Extending XCS with Representation in First-Order Logic

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

Since the introduction of XCS there have been many derivative systems supporting alternative rule languages such as languages over reals, fuzzy logic, S-expressions and even neural networks. This paper describes FOXCS, a derivative of XCS for learning rules in first-order logic. The FOXCS system is aimed at solving tasks in model-free, relational environments, and is generally applicable to Inductive Logic Programming (ILP) and Relational Reinforcement Learning (RRL). The system was evaluated on several benchmarking ILP tasks where it was found to perform at a level comparable to a number of well-known ILP algorithms with regard to its predictive accuracy. This finding validates the approach of using evolutionary heuristics for discovering rules in first-order logic under the reinforcement learning paradigm, and establishes the system as a promising alternative for RRL.

Categories and Subject Descriptors

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

General Terms

Algorithms, Languages

Keywords

First order logic, inductice logic programming, learning classifier system, relational reinforcement learning, XCS

1. INTRODUCTION

The accuracy based XCS [32, 33] is perhaps the most important Learning Classifier System (LCS) to emerge, realising the majority of the features of Holland's original framework [16], while overcoming the problem of strong overgeneral rules that had hindered earlier strength based systems [17]. Recent analysis supports the view that under favourable conditions XCS has an inductive bias towards the discovery of maximally general but accurate rules [7]. Good empirical results have also been reported, showing that the system is competitive with other machine learning approaches [1, 2].

Originally the XCS system was specified to use bit-string rules, but much subsequent research has focused on extending the representational capability of the system. For instance, rule languages over continuous spaces [34, 29, 30, 8], fuzzy logic [11], S-expressions [20, 19], and even multi-layer neural networks [6] have been previously investigated. This diversity of representations suggests that the framework underlying XCS is largely representationally neutral, and not dependent upon one rule language or another. Further evidence for this view can be found in the pressures which [9] identified as driving the inductive process within XCS, most of which function independently of the rule language.

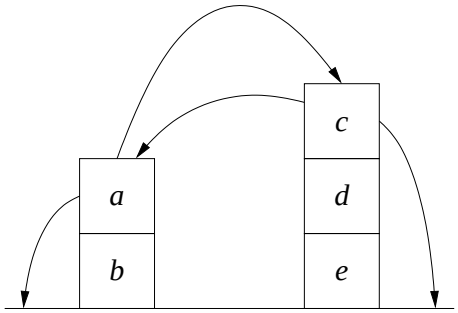
In this paper I extend XCS for representation with first-order logic. First-order logic is an important representational paradigm for machine learning, forming the basis of Inductive Logic Programming (ILP) [24, 22] and Relational Reinforcement Learning (RRL) [31]. The new system is aimed at solving tasks in relational environments where a model of the transition and reward dynamics is not available, and is generally applicable to both ILP and RRL. An advantage of using XCS as the parent system is that it performs both model-free learning *and* addresses automatic generalisation, where currently most RRL systems — with the notable exception of [14] and the family of systems based on it — either require a model of the environment or do not automatically induce generalisations.

2. SYSTEM DESIGN

This section gives a detailed description of the design of the FOXCS system. The description primarily focuses on the modifications made by the new system and assumes that the reader is familiar with the details of XCS. Readers unfamiliar with XCS might like to consult [32] for an introduction to XCS and [10] for a complete algorithmic specification. Note that an earlier version of FOXCS appears in [21]; the principal difference between the two versions is that the covering and mutation operations were domain specific in the earlier version, whereas here domain independent operations have been developed.

The FOXCS system, like its parent XCS, accepts inputs and produces rules, but unlike XCS these inputs and rules are expressed in languages over first-order logic. An input to the system describes the present state of the environment and the potential for action within it. More specifically, it is a pair $(s, \mathcal{A}(s))$, where $s \in \mathcal{S}$ is the current state and $\mathcal{A}(s)$

GECCO'06, July 8–12, 2006, Seattle, WA, USA.



$$s = \{cl(a), on(a,b), on\_fl(b), cl(c), on(c,d), on(d,e), on\_fl(e)\}$$
$$\mathcal{A}(s) = \{mv\_fl(a), mv(a,c), mv\_fl(c), mv(c,a)\}$$

Figure 1: A hypothetical system input.

is the set of actions that apply to s. The state, s, and the set of applicable actions, $\mathcal{A}(s)$, are each represented by a set of ground facts in first-order logic. In addition to the input describing the current condition of the environment, the system can have access to a background theory, B, which takes the form of a collection of rules that pertain to the environment. The use of background knowledge is a feature of many ILP and RRL systems.

