Complexity Results for Agent Design Problems

Paul E. Dunne Michael Laurence Michael Wooldridge

Department of Computer Science
University of Liverpool
Liverpool L69 7ZF, UK

{ped,mikel,mjw}@csc.liv.ac.uk

***Abstract*— The *Agent Design* problem involves determining whether or not it is possible to construct an agent capable of accomplishing a given task in a given environment. The simplest examples of such tasks are where an agent is required to bring about some goal (achievement tasks) or where an agent is required to maintain some invariant condition (maintenance tasks). Previous work has considered the complexity of achievement and maintenance agent design problems for a range of environmental properties. In this paper, we investigate the computational complexity of the agent design problem in three further settings. First, we investigate the issue of tasks that are specified as *Boolean combinations* of achievement and maintenance tasks. Second, we investigate the extent to which an agent's information about the history of the environment in which it operates affects the complexity of the problem: in the *bounded agent design problem*, an agent is constrained to have a 0, 1, or $k > 1$ bound on what it is permitted to "remember" about the history of the system. Finally, we investigate the complexity of *stochastic* agent design problems, where we ask whether there is an agent that has a probability of success at least p.**

***Index Terms*— multiagent systems, computational complexity.**

I. Introduction

In this paper, we are concerned with the computational complexity of one particular issue that arises in agent-based systems [15]: the *agent design problem*. This problem can be informally understood as follows:

> Given (representations of) an environment and a task to be carried out, does there exist an agent that can be guaranteed to carry out the task in the environment?

Although related to Markov decision problems and problems in AI planning, agent design is in fact quite distinct, being more akin to a game against nature, in which both nature and agent are assumed to be computational entities; we comment on the relationships to other work in section VI.

Wooldridge and Dunne investigated the computational complexity of the agent design problem in a range of different settings, and for a range of different types of tasks [14], [16]–[18]. The two most important types of tasks they considered were *achievement tasks* (where an agent is required to bring about some goal state), and *maintenance tasks* (where an agent is required to maintain some invariant in an environment). The complexity of these problems was shown to vary from NL-complete (and hence tractable) in the simplest case, to non-recursive (and hence undecidable) in the worst, depending upon the assumptions made about the environment. The main issues investigated to date relate to whether or not the environment is:

- *deterministic* or *non-deterministic* (i.e., whether or not the next state of the environment is uniquely determined, or whether multiple successors are possible);
- *history dependent* or *history independent* (i.e., whether or not the environment is allowed to observe the entire history of the environment in order to "choose" possible successors, or whether just the final state in the history is available); and
- *unbounded*, *bounded*, or *polynomially bounded* (i.e., whether any given "run" of an agent is guaranteed to terminate, and, if so, whether it will be guaranteed to terminate after only polynomially many actions).

Intuitively, non-deterministic environments are "harder" than deterministic environments, history dependent environments are harder than history independent environments, while unbounded environments are harder than bounded environments, which are in turn harder than polynomially bounded environments. The complexity results obtained previously bear out these intuitions fairly accurately. At worst, the agent design problem was shown to be undecidable – perhaps not surprisingly, the extent to which an environment was

ISSN 1109-9305

bounded was the major factor in determining undecidability. Under "reasonable" assumptions, (a non-deterministic, polynomially-bounded, history dependent environment) the agent design problem for both achievement and maintenance tasks was shown to be PSPACE-complete [14]. The easiest case considered was for deterministic, history dependent, polynomially-bounded environments, for which the achievement and maintenance design problems were shown to be decidable in polynomial-time.

In this paper we extend these previous studies in three ways:

- First, we consider richer task specifications. The idea is to specify tasks as *arbitrary Boolean combinations of achievement and maintenance tasks*. In such settings, a task is specified as a formula of propositional logic: atomic propositions denote sets of environment states, positive literals are interpreted as achievement tasks ("bring about one of these states"), and negative literals are interpreted as maintenance tasks ("avoid all these states"). Thus, for example, $p \vee \bar{q}$ is a task in which an agent must either bring about one of the states denoted by *p, or else* avoid all the states denoted by *q*.
- Second, we consider restrictions on the "power" of agents. That is, we ask whether there exist "weak", or *bounded* agents to accomplish a given task (cf. [12]). The specific notion of boundedness that we consider involves whether or not there exist agents capable of successfully accomplishing the task with given *memory bounds*, i.e., bounds on the length of history they are permitted to recall in order to make their decision. As an extremal case, we consider agents that are not permitted to observe the history at all, and are thus required to make a decision with no knowledge of the way in which the system has evolved.
- Third, we consider variants of the *stochastic* agent design problem. Previous work only considered agents that were either *guaranteed* to succeed with the task, or *had at least some chance* of success. In a stochastic agent design problem, we are given a rational probability *p*, and are asked whether there is an agent that succeeds with at least probability *p*.

We begin by introducing the formal model of agents, environments, and tasks upon which our results are established. We then summarize the results of [14], [16], [17], and examine the computational complexity of the agent design problem when Boolean task specifications are permitted. We consider bounded agent design problems in section IV, and in section V, we turn our attention to stochastic agent design problems. In Appendix , we provide a summary of the problems studied and the results obtained.

Throughout the paper, we assume familiarity with the theory of computational complexity [5], [8], [11].

II. Agents, Environments, and Tasks

In this section, we present an abstract formal model of agents and the environments they occupy; we then use this model to frame the decision problems we study. The systems of interest to us consist of an agent situated in some particular environment; the agent interacts with the environment by performing actions upon it, and the environment responds to these actions with changes in state. It is assumed that the environment may be in any of a finite set $E = \{e_0, e_1, \ldots, e_n\}$ of instantaneous states. Agents are assumed to have a repertoire of possible actions available to them, which transform the state of the environment. Let $Ac = \{\alpha_0, \alpha_1, \ldots, \alpha_k\}$ be the (finite) set of actions. The behaviour of an environment is defined by a *state transformer function*, τ. In our case τ is not simply a function from environment states and actions to sets of environment states, but from *runs* and actions to sets of environment states. This allows for the behaviour of the environment to be dependent on the *history* of the system – the previous states of the environment and previous actions of the agent can play a part in determining how the environment behaves.

Definition 1: An *environment* is a quadruple $Env = \langle E, e_0, Ac, \tau \rangle$, where E is a finite set of *environment states* with e_0 distinguished as the *initial state*; and Ac is a finite set of *available actions*. Let S_{Env} denote all sequences of the form:

$$e_0 \cdot \alpha_0 \cdot e_1 \cdot \alpha_1 \cdot e_2 \cdots$$

where e_0 is the initial state, and for each i, $e_i \in E$, $\alpha_i \in Ac$.

Let S^E be the subset of such sequences that end with a state. The *state transformer function* τ of the environment is a total mapping

$$\tau : S^E \times Ac \rightarrow \wp(E)$$

We focus on the the set of *runs* in the environment. This is the set $R_{Env} = \cup_{k=0}^{\infty} R^{(k)}$, where

$$\begin{aligned} R^{(0)} &= \{e_0\} \quad \text{and} \\ R^{(k+1)} &= \bigcup_{r \in R^{(k)}} \bigcup_{\{\alpha \in Ac \mid \tau(r,\alpha) \neq \emptyset\}} \{r \cdot \alpha \cdot e \mid e \in \tau(r,\alpha)\} \end{aligned}$$

An environment is *bounded* if $R^{(k)} = \emptyset$ for some k and *unbounded* otherwise. We denote by R^{Ac} the set $\{r \cdot \alpha \mid r \in R_{Env}\}$ so that we subsequently interpret τ as a total mapping, $\tau : R^{Ac} \rightarrow \wp(E)$, i.e., as describing the (possibly empty) set of states which may result by performing the action $\alpha \in Ac$ after a run $r \in R_{Env}$.

A run, r, has *terminated* if $\forall \alpha \in Ac$, $\tau(r \cdot \alpha) = \emptyset$. The subset of R_{Env} comprising all terminated runs is denoted T_{Env}. In bounded environments, every run is a prefix of some (set of) terminated runs.

The *length* of a run, $r \in R_{Env}$ is the total number of actions and states occurring in r and is denoted by $\mid r \mid$.

Unless otherwise stated it is assumed that environments are bounded.

The state transformer function, τ, is encoded in an input instance by a deterministic Turing machine description T_τ with the following characteristics: the input has the form $r\#e$, where $r \in R^{Ac}$, $e \in E$, and $\#$ is a separator symbol. The program T_τ accepts $r\#e$ if and only if $e \in \tau(r)$. The number of moves made by T_τ is bounded by a polynomial, $p(|r|)$. Using T_τ, the set of states $\tau(r)$ can be constructed in $|E|p(|r|)$ steps.

We view *agents* as performing actions upon the environment, thus causing the state of the environment to change. In general, an agent will be attempting to "control" the environment in some way, in order to carry out some task. However, the agent has at best partial control over the environment.

Definition 2: An *agent*, *Ag*, in an environment $Env = \langle E, e_0, Ac, \tau \rangle$ is a mapping

$$Ag : R_{Env} \rightarrow Ac \cup \{\otimes\}$$

The symbol $\otimes$ is used to indicate that the agent has finished its operation: an agent invokes this only on terminated runs, $r \in T_{Env}$, an event that is referred to as the agent having *no allowable actions*. A *system*, *Sys*, is a pair $\langle Env, Ag \rangle$ comprising an environment and an agent operating in that environment. A sequence $s \in R_{Env} \cup R^{Ac}$ is called a *possible run of the agent Ag in the environment Env* if

$$s = e_0 \cdot \alpha_0 \cdot e_1 \cdot \alpha_1 \cdots$$

satisfies

1) e_0 is the initial state of E, and $\alpha_0 = Ag(e_0)$;
2) $\forall k > 0$,

$$\begin{array}{rcl} e_k & \in & \tau(e_0 \cdot \alpha_0 \cdot e_1 \cdot \alpha_1 \cdots \alpha_{k-1}) \text{ where} \\ \alpha_k & = & Ag(e_0 \cdot \alpha_0 \cdot e_1 \cdot \alpha_1 \cdots e_k) \end{array}$$

It should be noted that, in general, the state transformer function τ (with domain R^{Ac}) is *non-deterministic*, whereas agents are *deterministic*.

An agent may have a number of different possible runs in any given environment. We will denote by $R(Ag, Env)$ the set of possible runs of agent Ag in environment Env and by $T(Ag, Env)$ the subset of these that are terminated, i.e., belong to T_{Env}. An agent, Ag, must define some allowable action in Ac, for every run in $R(Ag, Env) \setminus T_{Env}$, i.e., agents may not choose to halt arbitrarily.

We concentrate on the behaviour of agents which have some bound placed on the number of actions performed. More precisely, given $t : \mathbb{N} \rightarrow \mathbb{N}$, the set of $t(n)$-*critical runs* of an agent Ag in the environment Env is the set $C^{t(n)}(Ag, Env)$ defined by those runs of the agent in which exactly $t(n)$ actions have been performed or which have terminated after at most $t(n) - 1$ actions. Unless otherwise stated the bounding function $t(n) = n$ (with $n = |E \times Ac|$) is used.

