 0.06 \\
&P_0(normal\_error(30, T)) = 0.03 \\
&P_0(normal\_error(30, T)) = 0.01
\end{aligned}
$$

Similarly we can define $error\_reading$ which provides a probability distribution over error readings. There is nothing in principle that prevents us from having a non-discrete distribution, such as a normal (Gaussian) distribution over errors. We have not presented that here as the mathematics is more complicated; we would need to consider measurable sets of speeds rather than the speed themselves.

Whether the sensor is working at some time isn't independent of whether it's working at some other time. We need to axiomatise the dynamics of sensor failure (i.e., we need to specify the probability distribution over time). The sensor can break at any time; suppose it has a 2% chance of breaking at any time when it had been working and a 5% chance of being fixed up when it was broken (we could also axiomatise a more complicated dynamics of how the sensor can get fixed, but this will show the main point). $velocity\_sensor\_OK$ can be axiomatised as:

$$
\begin{aligned}
&velocity\_sensor\_OK(T+1) \leftarrow \\
&\quad velocity\_sensor\_OK(T) \wedge \\
&\quad \neg velocity\_sensor\_breaks(T). \\
&velocity\_sensor\_OK(T+1) \leftarrow \\
&\quad \neg velocity\_sensor\_OK(T) \wedge \\
&\quad velocity\_sensor\_fixed(T).
\end{aligned}
$$

$$\forall T \{velocity\_sensor\_breaks(T), velocity\_sensor\_remains\_OK(T)\} \in \mathcal{C}_0$$

$$\forall T \{velocity\_sensor\_fixed(T), velocity\_sensor\_remains\_broken(T)\} \in \mathcal{C}_0$$

with $P_0(velocity\_sensor\_breaks(T)) = 0.02$ and $P_0(velocity\_sensor\_fixed) = 0.05$. In addition, we need to define the initial value of $velocity\_sensor\_OK$, for example as a member of an alternative controlled by nature.

5.3 Executing a strategy

What does it mean for an agent to execute a strategy? If the strategy for the agent is pure, then it will tell the agent what to do based on its sensor values. The agent will "do" the unambiguous actions that are entailed. If the strategy for the agent isn't pure, then there may be a number of things that the agent could attempt to do based on its inputs. To follow a strategy it should pick the actions randomly according to the distribution specified in the strategy.

Picking strategies at random doesn't mean that there must be a random number generator in the robot (or access to some really random quantum phenomenon), although it could. For example, for the soccer playing robots we could compile in the randomness, by choosing offline whether it should go right or left according to the random strategy. If we did this then we would have to hide our design from our opponent designer and replace our robot after a single penalty kick. An alternative would be to have two non-random robots, where we choose one at random for each penalty kick. These two robots, together with the choosing mechanism, can be seen as one randomizing agent. The most important property is that the other agent isn't able to predict what our agent will do.

It's silly to think of an agent cheating, by not choosing from the random distribution. This is particularly the case when we consider that the agent gets to choose whichever strategy it wants. Cheating with one strategy is the same as choosing a different strategy. We will then consider it to be *that* strategy that the agent is carrying out. This doesn't mean that an agent can't lie about what strategy it's carrying out. What an agent says about what it will do will be another action of the agent, and it can do whatever it wants.

5.4 Non-deterministic Actions and the Frame Problem

There has been much work on logical specifications of actions and change. The specification of actions that solves the frame problem is well understood for the deterministic case with complete knowledge [27; 50; 45]. These axiomatisations assume the closure of the axiomatisation for change. See [2] for a detailed description of axiomatising change with acyclic programs.

What's added in this work is a way to handle non-deterministic actions and partial knowledge. The central idea is that determinism and complete knowledge (as assumed in negation as failure) occurs for each world. We can have a distribution over worlds. When we have uncertainty, it's useful to consider the question of "who chooses the value". Often it's random, but often it's another agent. What we require is that the axiomatiser resolves this ambiguity.

Example 5.2 We can axiomatise a coin toss, where, when a coin is tossed, it lands heads 50% of the time and tails 50% of the time; when it isn't tossed, it remains in the state (heads or tails) it was before:

$$\begin{aligned}
&heads(C, T+1) \leftarrow \\
&\qquad tossed(C, T) \wedge \\
&\qquad heads_turns_up(C, T). \\
&heads(C, T+1) \leftarrow \\
&\qquad \neg tossed(C, T) \wedge \\
&\qquad heads(C, T). \\
&tails(C, T) \leftarrow \neg heads(C, T)
\end{aligned}$$

where $heads(C, T)$ is true if coin C has heads up at time T. $heads\_turns\_up(C, T)$ is true at time T if heads would turn up on coin C is it were tossed at time T. It's defined as:

$$\forall C\ \forall T\ \{heads\_turns\_up(C, T), tails\_turns\_up(C, T)\} \in \mathcal{C}_0$$

with $\forall C\, \forall T\ P_0(heads\_turns\_up(C, T)) = 0.5$.