The rules in the new system contain a logical part which replaces the bit-string action and condition of their counterparts in XCS. All other parameters and estimates associated with rules in XCS are retained and function in the same fashion as they do in XCS. The logical part of the rule, Φ, is a *definite clause* in first-order logic, that is, it has the form:

$$\varphi_0 \leftarrow \varphi_1, \ldots, \varphi_n$$

where each φ_i is a predicate formula (literal). Each predicate formula φ_i has the form $f(t_1, \ldots, t_m)$, where f is a predicate symbol, and each t_i is a constant or variable (but not a function), and m is the arity of the formula. A special case occurs when the arity of φ_i is zero (i.e. $m = 0$): then φ_i is a *nullary predicate* and just has the form f, that is, the bracketed list is omitted. The head of the rule, φ_0, can be thought of as the rule's action, and the rule body, $\varphi_1, \ldots, \varphi_n$, can be thought of as the rule's condition.

With the new representation comes a corresponding redefinition for matching. The matching operation works as follows: a rule rl with logical part Φ successfully matches a ground state-action pair (s, a) if and only if Φ, s, and the background theory B, together entail a. This leads to the following definition of matching:

DEFINITION 1. *A given rule rl with logical part Φ* matches *a ground state-action pair (s, a) under background theory B, if and only if:*

$$\Phi \wedge s \wedge B \models a \quad (1)$$

FOXCS implements matching using the logic language Prolog: Φ and s are asserted to a Prolog knowledge base which already contains the background theory B, and then the query corresponding to a is posed. If the query succeeds then rl matches (s, a), otherwise it does not. Note that matching logical formula is a more expensive operation than the simple linear procedure used for matching bit-strings. This

input: the current input to the system, $(s, \mathcal{A}(s))$

1. *Construct match sets.* For each $a \in \mathcal{A}(s)$ there is a corresponding match set $[\mathrm{M}]_a$, where $[\mathrm{M}]_a = \{rl \in [\mathrm{P}] \mid \Phi_{rl} \text{ matches } (s, a)\}$.
2. *Trigger covering if required.* For each action $a \in \mathcal{A}(s)$ where $[\mathrm{M}]_a = \emptyset$, the covering operation is called to produce a rule covering (s, a).
3. *Calculate system predictions.* For each $a \in \mathcal{A}(s)$ the system prediction $p(a)$ is calculated as a fitness weighted sum of the predictions of the rules in $[\mathrm{M}]_a$.
4. *Select action.* An action $a^* \in \mathcal{A}(s)$ is selected ϵ-greedily over the system predictions. On greedy steps, if some actions are tied for maximum prediction then these actions are selected from at random uniformly. The action set is assigned: $[\mathrm{A}] := [\mathrm{M}]_{a^*}$.
5. *Execute action.* The selected action, a^* is executed, and a reward is obtained.
6. *Assign credit.* The parameters of the rules in the action set of the previous cycle, $[\mathrm{A}]_{-1}$ are updated. If it is the terminal step of an episode then the rules in [A] are also updated.
7. *Trigger mutation.* From time to time mutation is run on a parent rule selected from $[\mathrm{A}]_{-1}$. If it is the terminal step of an episode then mutation may be triggered on [A] also.

Figure 2: The algorithm for the operational cycle.

represents the cost of increasing the expressive power of the rule language, and occurs in all systems that upgrade from propositional representations to first-order logic.

The operational cycle for the new system, which is largely identical to its counterpart procedure in XCS, is shown in figure 2. The principal modification occurs at step 1 and arises because the system can generalise over actions as well as states: for each action $a \in \mathcal{A}(s)$ a separate match set, denoted $[\mathrm{M}]_a$, is constructed. Note that an individual rule can, and often does, belong to more than one match set if it contains an abstract action.

Rule discovery in FOXCS consists of covering plus various mutation operations. These operations have the same purpose as their counterparts in XCS but have been completely re-defined for representation with first-order logic. In order to create and vary rules the operations will need to know the predicates that constitute the rule language, which will depend on the task at hand. A language declaration command that enables such user provided definitions is a common feature of most ILP systems. In FOXCS the command is `mode`$(type, [min, max], neg, pred)$, which declares the predicate $pred$ for inclusion in the rule language. The arguments are: $type$, which indicates whether $pred$ is an action, state or background predicate (the values are `a`, `s` and `b` respectively); min and max, which are integers specifying the minimum and maximum number of occurrences of $pred$ allowed within an individual rule; and neg, which determines whether $pred$ may be negated or not.

