

First European Congress  
on Fuzzy and  
Intelligent Technologies  
Aachen, Germany,  
September 7–10, 1993  
Proceedings  
Volume 2

***EUFIT***  
**1993**

## Speeding Up Genetic Machine Learning – A Case for Fuzzy Rule Languages

Andreas Geyer-Schulz<sup>1</sup>

Department of Business Administration / Management Information Systems  
University of Augsburg, Memmingerstraße 18, D-8900 Augsburg, Germany

**Abstract:** Fuzzy classifier systems are genetic based machine learning systems which integrate a fuzzy rule base, a genetic algorithm and an apportionment of credit function. In this paper we show, that the computational complexity of classifier systems is determined by the syntactic structure of the formal language  $L$  of the production system. In particular, we are interested in comparing the computational complexity of fuzzy classifier systems with their corresponding crisp versions. We show that fuzzy classifier systems ideally provide an infinite speed-up when compared with crisp classifier systems. Even computer implementations which approximate fuzzy classifier systems learn considerably faster than crisp classifier systems.

### I. Introduction

In his 1936 abstract "Über die Länge von Beweisen" Kurt Gödel considered for the first time the question of comparing the length of proofs of formulas provable in a lower and a higher order formal system. He presented a speed-up theorem which states that migration from a formal system to a higher order one reduces the length of proofs albeit without giving a proof [Gödel, 1936; Feferman *et al.*, 1986]. In [Parikh, 1973] we find an analogue of Gödel's theorem for a formulation of Peano arithmetic with full induction and addition and multiplication being ternary relations as lower order system and, say analysis, as higher order system. [Statman, 1978] contains a study of the speed-up phenomenon for several classical formal systems.

The term "speed-up" is due to Blum [Blum, 1967] who discovered in 1967 the famous speed-up theorem of recursive function theory. Blum's speed-up theorem proves the existence of a total computable function which has no best program [Young, 1973; Cutland, 1980].

Salomaa proved a speed-up theorem for time-bounded grammars [Salomaa, 1973, p. 304]. The basic idea of Salomaa's speed-up theorem is to construct from a grammar  $G_1$  an equivalent grammar  $G_2$  by combining  $m$  derivation steps in  $G_1$  into a single step in  $G_2$ . For example,  $G_1 = \{S \rightarrow aS \mid a\}$  is transformed to  $G_2 = \{S \rightarrow aS \mid aaS \mid a \mid aa\}$ .

Our motivation is the following: We can achieve a substantial reduction in learning complexity by choosing very high-level production languages. It seems that for tasks like formal program verification, automated program testing, automatic programming and automated reasoning whose performance and – since performance is the limiting factor – feasibility is predominantly influenced by search space size, abstraction in the sense of moving to higher order formal systems is the only handle we have got to attack the problem. In the rest of this paper we show that fuzzy production languages constitute such higher order formal systems when compared with crisp production languages. We start, however, with a complete general setting.

### II. Comparing Two Search Spaces

How do we in general compare the learning complexity of two context-free languages which are related by a translation from one language into the other?

In order to answer this question, we assume that, in order to avoid counting problems, the grammars  $G_1$  and  $G_2$  of the languages  $L_1$  and  $L_2$  are  $\epsilon$ -free, do not contain useless symbols and are in the same normal form [Aho and Ullman, 1972] or have nearly identical structure (as in our examples). Ambiguity of the grammar is treated by viewing a word  $w$  of  $L(G)$  always as a pair consisting of a string from  $V_T^*$  and its leftmost derivation tree.

By considering the depth-bounded version of  $L(G)$  we can characterize the learning complexity by a formal power series  $A(x) = \sum_{d \geq 0} \text{card}(S_{L,d}) \cdot x^d$  whose coefficients  $\text{card}(S_{L,d})$  correspond to the number of

<sup>1</sup> On leave: Department of Applied Computer Science, Institute of Information Processing and Information Economics, Vienna University of Economics and Business Administration, Augasse 2-6, A-1090 Vienna, Austria. Phone: +43-1-3303636-31. EMAIL: geyers@wu-wien.ac.at

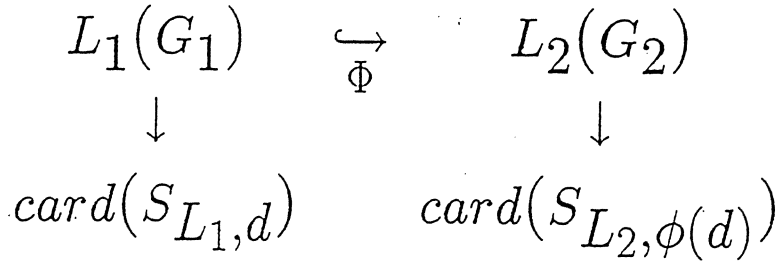


Figure 1: Comparing two search spaces

words which can be generated with at most  $d$  derivations. Moreover, from the productions of the depth-bounded grammar of  $L(G)$  we can automatically derive a definition of the formal power series  $A(x)$  in the form of a set of recurrence relations by a signature mapping. In figure 1 this kind of mapping is shown twice, for  $L_1(G_1)$  and  $L_2(G_2)$ . The arrows  $L_1(G_1) \rightarrow \text{card}(S_{L_1,d})$  and  $L_2(G_2) \rightarrow \text{card}(S_{L_2,\phi(d)})$  are of this type. In doing so, we exploit a natural interconnection between formal languages and formal power series [Goulden and Jackson, 1983; Kuich and Salomaa, 1986].

