A First Order Logic Classifier System

Drew Mellor
School of Electrical Engineering and Computer Science
The University of Newcastle, Callaghan, 2308, Australia
Telephone: (+612) 4921 6034, Facsimile: (+612) 4921 6929
dmellor@cs.newcastle.edu.au

ABSTRACT

Motivated by the intention to increase the expressive power of learning classifier systems, we developed a new XCS derivative, FOX-CS, where the classifier and observation languages are a subset of first order logic. We found that FOX-CS was viable at tasks in two relational task domains, poker and blocks world, which cannot be represented easily using traditional bit-string classifiers and inputs. We also found that for these tasks, the level of generality obtained by FOX-CS in the portion of population that produces optimal behaviour is consistent with Wilson's generality hypothesis.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*Predicate logic; Representations (procedural and rule-based)*; I.2 [**Artificial Intelligence**]: Learning—*Concept learning*

General Terms

Algorithms, Languages

Keywords

Relational learning, first order logic, learning classifier system, XCS, blocks world, poker

1. INTRODUCTION

Languages based on first order logic are highly expressive and capable of representing complex relationships between the attributes of a task domain. Thus, when addressing tasks set in relational domains (domains where the underlying regularity is relational in nature [21]), a language over first order logic can be appropriate for expressing data and acquired knowledge. The field of inductive logic programming (ILP) consists of methods for learning with first order logic, which are frequently adapted from propositional algorithms, like the well known FOIL algorithm [15] and others [3, 11]. A suitable candidate for an upgrade to first order logic is a learning classifier system (LCS) due to the parallel between a Horn clause and a classifier. Such an upgrade would extend the application of learning classifier systems to relational domains, and at the same time would extend ILP to reinforcement learning tasks, which has only been addressed recently with the advent of relational reinforcement learning [8, 20]. However, incorporating first order logic into the LCS framework has remained unrealised until this paper, which contributes FOX-CS, a learning classifier system that uses first order logic to represent classifiers and observations.

A learning classifier system is an adaptive rule-based system characterised by the combination of a genetic algorithm, which searches the space of rules, and a reinforcement learning-like algorithm, which estimates the rules' expected payoffs. An individual rule in the system is called a classifier and contains an action and a condition (in this paper we will use the notation *action* ← *condition* to represent a classifier), which the system interprets procedurally: "if the input matches *condition* then do *action*". In addition to the action and condition a classifier also contains a prediction value that estimates the expected long term payoff accrued by the system after following the rule. Ideally, the condition is a generalisation such that the classifier as a whole expresses a regularity in the payoff landscape (i.e. a collection of input-action pairs sharing equal expected payoff).

The standard classifier language is the bit-string, which supports fast matching operations (linear in the number of bits) and is adequately expressive for most attribute-value concepts [5]. A bit-string classifier expresses a condition as a string over the ternary alphabet {`0`,`1`,`#`}. To match a condition against a binary input string, the condition must match the input string precisely, symbol for symbol, except that the "don't care" symbol, `#`, can match either a `0` or a `1`. The `#` symbol supports generalisation by allowing a single condition to match a range of inputs, for example the condition string `10##` generalises over four input values. However, the `#` symbol cannot express *relationships* between the values at different bits, for example it cannot express "the first and fourth bits are equal and the second and third bits are equal" (i.e. `ABBA` where `A` and `B` are variables over bit values) [17]. It is this lack of expressive power that limits the usefulness of bit-strings in relational domains.

Although bit-string classifiers are typical, the LCS framework is general enough to support other classifier languages, such as intervals of reals [24], genetic programming inspired S-expressions [12], and weights for neural networks [6]. Classifier languages capable of representing relational concepts

GECCO'05, June 25–29, 2005, Washington, DC, USA.

have been used in the following systems: XCSL [12] and GP-CS [1], which partially express classifiers (the conditions and actions respectively) with S-expressions; and the proposed VCS system [18], which extends the bit-string alphabet to support variables. However, neither XCSL or GP-CS can express abstract relationships between the attributes of actions and the attributes of conditions (e.g. `AB` ← `ABBA`), and as far as we are aware no results for VCS have ever been published.

In this paper we present a system that uses first order logic to represent the classifiers and observations. In addition to facilitating the representation of relational concepts, including concepts that relate attributes of actions to those of conditions (like `AB` ← `ABBA`), other advantages which arise from the use of first order logic include support for the inclusion of background knowledge, and enhanced knowledge transparency due to the structure present in Horn clauses. It is also hoped that the new system will inherit some of the advantages of its parent, XCS, including its tendency to discover and proliferate maximally general classifiers in the population [23].

The remainder of the paper is set out as follows. First, we briefly present some definitions from first order logic that will be relevant for describing the new classifier and observation languages (section 2). Then we outline the XCS system (section 3), which the new system FOX-CS is based on. Next, FOX-CS is described in detail (section 4), followed by an empirical assessment of FOX-CS at single step (section 5) and multiple step (section 6) tasks. In the final section some concluding remarks are made (section 7).

2. DEFINITIONS IN FIRST ORDER LOGIC

An *atomic formula* $r(u_1, \ldots, u_n)$ is a predicate symbol r followed by a bracketed n-tuple of terms. A *predicate symbol* is a lower case letter followed by a string of zero or more lower case letters and digits. A *term* is either a constant, a variable or a function symbol followed by a bracketed n-tuple of terms. Both *constants* and a *function symbols* consist of a lower case letter followed by a string of zero or more lower case letters and digits (in this paper we will not be concerned with functions), whereas a *variable* is an upper case letter followed by a string of zero or more lower case letters and digits. The *arity* of an atomic formula is the number of terms in (the top level of) the formula. If A is an atomic formula then both A and $\neg A$ are *literals*, where A is called a positive literal and $\neg A$ is called a negative literal.