Many of the papers that present a solution to the frame problem [27; 50; 45; 2] use the situation calculus for representing change, rather than the discrete time model used here. See [40] for a description of how the situation calculus can be combined with the independent choice logic.

5.5 The Extensive Form of a Game

The extensive form of a game [53; 32; 17] is a representation of a game in terms of a *game tree*, a generalisation of a decision tree to include different agents making decisions at each node, and having information sets of nodes that agents can't distinguish.

"The extensive form of a game contains the following information:

1. the set of players
2. the order of the moves, i.e., who moves when
3. the players' payoffs as a function of the moves that were made
4. what the players' choices are when they move
5. what each player knows when he makes his choices
6. the probability distribution over any exogenous events."[17, p. 77]

There is a direct mapping between the dynamic ICL and the extensive form of a game. $\mathcal{A}$ is the set of players. The logic program specifies the payoffs as a function of the moves (actions) of the players. The set $\mathcal{C}_a$ specifies the players' choices when they move. The π function specifies what each player knows when it makes its move. P_0 provides a probability function over exogenous events represented as the independent random variables in $\mathcal{C}_0$.

The order of the moves is defined by the acyclicity of the knowledge base. The moves must be ordered so that the information is available before the decision is made. If there is some acyclicity ordering such that χ_1 is before χ_2 then χ_1 can be made before χ_2; if there is another acyclicity ordering where χ_2 is before χ_1, then it doesn't matter in which order the choices are made (as all of the information available for each choice can't depend on the other choice).

While the acyclicity of the rule base was chosen in order to allow for a simple semantic framework [36], it can be justified by appealing to the structure of games.

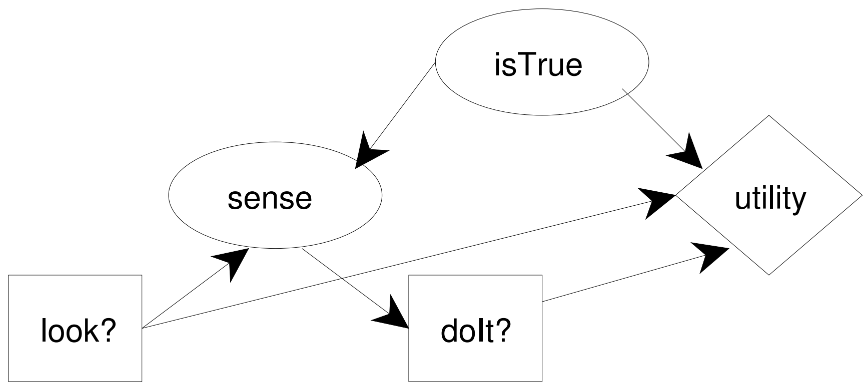

Figure 4: Influence diagram for our idealised example

6 Examples in Detail

In this section we present three different examples of using the ICL. The first demonstrates so-called "information seeking actions" and noisy sensors and actuators. The second presents a decision-theoretic planning example. The third defines a two-player, imperfect information game of blind tic-tac-toe. There are intended to show the details of the representation, and were not chosen because they are elegant examples for the formalism.

6.1 Information Seeking Actions and Noisy Sensors

In this section we give an idealised single agent in an environment example, showing how to model the following:

- Information producing actions, such as tests in diagnosis and positioning a camera in robotics, or asking a question in a user modelling situation
- Conditional plans (conditioning on sense values)
- How a passive sensor can be used to model an active sensor that "looks"
- How noisy sensors and actuators can be modelled (Section 6.1.4).

6.1.1 Information producing actions

The $look?$ decision of Figure 4 can be seen as an information producing action. It lets information about $isTrue$ be available to the next action. It also has a cost associated with it.

The agents action can either be $look$ or $dont\_look$. The action can be modelled by having

$$\{look, dont\_look\} \in \mathcal{C}$$

with our agent controlling this alternative.

In the rule base we model how actions by the agent and truths in the world affect sense values for the agent. Here is an idealised example for the case where there is no noise. (Section 6.1.4 considers noise.) Suppose that following the looking, the agent can sense either pos, neg or $nothing$. For the case with no noise, the environment model has the following axioms in $\mathcal{F}$:

$$sense(pos) \leftarrow look \wedge is\_true.$$
$$sense(neg) \leftarrow look \wedge \neg is\_true.$$
$$sense(nothing) \leftarrow dont\_look.$$

Thus "$look$" provides information about "is_true" to "do_it".

6.1.2 Conditioning on sense values

The agent can sense the world, and then decide what to do based on the sense values.