III. Boolean Task Specifications

We build agents in order to carry out *tasks* for us. The task to be carried out must be *specified* by us. An obvious question is how to specify these tasks. In this paper, we will be concerned with *predicate task specifications*. Such specifications take the form of a predicate over runs, i.e., a condition that runs either satisfy or fail to satisfy. We use Ψ to denote a predicate specification, and write $\Psi(r)$ to indicate that run $r \in R$ satisfies Ψ. Intuitively, $\Psi(r)$ means that the task specified by Ψ is successfully accomplished on run r. We are concerned in this paper with the *representation* of Ψ as a logical formula.

A *task environment* is a pair $\langle Env, \Psi \rangle$, where Env is an environment, and $\Psi : R \rightarrow \{0, 1\}$ is a predicate over runs. A task environment thus specifies the properties of the system the agent will inhabit (i.e., the environment Env), and also the criteria by which an agent will be judged to have either failed or succeeded (i.e., the specification Ψ).

One final consideration may, informally, be phrased as "what limits (if any) do we wish to place on how long an agent may take to succeed?" In this context we consider two basic decision problem formulations:

a) Is there an agent that *eventually* succeeds in its task?
b) Is there an agent the succeeds after *at most* some number of actions?

We now define our Boolean task specification language and formulate the decision problems studied. Let $X_n = \{x_1, \ldots, x_n\}$ be a set of n Boolean variables. A propositional formula $\Psi(X_n)$ over X_n is inductively defined by the following rules:

a) $\Psi(X_n)$ consists of single literal (x or $\bar{x}$ where $x \in X_n$).
b) $\Psi(X_n) = \Phi_1(X_n)\theta\Phi_2(X_n)$, where $\theta \in \{\vee, \wedge\}$ and Φ_1, Φ_2 are propositional formulae.

Let $|\Psi(X_n)|$ denote the total number of occurrences of *literals* in Ψ, and f_Ψ the Boolean logic function (of n variables) represented by Ψ. We say that $\Psi(X_n)$ is *non-trivial* if f_Ψ is not equivalent to a (Boolean) constant.

Definition 3: Let $Env = \langle E, e_0, Ac, \tau \rangle$ be an environment with state set E and actions Ac. Let $S =$

$\langle E_1, E_2, \ldots, E_n \rangle$ be an ordered collection of *pairwise disjoint* subsets of E (note that S does *not* have to be a partition of E) and $\Psi(X_n)$ be a non-trivial formula.

If $r \in R_{Env}$ is a run, then the instantiation, $\beta(r) = \langle b_1, b_2, \ldots, b_n \rangle$ of Boolean values to X_n *induced by* r is defined by:

$$b_i = \begin{cases} 1 & \text{if } r \text{ contains some state } e \in E_i \\ 0 & \text{if } r \text{ does not contain any state } e \in E_i. \end{cases}$$

A run, r, *succeeds with respect to* Ψ if $f_\Psi(\beta(r)) = 1$.

A *task specification* for an agent in an environment, $\langle E, e_0, Ac, \tau \rangle$ consists of a pair $\langle S, \Psi \rangle$ where S is an ordered collection of n pairwise disjoint subsets of E and Ψ is a non-trivial formula of n variables.

An agent, *Ag*, satisfies the task specification $\langle S, \Psi \rangle$ if and only if

$$\forall r \in T(Ag, Env) \qquad f_\Psi(\beta(r)) = 1$$

Example 1: Consider the environment whose state transformer function is illustrated by the graph in Figure 1. In this environment, an agent has just four available actions (α_1 to α_4 respectively), and the environment can be in any of six states (e_0 to e_5). History dependence in this environment arises because the agent is not allowed to execute the same action twice. Arcs between states in Figure 1 are labelled with the actions that cause the state transitions – note that the environment is non-deterministic. Suppose that we allow three primitive proposition letters to be used: p_1, p_2, p_3, where p_1 corresponds to environment states $\{e_2\}$, p_2 corresponds to $\{e_3\}$, and p_3 corresponds to $\{e_1\}$. Now consider the following task specifications:

- p_1
  This is an achievement task with goal states $\{e_2\}$. Since p_1 corresponds to $\{e_2\}$, an agent can reliably achieve p_1 by performing α_1, the result of which will be either e_1, e_2, or e_3. If e_1 results, the agent can perform α_0 to take it to e_5 and then α_2 to take it to e_2. If e_3 results, it can simply perform α_0.
- $\bar{p_1}$
  This is essentially a maintenance task with bad states $\{e_2\}$. Since the agent must perform either α_0 or α_1, no agent can guarantee to avoid e_2.
- $p_1 \vee p_2$
  Since there is an agent that can be guaranteed to succeed with task p_1, there is also an agent that can be guaranteed to succeed with task $p_1 \vee p_2$.
- $p_1 \wedge p_2$
  This task involves achieving *both* e_2 and e_3. There is no agent that can be guaranteed to succeed with this task.
- $p_1 \wedge (p_2 \vee p_3)$
  This task involves achieving e_2 and either e_3 or e_1. There exists an agent that can successfully achieve this task.
- $(p_1 \wedge \bar{p_2}) \vee (p_2 \wedge \bar{p_3})$
  This task involves either achieving e_2 and not e_3 or else achieving e_3 and not e_1. Since there exists an agent that can achieve e_2 and not e_3, there exists an agent that can succeed with this task.

We consider two decision classes of agent design problem: design tasks with *implicit* specification and design tasks with *explicit* specification.

Definition 4: Let Ψ be a non-trivial propositional formula of n variables. The decision problem *finite* Ψ-Design (Ψ-*FD*) takes as an instance: an environment *Env*, and *S*, an ordered collection of n pairwise-disjoint subsets of E. Ψ-$FD(\langle Env, S \rangle)$ returns "true" if and only if

$$\exists Ag \; \forall r \in C^{|E \times Ac|}(Ag, Env) \qquad f_\Psi(\beta(r)) = 1$$

It is important to note that Ψ is *not* part of the instance: each specific Ψ defines a design task for agents in *Env*. In contrast, we have the following decision problem.

Definition 5: The decision problem *Finite Agent Design* with *Explicit Task Specification* (*FED*) takes as an instance: a *non-trivial* propositional formula $\Psi(X_n)$, an environment *Env*, and *S*, an ordered collection of n pairwise-disjoint subsets of E. $FED(\langle \Psi, Env, S \rangle)$ returns "true" if and only if

$$\exists Ag \; \forall r \in C^{|E \times Ac|}(Ag, Env) \qquad f_\Psi(\beta(r)) = 1.$$

We employ the following shorthand in order to simplify subsequent notation. For $Q \in \{\Psi\text{-}FD, FED\}$ and $X \in \{det, non\text{-}det\}$ we denote by Q_X the decision problem Q with τ having the property X when given as an instance. For example, FED_{det} denotes the decision problem *Finite Agent Design with Explicit Task Specification for deterministic* τ.

A. Computational Complexity Results

Wooldridge and Dunne [14], [16], [17] considered the special cases $\Psi(x) = x$ and $\Psi(x) = \bar{x}$. In [17] the setting in which no restriction is placed on the number of actions that a successful agent is allowed, was also considered, i.e., the problem $\Psi(x)$-*Design* whose instances are pairs $\langle Env, S \rangle$ with S a subset of E concerned with the question of whether an agent exists that is guaranteed to reach at least one (respectively, avoid all) of the states in S. Theorem 1 summarises the results proved in [14], [16], [17] for these special cases.

Theorem 1: Let $\Psi(x)$-$Design^{\infty}$ (resp., $\Psi(x)$-$Design^{<\infty}$) denote the cases in which unbounded environments may (resp., may not) occur as instances:

a) x-$Design^{\infty}$ is recursively enumerable but not recursive.
b) $\bar{x}$-$Design^{\infty}$ is not recursively enumerable.
c) $\Psi(x)$-$Design^{<\infty}$ is recursive.

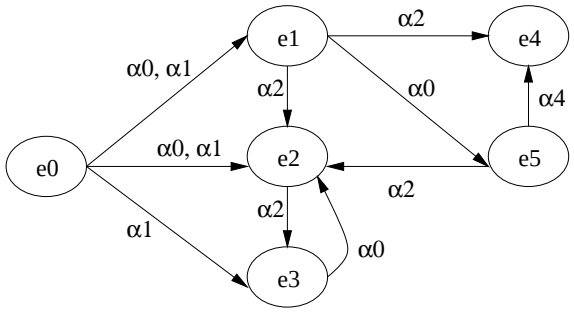

Fig. 1. The state transitions of an example environment: Arcs between environment states are labelled with the sets of actions corresponding to transitions. Note that this environment is *history dependent*, because agents are not allowed to perform the same action twice. So, for example, if the agent reached state e_2 by performing α_0 then α_2, it would not be able to perform α_2 again in order to reach e_3.

d) For any total recursive function $g : \mathbb{N} \rightarrow \mathbb{N}$ and any deterministic Turing machine program M deciding $\Psi(x)$-*Design*$^{<\infty}$: M exceeds $g(n)$ moves for infinitely many n, where n is the number of bits encoding an instance.

For the case of *finite* design problems.

e) $\forall X \in \{det, non\text{-}det\}$ x-$FD_X \equiv_{\log} \bar{x}$-$FD_X$.
f) $\Psi(x)$-$FD_{non\text{-}det}$ is PSPACE-complete.
g) $\Psi(x)$-FD_{det} is NP-complete.

Finally, if τ is *history independent*, i.e., the next environment state is dependent on the current state and chosen action only (not on the sequence of states and actions by which it was reached).

h) $\Psi(x)$-$FD_{non\text{-}det}$ is NL-complete.
i) $\Psi(x)$-FD_{det} is P-complete.

Our principal concern in this section is to generalize Theorem 1 to arbitrary n variable propositional functions. To begin, we note the following generalisation of Theorem 1 (a,b), in the case when the definition of Ψ-*Design* is modified to apply in unbounded environments with no restriction on the number of actions an agent can perform.

Theorem 2: Let Ψ-*Design*$^{\infty}$ denote the decision problem Ψ-*Design* with instances of *unbounded* environments allowed.

a) Ψ-*Design*$^{\infty}$ is recursive if and only if Ψ is trivial.
b) Ψ-*Design*$^{\infty}$ is recursively enumerable if and only if f_{Ψ} is monotone.

We now show that the decision problems Ψ-FD_X and FED_X are polynomial-time equivalent to the decision problems x-FD_X and $\bar{x}$-FD_X, irrespective of the specific non-trivial Ψ concerned.

Theorem 3: $\forall X \in \{det, non\text{-}det\}$, If Ψ is *any* non-trivial formula,

$$\Psi\text{-}FD_X \equiv_p x\text{-}FD_X \quad ; \quad FED_X \equiv_p x\text{-}FD_X$$

Proof: We omit the full technical details and simply outline the construction establishing Ψ-$FD_X \leq_p$ x-FD_X, i.e., that Ψ-FD_X is "no harder" than x-FD_X for any non-trivial Ψ. Given an instance $\langle Env, S\rangle$ of Ψ-FD_X, form an instance $\langle Env_A, G\rangle$ of x-FD_X by adding a new state $\{accept\}$ and action $\{test\}$. The set G contains the single state $\{accept\}$. We then modify the transition function for the instance of Ψ-FD_X so that if a run r has $f_{\Psi}(\beta(r)) = 1$ then $\tau_A(r \cdot test) = \{accept\}$. In this way any instance of Ψ-FD_X would have true returned if and only if the constructed instance also has true returned. ■

Corollary 1: For any non-trivial formula Ψ,

a) Ψ-$FD_{non\text{-}det}$ is PSPACE-complete.
b) Ψ-FD_{det} is NP-complete.