A *clause* is a finite set of literals that represents the disjunction of the literals. Any clause can be written as a formula of the form $A_1; \ldots; A_m \leftarrow B_1, \ldots, B_n$, where the A_i are all the positive literals in the clause, the B_i are all the negative literals in the clause, and the ";" symbol indicates disjunction, the "," symbol indicates conjunction, and the "←" symbol represents implication. A *Horn clause* is a clause that contains exactly one positive literal, and can be written as $A \leftarrow B_1, \ldots, B_n$. The positive literal, A, is called the head of the clause and the negative literals, $B_1, \ldots, B_n$, are called the body of the clause. A Horn clause with an empty body is called a *fact*. For more background on first order logic in the ILP setting see [14].

3. THE XCS SYSTEM

The XCS system [22, 23] is perhaps the most well known, thoroughly documented, and best performing LCS implementation. The principle characteristic distinguishing XCS from other LCS implementations is its accuracy based definition of fitness. Associated with each classifier cl are three parameters: the *prediction*, p_{cl}, which estimates the expected payoff for cl; the prediction *error*, ε_{cl}, which estimates the difference between p_{cl} and the actual payoff received; and the *fitness*, F_{cl}, which is based on the relative accuracy of p_{cl}. The *accuracy* of cl is 1 (perfect) if ε_{cl} lies within some tolerance of zero, otherwise it decreases towards 0 as ε_{cl} increases. In contrast to XCS, strength based classifier systems like Holland's LCS [10] use a single parameter, the strength, which serves for both the prediction and the fitness.

The operational cycle of XCS proceeds as follows. At each discrete time step, t, the match set $[M]$ is formed from the system's current population of classifiers, $[P]$, that match the current input. If the number of different actions in $[M]$ is less than some constant, a new classifier is created by the process of *covering*. Covering creates a classifier with a randomly selected action and a condition matching the current input, i.e. each condition bit will be equal to its equivalent input bit or `#`. For each action a specified by the classifiers of $[M]$, the action prediction $p(a)$ is calculated as the fitness weighted average prediction over all classifiers in $[M]$ advocating a. Based on the action predictions, an action a_t is selected, usually stochastically by either ϵ-greedy selection or roulette wheel selection weighted by the action predictions. Next, the action set $[A]$ is formed, containing the classifiers of $[M]$ that propose a_t. Then a_t is executed resulting in some immediate reward and a new input. Based on the immediate reward, the classifiers belonging to $[A]_{-1}$, the action set at time $t-1$, have their p, ε, and F parameters updated using a method similar to Q-Learning. Finally, from time to time a genetic algorithm is invoked on the members of $[A]_{-1}$ to discover new classifiers. If the size of $[P]$ exceeds a predetermined upper bound N after the insertion of a new classifier, then a classifier is deleted from $[P]$.

Two standard enhancements to XCS that reduce the matching overhead are macroclassifiers and subsumption. A macroclassifier cl has an additional *numerosity* parameter, n_{cl}, that is interpreted as the number of (micro)classifiers in the population that contain cl's action and condition. Each macroclassifier cl reduces the number of matching operations from n_{cl} to 1 because it replaces n_{cl} microclassifiers. In the remainder of this paper we will generally use the term "classifier" and reserve the macro- and micro- prefixes for when the distinction is important, e.g. the parameter N is given as a quantity of microclassifiers.

The subsumption technique also reduces the number of distinct classifiers in the population. If a classifier already has perfect accuracy then a specialisation of it adds no extra information to the system. Hence, if classifier i is a specialisation of another classifier j, and j has perfect accuracy, then j can subsume i by deleting i and incrementing n_j.

4. A FIRST ORDER LOGIC XCS SYSTEM

FOX-CS is a learning classifier system that uses first order logic classifier and observation languages. The system is based on the XCS specification given in [7], with modifications to handle matching, discovery and subsumption of first order logic classifiers. The modifications are detailed below after first discussing the classifier and observation languages.

The classifiers of FOX-CS are Horn clauses containing three parts — the usual action and condition parts plus a background part that can link with a domain specific background

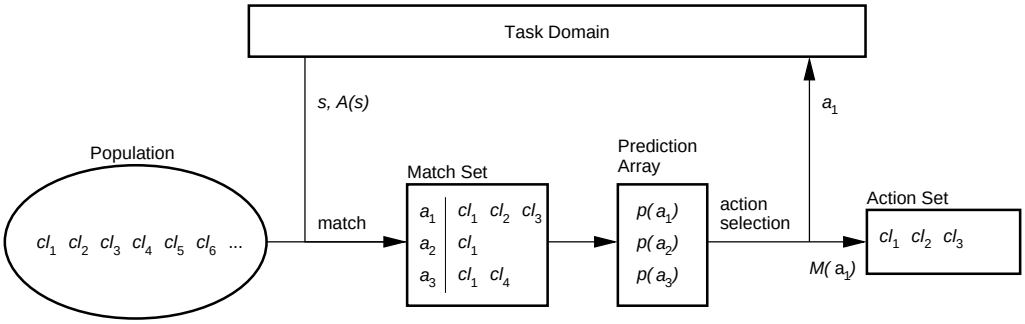

Figure 1: The performance subsystem of Fox-cs. The match set has been modified to associate with each action $a \in A(s)$ a set of classifiers $M(a)$ that match s and a. Note that a classifier can be associated with more than one action.

theory. Thus, the structure of the classifiers is:

$$Action \leftarrow Condition,\ Background$$

Each part consists of a list of atoms, which are predicate formulas that have the form $r(u_1, u_2, \ldots, u_n)$, whose arguments, $u_1, u_2, \ldots, u_n$, are variables or constants, and whose arity, n, satisfies $n \geq 0$. The action part always contains exactly one atom, and the background knowledge part can be empty. A domain specific grammar determines the vocabulary of predicate, variable and constant symbols, and how the atoms may combine to form a classifier. The classifiers have both a procedural interpretation as an action rule (at action execution time) and a declarative interpretation as a Horn clause (at matching time).

The observations input to Fox-cs are facts consisting of the same atoms found in the classifier's condition part, but without any variables. It is the use of variables in the classifier language that allows relationships between the attributes of the task domain to be expressed in an abstract way. Thus, variables perform a generalisation role, similar to the # symbol in bit-string classifiers. However, unlike the # symbol, variables may occur in actions. This means that in Fox-cs actions may be abstract and not directly executable. However, abstract actions can be made concrete because the variables that may occur in actions are constrained to occur in the condition also, hence the value of a variable in the action can be determined when the condition part is matched. The significance of permitting variables in actions is that it allows a correspondence between the attributes of an action and input to be expressed abstractly.

