To be presented at ES'95

A Hybrid Rule-based System with Rule-refinement Mechanisms

Riccardo Poli, Mike Brayshaw and Aaron Sloman

School of Computer Science
The University of Birmingham
Birmingham B15 2TT
United Kingdom
E-mail: {R.Poli,M.C.Brayshaw,A.Sloman}@cs.bham.ac.uk

September 27, 1995

Abstract

In this paper we present HTR, a hybrid trainable rule-based system. The key features of the system include: the heterogeneous integration of multiple control regimes, rule induction and refinement mechanisms, easy and concise specification of knowledge, treatment and support for uncertainty. All these features are obtained thanks to the introduction of a new formalism for rule specification that extends the behaviour of a traditional rule based system. The formalism utilises predicates as an intermediate layer between the conditions and the actions of rules. Different kinds of predicates support various sorts of control, distributed conflict resolution, and learning from examples. In particular we describe the characteristics of truth-table-based, collection-based and artificial-neural-net based predicates which are pre-defined in HTR. We also report on a method for automatic rule induction using genetic algorithms. Experimental results exploiting the afore-mentioned features are described.

1 Introduction

In symbolic AI the value of the integration of heterogeneous inference methods is already well understood, e.g. CAKE [9] and KEATS [5]. Therefore, it seems natural to try and combine standard symbolic inference techniques with other sub-symbolic or numeric models. One attempt in this direction has been the implementation of symbol systems using neural networks (e.g. [11, 14, 7]). This has the advantage of producing massively parallel, fault-tolerant systems that have the functionality typically associated with symbolic AI. However, in these systems, extensions often require retraining the nets or changing their topology, so that the symbol systems implemented may not be readily changeable. Another approach has been to use different technologies for different parts of the system. In general a neural network is used for pre-processing tasks, such as pattern recognition/classification, while a traditional inference system performs the higher-level tasks, such as reasoning. Schreinmakers [10] provides a good example of this sort. However, this weak integration of the different technologies does not allow the exploitation of their joint power within the same module.

Elsewhere, in machine learning, attention has also been turning to hybrid approaches. Two leading approaches use either empirical learning (learning from example sets) or domain knowledge, e.g. example-based learning (EBL) [4], to acquire new knowledge. However, typically empirical learning does not take into account domain knowledge when learning. Further EBL techniques require the domain theory to be complete and correct. This has led to attempts to produce hybrid systems that aim at integrating the two approaches, by improving existing domain theories using further examples. EITHER [6] used three different types of inferences to refine incorrect theories. KBANN [15] is a hybrid system that takes nearly complete rulebases, expressed as Horn clauses, and translates them into initial neural networks. Subsequently, these networks are further trained using a separate training set. Although clearly capable of impressive learning performance, the concepts developed are implicit within the neural representation. Further conventional cycling rules and variables are not currently supported.

In this paper we present an alternative approach to hybridisation which allows a more tight and natural integration between symbolic and and non-symbolic systems. Specifically, we have designed a system called HTR (Hybrid Trainable Rulebase) which allows useful properties of sub-symbolic/numeric systems to be integrated with the inference power of symbolic rules. In HTR, concepts and inference are modelled explicitly at the knowledge level, but other types of inference mechanisms, e.g. artificial neural networks or conventional or genetically-derived algorithms, can be used as control regimes over explicit symbolic concepts. Thus, where behaviour is best modelled by training a neural network (e.g. because of their noise tolerance) using a given dataset this can be done, yet within the confines of a conventional rulebased system.

The paper is organised as follows. We first describe an overview of our system and explain how hybrid reasoning capabilities may be integrated within a conventional recognise-act cycle. Then, we present four increasingly powerful techniques for rule induction/hybrid inference: one based on a truth-table representation, a second one based on a form of example based learning, a third one based on artificial neural nets and a fourth one based on genetic algorithms. Next, we report on some experimental results using such techniques, demonstrating how these ideas have been tried out in several different domains. Finally, we draw some conclusions.

2 The System

In this section we shall show how HTR extends conventional rule based systems to include the learning and control techniques present in other inference technologies. Learning capabilities have obvious potential in knowledge acquisition. However our system also aims to let the knowledge engineer model explicit knowledge whenever possible, yet be more flexible as to the inference paradigm applied to this knowledge. A side-effect of this is that large amounts of knowledge can be expressed in a very succinct way. In the next sections we will go on to describe how this can be done.

