Updating a Hybrid Rule Base with Changes to its Symbolic Source Knowledge

Jim Prentzas[1, 2] **and Ioannis Hatzilygeroudis**[1, 2]

Abstract. Neurules are a kind of hybrid rules that combine a symbolic (production rules) and a connectionist (adaline unit) representation. One way that neurules (target knowledge) can be produced is by converting symbolic rules (source knowledge). However, source knowledge may change, so that updating corresponding target knowledge is necessary. Changes concern insertion of new and removal of old symbolic rules. In this paper, methods for updating target knowledge to follow changes made in corresponding source knowledge are presented. The methods are efficient in the sense that they do not require retraining of the whole affected part of the target knowledge, but of as small portion of it as possible.

1 INTRODUCTION

There has been extensive research activity at combining (or integrating) the symbolic and the connectionist approaches for knowledge representation in expert systems [7, 8, 10]. Especially, there are a number of efforts combining symbolic rules and neural networks [2, 3, 6, 9]. They give pre-eminence to connectionism and use a neural network as a knowledge base. The main objective is to reduce knowledge elicitation from experts to a minimum. In such approaches, connectionism is mainly used as a means for refining an initial background rule-base. Integration with symbolic representation is rather loose. A weak point of them is that their knowledge base lacks the naturalness and modularity of symbolic rules; it is incomprehensible. Therefore, explanations are often provided in the form of if-then rules by rule extraction methods [1].

Neurules [4] integrate symbolic rules and connectionism, but in a different way. They give pre-eminence to the symbolic component. Neurocomputing is used within the symbolic framework to improve the performance of symbolic rules. A symbolic rule-base, called the *source knowledge*, is converted into a hybrid one, a neurule-base, called the *target knowledge*. The target knowledge base is quite smaller, since in average each neurule is a merger of more than one symbolic rule, retains the modularity of production rules, since it consists of autonomous units (neurules), and also retains their naturalness in a great degree, since neurules look much like symbolic rules. Also, the inference mechanism is a tightly integrated process, which results in more efficient inferences than those of symbolic rules. Finally, explanations, in the form of if-then rules, can be produced [5].

Given that the source knowledge may change, due to updates, the problem of maintaining the target knowledge to reflect the changes, without performing extended re-conversion, arises. As far as we know, this problem has not been addressed by other hybrid approaches.

In this paper, we present methods for efficient maintenance of the knowledge base (target knowledge) of a neurule-based expert system, due to changes to its source knowledge. Section 2 presents neurules. Section 3 introduces the methods, presents some examples and discusses the correctness of the methods. Finally, Section 4 concludes.

2 NEURULES

2.1 Syntax and semantics

Neurules are a kind of hybrid rules. The form of a neurule is depicted in Fig.1a. Each condition C_i is assigned a number sf_i, called its *significance factor*. Moreover, each rule itself is assigned a number sf_0, called its *bias factor*. Internally, each neurule is considered as an adaline unit (Fig.1b). The *inputs* C_i (i=1,...,n) of the unit are the (values of the) *conditions* of the rule. The weights of the unit are the significance factors of the neurule and its bias is the bias factor of the neurule. Each input takes a value from the following set of discrete values: [1 (true), -1 (false), 0 (unknown)]. The *output D*, which represents the *conclusion* (decision) of the rule, is calculated via the standard formulas:

$$D = f(a),\ a = sf_0 + \sum_{i=1}^{n} sf_i C_i \qquad (1)$$

$$f(\mathbf{a}) = \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

where **a** is the *activation value* and *f(x)* the *activation function*, a threshold function. Hence, the output can take one of two values ('-1', '1') representing failure and success of the rule respectively. The general syntax of a condition C_i and the conclusion D is:

<condition>::= <variable> <l-predicate> <value>

<conclusion>::= <variable> <r-predicate> <value>

where <variable> denotes a *variable*, that is a symbol representing a concept in the domain, e.g. 'sex', 'pain' etc, in a

[1] University of Patras, Dept of Computer Engineering & Informatics, 26500 Patras, Hellas (Greece), email: {prentzas, ihatz}@ceid.upatras.gr.

[2] Computer Technology Institute, P.O. Box 1122, 26110 Patras, Hellas (Greece).

medical domain. <l-predicate> and <r-predicate> are one of {is, isnot}. <value> denotes a value. It can be a *symbol* or a *number*.

Corresponding symbolic rules have the same syntax as that in Fig.1a, without the significant factors and where ',' denotes conjunction.