Given a classifier cl, a current input s and an action a (the origin of a is explained below), the matching operation is defined as

$$match(cl, s, a) = \begin{cases} true & \text{if } cl, s, B \models a \\ false & \text{otherwise.} \end{cases}$$

Thus, matching succeeds if and only if cl, s, and the background theory B, together entail a. To implement the matching operation, a search is run for a substitution of the variables of cl that satisfy the entailment constraint. Matching short-circuits if a successful substitution is found; but in the worst case (when matching fails) all substitutions are tried (at least implicitly), thus the time complexity of the matching operation is greater for first order logic classifiers than for bit-string classifiers. If there are n variables that each accept m possible values then the total number of substitutions is n^m, however the entire n^m substitutions would be rarely checked explicitly even when matching fails. This is because the substitutions can be organised into a tree structure such that each substitution corresponds to a path from the root to a leaf, thus if a partial substitution fails at some node in the tree then all full substitutions containing that node can be eliminated at once. Hence, although the worst case time complexity is $O(n^m)$ it must be emphasized that this is not a tight bound, and may be substantially less in practice.

With the matching operation modified, the performance subsystem now works as follows. First $A(s)$, the set of actions allowed in the current input s, is returned by querying the task domain. Then the match set is computed so that it associates with each $a \in A(s)$ the set of all classifiers matching s and a, which we denote by $M(a) = \{cl \in [P] \mid match(cl, s, a)\}$, where $[P]$ is the current population. Note that because an individual classifier can generalise over actions it may occur in the match set of more than one action (under different substitutions). Next, the prediction array is calculated, where for each action $a \in A(s)$ the prediction $p(a)$ is the fitness weighted mean prediction over all classifiers in $M(a)$, that is, $p(a) = \sum_{cl \in M(a)} F_{cl} p_{cl} / \sum_{cl \in M(a)} F_{cl}$. Finally, action selection proceeds as ϵ-greedy selection over the prediction array with uniform random selection of actions that are tied for the maximum prediction. The overall process is illustrated in figure 1.

As noted above, a single classifier cl can contribute to the prediction calculation of multiple actions in $A(s)$. However cl cannot achieve perfect accuracy unless each of these actions yields the same payoff. Thus, generalisations made over actions must still reflect uniformities in the payoff landscape or suffer low fitness. Hence Fox-cs should preserve the tendency of Xcs to form accurate generalisations.

Rule discovery in Fox-cs consists of mutation and covering. From time to time, a rule discovery module selects a classifier from the action set for reproduction and increments its numerosity with probability $1 - \mu$, or produces a mutated child with probability μ. The effect of mutation is to explore the space of classifiers allowed by the given task grammar. Covering operators are used when the match set is empty. Both mutation and covering employ domain specific operations, which are detailed in the task description sections below for the experiments reported in this paper. In its use of a "direct" representation (first order logic) and domain specific mutation operators, the evolutionary com-

A)

$$\begin{aligned} \mathit{Observation} &::= \mathbf{card}(\mathbf{1}, \mathit{Rank}, \mathit{Suit}), \mathbf{card}(\mathbf{2}, \mathit{Rank}, \mathit{Suit}), \mathbf{card}(\mathbf{3}, \mathit{Rank}, \mathit{Suit}), \mathbf{card}(\mathbf{4}, \mathit{Rank}, \mathit{Suit}), \\ &\quad \mathbf{card}(\mathbf{5}, \mathit{Rank}, \mathit{Suit}) \\ \mathit{Rank} &::= \mathbf{two} \mid \mathbf{three} \mid \mathbf{four} \mid \mathbf{five} \mid \mathbf{six} \mid \mathbf{seven} \mid \mathbf{eight} \mid \mathbf{nine} \mid \mathbf{ten} \mid \mathbf{jack} \mid \mathbf{queen} \mid \mathbf{king} \mid \mathbf{ace} \\ \mathit{Suit} &::= \mathbf{clubs} \mid \mathbf{spades} \mid \mathbf{diamonds} \mid \mathbf{hearts} \end{aligned}$$

$$\mathit{card}(1, \mathit{eight}, \mathit{spades}),\ \mathit{card}(2, \mathit{two}, \mathit{diamonds}),\ \mathit{card}(3, \mathit{eight}, \mathit{diamonds}),\ \mathit{card}(4, \mathit{eight}, \mathit{clubs}),\ \mathit{card}(5, \mathit{eight}, \mathit{hearts})$$

B)

$$\begin{aligned} \mathit{Classifier} &::= \mathit{Class} \leftarrow \mathit{Condition} \\ \mathit{Classifier} &::= \mathit{Class} \leftarrow \mathit{Condition}, \mathit{Background} \\ \mathit{Class} &::= \mathbf{nought} \mid \mathbf{pair} \mid \mathbf{twopair} \mid \mathbf{threeofakind} \mid \mathbf{straight} \mid \mathbf{flush} \mid \mathbf{fullhouse} \mid \mathbf{fourofakind} \\ \mathit{Condition} &::= \mathbf{card}(\mathbf{P1}, \mathit{Rank\_V}, \mathit{Suit\_V}), \mathbf{card}(\mathbf{P2}, \mathit{Rank\_V}, \mathit{Suit\_V}), \mathbf{card}(\mathbf{P3}, \mathit{Rank\_V}, \mathit{Suit\_V}), \\ &\quad \mathbf{card}(\mathbf{P4}, \mathit{Rank\_V}, \mathit{Suit\_V}), \mathbf{card}(\mathbf{P5}, \mathit{Rank\_V}, \mathit{Suit\_V}) \\ \mathit{Rank\_V} &::= \mathbf{R1} \mid \mathbf{R2} \mid \mathbf{R3} \mid \mathbf{R4} \mid \mathbf{R5} \mid \_ \\ \mathit{Suit\_V} &::= \mathbf{S1} \mid \mathbf{S2} \mid \mathbf{S3} \mid \mathbf{S4} \mid \mathbf{S5} \mid \_ \\ \mathit{Background} &::= \mathit{Succ} \mid \mathbf{not}(\mathit{Succ}) \\ \mathit{Succ} &::= \mathbf{succ}(\mathbf{R1}, \mathbf{R2}, \mathbf{R3}, \mathbf{R4}, \mathbf{R5}) \end{aligned}$$

$$\mathit{fullhouse} \leftarrow \mathit{card}(P1, R1, \_),\ \mathit{card}(P2, R1, \_),\ \mathit{card}(P3, R2, \_),\ \mathit{card}(P4, R2, S1),\ \mathit{card}(P5, R1, \_)$$

Table 1: The grammars, in BNF notation, defining the languages for expressing A) observations and B) classifiers for the poker classification task. An example is given underneath each grammar.

ponent of Fox-cs resembles evolutionary programming [9].

