On the Computational Complexity of Designing Bounded Agents

Michael Laurence Paul E. Dunne Michael Wooldridge

Department of Computer Science, University of Liverpool
Liverpool L69 7ZF, United Kingdom
{mikel, ped, mjw}@csc.liv.ac.uk

October 3, 2003

Abstract

In this paper, we determine the computational complexity of the *agent design problem* for several classes of *bounded* agents. Agent design is the problem of determining, for a given representation of an environment and representation of a task to be carried out in this environment, whether it is possible to construct an agent that can be guaranteed to accomplish the task in the environment. Previous research has determined the complexity of the agent design problem for various classes of environment and task, but where the agent was permitted *perfect recall* of prior events. In this paper, we investigate the complexity of the problem for agents that have various bounds placed on their memory. Specifically, we determine the complexity of the agent design problem for *reactive* agents, (which must make a decision about what to do based solely on the current environment state), *k-reactive* agents (which must make a decision based on the last k environment states), and *oblivious* agents (which have no information about the environment at all).

1 Introduction

In the literature on autonomous agents, there is a well-known distinction between what are often called *deliberative* or *cognitive* agents [3], and *behavioural* or *reactive* agents [1]. 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. However, comparatively little research has addressed the relative merits of the two approaches from a theoretical standpoint. In this paper, we do so, by using the tools of computational complexity theory [5].

In previous research, the complexity of designing agents to accomplish tasks in particular environments was investigated [9, 10, 2]. The agent design problem can be

crudely understood as follow: Given representations of a task to be carried out, and an environment in which this task is to be carried out, determine whether or not there exists an agent to carry out this task in this environment. If the answer is "yes", then the witness to this fact will be an agent that can carry out the task. In [9, 10, 2], this problem was studied for a range of different environments (e.g., deterministic *versus* non-deterministic, history dependent *versus* Markovian, finite *versus* infinite) and for a range of different types of tasks (e.g., achievement tasks — "bring about this state of affairs", maintenance tasks — "maintain this state of affairs", and Boolean combinations of achievement and maintenance tasks). However, a common factor in this prior research was that agents were permitted to have *perfect recall*, in that they were allowed to remember the complete sequence of events via which the current state of the environment was brought about. But of course, the fact that there exists such an agent *in principle* that can accomplish some task in the environment does not imply that there exists an agent that can be implemented *in practice*: an actual implementation of a 'perfect recall' agent might require too much memory, or an intractable computation to be carried out in constant time. (This is of course exactly the motivation behind Russell's notion of *bounded optimality* [7]).

For this reason, our aim in the present paper is to consider complexity issues for agent design problems when it is required that the agent program be ' bounded' or 'concise'. 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.

In the next section we review the formal model of agents, environments, and tasks from [10, 2] together with the distinct environment forms that agents may operate in. In Section 3 the reactive agent design decision problems are formulated, and it is proved that although markedly 'easier' than their non-reactive counterparts they remain intractable, being complete at the second level of the polynomial-time hierarchy. We further introduce a natural development of k-reactive agents – oblivious agents – whose chosen actions are determined solely by the length of time the agent has been operating, i.e. independently of the current environment state. Conclusions and further work are dealt with in the concluding section. Throughout, we assume some familiarity with computational complexity theory [5].

2 Agents, Environments, and Tasks

In this section, we present the abstract formal model of agents and the environments they occupy from [10, 2]. We then use this model to frame the decision problems studied in the present paper.