Continuing our example, suppose that the agent has possible actions do_it, $dont\_do\_it$, and has the sense values above. There are three independent choices the agent can make, namely whether or not to do_it for each of the three contingencies.

Within the ICL this can be modelled by having the axioms:

$$do\_it \leftarrow sense(pos) \wedge do\_if\_pos.$$
$$dont\_do\_it \leftarrow sense(pos) \wedge dont\_if\_pos.$$
$$do\_it \leftarrow sense(neg) \wedge do\_if\_neg.$$
$$dont\_do\_it \leftarrow sense(neg) \wedge dont\_if\_neg.$$
$$do\_it \leftarrow sense(nothing) \wedge do\_if\_nothing.$$
$$dont\_do\_it \leftarrow sense(nothing) \wedge dont\_if\_nothing.$$

and by having the following alternatives in $\mathcal{C}$, controlled by our agent:

$$\{do\_if\_pos, dont\_if\_pos\}$$
$$\{do\_if\_neg, dont\_if\_neg\}$$
$$\{do\_if\_nothing, dont\_if\_nothing\}$$

Within the dynamic ICL we specify:

$$\{do\_it, dont\_do\_it\} \in \mathcal{C}_{agent}$$
$$\mathcal{O}_{agent} = \{\{sense(pos), sense(neg), sense(nothing)\}\}$$
$$\pi(\{do\_it, dont\_do\_it\}) = \{\{sense(pos), sense(neg), sense(nothing)\}\}$$

The rules that need to be provided as part defining the ICL are created as part of the agent's strategy in the dynamic ICL.

6.1.3 Utility model

Finally, a utility model (part of the environment model) specifies how the utility varies depending on what's true and what the agent does. Here is an example of such an an axiomatisation (given that our agent is $agent_1$):

$$utility(agent_1, Prize - TC) \leftarrow$$
$$\quad test\_cost(TC) \wedge$$
$$\quad prize(Prize).$$
$$test\_cost(5) \leftarrow look.$$
$$test\_cost(0) \leftarrow dont\_look.$$
$$prize(10) \leftarrow do\_it \wedge is\_true.$$
$$prize(0) \leftarrow dont\_do\_it \wedge is\_true.$$
$$prize(8) \leftarrow dont\_do\_it \wedge \neg is\_true.$$
$$prize(1) \leftarrow do\_it \wedge \neg is\_true.$$

6.1.4 Noisy Actuators and Sensors

In Section 6.1.1 we assume that there were no noise in either actuator settings or in sense values. We can model actuator noise, for example in the "looking" actuator, by something like:

$$see \leftarrow look \wedge looking\_works$$
$$see \leftarrow dont\_look \wedge not\_looking\_doesnt\_work$$

with following in $\mathcal{C}$, controlled by nature:

$$\{looking\_works, looking\_doesnt\_work\}$$
$$\{not\_looking\_works, not\_looking\_doesnt\_work\}$$

$P_0(looking\_works)$ is the probability that the agent succeeds in seeing when it looks. $P_0(not\_looking\_doesnt\_work)$ is the probability that that agent sees when it doesn't look.

Noisy sensors can be modelled similarly. Assume that the value sensed only depends on whether the agent sees, but that there is no noise with respect to not seeing. There are two alternatives controlled by nature:

$$\{false\_positive, true\_negative\}$$
$$\{false\_negative, true\_positive\}$$

and the following facts:

$$sense(pos) \leftarrow see \wedge is\_true \wedge true\_positive.$$
$$sense(neg) \leftarrow see \wedge is\_true \wedge false\_negative.$$
$$sense(neg) \leftarrow see \wedge \neg is\_true \wedge true\_negative.$$
$$sense(pos) \leftarrow see \wedge \neg is\_true \wedge false\_positive.$$
$$sense(nothing) \leftarrow \neg see.$$

$P_0(false\_positive)$ is the probability of a false-positive; when the sensor reports positive when the value in the world is not true. $P_0(false\_negative)$ is the probability of a false-negative; when the sensor reports negative when the value in the world is true.

6.2 Shipping Widget s

In this section we present an example of Draper et. al. [14]. 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's done. Processing consists of rejecting flawed widgets and shipping unflawed widgets. The robot can inspect the widget to see if it's 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 robot. The robot has one sensor for detecting blemishes. $sense(blemished, T)$ is true if the robot senses that the widget is blemished at time T.

The robot has 6 actions (exactly one of which is possible at any time), namely to reject, ship, notify, paint or inspect the widget or do nothing. $do(A, T)$ is true if the robot does action A at time T.