Proof: Immediate from Theorem 1(f,g) and Theorem 3. ■

Theorems 2 and 3 extend the results of Theorem 1 (a,b,f,g) as regards environments in which the state transition function is *history-dependent*. For the remainder of this paper we deal with the remaining cases involving *history independent* τ. In a *history-independent* environment, the transition function, τ, acts upon the current state and action to determine the next state of the environment. In such environments τ presented as a directed graph, $H(V,A)$, whose vertices, V, are labelled with environment states and in which there is an edge labelled $\alpha \in Ac$ from e_i to e_j if $e_j \in \tau(e_i, \alpha)$. In retaining the assumption that environments are bounded, it follows that $H(V,A)$ is an *acyclic* directed graph.

For the case of arbitrary non-trivial formulae, Ψ, we have:

Theorem 4: If τ is history-independent, then: FED_{det} is NP-complete[1].

Proof: We first show that any instance $\langle \Psi(X_n), Env, S\rangle$ of FED_{det} having a history independent transition function can be decided in NP. An NP algorithm simply guesses which action $\alpha \in Ac$ (if any) to perform in each state. In the graph $H(V,A)$ these

[1]To be strictly accurate, we define $\langle \Psi_n \rangle_{n \geq 1}$, a sequence of non-trivial n-variable propositional formulae Ψ_n, such that the decision problem $FED_{det}(\Psi_n, Env, S_n)$ is NP-complete.

choices determine a single path (since τ is deterministic). Evaluating $f_\Psi(X_n)$ for this path is easily done in polynomially many steps in $|\Psi| + |Env|$.

To show that FED_{det} for history independent τ is NP-hard we use a reduction from *Paths with Forbidden Pairs* restricted to directed acyclic graphs – PFP-DAG. An instance of PFP-DAG consists of a directed acyclic graph $G(V,A)$, two specified vertices $s,t \in V$, and a collection

$$C = \{\langle a_1, b_1\rangle,\ \langle a_2, b_2\rangle,\ \ldots,\ \langle a_n, b_n\rangle\}$$

of pairs of vertices in V. An instance is accepted if there is a path from s to t in $G(V,A)$ that contains *at most one* vertex from each pair in C.

Given an instance $\langle G(V,A), s, t, C\rangle$ of PFP-DAG we construct, in time polynomial in $|A| + |C|$, an instance

$$\langle \Psi(X_n)_{G,C}, Env, S\rangle$$

of FED_{det} for which τ is history independent and there is an agent satisfying $\Psi(X_n)$ in Env if and only if there is a path from s to t in $G(V,A)$ that contains at most one vertex from each pair in C.

We set $Env = \langle E, e_0, Ac, \tau\rangle$ where $E = V \cup \{e_t\}$; $Ac = \{\alpha_1, \ldots, \alpha_d, \alpha_{d+1}\}$, where d is the maximum out-degree of $G(V,A)$. For v with out-degree k in $G(V,A)$ and each v_i such that $\langle v, v_i\rangle$ is an edge of $G(V,A)$ we set $\tau(v,\alpha_i)$ to be v_i. An additional action influences only the vertex t: for this we have a single action, α_{d+1} entering the final state e_t. Finally the initial state, $e_0 \in E$ is set to the state corresponding to vertex s of $G(V,A)$. The graph, $H(V \cup \{e_t\}, A)$ defined is identical to $G(V,A)$ (with the addition of the $\langle t, e_t\rangle$ edge), and has at most one out-going edge from any vertex labelled with a given action.

The formula Ψ constructed for the instance uses $|\cup_{i=1}^{n}\{a_i, b_i\}| + 1$ variables, where n is the number of pairs in C Let $\langle A_n, B_n, z\rangle$ denote the resulting variables (where it may be the case that $|A_n|, |B_n| < n$ or $A_n \cap B_n \neq \emptyset$). With these,

$$\Psi(A_n, B_n, z) \ = \ z \ \wedge \ \bigwedge_{i=1}^{n} (\bar{a}_i \vee \bar{b}_i)$$

The final stage of the construction is to define the set of (at most) $2n+1$ pairwise disjoint subsets of E that form S. Let m be the total number of distinct vertices referred to in C (so that Ψ depends on exactly $m+1$ variables). Then: $S_i = \{a_i\}$ (if i is odd and the vertex a_i does not occur in any pair $C_j \in C$ with $j < i$); $S_i = \{b_i\}$ (if i is even and the vertex b_i does not occur in any pair $C_j \in C$ with $j < i$); and $S_{m+1} = \{e_t\}$. The construction ensures that any state (i.e., vertex) occurs in at most one S_i set.

We claim that an agent satisfying the specification $\Psi(X_{m+1})$ in the environment just defined if and only if there is a directed path from s to t containing at most one vertex from each pair in C for the instance of PFP-DAG.

Suppose such an agent exists. Its actions define a directed path from the initial state (s) to some final state. This path must contain the state corresponding to e_t (since this is the $m+1$'st variable, z, used in defining Ψ). Consider the corresponding path from s in $G(V,A)$. This path must contain the vertex t since the state e_t reached by the agent can only be accessed from the state corresponding to t in the environment. Thus the agent describes some path from s to t in $G(V,A)$. This path cannot contains both vertices from some pair $\langle a_i, b_i\rangle$: for if x_i, x_j were the propositional variables associated with these, then $\Psi(X_{m+1})$ would be false under the instantiation induced by the agent due to the presence of a clause equivalent to $\bar{x}_i \vee \bar{x}_j$. It follows that from an agent satisfying the specification we can find a path from s to t in $G(V,A)$ meeting the forbidden pair requirements.

On the other hand, suppose $G(V,A)$ contains a path from s to t containing at most one vertex from each pair in C. From this path we can construct a sequence of actions for each state, (the action corresponding to the out-going edge from each vertex on the path). The agent thereby defined reaches the state t and carrying out the action α_{d+1} completes the transition into the required final state e_t. ■

We now consider the Ψ-$FD_{non\text{-}det}$ problem for history-independent environments. First, some technical definitions are required.

Definition 6: Let E be a set, let $S = \langle E_1, \ldots, E_n\rangle$ be an ordered collection of pairwise disjoint subsets of E, let $e \in E$ and let Ψ be a Boolean formula over the set $\{x_1, \ldots x_n, \bar{x}_1, \ldots, \bar{x}_n\}$. We define the Boolean formula $\Psi(S,e)$ as follows. Express Ψ in DNF as

$$\Psi = \Psi_1 \vee \ldots \vee \Psi_k,$$

where each Ψ_i is a conjunction of elements of $\{x_1, \ldots x_n, \bar{x}_1, \ldots, \bar{x}_n\}$. We define $\Psi(S,e) = \Psi'_1 \vee \ldots \vee \Psi'_k$, where each Ψ'_i is defined as follows. If $e \in E_l$ for any literal $\bar{x}_l$ occurring in Ψ_i, then we define $\Psi'_i =$ **false**. If not, and $e \in E_m$ for any literal x_m occurring in Ψ_i, then Ψ'_i is obtained from Ψ_i by deleting x_m. If neither of these cases occurs, then simply define $\Psi'_i = \Psi_i$.

Definition 7: Let Env be an environment with state set E and let $e \in E$. Then $Env(e)$ is the environment identical to Env except that e is the initial state of $Env(e)$.

The motivation for these definitions is the following, easily proved result.

Lemma 1: Let $Env = \langle E, e_0, Ac, \tau\rangle$ be a history-independent bounded environment with state $e \in E$ and let $\alpha \in Ac$. Let $S = \langle E_1, \ldots, E_n\rangle$ be an ordered

collection of pairwise disjoint subsets of E and let Ψ be a Boolean formula over the set $\{x_1, \ldots x_n, \bar{x}_1, \ldots, \bar{x}_n\}$. Then there is an agent Ag solving the problem Ψ-$FD_{non\text{-}det}$ for the instance $\langle Env(e), S\rangle$ and satisfying $Ag(e) = \alpha$ if and only if there is an agent solving the problem Ψ-$FD_{non\text{-}det}$ for the instance $\langle Env(f), S\rangle$ for every state $f \in \tau(e, \alpha)$.

This result then allows us to establish the following.

Theorem 5: The Ψ-$FD_{non\text{-}det}$ problem for history-independent environments is in P.

Proof: Given a history-independent bounded environment $Env = \langle E, e_0, Ac, \tau\rangle$ and an ordered collection $S = \langle E_1, \ldots, E_n\rangle$ of pairwise disjoint subsets of E, the algorithm below defines a predicate

$$eval(e, \Psi) \in \{\textbf{true}, \textbf{false}\}$$

for every state $e \in E$ and boolean formula Ψ over the set $\{x_1, \ldots x_n, \bar{x}_1, \ldots, \bar{x}_n\}$, such that $eval(e, \Psi) = \textbf{true}$ if and only if there is an agent solving the problem Ψ-$FD_{non\text{-}det}$ for the instance $\langle Env(e), S\rangle$.

1) For every pair (e, Ψ) such that e is a sink in the acyclic directed graph defined by Env, and define $eval(e, \Psi) = \textbf{true}$ if and only if the path e satisfies Ψ.
2) Let the set $X \subseteq E$; initially set X to be the set of sinks in the acyclic directed graph defined by Env.
3) Choose a state $e \in E - X$ such that every state f for which there is an edge $e \rightarrow_\alpha f$ for some $\alpha \in Ac$, lies in X.
4) For every Boolean formula Ψ, define $eval(e, \Psi) = \textbf{true}$ if and only if there exists $\alpha \in Ac$ such that $eval(f, \Psi(S, e)) = \textbf{true}$ for every state f for which there is an edge $e \rightarrow_\alpha f$.
5) Add e to X.
6) Go to (3).

By Lemma 1, the algorithm computes the function $eval$ correctly. The loop in steps 3 to 6 cannot be executed more that $|E|$ times, thus the algorithm requires only polynomial time. ■

IV. Bounded Agent Design

Thus far, we have said nothing about how agents might actually be *implemented* – instead, we have viewed them as simply abstract mathematical structures. Ideally, we would be able to say something precise about how hard it was *in practice* to actually implement an agent for a given task in a given environment. But, just as the theory of computational complexity does not really have anything to say about how hard it is to implement programs for specific problems in general, so we cannot really address this problem with the mathematical tools available to us. However, we can do something fairly close. We now study the complexity of the agent design problem assuming some precise bounds on the complexity of the agents to be designed. That is, we ask whether there is a *bounded* agent to accomplish a given task, where the notion of a bound is given a precise meaning.

The idea of considering bounded agents arises from the literature on autonomous agents, where there is a well-known distinction between what are often called *deliberative* or *cognitive* agents [6], and *behavioural* or *reactive* agents [2]. Deliberative agents are typically assumed to employ explicit symbolic representations of their environments, and generally make decisions about what action to perform by manipulating these representations, typically by means of symbolic reasoning. It is now widely accepted that both approaches have merits and drawbacks: deliberative, logic-based approaches benefit from a clear theoretical underpinning, and have an associated engineering methodology, but are computationally costly; in contrast, reactive agents tend to be economical in their use of computational resources, and are frequently very robust, but often suffer from the lack of an engineering methodology. Little research has addressed the relative merits of the two approaches from the standpoint of computational complexity, although Russell and Subramanian made important steps in this direction [12].