Subsumption reduces the number of matching operations required at each cycle. Because matching is a more costly operation using first order logic, subsumption can play an important role in Fox-cs. However, unlike bit-strings, it is difficult to detect if one expression in first order logic is a specialisation of another in a fast and general way. Hence, subsumption, like mutation and covering, is performed in a domain specific fashion.

In the following two sections, experiments with Fox-cs at a single step task (classifying hands of poker) and two multiple step tasks (stacking and unstacking blocks) are presented. The aims of the experiments are to access empirically the system's *i*) learning performance, and *ii*) generalisation ability. According to Wilson's generality hypothesis the system should tend to evolve a population of classifiers that are both accurate and maximally general and which cover the entire state-action space. Since it is complicated to determine all the accurate and maximally general conditions over the entire state-action space, we shall be content to consider just the optimal portion instead, i.e. the state-action pairs that lead to optimal performance. For the evaluation tasks in this paper, it can be verified whether an arbitrary classifier specifies optimal behaviour and is maximally general.

5. CLASSIFYING POKER HANDS

This multi-class classification task consists of recognising which of the following eight classes a hand of poker belongs to: *fourofakind*, *fullhouse*, *flush*, *straight*, *threeofakind*, *twopair*, *pair* or *nought*. The first seven classes are defined normally, with the exception that *flush* includes royal and straight flushes in addition to ordinary flushes, while the last class, *nought*, collects all the hands that don't belong to one of the other seven classes. This task, which was first subject to an ILP algorithm in [4], is inherently relational because it is the relationships between the ranks and suits of the cards in the hand, rather than their values, that define the classes. Xcs and other machine learning systems that use attribute value languages are unable to solve the task without using high level features to represent inputs.

5.1 Setup and Training

The observation and classifier languages for this task are generated by the grammars in table 1. The variables are interpreted in the following way: all occurrences of the same variable in a classifier refer to the same value, but separate variables in a classifier may never refer to the same value — the exception is the anonymous variable, denoted by "_", which may contain any value, similar to the don't care symbol # in bit-string classifiers. These rules for interpreting variables prevent separate atoms in the condition part from successfully matching against the same card in the input data, for example $\mathit{card}(P1, R1, \_)$ and $\mathit{card}(P2, R1, \_)$ must match separate cards in the input because $P1 \neq P2$. The formula $\mathit{succ}(R1, R2, R3, R4, R5)$ evaluates to true if the values of its arguments can be ordered to form a sequence of contiguous ranks. A classifier cannot contain the atom $\mathit{succ}(R1, R2, R3, R4, R5)$ unless each of the rank variables also occurs in the condition part.

When covering, a separate classifier is created for each of the eight classes of hand. The action is set to the class name, the condition is set to the current observation, and the background knowledge part is initially empty. Then the classifier is generalised by mapping each distinct rank and suit constant to a rank or suit variable.

There are two mutation operations, which are invoked with equal probability. The first selects an instance of a rank or suit variable from the condition part at random and renames it with a randomly selected variable name from $\mathit{Rank\_V}$ or $\mathit{Suit\_V}$, whichever is appropriate. The second appends or removes the $\mathit{succ}(R1, R2, R3, R4, R5)$ atom depending whether it is currently absent or present; when appending, the atom is negated half of the time. However if a mutation produces a classifier that violates the grammar — for example the atom $\mathit{succ}(R1, R2, R3, R4, R5)$ is present but one or more of the rank variables are missing — then the classifier is discarded and the mutation process is rerun.

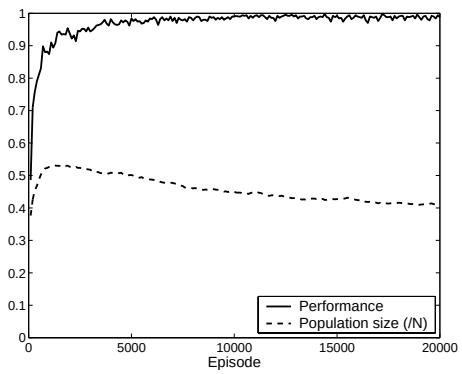

Figure 2: The system performance (solid line) and population size (dashed line) for the poker classification task. The population size is the number of macroclassifiers divided by N (500). Results are averaged over 10 separate runs.

A classifier cl_1 can be subsumed by cl_2 if and only if their expressions are identical or are identical except for either or both of the following specialisations: *i*) a named variable in cl_1 is replaced with "_" in cl_2; and *ii*) cl_1 contains the background atom and cl_2 does not. Other cases of specialisation can occur but are not detected.

The system was trained for 20,000 episodes. During each episode the system is presented with a randomly generated hand of poker and awarded 10 if it correctly classifies the hand, or -10 otherwise. The distribution of hands was uniform over the different classes. At the beginning of training the population was empty and initially created using covering. The initial classifier parameters were assigned to a random Gaussian with mean 0 and standard deviation 0.01 for prediction, p, and mean 0.01 and standard deviation 0.001 for error, ε, and fitness, F. Values for the system parameters were set as follows: $N = 500$, $\epsilon = 10\%$, $\alpha = 0.1$, $\beta = 0.1$, $\epsilon_0 = 0.0001$, $\nu = 5$, $\theta_{ga} = 50$, $\mu = 0.2$, $\theta_{sub} = 500$, $\theta_{del} = 20$, $\delta = 0.1$. The values for the parameters are typical in the XCS literature except for perhaps ϵ_0 and θ_{sub}, which are discussed later; the new mutation parameter μ was tried with a variety of values and is also discussed later.