The robot specification module for $robot$ is the tuple $\langle \mathcal{C}_{robot}, \mathcal{O}_{robot}, \pi \rangle$ where

$$\begin{aligned}
&\mathcal{C}_{robot} = \{\{do(reject, T), do(ship, T), do(notify, T), do(paint, T),\\
&\qquad\qquad do(inspect, T), do(nothing, T)\} : T \text{ is a time}\}.\\
&\mathcal{O}_{robot} = \{\{\neg sense(blemished, T), sense(blemished, T)\} : T \text{ is a time}\}.\\
&\pi(\{do(reject, T), do(ship, T), do(notify, T), do(paint, T),\\
&\qquad\qquad do(inspect, T), do(nothing, T)\})\\
&\quad = \{\{\neg sense(blemished, T'), sense(blemished, T')\} : T' \leq T\}.
\end{aligned}$$

In other words, at each time, the robot gets to choose which of the six actions it carries out. When it's making this decision, it knows whether or not it has sensed blemishes in the past.

NATURE MODULE: The remaining thing to define is the rules and alternatives controlled by nature. This specifies the dynamics of the world. We axiomatise how the robot's actions affect the world, how the world affects the senses of the robot.

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 doesn't depend on the time (a second painting will not make the widget more likely to be painted). Painting only works if it hasn't already been shipped or rejected. Once painted, a widget remains painted.

$$\begin{aligned}
&painted(T+1) \leftarrow\\
&\quad do(paint, T) \wedge\\
&\quad paint_works \wedge\\
&\quad \neg shipped(T) \wedge\\
&\quad \neg rejected(T).\\
&painted(T+1) \leftarrow\\
&\quad painted(T).
\end{aligned}$$

Painting succeeds 95% of the time:

$$\{paint\_works, paint\_fails\} \in \mathcal{C}_0$$
$$P_0(paint\_works) = 0.95, P_0(paint\_fails) = 0.05$$

Note that we have not parametrized $paint\_works$ by the time. This is lets us model the fact that repainting will not help when painting failed the first time. In any possible world where $paint\_fails$ is true, painting always results in the widget being painted, and if it is false, painting always results in the widget being painted.

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

$$\begin{aligned} & blemished(T) \leftarrow \\ & \quad flawed(T) \wedge \\ & \quad \neg painted(T). \end{aligned}$$

Note that the use of logic programs, assuming the stable model semantics entails that the rules mean "if and only if" (in the same way Clark's completion [11] does).

Whether the widget is flawed or not persists:

$$flawed(T+1) \leftarrow flawed(T).$$

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

$$\begin{aligned} & processed(T) \leftarrow \\ & \quad rejected(T) \wedge \\ & \quad flawed(T). \\ & processed(T) \leftarrow \\ & \quad shipped(T) \wedge \\ & \quad \neg flawed(T). \end{aligned}$$

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

$$shipped(T) \leftarrow do(ship, T).$$
$$shipped(T+1) \leftarrow shipped(T).$$

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

$$rejected(T) \leftarrow do(reject, T).$$
$$rejected(T+1) \leftarrow rejected(T).$$

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

$$
\begin{aligned}
& sense(blemished, T+1) \leftarrow \\
& \quad do(inspect, T) \wedge \\
& \quad blemished(T) \wedge \\
& \quad \neg false\_pos(T).
\end{aligned}
$$

The sensor gives a false positive with probability 0.1. Unlike whether painting succeeds, suppose the probability of the sensor giving a false positive at each time is independent of what happens at other times:

$$
\begin{aligned}
& \{false\_pos(T), not\_false\_pos(T)\} \in \mathcal{C}_0 \\
& P_0(false\_pos(T)) = 0.1 \\
& P_0(not\_false\_pos(T)) = 0.9
\end{aligned}
$$

30% of widgets are initially flawed:

$$
\begin{aligned}
& \{flawed(0), unflawed(0)\} \in \mathcal{C}_0 \\
& P_0(flawed(0)) = 0.3 \\
& P_0(unflawed(0)) = 0.7
\end{aligned}
$$

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

$$
\begin{aligned}
& utility(robot, 1) \leftarrow \\
& \quad do(notify, T) \wedge \\
& \quad \neg notified\_before(T) \wedge \\
& \quad painted(T) \wedge \\
& \quad processed(T). \\
& utility(robot, 0) \leftarrow \neg utility(robot, 1). \\
& notified\_before(T) \leftarrow T_1 < T \wedge do(notify, T_1).
\end{aligned}
$$

One (pure) policy for our robot is the logic program:

$$
\begin{aligned}
& do(inspect, 0). \\
& do(paint, 1). \\
& do(reject, 2) \leftarrow sense(blemished, 1). \\
& do(ship, 2) \leftarrow \neg sense(blemished, 1). \\
& do(notify, 3).
\end{aligned}
$$

This has expected utility 0.925. Note that in the problem formulation, we need to paint blemished widgets.