The general form of $pred$ itself is $name(arg_1, \ldots, arg_n)$, where $name$ is the name of the predicate, arg_i is a declaration for the ith argument of $pred$, and n is the predicate's arity (if a predicate has arity $n = 0$, then the form of $pred$ reduces to just $name$). Each arg_i is a list $[arg\_type, spec_1, spec_2, \ldots]$, where arg_type is a type specifier and the remaining arguments, $spec_1, spec_2, \ldots$, are symbols which determine how

input: a state-action pair (s, a)

1. create and initialise a new rule rl
2. $\Phi := \varphi_0 \leftarrow \varphi_1, \ldots, \varphi_n$, where $\varphi_0 = a$ and $\varphi_1, \ldots, \varphi_n$ is the clause containing all the facts in s (as part of this step each place p in Φ is associated with a set of mode symbols M_p according to the `mode` declarations)
3. $\theta^{-1} := \{\langle c_1, \{p_{1,1}, \ldots, p_{1,k_1}\}\rangle/v_1, \ldots, \langle c_l, \{p_{l,1}, \ldots, p_{l,k_l}\}\rangle/v_l\}$, where $c_1, \ldots, c_l$ are the constants of Φ, $p_{i,j}$ are the places in Φ where c_i occurs s.t. $\Theta(M_{p_{i,j}}) =$ `-`, and $v_1, \ldots, v_l$ are the variables to be substituted for $c_1, \ldots, c_l$, where each v_i is unique.
4. $\Phi := \Phi\theta^{-1}$
5. insert rl into $[P]$ with deletion

Figure 3: The algorithm for covering.

arg_i may be set. The symbols and their meanings are given below:

`+` *Input variable.* The argument may be set to a named variable (of the same type) that already occurs in the rule.

`-` *Output variable.* The argument may be set to a named variable that does not already occur in the rule.

`#` *Constant.* The argument may be a constant.

`_` *Anonymous variable.* The argument may be set to the anonymous variable.[1]

`!` If the argument is a variable then it must be unique within the literal's argument list.

The algorithm for covering is given in figure 3. Firstly, a new rule, rl, is created and its parameters (except for Φ) are initialised in the same way as XCS. Then at step 2, the logical part, Φ, is set to a clause derived from (s, a). At this stage the rule does not generalise because (s, a) is ground, thus steps 3 and 4 generalise the rule by creating and applying an inverse substitution which replaces some or all of the constants with variables, where the constants that will be replaced are determined by the `mode` declarations. Lastly, rl is inserted in the population where deletion may occur if the number of rules exceeds the system parameter N.

After an initial population is created through covering, further exploration of the rule space is achieved by applying mutation. The system uses three generalising and three specialising mutations, some of which were originally inspired by the ECL system [13]. The generalisation operations are:

DEL: This operation randomly selects a literal and deletes it.

C2V: This operation performs a inverse substitution on a randomly selected constant (i.e. occurrences of the constant are replaced with a variable).

V2A: This operation randomly selects a place containing a variable and replaces the variable with the anonymous variable.

[1]The role of the anonymous variable in first-order logic is analogous to role of the "don't care" symbol, `#`, in bit-string languages.

The specialisation operations are:

ADD: This operation creates and adds a new literal.

V2C: This operation performs a substitution on a randomly selected variable (i.e. occurrences of the variable are replaced with a constant).

A2V: This operation randomly selects a place containing an anonymous variable and replaces it with a named variable (i.e. an input or output variable).

In addition to the mutation operations there is also a reproduction operation:

REP: This operation increments the numerosity of the parent rule. It can be used to encourage the system to converge to a population containing the most highly fit rules.

When mutation is triggered the system randomly selects one of the above seven operations to apply. Each operation, $i \in \{\text{DEL,C2V,V2A,ADD,V2C,A2V,REP}\}$, is associated with a weight μ_i, and the selection probability for i is proportional to its relative weight, $\frac{\mu_i}{\sum_j \mu_j}$. If the operation fails to produce offspring then another randomly selected operation is applied, and so on, until one succeeds.

Finally, the subsumption deletion procedure makes use of θ-subsumption [25] to test whether one rule is more general than another.

3. EVALUATION

Tasks in ILP are essentially classification tasks, and it has been shown that XCS is a competitive method for classification [1, 2]. Therefore, in order to empirically validate the system I compared Foxcs to some well-known ILP systems to determine whether it performs as competitively as its parent, XCS.