More formally, rather than considering "perfect recall" agents, which prescribe an action for every possible run, we consider *reactive agents* that prescribe actions predicated on some *constant length* section of its current run: thus a *k-reactive* agent's action following r is determined solely by the final sequence of k state/action pairs in r rather than its entirety. The primary virtue of k-reactive agents is that their programs can always be implemented (at worst) by a look-up table of length $O(n^k)$ where n is the number of state/action pairs. (For a serious proposal to implement agents in this way, see [13], and for a critique of this proposal, see [7].)

A. Reactive Agent Design

The first bound we introduce requires agents to make a decision about what action to perform based *only* on the current state of the environment. Following the use of the term in the literature on autonomous agents, we refer to such agents as *reactive* [19].

Example 2: To better understand our notion of reactive agent, recall the environment whose state transformer function was illustrated in Figure 1. Now, consider the achievement task with goal states $\{e_2\}$. There is clearly a reactive agent to accomplish this task, defined by the following rules:

$$\begin{array}{ccc} e_0 & \longrightarrow & \alpha_1 \\ e_1 & \longrightarrow & \alpha_0 \\ e_3 & \longrightarrow & \alpha_0 \\ e_5 & \longrightarrow & \alpha_2 \end{array}$$

Formally, we have the following.

Definition 8: An agent, *Ag*, is *reactive* if for all pairs r, $r' \in R(Ag, Env)$ such that $last(r) = last(r')$ it holds that $Ag(r) = Ag(r')$.

Thus, a reactive agent, considers only its current state in choosing which action to perform, not the history leading to this.

It is important to note that there are environments and tasks for which there exists a successful agent (in the general sense), but for which no successful *reactive* agent exists. Moreover, the requirement that an environment is history independent is not sufficient to guarantee the existence of a successful reactive agent for any given task. That is, there are history independent environments and tasks for which there is no successful reactive agent, but there is a successful non-reactive agent – Example 4, below, illustrates this.

While limiting our attention to reactive agents has the disadvantage that potentially successful (non-reactive) agents may be overlooked, a significant gain is that the "program" for such agents requires (at most) $|E| \log |Ac|$ bits: the single action associated with each state of E.

Assuming τ to be history-dependent, we denote by $\mathrm{Q}_X^{(1)}$ the decision problem Q_X when solution agents are required to be *reactive*.

We recall that the complexity class Σ_2^p comprises those languages, L, membership in which is decidable by an NP program having (unit-cost) access to a co-NP oracle. Alternatively, L is defined via a ternary relation $R_L \subseteq W \times X \times Y$ for which $\langle w, x, y\rangle \in R_L$ can be decided in deterministic polynomial-time and R_L satisfies,

$$w \in L \;\Leftrightarrow (\exists x \in X \; \forall y \in Y \; \langle w, x, y\rangle \in R_L)$$

Theorem 6: $x\text{-FD}^{(1)}_{non\text{-}det}$ is Σ_2^p-complete.

Proof: To show that $x\text{-FD}^{(1)}_{non\text{-}det} \in \Sigma_2^p$ it suffices to observe that an instance $\langle\langle E, e_0, Ac, \tau\rangle, G\rangle$ is accepted if and only if there exists a reactive agent *Ag* all of whose $|E \times Ac|$-critical runs pass through an element of G. A reactive agent, *Ag* is defined by a mapping $\mu_{Ag} : E \to Ac$. Let S be the ternary relation

$$\begin{array}{c} (\langle Env, G\rangle, \mu_{Ag}, r) \in S \\ \Leftrightarrow \\ r \in C^{|E\times Ac|}(Ag, Env) \text{ includes some } g \in G \end{array}$$

where *Ag* is the reactive agent defined by the mapping μ_{Ag}. It is certainly the case that $(\langle Env, G\rangle, \mu_{Ag}, r) \in S$ can be decided in P. Moreover, $\langle Env, G\rangle$ is a positive instance of $x\text{-FD}^{(1)}_{non\text{-}det}$ if and only if

$$\exists \mu_{Ag}{:}E \to Ac \; \forall r \in C^n(Ag, Env) \; (\langle Env, G\rangle, \mu_{Ag}, r) \in S$$

and hence decidable by a Σ_2^p program.

To show that the problem is Σ_2^p-hard, we give a reduction from the Σ_2^p-complete problem QSAT_2. An instance of this is a Boolean formula $\varphi(X, Y)$ over disjoint variable sets X, Y with $|X| = |Y|$ which is accepted if $\exists \alpha_X \forall \beta_Y \varphi(\alpha_X, \beta_Y)$ holds, i.e., there is some instantiation (α_X) of X under which all instantiations (β_Y) of Y render $\varphi(X, Y)$ true. The Σ_2^p-completeness of QSAT_2 was demonstrated by Wrathall [20].

Given an instance, $\varphi(x_1, \ldots, x_n, y_1, \ldots, y_n)$ of QSAT_2, we construct an instance $\langle Env_\varphi, G_\varphi\rangle$ of $x\text{-FD}^{(1)}_{non\text{-}det}$ as follows. For $Env_\varphi = \langle E, e_0, Ac, \tau\rangle$ we set,

$$\begin{array}{lcl} E & = & \{x_1, \ldots, x_n, y_1^\top, \ldots, y_n^\top, y_1^\bot, \ldots, y_n^\bot, \top, \bot\} \\ Ac & = & \{\top, \bot, \rightarrow, eval\} \\ e_0 & = & x_1 \\ G_\varphi & = & \{\top\} \end{array}$$

The transition function $\tau(r \cdot \alpha)$ is

$$\left\{ \begin{array}{ll} \{y_i^\top, y_i^\bot\} & \text{if } last(r) = x_i \text{ and } \alpha \in \{\top, \bot\} \\ \{x_{i+1}\} & \text{if } last(r) = y_i^\top,\ i < n \text{ and } \alpha = \rightarrow \\ \{x_{i+1}\} & \text{if } last(r) = y_i^\bot,\ i < n \text{ and } \alpha = \rightarrow \\ \top & \text{if } last(r) \in \{y_n^\top, y_n^\bot\},\ \alpha = eval \text{ and } \varphi(\pi(r)) \\ \bot & \text{if } last(r) \in \{y_n^\top, y_n^\bot\},\ \alpha = eval \text{ and } \neg\varphi(\pi(r)) \\ \emptyset & \text{otherwise,} \end{array} \right.$$

where $\pi(r)$ is the instantiation of $\langle X, Y\rangle$ defined from

$$x_1 \cdot \alpha_1 \cdot y_1^{\beta_1} \cdot \rightarrow \cdot x_2 \cdot \alpha_2 \cdots y_k^{\beta_k} \cdot \rightarrow \cdot x_{k+1} \cdots \alpha_n \cdot y_n^{\beta_n} \cdot eval$$

through $x_i = \alpha_i$, $y_i = \beta_i$ for each $1 \le i \le n$.

Suppose $\langle Env_\varphi, \{\top\}\rangle$ is a positive instance of $x\text{-FD}^{(1)}_{non\text{-}det}$, i.e., there is a reactive agent, *Ag*, whose every critical run reaches the state $\top$. Consider the mapping $\mu_{Ag} : E \to Ac$ defining this agent. It is certainly the case that, $\mu_{Ag}(x_i) \in \{\top, \bot\}$ for each $x_i \in E$. Furthermore, $\mu_{Ag}(y_i^\top) = \mu_{Ag}(y_i^\bot) = \rightarrow$ for each $1 \le i < n$, and $\mu_{Ag}(y_n^\top) = \mu_{Ag}(y_n^\bot) = eval$, being the only allowable actions in these states. If we consider any critical run, r, of this agent then it ends in the state $\top$, and hence the instantiation of $\langle X, Y\rangle$ induced by $\pi(r)$ satisfies $\varphi(X, Y)$. Thus setting $x_i = \mu_{Ag}(x_i)$ yields an instantiation of α_X of X for which $\forall \beta_Y \varphi(\alpha_X, \beta_Y)$ holds. On the other hand if $\varphi(X, Y)$ is a positive instance of QSAT_2, witnessed by a setting $\alpha_X = \langle \alpha_1, \ldots, \alpha_n\rangle \in \langle \top, \bot\rangle^n$ of X, then the reactive agent defined by

$$\mu_{Ag}(e) = \left\{ \begin{array}{lll} \alpha_i & \text{if} & e \in \{x_1, \ldots, x_n\} \\ \rightarrow & \text{if} & e \in \{y_1^\top, y_1^\bot, \ldots, y_{n-1}^\top, y_{n-1}^\bot\} \\ eval & \text{if} & e \in \{y_n^\top, y_n^\bot\} \end{array} \right.$$

always achieves the state $\top$ and hence witnesses a positive instance of $x\text{-FD}^{(1)}_{non\text{-}det}$. ■

Theorem 6 indicates that deciding if a reactive agent exists is, under the usual complexity-theoretic assumptions, significantly "easier" (Σ_2^p-complete) in non-deterministic cases than deciding if an agent given the freedom to determine actions by its entire history can be used (PSPACE-complete). In contrast, our next result shows that for *deterministic* environments, there is no difference in complexity for these cases.

Theorem 7: x-FD$^{(1)}_{det}$ is NP-complete.

Proof: Membership in NP is obvious: given an instance $\langle Env, G\rangle$ of x-FD$^{(1)}_{det}$ simply non-deterministically guess an action for each state of *Env*, to define a reactive agent. Since *Env* is deterministic, this agent determines a unique run so it suffices to check whether this contains a state in G.

To prove NP-hardness, we give a reduction from SAT. Let $\psi(x_1,\ldots,x_n)$ be an instance of SAT. We define an instance $\langle Env_\psi, G\rangle$ of x-FD$^{(1)}_{det}$ with Env_ψ having state set $\{x_1,\ldots,x_n,\top,\bot\}$, initial state x_1, and actions $\{\top,\bot\}$. The transition function is given by $\tau(r\cdot\alpha) = x_{i+1}$ if $last(r) = x_i$ and $i < n$; if $last(r) = x_n$ then

$$\tau(x_1\cdot\alpha_1\cdot x_2\alpha_2\cdots x_n\cdot\alpha_n) = \psi(\alpha_1,\ldots,\alpha_n)$$

It is easy to see that $\langle Env_\psi,\{\top\}\rangle$ is a positive instance of x-FD$^{(1)}_{det}$ if and only if the formula $\psi(x_1,\ldots,x_n)$ is satisfiable, proving the theorem. ■

We note the following easy corollaries of Theorems 6, 7.

Corollary 2:

a) Ψ-FD$^{(1)}_{non\text{-}det}$ is Σ_2^p-complete.

b) Ψ-FD$^{(1)}_{det}$ is NP-complete.

Proof: The proof derives from the constructions in [4]: we omit the details. ■

B. k-Reactive Agent Design

By requiring agents to specify a single (re)action for each state, we gain a guaranteed "short program" if an appropriate agent exists, but at a potential cost of missing alternative solutions when no reactive agent is possible. It might be the case, however, that while an agent reacting to its current state only cannot be found, there are agents solving a specified design problem that, need only specify actions predicated on the last k action/state pairs in a run, without having to examine their whole history. Such "k-reactive" agents can be described by programs of size $O(|E\times Ac|^k \log |Ac|)$ bits, and thus are realistic for "small" values of k. We now consider this generalisation of the concept of reactivity to encompass k-reactive agents.