This policy isn't optimal. Policy:

$$
\begin{aligned}
&do(inspect, 0).\\
&do(inspect, 1).\\
&do(paint, 2).\\
&do(reject, 3) \leftarrow sense(blemished, 1).\\
&do(reject, 3) \leftarrow sense(blemished, 2).\\
&do(ship, 3) \leftarrow \neg sense(blemished, 1) \wedge \neg sense(blemished, 2).\\
&do(notify, 4).
\end{aligned}
$$

has expected utility 0.94715. There is no optimal policy for this example (it isn't a *finite* game so Nash's theorem doesn't apply here), we can add more "inspect"s to keep raising the expected utility.

The best policy without inspecting, namely $\{do(paint, 0), do(ship, 1), do(notify, 2)\}$ has expected utility 0.665.

Of course, we can always define the utility so that the robot is penalized for taking too much time, e.g., by defining utility by:

$$
\begin{aligned}
&utility(robot, 1 - T/10) \leftarrow rewarded(T).\\
&utility(robot, 0) \leftarrow \neg rewarded\_at\_some\_time.\\
&rewarded\_at\_some\_time \leftarrow rewarded(T).\\
&rewarded(T) \leftarrow\\
&\quad do(notify, T) \wedge\\
&\quad \neg notified\_before(T) \wedge\\
&\quad painted(T) \wedge\\
&\quad processed(T).
\end{aligned}
$$

Under the revised utility, the first policy above (with a single inspect) is optimal, with expected utility 0.625.

6.3 Blind tic-tac-toe

Koller and Pfeffer [26] present a game description language Gala. In that language they represent the game "blind tic-tac-toe". We represent the same game in the

ICL in order to present a parlour-game example and to enable us to compare the representation with Gala.

Blind tic-tac-toe is an imperfect information version of standard tic-tac-toe:

> "As in regular tic-tac-toe, the players take turns placing marks in squares. However, on his turn, each player can choose to mark either an X or an O; he reveals to his opponent the square in which he makes the mark, but not what type of make he makes. As usual, the goal is to complete a line of three squares with the same mark."[26]

The basic idea in defining such a game is to axiomatise the dynamics of the game in the logic. The rules should imply the consequences of the choices made by agents.

First we need a representation for the state of the game. In this game, the order of the moves is important as well as who put what where (as the last player who places a marker to make three in a row wins). We can represent the state of the game as a list of the form $put(X, Y, Who, What)$ which means that the player "Who" (either a or b) put "$What$" (an o or an x) at position (X, Y). The first element of the list was the last element placed there.

The first rule defines the first move. The second rule for the state progression defines subsequent moves:

$$
\begin{aligned}
&state([put(X, Y, Agent, What)], s(0)) \leftarrow \\
&\quad starts(Agent) \wedge \\
&\quad chooses(Agent, place(X, Y, What), 0). \\
&state([put(X, Y, Agent, What), put(Xp, Yp, Ap, Wp)|Rest], s(T)) \leftarrow \\
&\quad state([put(Xp, Yp, Ap, Wp)|Rest], T) \wedge \\
&\quad \neg finished([put(Xp, Yp, Ap, Wp)|Rest]) \wedge \\
&\quad opponent(Ap, Agent) \wedge \\
&\quad chooses(Agent, place(X, Y, What), T).
\end{aligned}
$$

We can define auxiliary relations such as who starts, and how the moves alternate, and when the game is finished.

$$
\begin{aligned}
&starts(a). \\
&opponent(a, b). \\
&opponent(b, a).
\end{aligned}
$$

$$
\begin{aligned}
&finished(S) \leftarrow \\
&\quad draw(S). \\
&finished(S) \leftarrow \\
&\quad wins(A, S). \\
&draw(S) \leftarrow \\
&\quad length(S, 9).
\end{aligned}
$$

We can axiomatise the utility functions. Note that we are relying on the fact that for any particular set of choices by agents, there is only one win state or one draw state.

$$
\begin{aligned}
&utility(a, 1) \leftarrow \\
&\quad wins(a, S). \\
&utility(a, 0) \leftarrow \\
&\quad wins(b, S). \\
&utility(a, 0.5) \leftarrow \\
&\quad draw(S). \\
&utility(b, 1) \leftarrow \\
&\quad wins(b, S). \\
&utility(b, 0) \leftarrow \\
&\quad wins(a, S). \\
&utility(a, 0.5) \leftarrow \\
&\quad draw(S).
\end{aligned}
$$

We can axiomatise the choices by the agents:[15]

$$
\begin{aligned}
&chooses(Agent, place(X, Y, What), T) \leftarrow \\
&\quad chooseX(Agent, X, T) \wedge \\
&\quad chooseY(Agent, X, T) \wedge \\
&\quad chooseWhat(Agent, What, T).
\end{aligned}
$$

[15] This is one simple way to axiomatise the choices. It means that agents can choose to place a mark in an occupied space (presumably they will be penalized by losing if they choose an occupied spot). Another method is that the agent can derive a list of free spaces from the sensed information, and then can only choose an element from this list.