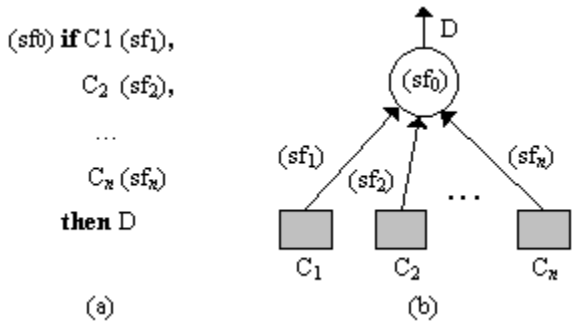

Figure 1. (a) Form of a neurule (b) a neurule as an adaline unit

2.2 Construction of a neurule-base

Construction of a neurule-base (NRB) from a symbolic rule-base (SRB) is made by the basic conversion algorithm (introduced in [4]), which is outlined here: (a) A symbolic rule with a unique conclusion is individually converted into a neurule using as training set the valid patterns of the truth table of its logical function (an AND function of their conditions). (b) Symbolic rules with the same conclusion constitute a *merger set*. From each merger set, a merger is produced, which is a neurule mould having as conditions all the conditions of the symbolic rules in the merger set (without duplications) and zero significant factors. Then, the training set for each merger is extracted from the truth table of the combined logical function of the rules in its merger set (an OR function of their AND functions), after a filtering process, during which some patterns of the truth table are eliminated (for a detailed treatment see [4]).

Table 1. An example merger set of symbolic rules

R1:**if** arterial-conc is slight-incr, blood-conc is normal, scan-conc is normal, capill-conc is mod-incr, venous-conc is highly-incr **then** disease is early-inflam	R2:**if** arterial-conc is mod-incr, blood-conc is highly-incr, scan-conc is normal, capill-conc is slight-incr, venous-conc is slight-incr, **then** disease is early-inflam
R3:**if** arterial-conc is mod-incr, blood-conc is normal, scan-conc is normal, capill-conc is slight-incr, venous-conc is normal **then** disease is early-inflam	R4:**if** arterial-conc is mod-incr, blood-conc is mod-incr, scan-conc is normal, capill-conc is mod-incr, venous-conc is slight-incr **then** disease is early-inflam
R5:**if** arterial-conc is mod-incr, blood-conc is normal, scan-conc is normal, capill-conc is mod-incr, venous-conc is slight-incr **then** disease is early-inflam	R6:**if** arterial-conc is mod-incr, blood-conc is slight-incr, scan-conc is normal, capill-conc is mod-incr, venous-conc is mod-incr **then** disease isearly-inflam

Training is performed using the standard LMS algorithm. When the algorithm succeeds, that is values for the significant factors are calculated that successfully classify all training examples, a neurule is produced. When it fails (case of inseparable training examples) a splitting process is followed, which produces more than one neurule having the same conclusions, called *sibling neurules*. To this end, the *closeness* between two rules is defined as the number of their common conditions. A *least closeness pair* (*LCP*) of rules in a merger set is a pair of rules with the *least closeness* (*LC*) in the set.

The process is the following:

1. Train the merger using the specified training set.
2. If training fails, find a LCP and produce two subsets of the merger set, each having as initial element (a different) one of the LCP rules, called its *pivot*. In each merger subset put the symbolic rules (of the initial merger set), which are closer to its pivot.
3. For each merger subset, apply step 1 recursively until either training succeeds or the merger subset contains only one rule (remaining rule)
4. Convert any remaining rule to a neurule.

Table 2. Neurules produced from the merger set of Table 1

NR1: (-11.4) **if** venous-conc is high-incr (3.3), arterial-conc is slight-incr (3.0), blood-conc is normal (2.8), scan-conc is normal (2.7), capill-conc is mod-incr (2.7) **then** disease is early-inflam	NR2: (-11.4) **if** venous-conc is slight-incr (3.3) arterial-conc is mod-incr (3.0), blood-conc is high-incr (2.8), scan-conc is normal (2.7), capill-conc is slight-incr (2.7) **then** disease is early-inflam
NR3: (-11.4) **if** venous-conc is normal (3.3), arterial-conc is mod-incr (3.0), blood-conc is normal (2.8), scan-conc is normal (2.7), capill-conc is slight-incr (2.7) **then** disease is early-inflam	NR4-5: (-7.8) **if** venous-conc is slight-incr (3.3) arterial-conc is mod-incr (3.0), blood-conc is mod-incr (2.8), scan-conc is normal (2.7), capill-conc is mod-incr (2.7), blood-conc is normal (2.6), **then** disease is early-inflam
NR6: (-11.4) **if** venous-conc is mod-incr (3.3), arterial-conc is mod-incr (3.0), blood-conc is slight-incr (2.8), scan-conc is normal (2.7), capill-conc is mod-incr (2.7) **then** disease is early-inflam	