5.2 Results

The system performance and the population size during training is shown in figure 2. The system performance was measured as the proportion of correct classifications over the last 50 non-exploratory episodes. The performance reaches about 98% by 10,000 episodes but never fully settles. Inspection of the training runs indicates that after this point the errors are primarily due to the presence of overgeneral classifiers in the population, which are continually evolving from maximally general classifiers. The overgeneral inherits its parent's optimal prediction value, which occasionally leads to the incorrect selection of the overgeneral's class for the given input, thus impairing the system's classification performance. After incorrect selection the overgeneral's prediction and fitness values are adjusted downward. However, sometimes the frequency of misclassifications is small and the overgeneral still has a high accuracy and therefore a high fitness. If so, the overgeneral may cause the average prediction for inputs it does correctly classify to be lowered enough to result in further classification errors. Eventually the situation is rectified when the overgeneral is deleted. Errors due to overgenerals are perhaps inevitable as long as the evolutionary component is operating, but they can be acceptable if the system's minimum performance lies within a specified tolerance of optimal. Here, a tolerance of 4% is enough to account for all errors after 10,000 episodes.

*	14	$straight \leftarrow card(F,D,M), card(G,L,\_), card(H,E,M), card(I,K,\_), card(J,C,A), succ\_rank(C,D,E,K,L)$
*	14	$threeofakind \leftarrow card(G,D,\_), card(H,D,\_), card(I,F,B), card(J,D,C), card(K,A,\_)$
*	13	$nought \leftarrow card(G,C,\_), card(H,F,B), card(I,L,\_), card(J,E,\_), card(K,M,N), not(succ\_rank(C,E,F,L,M))$
*	12	$fourofakind \leftarrow card(E,C,\_), card(F,\_,B), card(G,C,\_), card(H,C,\_), card(I,C,\_)$
*	10	$flush \leftarrow card(D,B,K), card(E,J,K), card(F,I,K), card(G,A,K), card(H,C,K)$
*	10	$pair \leftarrow card(F,E,D), card(G,C,\_), card(H,L,\_), card(I,M,\_), card(J,E,\_)$
*	8	$twopair \leftarrow card(C,H,\_), card(D,J,\_), card(E,H,\_), card(F,K,\_), card(G,J,A)$
*	6	$pair \leftarrow card(F,M,D), card(G,C,\_), card(H,L,\_), card(I,M,\_), card(J,E,D)$
*	5	$fourofakind \leftarrow card(E,C,\_), card(F,\_,\_), card(G,C,\_), card(H,C,\_), card(I,C,\_)$
*	4	$fullhouse \leftarrow card(D,J,B), card(E,J,A), card(F,C,I), card(G,C,B), card(H,J,\_)$
*	4	$fullhouse \leftarrow card(D,J,I), card(E,J,A), card(F,C,I), card(G,C,B), card(H,J,\_)$
*	4	$twopair \leftarrow card(C,H,B), card(D,J,\_), card(E,H,\_), card(F,K,\_), card(G,J,A)$
	3	$twopair \leftarrow card(C,H,B), card(D,J,\_), card(E,H,\_), card(F,K,\_), card(G,J,B)$
*	2	$fullhouse \leftarrow card(D,J,\_), card(E,J,A), card(F,C,I), card(G,C,B), card(H,J,\_)$
	1	$flush \leftarrow card(D,B,K), card(E,J,K), card(F,C,K), card(G,A,K), card(H,C,K)$
	1	$flush \leftarrow card(D,B,K), card(E,J,K), card(F,I,\_), card(G,A,K), card(H,C,K)$
	1	$fullhouse \leftarrow card(D,J,B), card(E,J,A), card(F,C,I), card(G,C,I), card(H,J,\_)$
	1	$fullhouse \leftarrow card(D,J,I), card(E,J,B), card(F,C,I), card(G,C,B), card(H,J,A)$
	1	$fullhouse \leftarrow card(D,J,I), card(E,J,B), card(F,C,I), card(G,C,B), card(H,J,\_)$
	1	$fullhouse \leftarrow card(D,J,K), card(E,J,B), card(F,C,I), card(G,C,B), card(H,J,A)$

Table 2: The sample of the population with perfect accuracy and prediction 10 after 25,000 episodes of the poker classification task, ordered by numerosity. In the first column, an asterisk "*" indicates that the classifier is maximally general. The second column gives the numerosity values. Note that in contrast to the grammar given in figure 1, single letter variables have been used here in order to be concise.

The population size rises quickly to a little over $0.5 \times 500 = 250$ macroclassifiers and then gradually decreases to about 200. The decline in the diversity of the macroclassifiers is expected when generalisation is occurring, because an accurate and specific classifier tends to be replaced by an accurate and general classifier.

To assess the level of generality attained, after a run was completed the final population was inspected to determine whether maximally general classifiers had evolved. For this task, a classifier cl is maximally general if every hand that matches cl's condition part belongs to cl's class, and if every hand belonging to cl's class does match cl's condition. Maximally general classifiers were found to evolve. After experimentation it was also found that the minimum number

of episodes required to evolve maximally general classifiers in greater proportion than sub-optimally general classifiers for all classes was approximately 25,000 when using the parameter settings given above. A sample of the final population for one arbitrarily selected run is given in table 2. Note that for some classes several different maximally general classifiers evolved, which is possible because the classifier language does allow for more than one maximally general expression for each of the classes. Most of the other classifiers shown are sub-optimally general, but some are overgeneral or contain conditions that cannot be met. The presence of these last two groups of classifiers can be explained as having evolved recently and awaiting either correction of their parameters or deletion. We conclude that for this task, the generalisation behaviour of Fox-cs is consistent with Wilson's generality hypothesis, at least for the portion of the population that makes correct classifications.