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 have a repertoire of possible actions available to them, which transform the state of the environment. Letting $Ac = \{\alpha_0, \alpha_1, \ldots, \alpha_k\}$ be the (finite) set of actions, the behaviour of an environment is defined by a *state transformer function*, τ. It is often the case that 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*, Env, is a tuple $\langle E, e_0, Ac, \tau \rangle$, 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 $e_0 \cdot \alpha_0 \cdot e_1 \cdot \alpha_1 \cdot e_2 \cdots$ with for each i, $e_i \in E$, $\alpha_i \in Ac$, and S^E be the subset of such sequences that end with a state. The* state transformer function *τ 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*

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

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}. The* length *of a run, $r \in R_{Env}$ is the total number of actions and states occurring in r and is denoted by $|r|$. Finally, for $r \in R_{Env}$, $last(r)$ denotes the final state of the run r, i.e. $r = s \cdot last(r)$, for $s \in R^{Ac}$.*

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, $p(|r|)$.[1] 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. The agent, however, 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*

[1] We adopt the convention that should T_τ fail to have halted after $p(|r|)$ moves on input $r\#e$ then $e \notin \tau(r)$

$s \in R_{Env} \cup R^{Ac}$ *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. $\forall k > 0$,
$$\begin{array}{rcl} e_k & \in & \tau(e_0 \cdot \alpha_0 \cdot e_1 \cdot \alpha_1 \cdots e_{k-1} \cdot \alpha_{k-1}) \text{ where} \\ \alpha_k & = & Ag(e_0 \cdot \alpha_0 \cdot e_1 \cdot \alpha_1 \cdots e_k) \end{array}$$

It should be noted that, in general, the state transformer function τ (with domain R^{Ac}) is *non-deterministic*, 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 $t : \mathbb{N} \to \mathbb{N}$, 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.

Study of agent design problems in this framework was initiated in [9] and extended in [10, 2]. Important decision problems introduced with these are the *finite achievement design* (FAD) and *finite maintenance design* (FMD) problems.

Definition 3 *An instance of the* Finite Achievement Design *problem (*FAD*) comprises an environment* $\langle E, e_0, Ac, \tau \rangle$ *and a subset G of E. An instance is accepted if there is an agent Ag for the environment such that,*

$$\forall r \in C^{|E \times Ac|}(Ag, Env)\ \exists g \in G : g \text{ occurs in } r$$

An instance of the Finite Maintenance Design *problem (*FMD*) comprises an environment* $\langle E, e_0, Ac, \tau \rangle$ *and a subset B of E. An instance is accepted if there is an agent Ag for the environment such that,*

$$\forall r \in C^{|E \times Ac|}(Ag, Env)\ \forall b \in B : b \text{ does not occur in } r$$

The analyses of [10, 9] consider these in settings where τ is non-deterministic, deterministic, history-dependent, and history-independent, i.e. $\tau(r \cdot \alpha)$ depends only on the state $last(r)$. Denoting these cases by FMD_X^Y, FAD_X^Y with $X \in \{det, non-det\}$, $Y \in \{h.d, h.i\}$, it has been shown that FAD_X^Y has equivalent complexity to FMD_X^Y and this varies from PSPACE-complete ($\text{FAD}_{non-det}^{h.d}$) through NP-complete ($\text{FAD}_{det}^{h.d}$) to P (in the history-independent cases). In [2], similar results were established for the problems $\Psi - \text{FD}$ where Ψ is a given Boolean function of k variables: each environment state is associated with at most one variable, so that any run r induces an instantiation of these via $x_i = \top$ (should any state mapped to x_i occur in r) and $x_i = \bot$ (if no such state occurs in r). Thus FAD and FMD are the special cases corresponding to the unary functions $\Psi(x) = x$ and $\Psi(x) = \neg x$ respectively. In history-dependent environments, [2] shows that whenever $\Psi(x_1, \ldots, x_k)$ is not a *constant* function, $\Psi - \text{FD}_X^{h.d}$ has equivalent complexity to $\text{FAD}_X^{h.d}$.