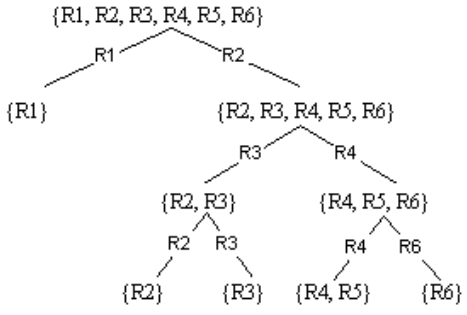

Figure 2. The splitting tree for the merger set of Table 1

For reasons that will become clear in the next section, for each initial merger set the splitting process is stored as a tree, which is called the *splitting tree*. The root of the tree corresponds to the initial merger set. The intermediate nodes and leaves correspond to the subsequent subsets into which the initial merger set was split. An intermediate node denotes a subset that was split, due to training failure, whereas a leave denotes a subset that was successfully trained and produced a neurule. The pivot of each (sub)set is attached to the corresponding edge of the tree. It can be easily seen that the merger (sub)set of the root or an intermediate node is a superset of the merger subsets related to its descendant nodes. Furthermore, the nearer one gets to the leaves, the greater the mean closeness between the rules of the corresponding merger subsets. Tree information is stored in the neurule-base alongside the produced neurules.

To illustrate how splitting is performed, we use the merger set of rules R1-R6 presented in Table 1, from which five neurules are produced, shown in Table 2. Figure 2 depicts the corresponding splitting tree. Due to inseparability, the initial merger set {R1, R2, R3, R4, R5, R6} is split in two subsets: {R1} and {R2, R3, R4, R5, R6}, with LCP: (R1, R2) (LC=1). {R1} produces neurule NR1. {R2, R3, R4, R5, R6} is split in {R2, R3} and {R4, R5, R6}, with LCP: (R3, R4) (LC=2). Then, {R2, R3} is split in {R2} and {R3} and NR2, NR3 are produced and so on.

3 UPDATE METHODS

The source knowledge (SRB), however, may change. This implies that the target knowledge (NRB) should be updated. The possible changes are due to (a) insertion of a new rule and (b) removal of an existing rule. Modification of a rule is equivalent to removal of the old rule and insertion of a new rule.

3.1 Rule insertion

Given the insertion of a new symbolic rule R in SRB, three cases can be distinguished:

(a) There is no sibling neurule of R in NRB.
(b) There is only one sibling neurule of R in NRB.
(c) There are more than one sibling neurule of R in NRB.

Case (a) is the simplest. The new symbolic rule is converted into a neurule and inserted into NRB.

If (b) is the case, it means that the corresponding merger set was not split. To handle this, the existing neurule is removed from NRB and a new merger set is formed containing the new symbolic rule and the symbolic rules of the initial merger set. The new merger is formed and trained. If training is successful, one neurule is produced. If training fails, two neurules are produced.

Case (c) is the most difficult. The existence of more than one neurule with the same conclusion means that, due to inseparability, the initial merger set was split into a number of merger subsets. There can be three approaches:

(i) Merely convert the new symbolic rule into a neurule and insert it into NRB. This method is computationally efficient but increases the number of neurules in NRB, which is not desirable.

(ii) The existing neurules are removed from NRB, the new symbolic rule is merged with the initial merger set and training of the new merger is performed to produce the new neurules. This approach is inefficient, especially when more than two neurules are produced from the initial merger set. The reason is that it discards the information contained in the splitting tree, thus performing extra training and splitting.