Because the use of variables in the classifier language increases the time cost of the matching operation, some attempt was made to minimise the population size limit N. A value of 500 was settled on because it was found that values of $N < 500$ occasionally resulted in deletion of maximally general classifiers, but this was never observed for $N \geq 500$. The mutation rate μ was tried for values in increments of 0.1 over the range $[0.1, 0.9]$. Mutation rates in the range $0.1 \leq \mu \leq 0.5$ all produced performances similar to those reported above, but as values increased above 0.5, instability in the system performance increased. The experience threshold for subsumption was set very high, at $\theta_{sub} = 500$, because it was found that with lower values, maximally general classifiers for *flush* were occasionally subsumed by overgeneral offspring like $flush \leftarrow card(A, F, \_)$, $card(B, G, K)$, $card(C, H, K)$, $card(D, I, K)$, $card(E, J, K)$, which is overgeneral because it will misclassify a *nought* containing four equal suits. Inspection of training runs revealed that the frequency of *nought* hands containing four equal suits was sometimes less than about $\frac{3}{500}$, which is low enough to delay the correction of the prediction and accuracy parameters of an overgeneral classifier like the one above until after it has potentially subsumed its maximally general parent when $\theta_{sub} < 500$. On one hand, this highlights the need for the distribution of data to be carefully selected so that the system can accurately differentiate between classifiers that predict over separate environmental niches, but it also shows that the Xcs methodology is robust enough to support learning under uneven training data distributions by setting the parameters appropriately.

6. BLOCKS WORLD TASKS

The second evaluation domain is the well known blocks world [19], which has been used extensively within AI, particularly for testing planning algorithms. The domain consists of a floor and a finite number of blocks that rest either on the floor or on each other. If a block has nothing on top of it, it is "clear" and can be moved to the floor or onto another clear block. A variety of multiple step tasks are possible within blocks world; in this paper Fox-cs is evaluated at two: *stacking* and *unstacking* [8]. The object of the stack task is to arrange all the blocks into a single stack where the order of the blocks is unimportant; the optimal stack policy simply places any clear block on top of the highest block. The object of the unstack task is to arrange the blocks so they are all on the floor, again order is not important; an optimal policy to unstack selects any clear block not already on the floor and moves it there. In blocks world the number of states increases exponentially with the number of blocks, thus as the number of blocks increases the tasks quickly become unsolvable without generalisation.

A) $Observation ::= Observation\_A \ \{, Observation\_A\}$
$Observation\_A ::= \mathbf{cl}(Block\_C) \mid \mathbf{on\_fl}(Block\_C) \mid \mathbf{on}(Block\_C, Block\_C)$
$Block\_C ::= \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \mid \ldots$

$cl(b), cl(c), on\_fl(a), on\_fl(c), on(b, d), on(d, a)$

B) $Classifier ::= Action \leftarrow Condition$
$Condition ::= Condition\_A \ \{, Condition\_A\}$
$Action ::= \mathbf{mv}(Block\_T, Block\_T) \mid \mathbf{mv\_fl}(Block\_T)$
$Condition\_A ::= \mathbf{cl}(Block\_T) \mid \mathbf{on\_fl}(Block\_T) \mid \mathbf{on}(Block\_T, Block\_T)$
$Block\_T ::= Block\_C \mid Block\_V$
$Block\_C ::= \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \mid \ldots$
$Block\_V ::= \mathbf{A} \mid \mathbf{B} \mid \mathbf{C} \mid \ldots$

$mv(A, B) \leftarrow cl(A), cl(B), on\_fl(C), on\_fl(B), on(A, C)$

Table 3: The grammars, in BNF notation, defining the languages for expressing A) observations and B) classifiers for the blocks world tasks. An example is given underneath each grammar.

6.1 Setup and Training

The observation and classifier languages are given by the grammars in table 3. As with the poker task, all occurrences of the same variable in a classifier refer to the same value, and separate variables in the same classifier may never refer to the same value. The motivation is again to prevent non-identical atoms in the condition from matching to the same atom in the observation. There is no background theory.

When covering, a separate classifier is created for each action $a \in A(s)$ with the action part set to a, the condition part set to the current observation, and the background knowledge part initially empty. Then the classifier is generalised so that each distinct block constant has a 40% probability of having all its occurrences replaced with a block variable not occurring in the classifier already.

The mutation operation either generalises by replacing all occurrences of a selected block constant with a new block variable, or it specialises by replacing all occurrences of a selected block variable with a block constant not contained already in the expression. When selecting a term for replacement, each distinct constant and variable has an equal probability of being chosen.

A classifier cl_1 can be subsumed by cl_2 only if their expressions are identical or are identical with the exception that each occurrence in cl_1 of a particular block constant is replaced in cl_2 with the same block variable, call it A, and no other block constant in cl_1 maps to A in cl_2.

The system was trained for 10,000 episodes. During each episode the system was initially presented with a randomly generated start state using the method described in [19]. If the start state happened to be a goal state then a new start state was generated. The episode continued until the task

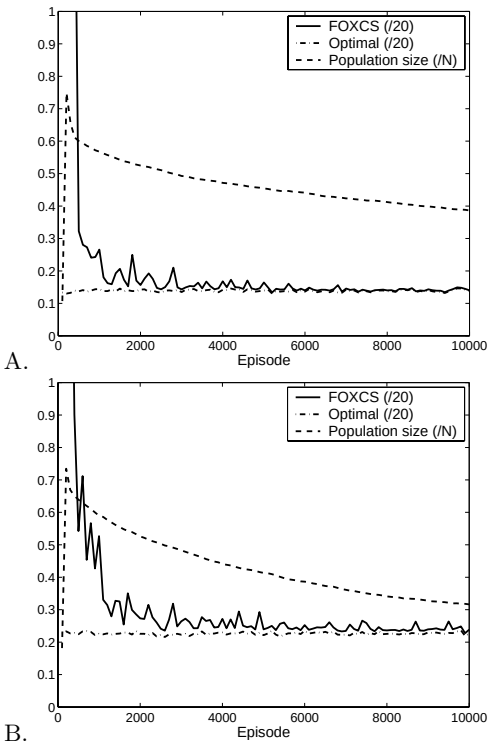

Figure 3: The system performance (solid line) and population size (dashed line) for A) stacking, and B) unstacking in a blocks world with 7 blocks. The performance of an optimal algorithm (dash-dot line) is given for comparison. The population size is the number of macroclassifiers divided by N (2,000). Results are averaged over 10 separate runs.

was complete or an upper limit on the number of steps taken was reached, which was set to 100. On each step the system received an immediate reward of -1. At the beginning of training the population was empty and initially created using covering. The initial classifier parameters were assigned to a random Gaussian with mean -0.1 and standard deviation 0.01 for prediction, p, and mean 0.01 and standard deviation 0.001 for error, ε, and fitness, F. Values for the system parameters were set as follows: $N = 2000$, $\epsilon = 10\%$, $\alpha = 0.1$, $\beta = 0.1$, $\epsilon_0 = 0.01$, $\gamma = 0.9$, $\nu = 5$, $\theta_{ga} = 50$, $\mu = 0.3$, $\theta_{sub} = 20$, $\theta_{del} = 20$, $\delta = 0.1$.