2.1 Knowledge Representation

We are all familiar with a typical if `IF <condition 1> AND <condition 2> .... AND <condition N> THEN <list of actions>` style of rules used in systems such as [2]. In HTR we have extended this format as follows:

```
Rule <rule name>:
  IF <condition 1>, <condition2>, ....  <condition N>
  SATISFY <predicate>
  THEN   <list of actions>.
```

Every time IF-SATISFY-THEN (IST) rules of this kind are considered by the interpreter, the pattern including the truth and falsity values of the conditions plus the variable instantiations associated with the conditions (we call it the *condition-pattern*) is used as argument for `<predicate>`. The predicate is satisfied if it returns a non-false result. If the predicate is satisfied (in one of the several ways described below), a corresponding instance of the rule is placed in the conflict set.

It is worth noting that if `<predicate>` is the standard `AND` predicate (or any more complex combination of `AND`, `OR` and `NOT`), the proposed system behaves exactly as a classical rule-based system having the same conditions and actions. However, this syntax allows for the definition of more general rules, thus leading to a reduction of the number and size of the rules in the system and to easier maintenance of the knowledge base. Let us consider some examples.

Let us suppose that `<predicate>` is the `MAJORITY` predicate (it is true if the majority of the conditions are true). Now, we can see that it is not so simple to write a set of rules equivalent to the rule:

```
Rule <rule name>:
  IF <condition 1>, <condition2>, ....  <condition N>
  SATISFY MAJORITY
  THEN <list of actions>
```

If N is small the task may be manageable, but for larger N about 2^{N-1} conjuncts of the conditions have to be handled making the task quite tedious, potentially error prone and involves use of several rules with overlapping sets of conditions, which may impose a significant run time cost. This happens because the decision on whether the actions have to be performed depends on properties (the number of true conditions in this case) of the whole set of conditions, as opposed to the truth or falsity of known logical combinations. Other examples of this kind of rule-control regimes include even and odd parity, all-but-one, and multiplexing. Other situations in which the proposed rule format can be quite useful may arise when the conditions record the current values of sensory transducers (e.g. in the control system for an industrial plant or a robot) and the actions depend on some non-Boolean combination of such values. In such cases, the calculations needed to make a decision could naturally be performed in `<predicate>`.

It is quite common when writing a set of rules, that several of them share the same conditions (although combined via different sets of AND, OR and NOT operators, in the classical formalism, or different predicates in our formalism) but have different actions. It would be desirable to be able to represent such rules with a single rule. In addition quite often the knowledge engineer wants only one rule of the set to fire (e.g. because the actions are mutually exclusive): this can be handled only with procedural hand coding or clever conflict-resolution mechanisms. To overcome these limitations we have extended the previous formalism as follows:

```
Rule <rule name>:
  IF <condition 1>, <condition 2>, ....  <condition N>
  SATISFY <vector predicate>
  THEN SELECT_ACTIONS <list of actions 1>; ...;<list of actions M>.
```

where `<vector predicate>` is a function that transforms the domain $\{\text{True}, \text{False}\}^N$ into the domain $\{\text{True}, \text{False}\}^M$.[1] In IF-SATISFIES-THEN-SELECT_ACTIONS (ISTS) rules the condition-pattern is passed to `<vector predicate>`. It returns a list, called *action-pattern*, containing M True or False values. The lists of actions for which a True value has been returned are added to the conflict set; the other lists of actions are ignored. With this formalism it is quite easy to represent a large number of rules with a single one, and to implement any form of distributed conflict resolution.

2.2 Hybrid Control and Learning Regimes

HTR is implemented in the Poplog environment [1]. The implementation gives the knowledge engineer complete freedom as to the kind of qualitative or quantitative models to be used as predicates and as to their implementation. In the Poplog environment external numeric/symbolic programs can be invoked as easily as defining internal Pop-11 predicates. However, some predefined predicates are available which enable several control and learning regimes. They are described below.

2.2.1 Truth-table-based Predicates

When an IST rule includes only conditions without variables or with Boolean variables only, the related predicate can be implemented via a truth table. One advantage of this representation is that it allows a simple (though limited) form of rule induction from examples. For example, the truth table of a predicate might be only partially specified at compile time and new entries can be entered at runtime.