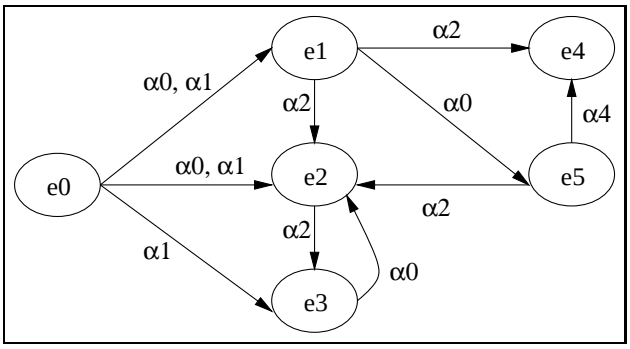

Figure 1: The state transitions of an example environment: Arcs between environment states are labelled with the sets of actions corresponding to transitions. Note that this environment is *history dependent*, because agents are not allowed to perform the same action twice. So, for example, if the agent 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.

3 Reactive Agent Design

While the intractability of $\text{FAD}^{h.d}_{non-det}$ creates one drawback, a further difficulty arises in that the length of program describing a *successful* agent may be exponential in $|E \times Ac|$ unless some concise encoding of the action required on each relevant run can be found. We can try and enforce a 'short program' regime, by limiting agents of interest to those which are only required to specify actions predicated on some constant length fragment of the agent's recent history.

Example 1 *To better understand our notion of reactive agent, 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. 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:*

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

Formally, we have the following.

Definition 4 *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 the state transition function, τ, continues to be *history-dependent*, and thus, the existence of an agent

solving $\text{FAD}^{h.d}_{not-det}$ in a particular instance does not *guarantee* the existence of a *reactive* agent doing so. While limiting attention to reactive agents has the disadvantage that potentially successful 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 $\text{Q}^{(1)}_X$ the decision problem Q_X (for $\text{Q} \in \{\text{FAD}, \text{FMD}, \Psi - \text{FD}\}$) when solution agents are required to be *reactive*.

We recall that the complexity class Σ^p_2 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 1 $\text{FAD}^{(1)}_{non-det}$ *is* Σ^p_2*-complete.*

Proof: To show that $\text{FAD}^{(1)}_{non-det} \in \Sigma^p_2$ 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

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

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 $\text{FAD}^{(1)}_{non-det}$ 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^p_2 program.

To show that the problem is Σ^p_2-hard, we give a reduction from the Σ^p_2-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^p_2-completeness of QSAT_2 was demonstrated by Wrathall[11].

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 $\text{FAD}^{(1)}_{non-det}$ as follows. For $Env_\varphi = \langle E, e_0, Ac, \tau\rangle$ we set,

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

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

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

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

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

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

Suppose $\langle \mathit{Env}_\varphi, \{\top\}\rangle$ is a positive instance of $\text{FAD}^{(1)}_{non-det}$, i.e. there is a reactive agent, *Ag*, whose every critical run reaches the state $\top$. Consider the mapping $\mu_{Ag} : E \rightarrow Ac$ 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 $1 \leq i < n$, and $\mu_{Ag}(y_n^\top) = \mu_{Ag}(y_n^\bot) = \mathit{eval}$, 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

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

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

Theorem 1 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 2 $\text{FAD}^{(1)}_{det}$ *is* NP*-complete.*

Proof: Membership in NP is obvious: given an instance $\langle \mathit{Env}, G\rangle$ of $\text{FAD}^{(1)}_{det}$ 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 $\langle \mathit{Env}_\psi, G\rangle$ of $\text{FAD}^{(1)}_{det}$ 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 $\mathit{last}(r) = x_i$ and $i < n$; if $\mathit{last}(r) = x_n$ 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 $\text{FAD}^{(1)}_{det}$ 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 1, 2.