Definition 9: Given an environment *Env* and a positive integer k, an agent *Ag* is k-reactive if for all r, $r' \in R(Ag, Env)$ with

$$\begin{aligned} r &= s\cdots e_l\cdot\alpha_l\cdots e_{l+k-1}\\ r' &= s'\cdots e_l\cdot\alpha_l\cdots e_{l+k-1}\end{aligned}$$

it holds that $Ag(r) = Ag(r')$.

We denote by $\mathrm{Q}^{(k)}_X$ the agent design problem Q_X in which solution agents are required to be k-reactive.

Theorem 8: For each $k \geq 1$, and $X \in \{non\text{-}det, det\}$, x-FD$^{(1)}_X \equiv_{\log}$ FD$^{(k)}_X$.

Proof: We first show that x-FD$^{(1)}_X \leq_{\log} x$-FD$^{(k)}_X$. An instance of the former consists of an environment *Env* and subset G of E. We define a new environment *Env*′ as follows. The basic idea is that every state in *Env* corresponds to a sequence of k states in *Env*′. For every state e of *Env*, the new environment *Env*′ has states $e(1),\ldots,e(k)$. The initial state of *Env*′ is $e_0(1)$, where e_0, as usual, is the initial state of *Env*. *Env*′ has an action μ in addition to all the actions of *Env*, and its set of legal runs is defined as follows. Suppose that *Env* has a run $e_0\cdot\alpha_0\cdot e_1\cdots\alpha_{n-1}\cdot e_n$, then *Env*′ has a run $e_0(1)\cdot\mu\cdot e_0(2)\cdots\mu\cdot e_0(k)\cdot\alpha_0\cdot e_1(1)\cdots\alpha_{n-1}\cdot e_n(1)\cdots e_n(k)$. Observe that *Env*′ is deterministic if and only if *Env* is. It is easy to see that a reactive agent for *Env* exists whose runs pass contain a state in G if and only if a k-reactive agent for *Env*′ exists whose runs contain a state in $\{g(1)\ldots g(k)|g\in G\}$.

To show that x-FD$^{(k)}_X \leq_{\log} x$-FD$^{(1)}_X$, given an instance $\langle Env, G\rangle$ of the former, we can create an instance $\langle Env', G'\rangle$ of the latter in which each state of *Env*′ corresponds to each distinct seqeunce of at most $k-1$ actions and k states in *Env*. We omit the details of the straightforward simulation establishing that a k-reactive agent solves $\langle Env, G\rangle$ if and only if a reactive agent solves $\langle Env', G'\rangle$. ■

We thus obtain the following.

Corollary 3:

a) Ψ-FD$^{(k)}_{non\text{-}det}$ is Σ_2^p-complete.

b) Ψ-FD$^{(k)}_{det}$ is NP-complete.

Proof: Immediate from Theorems 6–8 above. ■

C. Oblivious Agent Design

The concept of k-reactive ($k \geq 1$) agent offers one mechanism by which "concise" solutions to agent design tasks may be described in history-dependent environments. We can also, however, consider a superficially similar idea – that of an *oblivious* agent solution. Informally, an oblivious agent is one which takes no account of its current state in choosing an action, only of the *number* of actions its has performed so far. Thus one might regard an oblivious agent (with respect to our concept of reactivity) as being "0-reactive".

Example 3: Recall again the example presented earlier. In this environment, there is an oblivious agent to accomplish the achievement task with goal states $\{e_1, e_2\}$, by simply performing α_0.

More formally, we have the following.

Definition 10: An agent, *Ag*, is *oblivious* if for all pairs r, $r' \in R(Ag, Env)$ if $|r| = |r'|$ then $Ag(r) = Ag(r')$.

Thus, in settings where agents must cease their operations after some (maximum) number of actions, m say, an oblivious agent is specified by a mapping $\omega : \{1, 2, \ldots, m\} \rightarrow Ac$ describing what action is to performed at each stage i from the initial state ($i = 1$) onwards. For the design problem x-FD$_X$ on which we have focused we can introduce an oblivious variant as follows.

Definition 11: An instance of the *oblivious achievement design* problem (x-FD$_X^{(0)}$) consists of an environment $Env = \langle E, e_0, Ac, \tau \rangle$, a subset G of E, and a positive integer, m. An instance is accepted if there is a mapping $\omega : \{1, 2, \ldots, t\} \rightarrow Ac$ such that $t \leq m$ and for Ag_ω the agent defined by

$$Ag_\omega(r) = \begin{cases} \omega((|r|+1)/2) & \text{if} \quad |r| \leq 2t-1 \\ \otimes & \text{if} \quad |r| \geq 2t \end{cases}$$

then every run $r \in R(Ag_\omega, Env)$ contains some state of G.

An oblivious agent can be specified using at most $m \log_2 |Ac|$ bits, and thus if $m \ll |E \times Ac|$ may represent a significant saving over even 1-reactive agents.

From our earlier results, however, it turns out that deciding if oblivious agents exist is "no easier" than deciding if reactive ones do.

Theorem 9:

a) x-FD$^{(0)}_{non\text{-}det}$ is Σ_2^p-complete.

b) x-FD$^{(0)}_{det}$ is NP-complete.

Proof: For (a) the reduction of Theorem 6 is used to construct a instance of x-FD$^{(0)}_{non\text{-}det}$ in which $m = 2n$. It then suffices to observe that x_i actions are determined on *odd* indexed moves and the only applicable actions on even indexed moves are $\rightarrow$ and (finally) *eval*. Thus an oblivious agent can be specified if and only if the instance of QSAT$_2$ from which $\langle Env_\varphi, G_\varphi, 2n \rangle$ results would be accepted. The proof of (b) is similar. ■

So far we have considered only agents that must realise their specified task (whether achievement or maintenance) within some limited number of actions, i.e., with respect to the critical runs of length $|E \times Ac|$. Of course, within deterministic environments even using oblivious agents, determining whether there exists an agent that "eventually" realises an achievement task is easily shown to be undecidable. This situation changes when we consider agents operating in history-*independent* environments, i.e., where the change in environment state specified by τ depends only on its current state and the action chosen. We conclude this section by presenting some results concerning oblivious agent design tasks in history-independent *non-deterministic* environments.

Any environment of this form is naturally modeled by a *directed graph* $H(E, A)$ whose vertices correspond to the possible states and in which there is an edge $\langle e_i, e_j \rangle$ labelled $\alpha \in Ac$ whenever $e_j \in \tau(e_i, \alpha)$. Oblivious agents in this setting may be regarded simply as mappings $Ag : \mathbb{N} \rightarrow Ac \cup \{\otimes\}$. Of course, in non-deterministic environments, it could happen that such an agent reaches different states e_i and e_j after k actions are performed, but these are such that

$$\tau(e_i, Ag(k+1)) \neq \emptyset \quad ; \quad \tau(e_j, Ag(k+1)) = \emptyset$$

It is convenient to pre-empt this possibility by adding a "special" *dead* state, $e_\emptyset$ to the environment such that for each pair $\langle e, \alpha \rangle \in E \times Ac$, should $\tau(e, \alpha) = \emptyset$, then the directed graph $H(E, A)$ contains an edge $\langle e, e_\emptyset \rangle$ labelled α; additionally there are edges $\langle e_\emptyset, e_\emptyset \rangle$ labelled α for each $\alpha \in Ac$. It is noted that we neither require nor assume $H(E, A)$ to be *acyclic*.

The most "general" form of the oblivious agent design problem that we consider in this context of history-independent, non-deterministic environments is that defined below.

Definition 12: Let $\Psi(x_1, \ldots, x_t)$ be a (non-constant) propositional logic function of t variables. An instance of the Ψ-*Oblivious Agent Design* problem (Ψ-OAD) consists of: an edge-labelled directed graph $H(E, A)$ as arising from a history-independent, non-deterministic environment $\langle E, e_0, Ac, \tau \rangle$; and a partial mapping $\Pi : E \rightarrow \{x_1, \ldots, x_t\}$ associating each state with (at most) one variable x_i of Ψ. An instance is accepted if there is a value $n \in \mathbb{N}$ and an oblivious agent $Ag : \mathbb{N} \rightarrow Ac \cup \{\otimes\}$ for which

$$\begin{array}{ll} Ag(m) \in Ac & \textit{if} \quad m < n \\ Ag(m) = \otimes & \textit{if} \quad m \geq n \end{array}$$

and if $r = e_0 \cdot s_1 \cdot s_2 \cdots s_m$ is any sequence of $m+1$ states traversed by Ag in $H(E, A)$ then $\Psi(\pi(r)) = \top$, where $\Psi(r)$ is the instantiation of $\langle x_1, \ldots, x_t \rangle$ defined through: $x_k = \top$ if any state, e with $\Pi(e) = x_k$ occurs in r; $x_k = \bot$ if no such state does.

For reasons of limited space the following results are merely stated without proof.

Theorem 10: For all propositional functions $\Psi(x_1, \ldots, x_k)$, the problem Ψ-OAD is decidable.

Theorem 11: $(x_1 \wedge \bar{x_2})$-OAD is PSPACE-hard.

Some remarks regarding the relationship between Theorem 11 and Theorem 9 may seem in order: the former appearing to claim that the history-independent version of a problem is rather more difficult that its history-dependent counterpart. The important difference between these two cases is that, in contrast to Ψ-OAD, an instance of x-FD$^{(0)}_{non\text{-}det}$ include a stated bound on the number of actions an oblivious agent is allowed to perform: in Ψ-OAD whether any such bound *exists* has

to be decided. As we noted earlier, the "exact" analgoue of Ψ-OAD for history-dependent environments is, in fact, undecidable, hence the relevance of Theorem 10.

D. History-Independent Environments and Reactive agents

We write FC to refer to the *finite conjunction* problem: in such a problem, we are given a set of "good" states, and a set of "bad" states, and asked whether there exists an agent that can bring about a good state while avoiding all bad states. Clearly FC is a special case of Ψ-$FD_{non\text{-}det}$ in which the formula Ψ is of the form $x \wedge \bar{y}$.

Definition 13: Given an environment $Env = \langle E, e_0, Ac, \tau \rangle$ and sets $G, B \subseteq E$, we say that an agent Ag solves the Conjunction problem for the tuple $\langle Env, G, B \rangle$ if every run of $\langle Env, Ag \rangle$ passes through G and none passes through B.

In this subsection we consider the FC problem under the restriction that the agent is reactive, and the environment is history independent. In one sense, this captures the idea that the agent and environment have the "same power". Our first result is as follows.

Lemma 2: Let $Env = \langle E, e_0, Ac, \tau \rangle$ be a history-independent bounded environment and let $G, B \subseteq E$. If there exists an agent Ag solving $\text{FC}_{non\text{-}det}$ for the instance $\langle Env, G, B \rangle$, then there exists a *reactive* agent solving this problem for the same instance.