For example, a rule with two conditions and a single list of actions could be represented by the truth table shown in Table 1, where `FIREABLE` is a

[1]The keyword `SELECT_ACTIONS` is optional. It has been introduced for clarity of notation.

value that represents the "output" of the `<predicate>` (i.e. the action-pattern of the rule). Then, every time the rule is considered by the interpreter the condition-pattern (e.g. the pattern [False,True]) is looked up in the truth table and the related `FIREABLE` value is considered. If such a value is True or False it is used to make `<predicate>` succeed or fail, respectively, if it is Unknown the user (at first probably the expert) is asked to provide a value for `FIREABLE`. Such a value is then stored in the truth table and used thereafter. Alternatively, after some number of such interactions a stochastic test could be induced.

`<condition 1>`	`<condition 2>`	FIREABLE
False	False	True
False	True	Unknown
True	False	True
True	True	False

Table 1: Truth-table-based predicate implementation.

In addition to allowing learning, truth-table-based predicates are an extra validation trap to catch cases that may have accidentally been missed out. This is useful for applications such as real-time control or medical systems, where it seems safer to have a system asking for advice in the presence of a new/unexpected situations than a system that simply ignores such situations.

2.2.2 Collection-based Predicates

The previous idea is extended by another type of predicate that instead of using truth tables uses *collections* of pattern pairs (implemented with hash tables). Each pair includes an input pattern representing an instantiation for the condition pattern of a rule and an output pattern of values in the range [0,1] termed the `FIREABILITY` pattern. Each element of the `FIREABILITY` pattern represents the level of confidence that the related list of actions should be fired in the presence of the given condition-pattern.

Every time a collection-based predicate is called the actual condition-pattern is searched for in the collection. If a pair is found whose condition-pattern is equal to the given pattern then the related `FIREABILITY` pattern is used to evaluate the `FIREABLE` values (the components of the action-pattern) through the following formula:

$$\text{FIREABLE} = \begin{cases} True & \text{if FIREABILITY} > P, \\ False & \text{otherwise.} \end{cases}$$

where P is a parameter in the range $[0,1]$, termed the *prudence* of the system (which may or may not vary at run time). Once all the `FIREABLE` values are known the behaviour of the system is the same as for truth-table-based predicates.

Collection-based predicates can also be used to refine or induce new rules. In fact, as in the previous method, if the condition-pattern is not found among the stored pattern-conditions, the user is asked to provide a value for `FIREABILITY` which can be stored for future use. An example of reasonable options and the related meanings is shown in Table 2.[2]

`FIREABILITY`	*Meaning*
1	"I'm sure the action should be fired"
0.75	"Probably the action should be fired"
0.5	"I'm not sure the action should be fired"
0.25	"Probably the action should NOT be fired"
0	"I'm sure the action should NOT be fired"

Table 2: A possible set of meanings for `FIREABILITY` values.

The prudence parameter P can be used to modify the behaviour of collection-based rules and, therefore, of the system: the greater the *prudence*, the more confidence (`FIREABILITY`) is required for an action to be fired.

2.2.3 Artificial-Neural-Network-based Predicates

Collection-based predicates are used as a basis to implement ANN-based predicates and obtain symbolic/sub-symbolic hybrid rules. ANN predicates work as follows.

Initially when an ANN-based predicate is called the user is asked to provide `FIREABILITY` patterns that are stored as in the case of collection-based predicates. After a predefined number of pairs have been stored, their components are transformed into values in the range [0,1] via rescaling and mapping. The resulting numeric patterns are used as examples for a neural network trained with some suitable algorithm (e.g. the backpropagation rule). Afterwards, when the ANN-based predicate is called the condition-pattern is fed into (and propagated through) the net and the output is taken to be the `FIREABILITY` pattern of the rule and is used accordingly.

In cases where one can rely on the properties of generalisation, noise rejection, input rectification, etc. of a neural network, ANN-based predicates also behave sensibly in the presence of new or partially inconsistent condition-patterns, without requiring the intervention of the user.

2.2.4 GA-based Learning

The previous mechanisms for inducing or refining the `<predicate>` of a rule from examples require interaction with experts. However, there may be situations when the relevant conditions and actions of a rule are known, but there is no easy way of deciding, for a given condition pattern, if and which actions should be chosen (e.g. rarely seen contingencies for which a decision might require tedious or complex calculations or expensive consultations with other experts). For such cases we have implemented an

[2]An alternative view is that of considering the `FIREABILITY` values of a rule as values quantifying the percentage of cases in which such rule is true, i.e. the *reliability* of the rule (e.g. 1 could mean "always valid", 0.75 "often", etc.).

alternative, although computationally expensive, predicate-induction mechanism not requiring rule-by-rule examples. It is based on Genetic Algorithms (GAs) [3].