The agents get to choose the X position, the Y position, and what mark they make. Thus,

$$\forall T\ \{chooseX(Agent,1,T), chooseX(Agent,2,T), chooseX(Agent,3,T)\} \in \mathcal{C}_{Agent}$$
$$\forall T\ \{chooseY(Agent,1,T), chooseY(Agent,2,T), chooseY(Agent,3,T)\} \in \mathcal{C}_{Agent}$$
$$\forall T\ \{chooseWhat(Agent,o,T), chooseWhat(Agent,x,T)\} \in \mathcal{C}_{Agent}$$

Now we have to decide what an agent gets to observe when making their decision. We assume that for each of these choices the agent gets to observe a filtered version of the state, which consists of a list of $pos\_who(X,Y,Who)$ for each square and a list of $pos\_what(X,Y,What)$ for the squares they occupy:

$$\begin{aligned}
&sense(Agent, Pos_Who_List, Pos_What_List, T) \leftarrow\\
&\quad state(S,T) \wedge\\
&\quad extract_pos_who_list(S, Pos_Who_List) \wedge\\
&\quad extract_pos_what_list(Agent, S, Pos_What_List).\\
&extract_pos_who_list([],[])\\
&extract_pos_who_list([put(X,Y,Agent,What)|S1],[pos_who(X,Y,Who)|P1]) \leftarrow\\
&\quad extract_pos_who_list(S1,P1).
\end{aligned}$$

Similarly for $extract\_pos\_what\_list$.

The observable function can be given by:[16]

$$\begin{aligned}
&\pi(\{chooseX(Agent,1,T), chooseX(Agent,2,T), chooseX(Agent,3,T)\})\\
&\quad = \{\{sense(Agent, Pos_Who_List, Pos_What_List, T)\}\\
&\qquad : Pos_Who_List, Pos_What_List \text{ are appropriate lists}\}.
\end{aligned}$$

The other two choices have similar information sets, but include the previous decisions (i.e., the agent knows which X it chose when choosing a Y).

For those who don't like to read declarative logical formulae, the best way to understand these rules is to think about building a game tree by forward chaining on the rules. a starts and so must make a choice of the X position, the Y position, and what mark they are making. This forms an 18-way split in the game tree (there are 18 different choices available to a). Then the state evolves by b making a move. There are 9 different information states for b, and they have to choose one

[16]We have neither presented a syntax for π, nor a syntax for the choices. This is because we only wanted to present object level rules, in order to not have to present two different logical formalisms. It's hoped that this set notation gets the general idea across.

of 18 choices (or 16 if the alternative axiomatisation is made). And so on building the game tree.

It's envisioned that such a representation could be used to build the same game tree as for Gala [26], and can use the same efficient algorithms. The representation proposed in this paper is more declarative (in that we can give a declarative possible worlds semantics for the whole framework, and all of the logical rules can be interpreted as statements about the domain), and more general in that it's not tuned specifically to 2-person alternating games. In fact the ICL isn't tuned for any particular application; there are no built in predicates, and no syntax beyond that of the logic. This may mean that Gala is more natural for those games it's designed for, but we believe that the more general language will be more useful for general specification of decision problems under uncertainty.

7 Conclusion

This paper has presented a logic that allows for what's arguably a natural specification of multi-agent decision problems. There is a simple semantic framework in terms of possible worlds semantics. It lets us use logic to specify the dynamics of the world, while retaining the elegance and generality of game theory.

What we are adding to game theory is an object-level representation of the domain. We can axiomatise how actions (moves) affect the world, how the utility is derived from simpler components, and how sensors work. All of these axioms can be interpreted within the simple logic. It allows us to represent the probabilistic dependencies in a domain, in much the same way that influence diagrams provide a more intuitive representation for many problems than decision trees [23; 22]. We also allow for a form of parametrized rules by the use of logical variables that allow us to construct large game trees from smaller components.

We are adding to influence diagrams, the ability to represent multiple agents, the ability to represent[17] structured probability and decision tables, and a way to have a dynamic construction of influence diagrams (with a similar motivation to Breese [9], but having logic programs as object-level statements about the world rather than at the meta-level as does Breese). We also allow for the designer to axiomatise the dynamics of the system, instead of having to summarize it in a single step as a probability distribution.