(iii) The third approach is as follows:

Starting from the root, traverse the splitting tree:

1. If the current node is not a leaf, check whether the merger (sub)set corresponding to the node contains a rule R' whose closeness to R is less than the LC of the (sub)set. If there is no such rule, insert R into the merger (sub)set of the node and execute this step recursively for the child of the node denoted by the LCP member closer to R. If there is such a rule R', do:
   1.1 Stop traversing the splitting tree.
   1.2 Remove from NRB all neurules corresponding to the leaves descending from this node.
   1.3 Insert R into the corresponding merger (sub)set and split it in two subsets with LCP: (R, R').
   1.4 Train each one of the mergers formed from the two subsets, produce corresponding neurules (reusing parts of the initial splitting tree to avoid unnecessary training or splitting), insert the produced neurules into NRB and update the splitting tree.
2. If the current node is a leaf, remove the corresponding neurule, insert R into its merger set and train the merger. If training fails, split the merger set, produce the two neurules, insert them into NRB and update the splitting tree. If training is successful, do the following:
   2.1 If the sibling node of the leaf is also a leaf and introduction of R into their parent's merger subset increases the mean closeness between its rules, and the parent node's new merger can be successfully trained, do the following:
      2.1.1 Remove the neurule corresponding to the sibling leaf from NRB.
      2.1.2 Insert the neurule produced from the parent node's new merger into NRB.
      2.1.3 Update the splitting tree.
   2.2 Else do the following:
      2.2.1 Insert the neurule produced from the leaf's new merger into NRB.
      2.2.2 Update the splitting tree.

Table 3. Inserted rule R7 and resulted neurule NR2-3-7

R7: **if** arterial-conc is mod-incr, blood-conc is normal, scan-conc is normal, capill-conc is slight-incr, venous-conc is slight-incr **then** disease is early-inflam	NR2-3-7: (-20.4) **if** venous-conc is slight-incr (8.7), arterial-conc is mod-incr (8.4), scan-conc is normal (8.1), capill-conc is slight-incr (8.1), blood-conc is normal (8.0), blood-conc is highly-incr (4.6) venous-conc is normal (1.5), **then** disease is early-inflam

As an example, consider the rules in Table 1 as SRB and those in Table 2 as NRB. Suppose that rule R7 (Table 3) is to be inserted. Given that more than one neurule have the same

conclusion with R7, it is a (c) case. Following approach (iii), traversing ends at the leaf related to subset {R2} (Fig. 3). Training of the new merger (sub)set {R2, R7} is successful. Thus NR2 is removed from NRB. The sibling node of the {R2} leaf is also a leaf and insertion of R7 into the subset {R2, R3}, related to their 'parent' node, increases its mean closeness from 3 ({R2, R3}) to 11/3 ({R2, R3, R7}). Therefore, training of the merger of {R2, R3, R7} is tried. It is successful and NR2-3-7 is produced and inserted into NRB. Meanwhile, NR3 is removed from NRB. The splitting tree takes the form in Fig. 4. So, insertion of R7 finally decreases the total number of neurules in NRB from five to four (NR1, NR2-3-7, NR4-5, NR6).

Notice that approach (iii) focuses on subset {R2, R3}, which includes the closest rules to R7. Thus, only the necessary part of NRB was (re)trained. The part of NRB produced from R1, R4, R5 and R6 remains intact. Approach (ii) would have produced the same neurules, requiring though unnecessary training and splitting. Approach (i) would have inserted the neurule produced from R7 into NRB. At the end, NRB would contain six neurules instead of the four resulted by approach (iii).

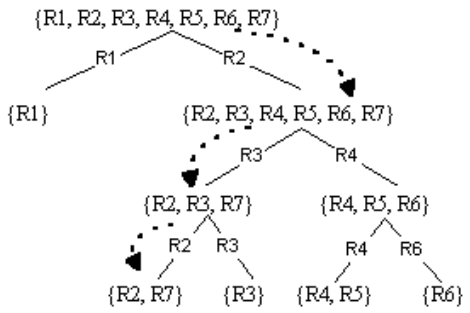

Figure 3. Traversal of the splitting tree to insert R7

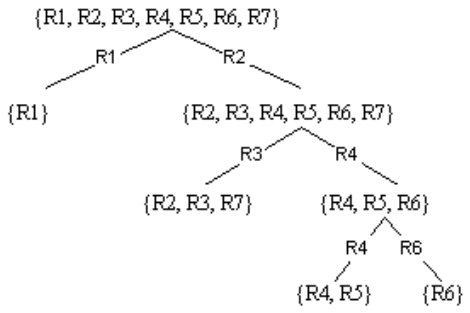

Figure 4. The splitting tree after insertion of R7

3.2 Rule removal

Given the removal of a symbolic rule R from SRB, two cases can be distinguished:

(a) There is only one sibling neurule of R in NRB.