To explain how the mechanism works, let us consider the simple problem of finding the truth tables needed for an ISTS rule including N conditions and M actions. The truth table includes $2^N \times M$ `FIREABLE` entries, that can be represented as a bit string. If we define an objective (or fitness) function that scores each possible instance of the string, we can then easily apply a GA for finding the best rule.

Typically, one will want to induce genetically several IST(S) rules (by chaining the bit strings representing each truth table) and such rules will be in a rulebase including other non-trainable rules. In addition, the behaviour of a rulebase depends on the initial data in working memory. So, in HTR we have used fitness functions that include terms that consider the accuracy of the conclusions reached with a set of initial databases and also terms that consider the amount of computation (number of interpreter cycles) needed to reach such conclusions. The general form of such functions is:

$$f(R) = \sum_j [A(D_j) + \lambda \times C(D_j)]$$

R being a given rule-base, D_j a initial database of facts, $C(D_j)$ the computation required to reach a conclusion for database D_j, $A(D_j)$ the accuracy of the conclusions obtained with database D_j and λ a constant factor. (Computation and accuracy have problem-dependent operative definitions.)

Extensions of this method to the other types of predicates have also been implemented. For example, to induce ANN-based predicates, we encode the weights and the biases of a neural net into the bit strings that undergo genetic optimisation via a fixed-point number-representation.

2.3 Implementation Issues

The implementation of HTR has undergone several phases. A system having a subset of the current capabilities of HTR was originally implemented in Prolog [8]. Successively, some of the ideas of this system where included (after being extended) in POPRULEBASE (PRB), a rule-based system in Pop-11 designed by Aaron Sloman [12]. (These new features were later used in experiments with SIM_AGENT, a toolkit for the design of intelligent agents based on PRB [13].) PRB includes many unusual facilities to control the matching of conditions and the firing of actions, including the possibility of running plain Pop-11 code within conditions or actions. These facilities enabled us easily to (re)implement HTR in Pop-11 using PRB as basic inference engine.

The system allows both normal IF-THEN rules and the new IF-SATISFY-THEN rules to be used in the same rulebase. Standard HTR rules are simply translated into equivalent standard PRB rules. The translation of IST(S) rules is more complicated as the original conditions of such rules are interleaved by several calls to Pop-11 code. Such code has to perform the following tasks: a) to assign the correct value to the condition variables (variables that state if the related conditions are true or false); b) to call the predicate to be satisfied by the condition variables and the variables bound in the conditions, if any (such variables are undefined if the related condition is false); c) to store the output of the predicate into a variable. Such a variable is used by a PRB action (called SELECT) to decide which actions to execute.

An unusual condition of PRB, called CUT, is used to control the pattern matcher's attempts to instantiate the conditions in IST rules. HTR syntax allows the user to decide whether the CUT condition has to be used before or after the result of the predicate is known, and also whether the CUT should not be used at all. If the CUT is before the predicate, the pattern matcher does not attempt to reinstantiate the preceding conditions, whatever the result of the predicate. If it is used after the predicate and the predicate fails, the pattern matcher tries to find other matches for the conditions which might allow the predicate to succeed. If no CUT is used, the pattern matcher instantiates the rule with all possible matches for the conditions for which the predicates produces a non-false result.

Predefined predicates are implemented by redefining syntax words which once compiled produce Pop-11 code via a process similar to macro expansion. This is completely transparent to the user.

3 Experimental Results

We have implemented and used the previously described formalism and the related rule induction mechanisms in several experiments, some of which are described in the following subsections. A preliminary set of experiments with Winston's animals [16] showing some of the features of our approach was described in [8]. In this paper we present three examples that show the advantages of our approach in practical applications.

3.1 A Hybrid System for Diagnosing and Treating Hypertension

Standard rules as well as IST and ISTS rules with numeric and collection-based predicates have been used in the development of a hybrid rulebase for the diagnosis and treatment of hypertension (high blood pressure). The objective of the system is to perform an analysis of some blood pressure (BP) measurements and anamnestic data of a patient in order to decide: a) if he/she is hypertensive, b) if so, whether to treat him/her, and c) what kind of drugs are more appropriate, if treatment is required.