We are adding to logic a new way to handle and think about non-determinism

[17] In other papers we show how the structure can be exploited for computational gain [38; 41].

and uncertainty. Rather than just using disjunction which doesn't seem to be subtle enough for the range of forms of uncertainty that we need to handle, we provide a mechanism in terms of independent choices to handle uncertainty. We argue that considering different agents making choices is the right way to think about uncertainty in a logical formalism. The ICL is weaker than other mixes of logic and decision theory for modelling agents [3; 24; 20; 19] which have added probability and decisions to a rich logic. They don't have general independence assumptions (although they can state independence assumptions), and have to cope with many different forms of uncertainty (e.g., disjunction as well as choices by agents). The goals of this paper are different: we are investigating a different way of viewing uncertainty for modelling agents. We are looking for ways to make representations of the world simpler. Whether we have succeeded in this is an open question.

Conspicuous by its absence in this paper is a discussion on computation. In this context, computation can mean three things: (1) building a situated agent that embodies a strategy; (2) simulating a policy and environment; or (3) finding an optimal strategy. A Prolog implementation of the second that finds expected utilities of strategies is available from the author's web page.[18] It should be noted that the computational complexity of finding Nash equilibria, even the propositional, single-agent without perfect recall case is exponentially harder to solve than an influence diagram [25]. Intuitively this is because dynamic programming doesn't work when we have a forgetful agent; we can't solve the last decision independently of the earlier decisions [54]. This isn't a problem with the representation; it's the problems that are difficult. It isn't clear whether the representation in this formalism makes the problems more difficult to solve. There is some, however, evidence that the representation presented here makes solving a decision problem easier than in an influence diagram, as we can exploit the rules structure. [5; 38; 41] There is much more work to be done on exact and approximate algorithms for the problems represented in the ICL.

References

[1] J. S. Albus. *Brains, Behavior and Robotics*. BYTE Publications, Peterborough, N.H., 1981.

[2] K. R. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9(3-4):335–363, 1991.

[18] `http://www.cs.ubc.ca/spider/poole`

[3] F. Bacchus. *Representing and Reasoning with Uncertain Knowledge*. MIT Press, Cambridge, Massachusetts, 1990.

[4] F. Bacchus, A. J. Grove, J. Y. Halpern, and D. Koller. From statistical knowledge bases to degrees of belief. *Artificial Intelligence*, 87(1-2):75–143, 1996. ftp://logos.uwaterloo.ca/pub/bacchus.

[5] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proc. 14th International Joint Conf. on Artificial Intelligence (IJCAI-95)*, pages 1104–1111, Montreal, Quebec, 1995.

[6] C. Boutilier and N. Friedman. Nondeterministic actions and the frame problem. In *Working Notes AAAI Spring Symposium 1995 — Extending Theories of Actions: Formal Theory and Practical Applications*, March 1995.

[7] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In E. Horvitz and F. Jensen, editor, *Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence (UAI-96)*, pages 115–123, Portland, Oregon, 1996.

[8] C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proc. 13th National Conference on Artificial Intelligence*, pages 1168–1174, Portland, OR, 1996.

[9] J. S. Breese. Construction of belief and decision networks. *Computational Intelligence*, 8(4):624–647, 1992.

[10] R.A. Brooks. A robust layered control system for a mobile robot. *I.E.E.E. Journal of Robotics and Automation*, 2(1):14–23, 1986.

[11] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, 1978.

[12] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[13] T. L. Dean and M. P. Wellman. *Planning and Control*. Morgan Kaufmann, San Mateo, California, 1991.

[14] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 31–36, Menlo Park, CA, 1994.

[15] R. Y. Fagin, J. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, Mass., 1994.

[16] J. Forbes, T. Huang, K. Kanazawa, and S. Russell. The BATmobile: Towards a Bayesian automated taxi. In *Proc. 14th International Joint Conf. on Artificial Intelligence (IJCAI-95)*, pages 1878–1885, Montreal, August 1995.

[17] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, Cambridge Massachusetts, 1992.

[18] M. Gelfond and V Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth Logic Programming Symposium*, pages 1070–1080, Cambridge, Mass., 1988.

[19] P. Haddawy. *Representing Plans Under Uncertainty: A Logic of Time, Chance, and Action*, volume 770 of *Lecture notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1994.

[20] J. Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3):311–350, 1990.

[21] J.Y. Halpern and M.R. Tuttle. Knowledge, probability, and adversaries. *Journal of the ACM*, 40(4):917–962, 1993.

[22] R. Howard. From influence to relevance to knowledge. In R. M. Oliver and J. Q. Smith, editor, *Influence Diagrams, Belief Nets and Decision Analysis*, chapter 1, pages 3–23. Wiley, 1990.

[23] R. A. Howard and J. E. Matheson. Influence diagrams. In R. A. Howard and J. Matheson, editors, *The Principles and Applications of Decision Analysis*, pages 720–762. Strategic Decisions Group, CA, 1981.