(b) There are more than one sibling neurule of R in NRB.

If (a) is the case, it means that corresponding merger set was not split. So, the corresponding neurule is removed from NRB. If the merger set includes only the removed rule, nothing else is done. Otherwise, a new merger set is formed, including the rules of the initial merger set, but excluding the removed rule. The new merger is trained. If training is successful, one neurule will be produced. If it fails, the merger set is split and, after training, two neurules are produced.

There can be three approaches to handle case (b):

(i) In this approach, only the neurule whose merger set included the removed rule is affected. The neurule is removed from NRB. If the merger set included only the rule that is removed, nothing else is done. Otherwise, the new merger (formed as in (a)) is trained and corresponding neurule(s) is(are) produced. This method is computationally efficient, but may increase the number of sibling neurules.

(ii) All the neurules derived from the initial merger set are removed from NRB and the merger of the initial merger set, after excluding the removed rule, is trained. After possible splitting, the corresponding neurule(s) is(are) produced. This approach is inefficient, since it discards the information contained in the splitting tree and performs training and splitting that could have been avoided.

(iii) The third approach is as follows:

Starting from the root, traverse the splitting tree:

1. If the current node is not a leaf, check whether R is a member of the LCP of its merger (sub)set. If it isn't, remove R from the merger (sub)set and execute this step recursively for the child of the node on the edge denoted by the LCP's member closer to R. If it is, do:
   - 1.1 Stop traversing the splitting tree.
   - 1.2 Remove the neurules produced from this merger (sub)set.
   - 1.3 Remove R from the node's merger (sub)set and train the resulted merger (possibly reusing parts of initial splitting tree).
   - 1.4 Insert the produced neurules into NRB and update the splitting tree.
2. If the current node is a leaf, do:
   - 2.1 Remove the neurule whose merger set included R from NRB.
   - 2.2 Remove R from the leaf's merger set.
   - 2.3 If R was the only member of the merger set, do:
     - 2.3.1. If the sibling node of the leaf node is also a leaf and the sibling node of their parent node is a leaf, check the parent node F of the leaves' parent node.
     - 2.3.2. If removal of R from the merger set of F increases its mean closeness and F's new merger can be successfully trained, do:
       - 2.3.2.1. Remove the neurules corresponding to the leaves descending from F.
       - 2.3.2.2. Insert the neurule corresponding to the new merger set of F into NRB.
       - 2.3.2.3. Update the splitting tree.
     - 2.3.3. Else update the splitting tree.
   - 2.4 If not, form the new merger and train it.
   - 2.5 If training fails, produce the neurules (after splitting), insert them into NRB and update the splitting tree.
   - 2.6 If training succeeds, do:
     - 2.6.1 If the sibling of the leaf node is also a leaf and removal of R from the merger set of their parent increases its mean closeness and the parent's new merger can be successfully trained, do:
       - 2.6.1.1. Remove the neurule corresponding to the sibling leaf from NRB.

2.6.1.2. Insert the neurule produced from the parent's new merger into NRB.
2.6.1.3. Update the splitting tree.
2.6.2 Else do:
2.6.2.1. Insert the neurule produced from the leaf's new merger into NRB.
2.6.2.2. Update the splitting tree.

As a first example, suppose that R5 is to be removed (after R7 insertion, Fig.4). It's a (b) case and following approach (iii), since R5 is not a member of any LCP, NR4-5 is removed from NRB, the merger of {R4} is trained and NR4 is produced. The sibling node of leaf {R4} is also a leaf ({R6}). Removal of R5 decreases the mean closeness of their parent node's subset from 10/3 ({R4, R5, R6}) to 3 ({R4, R6}). Therefore, no training of the merger of {R4, R6} is tried. NR4 is inserted into NRB and the splitting tree is updated (Fig. 5). Once again a large portion of NRB remains intact. The new NRB consists of NR1, NR2-3-7, NR4, NR6.