Thanks to the expressive power of ISTS rules the rulebase includes only seven rules. Three rules compute the average and the standard deviation of systolic and diastolic BP measurements. Another rule evaluates the mean BP (MBP) and its standard deviation. These are all standard IF-THEN rules. The others are ISTS rules which make all the important medical decisions.[3]

[3]For brevity, output messages and some anamnestic data have been removed from ISTS

The first ISTS rule,

```
Rule check_hypertension:
IF
    sex ?sex,
    age ?age,
    mbp ?mbp,
    mbp_std_dev ?mbpsd
SATISFY hyper_predicate
THEN SELECT_ACTIONS
    hypertensive subject;
    normal subject;
    possible borderline subject;
    STOP.
```

is a hybrid numeric/symbolic rule to determine if the patient is hypertensive. It calls a user-defined predicate which receives as input a condition-pattern including eight values: four represent the truth or falsity of the conditions, the other four are the values of the variables in such conditions. On the basis of this information the predicate tries to estimate a BP limit with which to compare the MBP of the patient. In general such limit is a function of age and sex (if sex and/or age are not available, alternative calculations are used). The limit is then compared with MBP to determine if the patient is hypertensive. Numeric calculations use also the standard deviation of MBP to determine if borderline conditions are present.

The second ISTS rule,

```
Rule check_whether_to_treat:
IF
    hypertensive subject,
    hypertensive relative ==,
    left ventricular hypertrophy,
    retinopathy,
    possible borderline subject
SATISFY treat_predicate
THEN SELECT_ACTIONS
    treat subject;
    do not treat subject;
    STOP.
```

makes a decision as to whether to treat a hypertensive patient (mild or borderline hypertension is sometimes not treated unless there is evidence of some form of end-organ damage). It invokes a collection-based predicate.

The last ISTS rule,

```
Rule select_treatment:
IF
    treat subject,
    asthma ?degree,
    diabetes ?type
SATISFY drug_predicate
THEN SELECT_ACTIONS
```

rules.

Patient 1 (Normal)	*Patient 2* (Borderline Hypertensive)	*Patient 3* (Borderline Hypertensive)	*Patient 4* (Hypertensive)
`sex male`	`sex female`	`sex female`	`sex male`
`age 30`	`age 25`	`age 25`	`age 55`
`bp 82 120`	`bp 80 130`	`hypertensive relative (mother)`	`hypertensive relative (mother and brother)`
`bp 75 136`	`bp 95 134`	`bp 80 130`	`asthma severe`
`bp 85 143`	`bp 96 164`	`bp 95 134`	`bp 100 145`
`bp 88 122`	`bp 90 140`	`bp 96 164`	`bp 102 154`
`bp 86 138`	`bp 87 140`	`bp 90 140`	`bp 93 148`
		`bp 87 140`	`bp 105 176`
			`bp 108 155`

Table 3: Initial working memories for four patients.

```
    administer beta-adrenoceptor  blocking   drug;
    administer angiotensin converting enzyme inhibitor;
    administer calcium-channel blocking drug;
    administer diuretic;
    administer alternative drugs;
    STOP.
```

has the function of selecting the most appropriate drug (or combination of drugs) for the treatment. For brevity, only two (out of fifteen) anamnestic data entries which determine the drug-compatibility of a patient are included. The rule calls a collection-based predicate.

Let us now consider the behaviour of the system during the pre-release phase in which the last two IST rules are to be induced completely (i.e. their collections are empty). The system has the capability of determining if a patient is hypertensive and if borderline conditions exist. The expert has to run the system with the data of several patients to "fill the gaps". Here we report on the experiments performed with the data of the four patients described in Table 3.[4]

In a first experiment the system was run with prudence $P = 1$. Patient 1 was classified as normal by `hyper_predicate` and the system was stopped without any user interactions (i.e. before the other IST rules where checked).

Patient 2 was first classified as borderline hypertensive, then rule `check_whether_to_treat` was checked and `treat_predicate` was called with condition-pattern [True,False,False,False,True]. As the collection did not contain such pattern, the medical user was asked to provide a `FIREABILITY` pattern. She gave [0.7,0.3,0] to indicate that probably the patient should have been treated. However, as $P = 1$ the fact "treat subject" was not added to the database and the system halted.