[24] K. Kanazawa. A logic and time nets for probabilistic inference. In *Proc. 9th National Conference on Artificial Intelligence*, pages 360–365, Anaheim, California, 1991.

[25] D. Koller and N. Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4:528–552, 1992.

[26] D. Koller and A. J. Pfeffer. Generating and solving imperfect information games. In *Proc. 14th International Joint Conf. on Artificial Intelligence (IJCAI-95)*, pages 1185–1192, Montreal, 1995.

[27] R. Kowalski. *Logic for Problem Solving*. Artificial Intelligence Series. North Holland, New York, 1979.

[28] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming, Special issue on Reasoning about Action and Change*, to appear, 1996.

[29] D. G. Luenberger. *Introduction to Dynamic Systems: Theory, Models and Applications*. Wiley, New York, 1979.

[30] J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28(1):89–116, February 1986.

[31] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In M. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[32] R. B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge, MA, 1991.

[33] N. J. Nilsson. Logic and artificial intelligence. *Artificial Intelligence*, 47:31–56, 1991.

[34] R. M. Oliver and J. Q. Smith, editors. *Influence Diagrams, Belief Nets and Decision Analysis*. Series in probability and methematical statistics. Wiley, Chichester, 1990.

[35] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[36] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[37] D. Poole. Abducing through negation as failure: Stable models within the independent choice logic. Technical Report, Department of Computer Science, UBC, ftp://ftp.cs.ubc.ca/ftp/local/poole/papers/abnaf.ps.gz, January 1995.

[38] D. Poole. Exploiting the rule structure for decision making within the independent choice logic. In P. Besnard and S. Hanks, editor, *Proc. Eleventh Conf. on Uncertainty in Artificial Intelligence (UAI-95)*, pages 454–463, Montreal, 1995.

[39] D. Poole. Logic programming for robot control. In *Proc. 14th International Joint Conf. on Artificial Intelligence (IJCAI-95)*, pages 150–157. 1995.

[40] D. Poole. A framework for decision-theoretic planning I: Combining the situation calculus, conditional plans, probability and utility. In E. Horvitz and F. Jensen, editor, *Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence (UAI-96)*, pages 436–445, Portland Oregon, 1996.

[41] D. Poole. Probabilistic partial evaluation: Exploiting rule structure in probabilistic inference. In *Proc. 15th International Joint Conf. on Artificial Intelligence (IJCAI-97)*, page to appear, Nagoya, Japan, 1997. `ftp://ftp.cs.ubc.ca/ftp/local/poole/papers/pro-pa.ps.gz`,

[42] D. Poole, A.K. Mackworth, and R.G. Goebel. *Computational Intelligence: A Logical Approach.* Oxford University Press, New York, 1997.

[43] T.C. Przymusinski. Three-valued nonmonotonic formalisms and semantics of logic programs. *Artificial Intelligence*, 49:309–343, 1991.

[44] M. L. Puterman. Markov decision processes. In D. P. Heyman and M. J. Sobel, editor, *Handbooks in OR and MS, Vol. 2*, chapter 8, pages 331–434. Elsevier Science Publishers B. V., 1990.

[45] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and the Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, California, 1991.

[46] R. L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[47] S. J. Rosenschein and L. P. Kaelbling. A situated view of representation and control. *Artificial Intelligence*, 73:149–173, 1995.

[48] S. J. Russell and D. Subramanian. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 2:575–609, 1995.

[49] L. J. Savage. *The Foundation of Statistics*. Dover, New York, 2nd edition, 1972.

[50] L. K. Schubert. Monotonic solutions to the frame problem in the situation calculus: An efficient method for worlds with fully specified actions. In H. E. Kyburg, R. P. Loui and G. N. Carlson, editor, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer Academic Press, Boston, Mass., 1990.

[51] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[52] J. E. Smith, S. Holtzman, and J. E. Matheson. Structuring conditional relationships in influence diagrams. *Operations Research*, 41(2):280–297, 1993.

[53] J. Von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, third edition, 1953.

[54] L. Zhang, R. Qi, and D. Poole. A computational theory of decision networks. *International Journal of Approximate Reasoning*, 11(2):83–158, 1994.

[55] Y. Zhang. *A Foundation for the Design and Analysis of Robotic Systems and Behaviours*. PhD thesis, Department of Computer Science, University of British Columbia, September 1994.

[56] Y. Zhang and A. K. Mackworth. Will the robot do the right thing? In *Proc. 10th Canadian Artificial Intelligence Conf.*, pages 255–262, Banff, May 1994.

[57] Y. Zhang and A. K. Mackworth. Constraint nets: a semantic model for hybrid dynamic systems. *Theoretical Computer Science*, 138:211–239, 1995.