Corollary 1 *For* $\text{Q} \in \{\text{FMD}, \Psi - \text{FD}\}$,

a) $\text{Q}^{(1)}_{non-det}$ *is* Σ^p_2*-complete.*

b) $\text{Q}^{(1)}_{det}$ *is* NP*-complete.*

Proof: For FMD simply choose the set B as $\{\bot\}$ in the reductions from QSAT_2 and SAT. One minor modification is required to the definition of τ: since a reactive agent could simply associate a disallowed action with each state, e.g. $\rightarrow$ as the action in state x_1, instead of $\tau(r \cdot \alpha) = \emptyset$, we set $\tau(r \cdot \alpha) = \bot$ for such cases. The proof for Ψ-FD derives from the constructions in [2]: we omit the details. □

4 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 at $O(|E \times Ac|^k \log |Ac|)$ 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 5 *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 $\text{Q}^{(k)}_X$ the agent design problem Q_X in which solution agents are required to be k-reactive.

Theorem 3 *For each* $k \geq 1$*, and* $X \in \{non - det, det\}$, $\text{FAD}^{(1)}_X \equiv_{\log} \text{FAD}^{(k)}_X$.

Proof: We first show that $\text{FAD}^{(1)}_X \leq_{\log} \text{FAD}^{(k)}_X$. 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 $\text{FAD}_X^{(k)} \leq_{\log} \text{FAD}_X^{(1)}$, 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 $\langle Env', G' \rangle$. □

Corollary 2 *For* $\text{Q} \in \{\text{FAD}, \text{FMD}, \Psi - \text{FD}\}$,

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

b) $\text{Q}_{det}^{(k)}$ *is* NP*-complete.*

Proof: Immediate from Theorems 1–3 above. □

5 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 2 *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_1.

More formally, we have the following.

Definition 6 *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 FAD_X on which we have focused we can introduce an oblivious variant as follows.

Definition 7 *An instance of the* oblivious achievement design *problem (*$\text{FAD}_X^{(0)}$*) consists of an environments* $Env = \langle E, e_0, Ac, \tau \rangle$*, 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*

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

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 4

a) $\text{FAD}^{(0)}_{non-det}$ *is* Σ^p_2*-complete.*

b) $\text{FAD}^{(0)}_{det}$ *is* NP*-complete.*

Proof: For (a) the reduction of Theorem 1 is used to construct a instance of $\text{FAD}^{(0)}_{non-det}$ 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 an agent that 'eventually' realises an achievement tasks 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 reaches different states e_i and e_j after k actions are performed, but these are such that

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

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 8 *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 ($\Psi - \text{OAD}$) 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*

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

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

Theorem 6 $(x_1 \wedge (\neg x_2)) - \text{OAD}$ *is* PSPACE*-hard.*

Some remarks regarding the relationship between Theorem 6 and Theorem 4 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 $\text{FAD}^{(0)}_{non-det}$ 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 5.

6 Conclusions and Related Work

In this paper, we have determined the computational complexity of designing agents with various types of imperfect recall. In general, we show that the problem is apparently easier than the corresponding setting in which agents have perfect recall (Σ_2^p-complete, as opposed to PSPACE-complete), although in simpler types of the problem setting (for example where the environment is deterministic) the complexity of the problems in the perfect and imperfect recall case coincide.

To the best of our knowledge, this work represents the first attempt to apply the tools of complexity theory to the comparative study of reactive and other types of agents, although there is some related work in the literature. [4] argues against reactive agents in general, (and the 'Universal Plans' approach of [8] in particular), by arguing that the approach will not scale due to the size of the representation required; the 'universal plans' agents referred to here correspond almost precisely with our notion

of a reactive agent. [7] argue against the prevailing AI notion of 'optimal' agent as the one that chooses the best action in principle, and argues forcefully that we must focus on finding agents that can make the best choice given the computational (memory and processor) constraints under which they must operate. [6] studies various bounds that may be placed on the memory size of agents from a game theoretic point of view; the intuition is exactly the same as ours, but no complexity-theoretic results are obtained.

References

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

[2] 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.

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

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

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

[6] A. Rubinstein. *Modeling Bounded Rationality*. The MIT Press: Cambridge, MA, 1998.

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

[8] 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.

[9] 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.

[10] 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.

[11] C. Wrathall. Complete sets for the polynomial hierarchy. *Theoretical Computer Science*, 3:23–24, 1976.