Having the same numeric data as patient 2, also patient 3 was classified as borderline hypertensive. However, in this case `treat_predicate` was called with condition-pattern [True,True,False,False,True] as the subject's

[4]Columns 2 and 3 actually show data which relate to the same subject. The only difference is the presence of information about an hypertensive relative.

mother was hypertensive. For this reason the doctor gave a `FIREABILITY` pattern [0.8,0.2,0] as she was nearly sure that the patient should have been treated. However, with $P = 1$ no further inference was possible.

Patient 4 was classified as definitely hypertensive and `treat_predicate` was called again with condition-pattern [True,True,False,False,True]. As such a pattern was now in the collection, a `FIREABILITY` pattern [0.8,0.2,0] was retrieved but no further inference was possible.

Obviously, after the first experiment, running the system on the same data produced the same results. The difference was that no user interaction was then needed.

Reducing the prudence to $P = 0.75$ did not alter the inferences on patients 1 and 2. However, it enabled rule `select_treatment` to be checked for patients 3 and 4, as rule `check_whether_to_treat` was able to determine that they had to be treated. For patient 3 `drug_predicate` was called with only the condition "treat subject" satisfied and the doctor was asked to provide a `FIREABILITY` pattern. She gave [1,0,0,0,0,1] to mean: "definitely give a beta-adrenoceptor blocking drug and stop". For patient 4 also the condition about asthma was true and the variable `?degree` was instantiated to the value `severe`. As in this situation beta-adrenoceptor blocking drugs are contra-indicated, the user gave [0,0,0,1,0,1] to mean: "give diuretics and stop".

Setting the prudence $P = 0.5$ allowed to make additional inferences on patient 2 (no change happened for patient 1, 3 and 4 with respect to the previous case). In particular the system was able to establish that the subject had to be treated as the first element of the pattern [0.7,0.3,0] recalled in treat_predicate was now above P. The treatment suggested was the same as for patient 3. The decision to treat patient 2 was considered imprudent by the doctor.

3.2 Robot Motor Control

In order to show the advantages of using predicates based on artificial neural networks we now consider a motor control problem. The problem consists of designing a set of rules that allow a robot to react properly in the presence of obstacles.

The robot has engines and wheels that allow it to move in (at least) four main directions: forward, backward, left and right. It has eight proximity sensors that reveal the presence of obstacles at 0°, 45°, 90°, ... 315° (clockwise with respect to its current heading). As two or more sensors can detect the presence of obstacles for any given position (e.g. because the robot is in a corner or in a tight corridor), writing a set of rules for the four possible motor actions could be relatively difficult with a non-trainable rule-based system.

Instead, with our formalism, we need only the following rule

```
Rule obstacle_avoidance:
IF Obstacle at 0 degrees, Obstacle at 45 degrees,
   Obstacle at 90 degrees, Obstacle at 135 degrees,
   Obstacle at 180 degrees, Obstacle at 225 degrees,
   Obstacle at 270 degrees, Obstacle at 315 degrees
SATISFY obstacle_predicate
THEN SELECT_ACTIONS Go forward; Go backward; Go left; Go right.
```

where obstacle_predicate is a vector predicate implemented via a neural network plus a winner-takes-all output filter that prevents more than one True value being present in the action-pattern.

In this experiment we have allowed the ANN-based predicate to collect a training set including a total of 25 examples (out of the 256 possible condition-patterns). Eight condition-patterns had only one True value (only one sensor detecting a collision); 17 included two True values. Then, we trained the net and used it thereafter.

Figure 1 reports on some results obtained with this scenario. The lower part of the figure shows six different situations, labeled (a)–(f), in which the robot (represented as an octagon) can collide with obstacles. The table in the upper part of the figure reports the sensory conditions for each situation and the correspondingly selected action. The local conflict resolution implemented via the winner-takes-all filter prevents more than one action from being fired. The generalisation properties of the neural net provide the correct behaviour even in the presence of new condition-patterns containing two or more True values (situations (c)–(f)). A variant of this rule (with a different syntax which is functionally equivalent to the one presented in this paper) has been used as a behaviour sub-system for the subsumptive architecture of a simulated robot, as described in [13].

3.3 Inter-Agent Communication

The GA-based rule induction mechanism described in Section 2.2.4 has been (and it is currently) used, in more complicated experiments, for the development of communication between agents. In the following we describe the simplest of these experiments (the experiment has been implemented using SIM_AGENT [13]).