6.2 Results

The system performance and population size during training is shown in figure 3 for both stacking and unstacking. After every 100 training episodes, all learning and exploratory behaviour is switched off, and the performance of the current population of rules is measured over 25 evaluation episodes that start from randomly generated initial states. The performance measure is the average number of steps taken to reach the goal state. For both tasks the performance is quite poor in the early stages of training up to about 500 episodes, then performance improves rapidly until 3,000 episodes or so, after which it levels off until near optimal performance is attained at about 7,000 episodes. Inspection of evaluation episodes revealed that the initial poor performance was due

*	72	$mv(A,F) \leftarrow cl(A), cl(B), cl(F), on\_fl(B), on\_fl(C), on\_fl(G), on(A,G), on(D,E), on(E,C), on(F,D)$
*	72	$mv(D,B) \leftarrow cl(B), cl(D), on\_fl(D), on\_fl(E), on(A,F), on(B,G), on(C,E), on(F,C), on(G,A)$
*	69	$mv(E,B) \leftarrow cl(B), cl(D), cl(E), cl(F), on\_fl(A), on\_fl(C), on\_fl(D), on\_fl(F), on(B,G), on(E,C), on(G,A)$
*	67	$mv(C,F) \leftarrow cl(B), cl(C), cl(F), cl(G), on\_fl(B), on\_fl(C), on\_fl(E), on\_fl(G), on(A,D), on(D,E), on(F,A)$
*	67	$mv(A,B) \leftarrow cl(A), cl(B), cl(C), on\_fl(D), on\_fl(E), on\_fl(F), on(A,D), on(B,G), on(C,F), on(G,E)$
*	64	$mv(F,D) \leftarrow cl(D), cl(F), on\_fl(E), on\_fl(G), on(A,E), on(B,G), on(C,B), on(D,C), on(F,A)$
*	64	$mv(E,F) \leftarrow cl(E), cl(F), on\_fl(A), on\_fl(D), on(B,G), on(C,D), on(E,A), on(F,B), on(G,C)$
*	61	$mv(G,D) \leftarrow cl(B), cl(D), cl(G), on\_fl(B), on\_fl(E), on\_fl(G), on(A,F), on(C,A), on(D,C), on(F,E)$
*	58	$mv(C,B) \leftarrow cl(B), cl(C), cl(E), on\_fl(A), on\_fl(E), on\_fl(G), on(B,D), on(C,F), on(D,G), on(F,A)$
*	46	$mv(F,A) \leftarrow cl(A), cl(C), cl(D), cl(F), on\_fl(B), on\_fl(E), on\_fl(F), on\_fl(G), on(A,B), on(C,E), on(D,G)$
	41	$mv\_fl(A) \leftarrow cl(A), cl(B), on\_fl(B), on\_fl(F), on(A,D), on(C,E), on(D,C), on(E,G), on(G,F)$
	40	$mv(C,G) \leftarrow cl(A), cl(C), cl(G), on\_fl(A), on\_fl(B), on\_fl(G), on(C,E), on(D,F), on(E,D), on(F,B)$
	40	$mv(C,D) \leftarrow cl(C), cl(D), on\_fl(D), on\_fl(G), on(A,G), on(B,E), on(C,B), on(E,F), on(F,A)$
*	39	$mv(A,F) \leftarrow cl(A), cl(B), cl(C), cl(E), cl(F), on\_fl(A), on\_fl(B), on\_fl(D), on\_fl(E), on\_fl(G), on(C,D), on(F,G)$
	34	$mv\_fl(F) \leftarrow cl(D), cl(F), cl(G), on\_fl(C), on\_fl(D), on\_fl(G), on(A,E), on(B,A), on(E,C), on(F,B)$
	33	$mv(B,D) \leftarrow cl(A), cl(B), cl(D), on\_fl(B), on\_fl(C), on\_fl(D), on(A,E), on(E,F), on(F,G), on(G,C)$
	26	$mv\_fl(E) \leftarrow cl(A), cl(E), on\_fl(B), on\_fl(G), on(A,C), on(C,D), on(D,F), on(E,B), on(F,G)$
	22	$mv\_fl(C) \leftarrow cl(C), cl(F), on\_fl(A), on\_fl(D), on(B,G), on(C,E), on(E,D), on(F,B), on(G,A)$
	8	$mv(G,F) \leftarrow cl(D), cl(F), cl(G), on\_fl(C), on\_fl(D), on\_fl(G), on(A,E), on(B,A), on(E,C), on(F,B)$
	5	$mv(C,E) \leftarrow cl(B), cl(C), cl(d), cl(E), cl(F), on\_fl(A), on\_fl(B), on\_fl(C), on\_fl(d), on\_fl(E), on(F,G), on(G,A)$
	5	$mv(D,C) \leftarrow cl(A), cl(C), cl(D), cl(E), cl(g), on\_fl(A), on\_fl(B), on\_fl(D), on\_fl(E), on\_fl(g), on(C,F), on(F,B)$
*	5	$mv(D,C) \leftarrow cl(A), cl(C), cl(D), cl(E), cl(G), on\_fl(A), on\_fl(B), on\_fl(D), on\_fl(E), on\_fl(G), on(C,F), on(F,B)$
	4	$mv(B,G) \leftarrow cl(B), cl(C), cl(d), cl(G), on\_fl(B), on\_fl(d), on\_fl(E), on\_fl(G), on(A,F), on(C,A), on(F,E)$
	1	$mv(D,C) \leftarrow cl(a), cl(C), cl(D), cl(E), cl(G), on\_fl(a), on\_fl(B), on\_fl(D), on\_fl(E), on\_fl(G), on(C,F), on(F,B)$
	1	$mv(C,g) \leftarrow cl(A), cl(C), cl(g), on\_fl(A), on\_fl(B), on\_fl(g), on(C,E), on(d,F), on(E,d), on(F,B)$
	1	$mv(A,C) \leftarrow cl(A), cl(C), cl(e), cl(F), on\_fl(A), on\_fl(B), on\_fl(F), on\_fl(g), on(C,D), on(D,g), on(e,B)$