Proof: It suffices to prove the following; given an agent Ag such that every run through Env permitted by Ag ending at a sink in the directed graph $H(V, A)$ defined by Env passes through G but not through B, there is a reactive agent A satisfying this condition. We define A as follows; let $e \in E$. If no run permitted by Ag ends at e then define the action $A(e)$ arbitrarily. If there does exist a run permitted by Ag and ending at e, but all such runs pass through G at some point, let r be any such run and define $A(e) = Ag(r)$. Lastly, if there is a run s permitted by Ag and ending at e, which does not pass through G, define $A(e) = Ag(r)$.

Given a run r permitted by A which passes through the states $e_0, \ldots, e_n$ in succession, the following is easily proved by induction on n; there is a run permitted by Ag passing through e_n, and if none of the states e_i lies in G, then there exists a run permitted by Ag and ending at e_n, which does not pass through G. From the first assertion, it follows that no run permitted by A passes through B. To show that every run permitted by A passes through G, assume that this is false. Thus since Env is bounded, there is a run r permitted by A which passes through the states $e_0, \ldots, e_n$ in succession, and such that the last state e_n is a sink in the directed graph $H(V, A)$ defined by Env. From the second assertion above, there is a run permitted by Ag which ends at the sink e_n and does not pass through G, contradicting the hypotheses. ■

We can now prove the following.

Theorem 12: If the state transformer function is history-independent, then:

a) FC_{det} with respect to reactive agents is NL-complete.
b) $\text{FC}_{non\text{-}det}$ with respect to reactive agents is P-complete.

Proof:

a) Given an instance $\langle Env, G, B \rangle$ of this problem, let H be the acyclic directed graph defined by Env with all states in B and incident edges deleted. We prove that problem is solvable in NL-space first. Assume first that $G = \{g\}$. Then the problem is solvable for the instance $\langle Env, \{g\}, B \rangle$ if and only if there is a directed path in H from the initial state of Env to g *and* there is a directed path in H from $\{g\}$ to a sink in Env. Both problems are decidable in NL-space [11, p398], and thus the general problem for arbitrary G lies in this class. To show that the problem is NL-space hard, observe that the graph reachablity problem for finite directed graphs, which is NL-space hard [11, p398], can easily be reduced to an instance of our problem for which $B = \emptyset$.
b) By Lemma 2, the restriction that the agent be reactive may be ignored. Thus membership in P follows from Theorem 5. The proof of P-hardness follows a reduction from the monotone circuit reduction problem.

■

Lemma 2 does not generalise to arbitrary formulae. We now give an example to show that the requirement for an agent to be reactive is a real restriction for Ψ-$FD_{non\text{-}det}$.

Example 4: Let Env be the (history-independent) environment with state set $\{e_0, \ldots, e_5\}$, of which e_0 is the initial state. Env has action set $\{\alpha, \beta\}$ and transition function τ given by $\tau(e_0, \alpha) = \{e_1, e_2\}$, $\tau(e_1, \alpha) = \tau(e_2, \alpha) = \{e_3\}$, $\tau(e_3, \alpha) = \{e_4\}$, $\tau(e_3, \beta) = \{e_5\}$ and τ returns the empty set in all other cases. Consider the problem $(x \wedge y)$-$FD_{non\text{-}det}$. This is solvable for the instance $\langle Env, \{e_1, e_4\}, \{e_2, e_5\} \rangle$, but the only agent solving it is the non-reactive agent Ag defined by $Ag(e_0, \alpha, e_1, \alpha, e_3) = \beta$ and $Ag(e_0, \alpha, e_2, \alpha, e_3) = \alpha$. However we do get a partial result.

Definition 14: Let $\Psi = x_1 \wedge \ldots x_n \wedge \bar{x}_{n+1}$ be a formula for some $n \geq 1$. The *restricted* Ψ-$FD_{non\text{-}det}$ problem is defined as the normal Ψ-$FD_{non\text{-}det}$ problem, but subject to the additional constraint that for each $i \in \{2, \ldots, n\}$, the set of states defined by the atomic proposition x_i in the particular environment must be a singleton.

Lemma 3: Let $Env = \langle E, e_0, Ac, \tau \rangle$ be a history-free bounded environment and let $\Psi = x_1 \wedge \ldots x_n \wedge \bar{x}_{n+1}$ be a formula. Let $S = \langle E_1, \ldots, E_{n+1} \rangle$ be an ordered collection of pairwise disjoint subsets of E satisfying $|E_i| = 1$ for each $i \in \{2, \ldots, n\}$. If there exists an agent *Ag* solving Ψ-*FD*$_{non\text{-}det}$ for the instance $\langle Env, S \rangle$, then there exists a *reactive* agent solving this problem for the same instance.

Proof: Given a state $f \in E$, let us call an agent for *Env* f-reactive if it has the same image for every run r with $last(r) = f$. Since *Env* is history-free and bounded, its underlying graph is acyclic, and so if the agent *Ag* is not g-reactive, but is f-reactive for every state f reachable from g in the underlying graph of *Env*. We will construct an agent Ag' also solving Ψ-*FD*$_{non\text{-}det}$ for the instance $\langle Env, S \rangle$, and which is g-reactive and differs from *Ag* only on runs whose last state is g. By repeating this process we eventually construct a reactive agent.

Let r and s be two runs permitted by *Ag* and ending at g and let $i \in \{2, \ldots, n\}$. Suppose that r passes through the singleton E_i but s does not. Thus s can be extended to a run ss' permitted by *Ag* such that s' passes through E_i. Since *Ag* is f-reactive for every state f reachable from g in the underlying graph of *Env*, the run rs' is also permitted by *Ag* and passes twice through the singleton E_i, giving a contradiction since the underlying graph of *Env* is acyclic. Thus we have shown that all runs or no runs ending at g and permitted by *Ag* pass through E_i.

We define the new agent Ag' as follows. Let $Ag'(g) = Ag(r)$, where r is a run permitted by *Ag* with $last(r) = g$ chosen as follows. If there is a run s with $last(s) = g$, such that all runs ss'' permitted by *Ag* pass through an element of E_1 *after* g (that is, s' passes through an element of E_1) then let $r = s$; otherwise let r be arbitrary. In the former case, every run permitted by Ag' and passing through g will pass through an element of E_1 after g; in the latter case, by the choice of *Ag*, every run passing through g will pass through an element of E_1 before g (or $g \in E_1$). If $i \in \{2, \ldots, n\}$, and r passes through E_i, then by the observation above, all runs passing through g and permitted by Ag' pass through E_i before g; otherwise every maximal extension of r permitted by *Ag* must pass though E_i after g, and hence all maximal runs passing through g and and permitted by Ag' pass through E_i before g. Similarly, runs permitted by Ag' can be shown not to pass through E_{n+1}. Lastly, if a run permitted by Ag' does not pass through g, then it is also permitted by *Ag*. Hence we have shown that Ag' solves Ψ-*FD*$_{non\text{-}det}$ for the instance $\langle Env, S \rangle$, as required. ■

Theorem 13: Let $\Psi = x_1 \wedge \ldots x_n \wedge \bar{x}_{n+1}$ be a formula. Then the *restricted* Ψ-*FD*$_{non\text{-}det}$ problem for history-free environments and reactive agents lies in P.

Proof: This follows immediately from Lemma 3 and Theorem 5. ■

V. Stochastic Agent Design

So far, we have been considering a rather pessimistic notion of success with respect to tasks: an agent must be guaranteed to satisfy its task on *all* runs, otherwise it is deemed to be no good. At the other extreme, a rather too *optimistic* notion of success was studied in [16], where an agent was considered acceptable if it succeeded to accomplish its designated task on *at least one* run. It was shown in [16] that the optimistic agent design problem was "easier" than pessimistic agent design: versions of the agent design problem that are PSPACE-complete are when considering pessimistic agent design are "merely" NP-complete in the optimistic case.

Whereas the strict, pessimistic notion of success that we have discussed so far in this paper is perhaps too strong, the optimistic agent design problem is too weak. In practice, we would probably want to know whether the agent could be *expected* to succeed – that is, given a rational number $p \in [0, 1]$, whether this agent will succeed with probablity greater than p. We refer to this problem as *stochastic agent design*. The critical assumption that we must make in order to study this problem is that all (immediate) outcomes of an action are equiprobable.

Definition 15: The Ψ-*Stochastic Agent Design* problem (Ψ-SAD) takes as an instance an environment $Env = \langle E, e_0, \tau \rangle$, state sets $E_1, \ldots, E_n \subseteq E$ (where the formula Ψ is over the atomic propositions $x_1, \ldots, x_n$) and rational $p \in [0, 1]$, and returns "true" if and only if there is an agent that succeeds with this task (in the sense of Definition 3) with probability greater than p, "no" otherwise.

It should be noted that not all runs through an environment have the same probability of occurring and so an agent having a probability of success greater than p is *not equivalent* to an agent succeeding on proportion p of its runs. To see this, we give an example for $p = 1/2$. Consider an environment in which there is a single action α which leads to 2 states from the initial state; one of these has no available actions; the other (for some value k) leads to a binary computation tree of height k in which all but one of the states at the leaves is a state in S: the unique agent in this environment has $2^k + 1$ runs of which the majority $(2^k - 2)$ succeed. The probability of success, however, is less than $1/2$: the state with further available actions is chosen with probability $1/2$, but with non-zero probability a run from this state does not succeed.

Definition 16: Let Ψ be a Boolean formula over a set $\{x_1, \ldots x_m, \bar{x}_1, \ldots, \bar{x}_m\}$. The Ψ-Stochastic Agent Design problem (Ψ-SAD) takes as instance a tuple $\langle Env, S, p \rangle$,

where *Env* is a bounded environment, $S = \langle E_1, \ldots, E_m \rangle$ is an ordered collection of pairwise disjoint subsets of the state set of *Env* and p is a rational number lying in $[0, 1]$. The problem has answer "yes" if and only if there is an agent *Ag* such that the pair $\langle Env, Ag \rangle$ satisfies Ψ with respect to the set sequence S with probability greater than p. If $\Psi = x_1$, then we refer to Ψ-SAD as the Achievement Stochastic Agent Design Problem (ASAD); if $\Psi = \bar{x}_1$, then we refer to Ψ-SAD as the Maintenence Stochastic Agent Design Problem (MSAD).

Lemma 4: Ψ-SAD lies in PSPACE for all Boolean formulae Ψ.