Four well-known ILP algorithms were selected for comparison on the basis that they have results published for the same benchmarking data: FOIL [26], PROGOL [23], ICL [12], and TILDE [4]. The first three are rule-based systems, while TILDE uses a tree-based representation. An evolutionary ILP algorithm was also selected, ECL [13], which employs a memetic search that hybridises evolutionary and ILP search heuristics. The following three benchmarking data sets were used: Mutagenesis [28], Biodegradability [5] and Traffic [15].

For classification a typical performance measure is the predictive accuracy under 10-fold cross-validation. However, care must be taken because FOXCS is not a deterministic system and produces different rules when an experiment is re-run, which potentially reduces the reproducibility of the result. In order to minimise the effects of non-determinism, 10-fold cross-validation was repeated ten times on the same data partition. The predictive accuracies reported for the comparison systems were determined by a single 10-fold cross validation for the Mutagenesis and Traffic data sets, and by five repetitions of 10-fold cross validation (on five different 10-fold partitions) for Biodegradability.

The following system parameters were used for the FOXCS system: $N = 1000$, $\epsilon = 10\%$, $\alpha = 0.1$, $\beta_F = 0.1$, $\epsilon_0 = 0.01$, $\nu = 5$, $\theta_{ga} = 50$, $\theta_{sub} = 20$, $\theta_{del} = 20$, $\delta = 0.1$, $\mu_{rep} = 1$, $\mu_{add} = 3.25$ and $\mu_{del} = 0.75$. The learning rate was annealed and both GA and action set subsumption were used. The system was trained for 100,000 steps, but mutation was

Algorithm	*Predictive Accuracy (%)*	
	Mutagenesis (NS+S1)	*Mutagenesis (NS+S2)*
FOXCS	84 (2)	87 (1)
ICL	87 (10)	88 (8)
TILDE	85	86
PROGOL	82 (3)	88 (2)
FOIL	83	82
ECL	–	90 (1)
	Biodegradability	*Traffic*
FOXCS	72 (2)	94 (1)
ICL	75 (1)	93 (4)
TILDE	74 (1)	94 (4)
PROGOL	–	94 (3)
FOIL	–	–
ECL	74 (4)	93 (2)

Table 1: Comparison between the predictive accuracy of Foxcs and selected ILP algorithms on the benchmarking data sets. The standard deviation, where available, is given in parentheses.

switched off after 90,000 steps in order to encourage convergence of the system prediction. The reward was 10 for a correct classification and -10 for an incorrect classification.

Table 1 compares the predictive accuracy of FOXCS to the above ILP algorithms on the three data sets.From the table it can be seen that FOXCS performs at a level comparable to the other systems. Differences in the accuracy rates of the systems may be due to variations in their particular inductive and language biases. More sophisticated methods for handling numerical attributes were used by TILDE, ICL and ECL, which may also partially account for their better performance at Mutagenesis and Biodegradability, both of which contain numerical data.

In summary, the results for FOXCS are consistent with previous findings that XCS is a competitive approach to machine learning, validating the effectiveness of FOXCS as a method for evolving rules in first-order logic under the reinforcement learning paradigm.

4. CONCLUSION

In this paper the XCS system was extended with representation in first-order logic, deriving a system generally applicable to both ILP and RRL tasks. It was found that the new system, FOXCS, performs at a level comparable to several well-known ILP algorithms, which is consistent with previous findings for XCS on classification tasks, validating the use of evolution for discovering rules in first-order logic in the context of reinforcement learning. The wider significance of the result is two-fold. Firstly, for the field of LCS it suggests that the mechanisms driving induction in XCS are largely representationally neutral, especially when considered with the growing number of systems that extend the representational capabilities of XCS. The finding also establishes the XCS framework as a promising alternative for RRL. An advantage of using the XCS framework for RRL is that it both supports online learning and automatically induces generalisations, and furthermore it avoids the need for complex and possibly unnatural gradient calculations through the use of evolutionary heuristics. Further work will look at applying FOXCS to multi-step RRL tasks, such as blocks world.

5. REFERENCES

[1] Ester Bernadó, Xavier Llorà, and Joesp M Garrel. XCS and GALE: A comparative study of two learning classifier systems on data mining. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Advances in Learning Classifier Systems: 4th International Workshop, IWLCS 2001*, volume 2321 of *LNCS*, pages 115–132. Springer-Verlag, 2002.

[2] Ester Bernadó-Mansilla and Josep Maria Garrell-Guiu. Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks. *Evolutionary Computation*, 11(3):209–238, 2003.

[3] Hans-Georg Beyer and Una-May O'Reilly, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2005, Washington DC, USA, June 25-29, 2005*. ACM Press, 2005.