Table 4: The sample of the population with perfect accuracy after 10,000 episodes of stacking, ordered by numerosity. In the first column an asterisk "*" indicates maximally general classifiers. The second column gives the numerosity values.

to cyclic behaviour, which would only terminate when the 100 step limit was reached.

The average population size rises very quickly to about $0.75 \times 2,000 = 1,500$ macroclassifiers and subsequently decreases, at first sharply and then more gradually, to about 800 and 600 for stacking and unstacking respectively. Again, the decrease in population size is expected from the replacement of specific classifiers with more general ones in the accurate portion of the population.

To assess the level of generality attained, the final population was inspected to determine if it contained the minimum number of classifiers required to cover all states while still preserving optimal performance. For blocks worlds with 7 blocks the minimal cover is 14 for both tasks, although there are several different possible minimal covers. In all training runs observed, minimal or near minimal covers evolved. The

near minimal covers contained at least 12 classifiers and covered 99.9% (for stacking) or 86.5% (for unstacking) of the total state space. Out of the minimal or near minimal covers evolved, about 10 classifiers would usually be the highest ranked by numerosity, with most of the others not far behind. The near minimal cover evolved for stacking during one arbitrarily selected run is shown in table 4; limited space prevents the inclusion of a similar table for unstacking. We conclude that for stacking and unstacking, the generalisation behaviour of Fox-cs is again consistent with Wilson's generality hypothesis, as far as can be determined from the classifiers that produce optimal behaviour.

7. CONCLUSION

We presented a new Xcs derivative, Fox-cs, where the bit-string classifiers are replaced by Horn clauses in first order logic. The system was able to attain near optimal performance at tasks in two relational domains, poker and blocks world, that would be difficult for systems using attribute value languages to represent. In addition, repeated inspection of the system's population showed that, in line with Wilson's generality hypothesis, given sufficient training maximally general classifiers evolved consistently in the portion of the population that produced optimal behaviour. This finding is obviously valuable for Fox-cs, but its wider significance is that it suggests the validity of the generality hypothesis is not conditional upon implementing Xcs with bit-string classifiers and a genetic algorithm.

Future work on Fox-cs will focus on evolving classifiers that accurately generalise over different payoff levels. For example, recall that the optimal unstack policy is to place any clear block onto the floor if it isn't already there, and the optimal stack policy is to place any clear block onto the highest block. These policies give rise to generalisations that apply over different payoff levels, but multi-payoff level generalisations are not supported by Xcs or the traditional learning classifier system framework. Our approach will evolve a secondary population of classifiers whose prediction parameters estimate optimality rather than payoff.

8. REFERENCES

[1] M. Ahluwalia and L. Bull. A genetic programming-based classifier system. In Banzhaf et al. [2], pages 11–18.

[2] W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors. *GECCO'99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 1999.

[3] H. Blockeel and L. D. Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1–2):285–297, 1998.

[4] H. Blockeel, L. D. Raedt, N. Jacobs, and B. Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3(1):59–93, 1999.

[5] L. B. Booker. Representing attribute-based concepts in a classifier system. In G. J. E. Rawlins, editor, *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA91)*, pages 115–127, San-Maeto, 1991. Morgan Kaufmann.

[6] L. Bull and T. O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In *GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 905–911. Morgan Kaufmann, 2002.

[7] M. V. Butz and S. W. Wilson. An algorithmic description of XCS. *Soft Computing*, 6(3–4):144–153, 2002.

[8] S. Džeroski, L. D. Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43(1–2):7–52, 2001.

[9] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, 1995.

[10] J. H. Holland. Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, an Artificial Intelligence Approach*, pages 593–623, Los Altos, California, 1986. Morgan Kaufmann.

[11] W. V. Laer. *From Propositional to First Order Logic in Machine Learning and Data Mining*. PhD thesis, Katholieke Universiteit Leuven, 2002.

[12] P. L. Lanzi. Extending the representation of classifer conditions, part II: From messy codings to S-expressions. In Banzhaf et al. [2], pages 345–352.

[13] P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors. *Learning Classifier Systems: from Foundations to Applications*. Springer-Verlag, 2000.

[14] S. Muggleton. Inductive Logic Programming. In *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*. Academic Press, 1992.

[15] J. R. Quinlan. Learning logical definition from relations. *Machine Learning*, 5(3):239–266, 1990.

[16] J. D. Schaffer, editor. *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, 1989. Morgan Kaufmann.

[17] D. Schuurmans and J. Schaeffer. Representational difficulties with classifier systems. In Schaffer [16], pages 328–333.

[18] L. Shu and J. Schaeffer. VCS: Variable classifier system. In Schaffer [16], pages 334–339.

[19] J. Slaney and S. Thiébaux. Blocks World revisited. *Artificial Intelligence*, 125:119–153, 2001.

[20] P. Tadepalli, R. Givan, and K. Driessens, editors. *Proceedings of the ICML'04 Workshop on Relational Reinforcement Learning*, 2004. http://eecs.oregonstate.edu/research/rrl/index.html.

[21] C. Thornton. *Truth from Trash: How Learning Makes Sense*. The MIT Press, 2000.

[22] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[23] S. W. Wilson. Generalization in the XCS classifier system. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

[24] S. W. Wilson. Get real! XCS with continuous-valued inputs. In Lanzi et al. [13], pages 209–222.