Proof: Let $\langle Env, S, q \rangle$ be an instance of Ψ-SAD and let $n = |E \times Ac|$. To show that Ψ-SAD $\in$ PSPACE, for each Boolean formula θ over the set $\{x_1, \ldots x_m, \bar{x}_1, \ldots, \bar{x}_m\}$ we consider a function – Ext_θ – that maps possible runs (ending in a state) from Env_θ to *rational* values, p, in the range $[0, 1]$. This function $Ext_\theta(r \cdot e)$ has value 0 if the length of $r \cdot e$ exceeds n. If not, and if $\tau(r \cdot e \cdot \alpha = \emptyset$ for all actions α, then $Ext_\theta(r \cdot e)$ is 1 if the set $\{e\}$ satisfies θ with respect to S, and 0 otherwise. In all other cases, $Ext_\theta(r \cdot e)$ is:

$$\max_{\alpha : \tau(r \cdot e \cdot \alpha) \neq \emptyset} \frac{1}{|\tau(r \cdot e \cdot \alpha)|} \sum_{d \in \tau(r \cdot e \cdot \alpha)} Ext_{\theta(S,e)}(r \cdot e \cdot \alpha \cdot d)$$

The function $Ext_\theta(r \cdot e)$ is the (maximum) probability that an agent continuing a run $r \cdot e$ attains one of the goal states. It follows that $\langle Env, S, q \rangle$ is a positive instance of Ψ-SAD if and only if $Ext(e_0) > q$ and so it suffices to show that the value $Ext_\Psi(e_0)$ can be computed in PSPACE. In order to simplify the algorithm presentation we use a *non-deterministic* machine to "guess" the maximising action for its input run: certainly there is (at least) one pattern of such guesses which will correctly compute $Ext_\Psi(e_0)$. The algorithm is recursive and given a run $r \cdot e$ and a Boolean formula Ψ over the set $\{x_1, \ldots x_m, \bar{x}_1, \ldots, \bar{x}_m\}$ proceeds as follows:

1. If $|r \cdot e| > n$ return 0 else
   - if $\forall \alpha$, $\tau(r \cdot e \cdot \alpha) = \emptyset$ return 1 if the set $\{e\}$ satisfies Ψ with respect to S, and 0 otherwise, else
2. Non-determininistically choose an available action α.
   2.1 *count* $:= 0$.
   2.2 for each $d \in \tau(r \cdot e \cdot \alpha)$,

$$count := count + \frac{Ext_{\Psi(S,e)}(r \cdot e \cdot \alpha \cdot d)}{|\tau(r \cdot e \cdot \alpha)|}$$

   2.3 return *count*

It remains to observe that this computation can be carried out in NPSPACE: only rational values are involved, and using representation of these by two integer values (i.e., numerator and denominator), the value of the denominator never exceeds n^n and so can be stored in $O(n \log n)$ bits. The depth of the recursion is $O(n)$ and the computation of $\Psi(S, e)$ requires only polynomial space, so the entire procedure can be realised using only polynomial (in n) space. Since NPSPACE = PSPACE the membership proof is completed. ■

Lemma 5: The MSAD and ASAD problems are PSPACE-hard.

Proof: We first give the proof for the ASAD problem, and then give the slight modification needed for the proof for MSAD.

We show how the problem of *stochastic satisfiability* (SSAT) can be reduced to ASAD. The instance of ASAD that we will construct will have probability $p = 1/2$. An instance of SSAT is given by a formula with the form:

$$\exists x_1 . \mathsf{R} x_2 . \exists x_3 . \mathsf{R} x_4 \cdots Q x_n . \ \varphi(x_1, \ldots, x_n) \qquad (1)$$

where:

- each x_i is a Boolean variable (it does *not* need to be a collection of variables);
- R is a "random" quantifier, with the intended interpretation "with some randomly selected value"; and
- Q is $\exists$ if n is odd, and R otherwise.

The goal of SSAT is to determine whether there is a strategy for assigning values to existentially quantified variables that makes (1) true with probability great than $\frac{1}{2}$, i.e., if

$$\exists x_1 . \mathsf{R} x_2 . \exists x_3 . \mathsf{R} x_4 \cdots Q x_n . \ \mathsf{prob}[\varphi(x_1, \ldots, x_n) = \top] > \frac{1}{2}$$

We give an outline of the reduction first. To reduce an instance (1) to ASAD, we proceed as follows. The idea is to create an environment which forces the agent and environment to take it in turns, starting with the agent, to assign values (truth or falsity) to the variables in (1) – the agent assigns values to $\exists$-quantified variables, the environment assigns values to R-quantified variables. The environment is non-deterministic, with the actual assignment of values done at random. After all variables have been assigned values, the environment returns the goal state, $e^\top$, if the formula $\varphi(x_1, x_2, x_3, \ldots, x_n)$ is made true by the valuation traced out by agent and environment, and a dummy, "fail" state, $e^\perp$, otherwise. Each run thus corresponds to a possible valuation for the variables, and all possible valuations will be runs. Recall that the goal of the SSAT problem is to determine whether there is a strategy for assigning values to $\exists$-variables that makes the formula true in the majority of runs: this will clearly be the case just in case there is an agent that can succeed with the task (make the formula true) with probability greater than $1/2$.

The details are as follows. For each $\exists$-quantified variable x_i, we create two actions, $\alpha_i^\top$ and $\alpha_i^\perp$, corresponding to the assignment of truth or falsity to x_i,

respectively. For each R-quantified variable x_j, we create two environment states, $e_j^\top$ and $e_j^\bot$, again corresponding to an assignment of truth or falsity to x_j. We also create three additional environment states, e_0, $e^\top$ and $e^\bot$, let the initial state be e_0, and let the set of goal states S be a singleton containing $e^\top$. The state transformer function of the environment behaves as follows. The agent and environment alternate to give valuations to the quantified variables, with the agent going first, working from outermost variable to innermost. The agent is first allowed to pick a value for x_1 by performing $\alpha_1^\top$ or $\alpha_1^\bot$, to which the environment responds with either $e_2^\top$ or $e_2^\bot$, to which the agent must respond with either $\alpha_3^\top$ or $\alpha_3^\bot$, and so on. When all variables have been given values, the environment responds with either $e^\top$ if $\varphi(x_1, x_2, x_3, \ldots, x_n)$ is true under the valuation traced out, and $e^\bot$ otherwise. For example, the following run

$$e_0 \xrightarrow{\alpha_1^\top} e_2^\top \xrightarrow{\alpha_3^\bot} e^\top$$

is a run in which the variables $x_1, \ldots, x_3$ are assigned the values true, true, and false respectively; the environment indicates that the formula under question is true under this valuation. To ensure that the environment can always respond with a final valuation, we can "pad" runs by allowing an additional dummy action by the agent.

In the MSAD case, the environment constructed is identical and the "bad" state set is defined to be $\{e^\bot\}$. ■

Lemma 6: Let Ψ be a Boolean formula over a set $\{x_1, \ldots x_m, \bar{x}_1, \ldots, \bar{x}_m\}$. If Ψ is not constant (that is, Ψ is neither a tautology nor unsatisfiable) then for some $i \in \{1, \ldots, m\}$, there is an interpretation $x_j \rightarrow T_j \in \{true, false\}$ for each $j \neq i$ such that the Boolean formula $\Psi[x_j \rightarrow T_j; j \neq i]$ is equivalent to either x_i or $\bar{x}_i$.

Proof: This follows by induction on m. If the lemma is false for $i = 1$, then for every interpretation $x_j \rightarrow T_j \in \{true, false\} | j > 1$ the formula $\Psi[x_j \rightarrow T_j \in \{true, false\} | j > 1]$ over $\{x_1, \bar{x}_1\}$ is equivalent to either *true* or *false*, and hence Ψ is equivalent to a non-constant formula Ψ which does not contain x_1 or $\bar{x}_1$. Thus the lemma follows by the inductive hypothesis applied to Ψ. ■

Theorem 14: Ψ-SAD is PSPACE-complete for all Boolean formulas Ψ.

Proof: Membership in PSPACE is given by Lemma 4. To show PSPACE-hardness, we first show that for every Boolean formula Ψ, there is a problem $Q \in \{\text{ASAD}, \text{MSAD}\}$ such that any instance of Q can be regarded as an instance of Ψ-SAD. For this we use Lemma 6. We assume that Ψ is over the set $\{x_1, \ldots x_m, \bar{x}_1, \ldots, \bar{x}_m\}$ of atoms. Thus for some $i \in \{1, \ldots, m\}$, there is an interpretation $x_j \rightarrow T_j \in \{true, false\}$ for each $j \neq i$ such that the Boolean formula $\Psi[x_j \rightarrow T_j; j \neq i]$ is equivalent to either x_i or $\bar{x}_i$. We will assume the former; the latter case is similar. An instance $\langle Env, \langle S_i \rangle, p \rangle$ of x_i-SAD (that is, of ASAD) defines the same answer to ASAD as the instance $\langle Env, \langle S_1, \ldots, S_m \rangle, p \rangle$ does to Ψ-SAD provided that for each $j \neq i$, the set $S_j = E$ if $T_j = true$ and $S_j = \emptyset$ otherwise, where E is the state set of *Env*. Since the instance $\langle Env, \langle S_1, \ldots, S_m \rangle, p \rangle$ can be constructed in polynomial time from $\langle Env, \langle S_i \rangle, p \rangle$, and ASAD is PSPACE-hard by Lemma 6, we have shown that Ψ-SAD is PSPACE-hard. ■

You might now expect us to consider the Ψ-SAD$_{det}$ problem, i.e., the stochastic finite agent design problem for deterministic environments. For deterministic environments, however, the stochastic agent design problem is not meaningful: an agent will only ever have one run in a deterministic environment, and it thus makes no sense to ask whether the agent succeeds with probability exceeding $1/2$ – its probability of success is either 0 or 1.

However, we can ask a closely related question, namely, do most *agents* satisfy the task in a deterministic environment. That is, given a deterministic environment and a particular (achievement) task, do the majority of agents satisfy this task in the environment? Another way of looking at this problem is to say that if we selected an agent at random, are the chances better than even that this agent would succeed with the task in the environment? We call this problem *majority agent design* (MAJD):

Definition 17: The *Majority agent design* (MAJD) problem takes as an instance an achievement agent design task with environment $Env = \langle E, e_0, \tau \rangle$ and goal states $S \subseteq E$. It returns "true" if and only if *the majority* of agents achieve the task, "no" otherwise.

We can immediately show:

Theorem 15: MAJD$_{det}$ is PP-complete.

Proof: We must first show the problem is in PP. The following PP algorithm decides the problem (see [11, pp.256–257] for the definition of the class PP):

1) a computation non-deterministically "guesses" a run consistent with the state transformer function τ of the environment;
2) a computation "accepts" if the task is accomplished on this run, otherwise it "rejects".

The PP machine accepts if the majority of its computations accept. Each different run corresponds to a different agent. Since we are considering finite runs, each run is guaranteed to have length at most polynomial in $|E \times Ac|$, the machine decides the problem.

To show that the problem is PP-hard, we reduce the MAJSAT problem [11, p.256]. An instance of MAJSAT is simply a propositional logic formula φ over Boolean

variables $x_1, \ldots, x_n$. The goal is to answer "yes" if the formula is true under most valuations of these variables, "no" otherwise. We create an instance of MAJD_{det} as follows. The idea is that any given agent simply traces out a valuation for the variables: every different agent thus corresponds to a different valuation. Formally, the environment we create has four states: $\{e_0, e^\top, e^\bot\}$, with the initial state being e_0, and the states $e^\top$ and $e^\bot$ respectively used to indicate whether or not the valuation traced out makes the formula true or false respectively; the set of goals states is a singleton containing $e^\top$. Until the agent has completed assigning values to variables, the environment responds to all assignments with e_0. When all Boolean variables have been assigned values, the environment responds with $e^\top$ (if the formula is true under the assignment traced out), or $e^\bot$ otherwise; the run then ends. ■

The MAJD problem gives us a (somewhat crude) measure of the inherent hardness or easiness of a task: if there is a better than evens chance that an agent selected at random will succeed with the task, then the task itself must be easy to solve.