[4] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1–2):285–297, 1998.

[5] Hendrik Blockeel, Sašo Džeroski, Boris Kompare, Stefan Kramer, Bernhard Pfahringer, and Wim Van Laer. Experiments in predicting biodegradability. *Applied Artificial Intelligence*, 18(2):157–181, 2004.

[6] Larry Bull and Toby O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In Langdon et al. [18], pages 905–911.

[7] Martin Butz, Tim Kovacs, Pier Luca Lanzi, and Stewart W. Wilson. Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, 8(1):28–46, 2004.

[8] Martin V. Butz. Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In Beyer and O'Reilly [3], pages 1835–1842.

[9] Martin V. Butz and Martin Pelikan. Analyzing the evolutionary pressures in XCS. In Spector et al. [27], pages 935–942.

[10] Martin V. Butz and Stewart W. Wilson. An algorithmic description of XCS. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems. Third International Workshop (IWLCS-2000)*, pages 253–272. Springer-Verlag, 2001.

[11] Jorge Casillas, Brian Carse, and Larry Bull. Fuzzy XCS: an accuracy-based fuzzy classifier system. In *Proceedings of the XII Congreso Espanol sobre Tecnologia y Logica Fuzzy (ESTYLF 2004)*, pages 369–376, 2004.

[12] Luc De Raedt and Wim Van Laer. Inductive constraint logic. In K. P. Jantke, T. Shinohara, and T. Zeugmann, editors, *Proceedings of the Sixth International Workshop on Algorithmic Learning Theory*, volume 997 of *LNCS*, pages 80–94. Springer-Verlag, 1995.

[13] Federico Divina and Elena Marchiori. Evolutionary concept learning. In Langdon et al. [18], pages 343–350.

[14] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1–2):7–52, 2001.

[15] Sašo Džeroski, Nico Jacobs, Martin Molina, Carlos Moure, Stephen Muggleton, and Wim. van Laer. Detecting traffic problems with ILP. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, volume 1446 of *LNCS*, pages 281–290. Springer-Verlag, 1998.

[16] John H. Holland. Adaption. In R. Rosen and F. M. Snell, editors, *Progress in Theoretical Biology*, volume 4, NY, 1976. Plenum.

[17] Tim Kovacs. Towards a theory of strong overgeneral classifiers. In Worthy Martin and William Spears, editors, *Foundations of Genetic Algorithms 6*, pages 165–184. Morgan Kaufmann, 2001.

[18] William B. Langdon, Erick Cantú-Paz, Keith E. Mathias, Rajkumar Roy, David Davis, Riccardo Poli, Karthik Balakrishnan, Vasant Honavar, Günter Rudolph, Joachim Wegener, Larry Bull, Mitchell A. Potter, Alan C. Schultz, Julian F. Miller, Edmund K. Burke, and Natasa Jonoska, editors. *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002*. Morgan Kaufmann, 2002.

[19] Pier Luca Lanzi. Extending the representation of classifer conditions, part II: From messy codings to S-expressions. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 345–352. Morgan Kaufmann, 1999.

[20] Pier Luca Lanzi. Mining interesting knowledge from data with the XCS classifier system. In Spector et al. [27], pages 958–965.

[21] Drew Mellor. A first order logic classifier system. In Beyer and O'Reilly [3], pages 1819–1826.

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

[23] Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3–4):245–286, 1995.

[24] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[25] Gordon D Plotkin. *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University, 1971.

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

[27] Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, California, USA, July 7-11 2001. Morgan Kaufmann.

[28] Ashwin Srinivasan, Stephen Muggleton, Michael J. E. Sternberg, and Ross D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.

[29] Christopher Stone and Larry Bull. For real! XCS with continuous-valued inputs. *Evolutionary Computation*, 11(3):299–336, 2003.

[30] Christopher Stone and Larry Bull. An analysis of continuous-valued representations for learning classifier systems. In Larry Bull and Tim Kovacs, editors, *Foundations of Learning Classifier Systems*, volume 183 of *Studies in Fuzziness and Soft Computing*, pages 127–176. Springer-Verlag, 2005.

[31] Martijn van Otterlo. A survey of reinforcement learning in relational domains. Technical Report TR-CTIT-05-31, University of Twente, The Netherlands, 2005.

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

[33] Stewart W. Wilson. Generalization in the XCS classifier system. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

[34] Stewart W. Wilson. Get real! XCS with continuous-valued inputs. In Pier Lucs Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Learning Classifier Systems: from Foundations to Applications*, volume 1813 of *LNCS*, pages 209–222. Springer-Verlag, 2000.