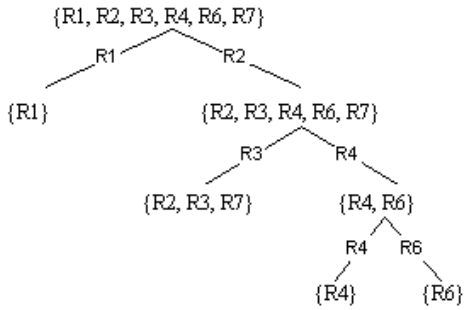

Figure 5. Splitting tree after removing R5

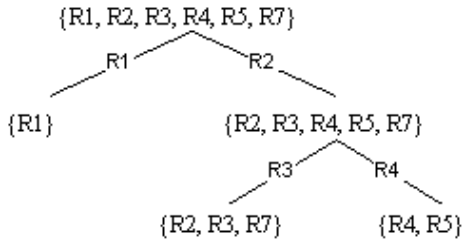

Figure 6. The splitting tree after R6 removal

As a second example, suppose that R6 is to be removed (after R7 insertion). R6 is not member of any LCP as far as node {R4, R5, R6}, so R6 is removed from the subsets of its ancestor nodes. NR4-5 and NR6 are removed from NRB and R6 also from {R4, R5, R6}. NR4-5 is inserted in NRB and the splitting tree is updated (Fig. 6). Notice that NR4-5 needs no reproduction, is reused.

3.3 Discussing correctness

Given that the above algorithms concern indirect changes to target knowledge, the question about their correctness is reasonably set. The basis of the algorithms is the basic conversion algorithm (introduced in [4] and outlined in Section 2.2). The criterion for the correctness of that algorithm is to preserve inference equivalence between SRB and NRB, which means that all inferences performed in SRB should also be performed in NRB (with identical results). This is implicitly proved, and also experimentally confirmed, in [4]. The proof is based on the fact that the training set of each merger includes the (valid) rows of the truth table of the combined logical function of the rules in the corresponding merger set. Given that training is complete, that is all training patterns are successfully classified, inference equivalence is assured.

Inserting a new rule in SRB means that a new inference is introduced. So, the rule insertion update algorithms of Section 3.1, to be correct, should assure that the new inference can also be performed in the updated NRB. This is rather obvious, since in any case the truth table of the new inserted rule is taken into account in creating the new training set for the partial reconversion. A similar thing happens when a rule is removed from SRB. The patterns related to its truth table are removed from the training set of the corresponding merger. So, correctness of the rule removal update algorithms of Section 3.2 is also assured.

4 CONCLUSIONS

In this paper, we present methods for efficiently updating target knowledge (a hybrid rule base) to reflect changes in its source knowledge (a symbolic rule base). Target knowledge consists of neurules, a type of hybrid rules. Neurules are produced from symbolic rules via a conversion mechanism. Methods for updating target knowledge in the cases of a rule insertion in and a rule removal from source knowledge are introduced. Efficiency refers to updating target knowledge (a) with as little reconversion as possible and (b) preserving the number of neurules as small as possible. We achieve that by storing information related to the conversion process in a tree, called splitting tree. The methods are also argued to be correct.

REFERENCES

[1] R. Andrews, J. Diederich and A. Tickle, 'A survey and critique for extracting rules from trained ANN', *Knowledge-Based Systems*, **8(6)**, 373-389 (1995).

[2] L-M Fu and L-C Fu, 'Mapping rule-based systems into neural architecture', *Knowledge-Based Systems* **3**, 48-56 (1990).

[3] L-M Fu, 'A Connectionist Approach to Rule Refinement', *Applied Intelligence*, **2**, 93-103 (1992).

[4] I. Hatzilygeroudis and J. Prentzas, 'Neurules: Improving the Performance of Symbolic Rules', *International Journal on AI Tools* **9(1)**, 113-130, (2000).

[5] I. Hatzilygeroudis and J. Prentzas, 'An Efficient Hybrid Rule Based Inference Engine with Explanation Capability', Proceedings of the 14th International FLAIRS Conference, Key West, FL, 227-231 (2001).

[6] J.J. Mahoney, 'Combining Symbolic and Connectionist Learning Methods to Refine Certainty-Factor Rule-Bases', PhD Dissertation, University of Texas at Austin, 1996.

[7] K. McGarry, S. Wertmer, and J. MacIntyre, 'Hybrid neural systems: from simple coupling to fully integrated neural networks', *Neural Computing Surveys*, **2**, 62-93, (1999).

[8] R. Sun and E. Alexandre (eds), *Connectionist-Symbolic Integration: From Unified to Hybrid Approaches*, Lawrence Erlbaum, 1997.

[9] G. Towell and J. Shavlik, 'Knowledge-Based Artificial Neural Networks', *Artificial Intelligence*, **70(1-2)**, 119-165, (1994).

[10] S. Wermter and R. Sun (eds), *Hybrid Neural Systems*. Springer-Verlag, Heidelberg, 2000.