VI. Related Work

A. Relationship to AI Planning

In the AI literature, the most closely related work to our own is on the complexity of the planning problem: Bylander was probably the first to undertake a systematic study of the complexity of the planning problem; he showed that the (propositional) STRIPS planning problem is PSPACE-complete [3]. Building on his work, many other variants of the planning problem have been studied – recent examples include [1], [10]. The main differences between our work and the work on AI planning is as follows:

- The notion of an agent in our work (as a function that maps runs to selected actions) is more general than the notion of a plan as it commonly appears in the planning literature. Our agents are more akin to the notion of a *strategy* in game theory. The obvious advantage of our approach is that our results are not bound to a particular plan representation. The obvious disadvantage is that having a positive answer to one of our agent design problems does not imply that an agent to carry out the task will be *implementable* (cf. [12]).
- Most complexity results in the planning literature are bound to particular *representations* of goals and actions. The STRIPS notation in Bylander's work is one example [3]; Baral *et al.* use the action description language $\mathcal{A}$ [1]; in the work of Littman *et al*, the representation chosen is ST [10]. In some cases, it is not clear whether results reflect the complexity of the decision problem, or whether they are at least in part an artifact of the representation. We have adopted the most general representations possible, for example representing achievement and maintenance tasks through sets of states, rather than through a particular (e.g., logical) formalism, and consider several possible representations of the environment.
- Most complexity results in planning assume simple (deterministic, history independent) environments.
- The focus in the planning literature has been almost exclusively on achievement tasks. We have also considered maintenance tasks, and elsewhere, have considered richer task specifications [16].

B. Relationship to Game and Decision Theory

In the computational complexity literature, the most relevant problems are those of determining whether or not a given player has a winning strategy in a particular two-player game. PSPACE-completeness appears to be the characteristic complexity result for such problems [11, pp459–480].

VII. Conclusions and Future Work

In this paper we have extended the results of [14], [16], [17] in three ways.

First, we considered the specification of task via arbitrary Boolean combinations of achievement and maintenance problems. It was shown that with history-dependent environments the the decision problem with arbitrary non-trivial formulae is "no harder" than that for single variable formula specifications. In contrast, one setting of these problems in history-independent environments gives rise to NP-complete decision problems.

Second, we considered *bounded* agent design problems, where a successful agent is constrained to have some "memory bound". We showed that, for non-deterministic history-dependent environments, the agent design problem for such bounded agents was easier (under standard complexity-theoretic assumptions) than its more general counterpart. For deterministic environments, the complexity of the problems coincided.

We also considered stochastic variants of agent design problems, where we ask whether there is an agent that succeeds with probability greater than p, for some rational p in the interval $[0, 1]$.

There are many issues that demand attention in future work. One is the relationship of our work to that of solving Markov decision problems [9]. Another key problem is that of determining the extent to which our positive (polynomial time) results can be exploited in

practice. Yet another is on extending our task specification framework to allow richer and more complex tasks.

ACKNOWLEDGEMENTS

This research was supported by the EPSRC under grant GR/R60836/01 ("Algorithmics for agent design and verification").

APPENDIX

For convenience, we provide a summary of the problems that we have studied in this paper and the results obtained. Recall that if $X_n = \{x_1, \ldots, x_n\}$ is a set of Boolean variables, then the notation $\Psi(X_n)$ is used to denote a propositional logic formula over variables X_n. Note that:

- We assume *bounded* environments unless explicitly stated otherwise.
- We frequently make reference to *achievement* and *maintenance* task – recall that an achievement task is specified by a set of "good" or "goal" states, and an agent succeeds with such a problem in a given environment if, for every possible run of the agent in the environment, at least one of the good states occurs (it does not need to be the same state on every run); in contrast, a maintenance task is specified by a set of "bad" states, the idea being that an agent succeeds with such a task if, on every possible run of the agent in the environment, no bad state occurs.
- We also distinguish between *non-deterministic* and *deterministic* environment. A non-deterministic environment is one in which there are potentially more than one possible successor states, whereas in a deterministic environment, there is always at most one possible successor state. Given a particular problem P, we use the notation P_{det} and $\text{P}_{non\text{-}det}$ to distinguish between the variants of this problem corresponding to the assumption of deterministic and non-deterministic environments, respectively.
- We also distinguish between environments that are *history dependent* and *history independent*: a history dependent environment is one that is permitted to make its "decision" about the next possible states of the environment based on the entire run so far, while a history independent environment is one that must make its "decision" based only on the final state of the environment and action performed.

For completeness, we begin by summarising results obtained in previous work:

x-Design$^{\infty}$
: The achievement design problem for unbounded environments. Recursively enumerable but not recursive. (Theorem 1.)

$\bar{x}$-*Design*$^{\infty}$
: The maintenance design problem for unbounded environments. Not recursively enumerable. (Theorem 1.)

$\Psi(x)$-*Design*$^{<\infty}$
: The achievement or maintenance design problem for finite environments (observe that $\Psi(x)$ is a propositional logic formula of one variable). Recursive but non-elementary in general. (Theorem 1.)

x-FD$_X$ for $X \in \{det, non\text{-}det\}$
: The achievement design problem for deterministic/non-deterministic bounded environments. PSPACE-complete for history-dependent non-deterministic environments, NP-complete for history-dependent deterministic environments, NL-complete for history independent non-deterministic environments, and P-complete for history-independent deterministic environments. (Theorem 1.)

$\bar{x}$-*FD*$_X$ for $X \in \{det, non\text{-}det\}$
: The maintenance design problem for deterministic/non-deterministic bounded environments: equivalent via logspace reductions to the corresponding achievement design problem (see preceding entry), and hence PSPACE-complete for history-dependent non-deterministic environments, NP-complete for history dependent deterministic environments, NL-complete for history independent non-deterministic environments, and P-complete for history-independent deterministic environments. (Theorem 1.)

The following results were obtained in the present paper:

Ψ-*Design*$^{\infty}$
: The agent design problem for unbounded environments, where a task is specified as an arbitrary propositional logic formula, but where this formula is fixed, and *does not form part of a problem instance*. Recursive if, and only if, Ψ is trivial (logically equivalent to a Boolean constant), and recursively enumerable if, and only if, the Boolean function f_Ψ corresponding to Ψ is monotone. (Theorem 2.)

Ψ-*FD*$_X$ for $X \in \{det, non\text{-}det\}$
: The agent design problem where a task is specified as an arbitrary propositional logic formula, but where this formula is

fixed, and *does not form part of a problem instance*. Equivalent via polynomial time reductions to *x*-FD_X for history-dependent environments, and hence PSPACE-complete for history-dependent non-deterministic environments, and NP-complete for history-dependent deterministic environments. (Theorem 3.) For history-independent environments, Ψ-FD_X is in P even if the environment is non-deterministic, i.e., $X = \textit{non-det}$ (Theorem 5.)

Ψ-FED_X for $X \in \{\textit{det}, \textit{non-det}\}$
: The agent design problem where a task is specified as an arbitrary propositional logic formula *which forms part of a problem instance*, i.e., is given "explicitly". For history-dependent environments, equivalent via polynomial time reductions to *x*-FD_X, and hence PSPACE-complete for non-deterministic history-dependent environments, and NP-complete for non-deterministic history dependent environments. (Theorem 3.) For history-independent environments, however FED_{det} i.e., the problem for deterministic environments, is NP-complete (and hence apparently harder than in the case where the task specification is implicit). (Theorem 4.)

Ψ-$\text{FD}_X^{(k)}$ for $X \in \{\textit{det}, \textit{non-det}\}$
: The agent design problem for non-deterministic environments and *k*-bounded agents, where tasks are implicitly specified via the propositional formula Ψ – i.e., the problem of determining whether an agent exists that will accomplish the task even when that agent is only allowed to "remember" the final *k* states of the environment. For achievement and maintenance tasks (i.e., $\Psi = x$ and $\Psi = \bar{x}$), Σ_2^p for non-deterministic history-dependent environments (Theorems 6 and 9), and NP-complete for deterministic history-dependent environments (Theorems 7 and 9).

Ψ-OAD
: The oblivious agent design problem for history-independent non-deterministic environments, with tasks specified by a propositional logic formula Ψ, where the task Ψ is implicit (i.e., does not form part of the input). Recall that an agent is oblivious if it is 0-bounded, i.e., if it is permitted no information about the way the system has evolved at all. The problem is decidable for all propositional formulae Ψ, and PSPACE-complete for Ψ of the form $x \wedge \bar{y}$.

FC
: The finite conjunction problem: where we ask whether there is an agent that can bring about some "good" state while avoiding some "bad" state (and hence a special case of Ψ-FD where Ψ is of the form $x \wedge \bar{y}$). For history-independent environments, considering reactive agents, the problem is P-complete for non-deterministic environments, and NL-complete for deterministic environments. (Theorem 12.)

Ψ-SAD
: The stochastic agent design problem, where the task is specified by both propositional logic formula, (which is assumed to be implicit, i.e., not given as part of the instance), and a rational number p in the range $[0, 1]$. We are asked whether there exists an agent that succeeds with the task with probability greater than p. For non-deterministic environments, the problem is PSPACE-complete, for all Ψ. (Theorem 14.) The two special cases of Ψ-SAD corresponding to $\Psi = x$ (achievement) and $\Psi = \bar{x}$ (maintenance) are denoted by ASAD and MSAD respectively.

MAJD_{det}
: The problem of determining whether a majority of agents accomplish an achievement task in a deterministic environment. The problem is PP-complete. (Theorem 15.)

REFERENCES

[1] C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in presence of incompleteness. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden, 1999.

[2] R. A. Brooks. *Cambrian Intelligence*. The MIT Press: Cambridge, MA, 1999.

[3] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[4] P. E. Dunne, M. Wooldridge, and M. Laurence. The computational complexity of boolean and stochastic agent design problems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002)*, pages 976–983, Bologna, Italy, 2002.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of* NP*-Completeness*. W. H. Freeman: New York, 1979.

[6] M. R. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers: San Mateo, CA, 1987.

[7] M. L. Ginsberg. Universal planning: An (almost) universally bad idea. *AI Magazine*, 10(4):40–44, 1989.

[8] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science Volume A: Algorithms and Complexity*, pages 67–161. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.

[9] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[10] M. L. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Journal of AI Research*, 9:1–36, 1998.

[11] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley: Reading, MA, 1994.

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

[13] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 1039–1046, Milan, Italy, 1987.

[14] M. Wooldridge. The computational complexity of agent design problems. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 341–348, Boston, MA, 2000.

[15] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.

[16] M. Wooldridge and P. E. Dunne. Optimistic and disjunctive agent design problems. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII: Proceedings of the Seventh International Workshop on Agent Theories, Architectures, and Languages, ATAL-2000 (LNAI Volume 1986)*, pages 1–14. Springer-Verlag: Berlin, Germany, 2000.

[17] M. Wooldridge and P. E. Dunne. The complexity of agent design problems: determinism and history dependence. Technical Report ULCS-01-010, University of Liverpool, Dept of Computer Science, 2001.

[18] M. Wooldridge and P. E. Dunne. The computational complexity of agent verification. In J.-J. Ch. Meyer and M. Tambe, editors, *Intelligent Agents VIII: Proceedings of the Eigth International Workshop on Agent Theories, Architectures, and Languages, ATAL-2001 (LNAI Volume 2333)*, pages 115–127. Springer-Verlag: Berlin, Germany, 2002.

[19] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[20] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1976.