The experiment involves two agents: a blind agent and a lazy one. The blind agent can move in the world (the 2-D Cartesian plane) and can receive messages from the other agent, but is not capable of "visually" perceiving it. The lazy agent can perceive the (roughly quantised) relative position of the blind agent and can send messages, but cannot move. Both agents are implemented via a small set of rules the most important of which are:

```
Rule lazy_message_transmission:
IF Blind is north, Blind is south, Blind is east, Blind is west
SATISFY lazy_ga_predicate
THEN SELECT_ACTIONS Send word 1; Send word 2; Send word 3; Send word 4.

Rule blind_message_interpretation:
IF Received word 1, Received word 2,
   Received word 3, Received word 4
SATISFY blind_ga_predicate
THEN SELECT_ACTIONS Go east; Go west; Go north; Go south.
```

where `lazy_ga_predicate` and `blind_ga_predicate` are two vector predicates implemented via truth tables. The objective of the experiment is to

	Conditions								*Actions*			
Exp.	0°	45°	90°	135°	180°	225°	270°	315°	Forward	Backward	Left	Right
(a)	×									×		
(b)	×	×								×		
(c)	×	×	×								×	
(d)	×	×	×	×							×	
(e)	×	×	×	×	×						×	
(f)	×	×		×	×						×	

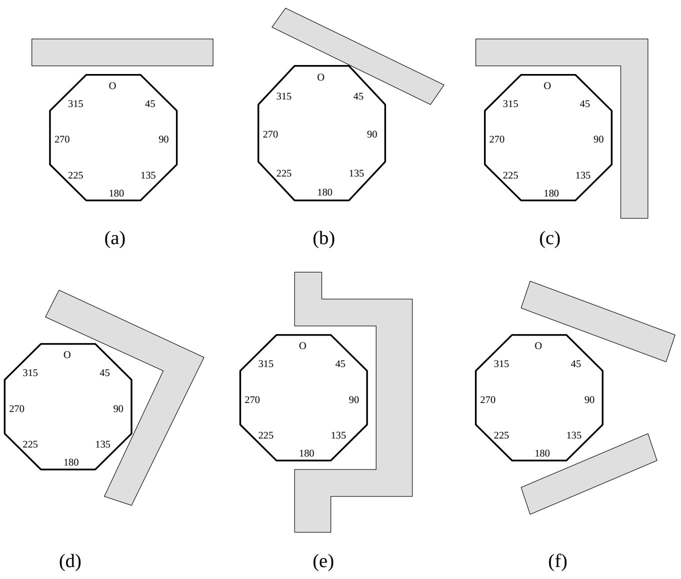

Figure 1: Experimental results for robot obstacle avoidance.

induce such predicates (by means of a genetic algorithm) so as to obtain cooperation (via communication) between the two agents that results in the blind agent moving towards and finally reaching the lazy one, for any possible initial mutual positions.

In the simulation message sending and position perception are obtained by a scheduler that repeatedly runs the rulebase of each agent and changes the database of each one so as to include new sensory data or new messages (clean-up rules remove the old data from each database).

In order to represent the truth tables of the aforementioned rules bit strings including 128 bits are needed. Each of such bit strings represents a possible rule-base. We associate to each rulebase a fitness which is the negative of the sum of the distances between the blind and the lazy agents measured at the end of four different runs of the simulation. In each run the lazy and the blind agents start in different relative locations (the furthest corners of a square).

The truth tables induced by running a GA with a population of 20 rule-sets (bit strings) for 50 iterations are shown in Tables 4 and 5.[5] The

[5] Some of the entries of these truth tables have been removed as not all the combinations of conditions can actually occur.

corresponding rules provide the two agents with the required behaviour, i.e. they meet each other within a minimum number of steps. An example of such a behaviour is shown in Figure 2. The figure is a graphical representation of a run (not included in the training set) in which the lazy agent (the circle marked with a dot) was at $(0.2, 0.3)$ and the blind one started from $(0.9, 0.7)$. Note how the blind agent follows the shortest route to the lazy agent as a result of the communication.

Conditions				*Actions*			
Blind is north	Blind is south	Blind is east	Blind is west	Send word 1	Send word 2	Send word 3	Send word 4
	×		×		×		
	×	×					×
×			×		×	×	×
×		×				×	

Table 4: Truth table representing `lazy_ga_predicate`.

Conditions				*Actions*			
Received word 1	Received word 2	Received word 3	Received word 4	Go east	Go west	Go north	Go south
			×		×	×	
		×			×		×
	×			×		×	
	×	×	×	×			×

Table 5: Truth table representing `blind_ga_predicate`.

4 Conclusions

In this paper we have presented HTR a hybrid system in which different control and learning regimes co-operate tightly to produce robust, flexible, trainable and efficient architectures. All this has been obtained thanks to a new formalism for rule representation that in addition to lending itself to rule induction and refinement is also very concise.

On the grounds of the experimental results, we believe that HTR is not "yet-another-rule-based system" but one that offers new powerful solutions for problems in which knowledge is uncertain, inconsistent, incomplete or variable.

Acknowledgements

The authors wish to thank Dr. Riccarda Del Bene (Department of Cardiovascular Medicine, Queen Elizabeth Hospital, The University of Birmingham) for her useful suggestions and help in developing the rulebase described in Section 3.1.

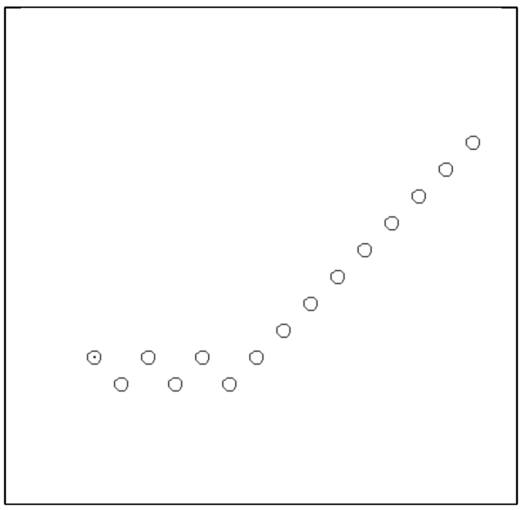

Figure 2: A run of the communication induction experiment.

References

[1] J.A.D.W. Anderson, editor. *POP-11 Comes of Age: The Advancement of an AI Programming Language*, Chichester, 1989. Ellis Horwood.

[2] C. L. Forgy. *OPS5 User's Manual.* Carnegie-Mellon University, Pitsburgh, PA, 1981. Tech. Report CMU-CS-81-135.

[3] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[4] T. M. Mitchell, R. M. Keller, and S. Kedar-Cabelli. Explanation-based generalization: a unifying view. *Machine Learning*, 1(1):47–80, 1986.

[5] E. Motta, M. Eisenstadt, M. West, K. Pitman, and R. Evertsz. Keats: The knowledge engineers assistant. *Expert Systems: The International Journal of Knowledge Engineering*, 1988.

[6] D. Ourston and R. J. Mooney. Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66:273–309, 1994.

[7] R. Poli, S. Cagnoni, R. Livi, G. Coppini, and G. Valli. A neural network expert system for diagnosing and treating hypertension. *IEEE Computer*, 24(3):64–71, 1991.

[8] Riccardo Poli and Mike Brayshaw. A hybrid trainable rule-based system. Technical Report CSRP-95-3, University of Birmingham, March 1995.

[9] C. Rich. Cake: An implemented hybrid knowledge representation and limited reasoning system. *SIGART Bulletin*, 2(3):120–127, 1991.

[10] J. F. Schreinmakers. *Pattern Recognition and Symbolic Approaches to Diagnosis*. Eburon:Delft, The Netherlands, 1991.

[11] L. Shastri. A connectionist approach to knowledge representation and limited inference. *Cognitive Science*, pages 331–392, 1988.

[12] A. Sloman. Poprulebase help file, 1995. Available at URL ftp://ftp.cs.bham.ac.uk/pub/dist/poplog/prb/help/poprulebase.

[13] Aaron Sloman and Riccardo Poli. SIM_AGENT: A toolkit for exploring agent designs. In *Intelligent Agents – Proceedings of the 1995 Workshop on Agent Theories, Architectures and Languages*. Springer-Verlag (LNAI Series), 1995.

[14] D. S. Touretzky and G. E. Hinton. A distributed connectionist production system. *Cognitive Science*, 1988.

[15] G. G. Towell and J. W.Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70:119–165, 1994.

[16] P. H. Winston. *Artificial Intelligence*. Addison-Wesley, third edition, 1992.