The basic idea which allows us a comparison of the two series is the following: We use the embedding  $\Phi : L_1(G_1) \hookrightarrow L_2(G_2)$  in order to construct a function  $\phi : N \rightarrow N_0$  from the depth-bound  $d$  on the grammar  $G_1$  to the depth bound  $\phi(d)$  on the grammar  $G_2$ . The *depth-bound function*  $\phi(d)$  ensures, that for each depth-bound  $d$  of  $L_1(G_1)$ ,  $\phi(d)$  is the smallest depth-bound on  $L_2(G_2)$  such that the translations  $\Phi(w)$  of all words  $w$  of  $S_{L_1,d}$  are contained in  $S_{L_2,\phi(d)}$ . We call  $\Phi$  a *bounding relation*, because  $\Phi$  induces the depth-bound function  $\phi(d)$  which yields for each coefficient of the first series the index of the coefficient of the second series with which it should be compared. Figure 1 summarizes the whole idea.

What remains to be done is to establish that  $\text{card}(S_{L_1,d}) < \text{card}(S_{L_2,\phi(d)})$  almost everywhere, which might be quite a cumbersome task. Fortunately, for most practical applications calculation of the coefficients of the two formal power series up to some (small)  $n$  suffices, because in practice (at least in our examples) the largest derivation depth used is quite small.

### III. A Meta-Theorem

In order to prove theorems of the following kind

**Theorem 1** *Given two context-free languages,  $L_1(G_1)$  and  $L_2(G_2)$ , and an effective (computable) embedding  $\Phi : L_1(G_1) \hookrightarrow L_2(G_2)$  (a translation from  $L_1(G_1)$  to  $L_2(G_2)$ ),  $\text{card}(S_{L_1,d}) < \text{card}(S_{L_2,\phi(d)})$  holds, for almost all  $d \in N$ .*

we outline the proof steps in the next section.

### IV. A Proof Skeleton

1. The arrows,  $L_1(G_1) \rightarrow \text{card}(S_{L_1,d})$  and  $L_2(G_2) \rightarrow \text{card}(S_{L_2,\phi(d)})$  respectively, denote recursive word counting functions (WCF) which count the number of words in  $S_{L_1,d}$  and  $S_{L_2,\phi(d)}$ . For  $L(G)$  the recursive word counting function  $\Pi$  can be automatically derived by a signature mapping  $F_1 : \sum_{BNF} \rightarrow \sum_{WCF}$  from the productions  $P$  of  $L(G)$  which are usually given in Backus-Naur Form (BNF).

$$\sum_{BNF} = \langle V_T, V_N, :=, |, \star \rangle \quad (1)$$

is the signature of the BNF language, with  $V_T$  denoting the terminal alphabet,  $V_N$  the nonterminal alphabet,  $:=$  denoting *derives to*,  $|$  denoting *or* and  $\star$  denoting *catenation* (usually catenation is denoted by juxtaposition of symbols). Since  $y \in V_N$  may appear on both sides of a production,  $y_{LHS}$  indicates its appearance on the left hand side and  $y_{RHS}$  its appearance on the right hand side of a production. The signature of the recursive word counting function language is

$$\sum_{WCF} = \langle \Pi(x, d), \Pi(y, d), =, +, \circ \rangle \quad (2)$$

$\Pi(x, d)$  is the set of all invocations and definitions of the recursive word counting function  $\Pi$  with  $x \in V_T$  and  $d \in N_0$  as argument.  $\Pi(y, d)$  is the set of all invocations and definitions of the recursive word counting function  $\Pi$  with  $y \in V_N$  and  $d \in N$  as argument.  $=$  denotes *is defined by* and  $+$  denotes addition.  $\circ$  is a kind of a ternary function composition operation.  $\circ(\Pi(y_1, r_1), \Pi(y_2, r_2))$  with  $r_1 + r_2$  calculates the number of words derivable from the symbol string  $y_1 y_2$  in  $d$  derivations.

The recursive word counting function scheme  $\Pi : (V_T \times N_0) \cup (V_N \times N) \rightarrow N_0$  is now derived by the signature mapping  $F_1 : \sum_{BNF} \rightarrow \sum_{WCF}$ :

(a) For all elements of  $V_T$ ,  $F_1$  is defined by

$$\forall x \in V_T : x \rightarrow \Pi(x, d - 1).$$

We add for all  $x \in V_T$  a clause of the following kind to the recursive word function scheme  $\Pi$ :

$$\Pi(x, d) = 1 \cdot (d = 0)$$

For a terminal symbol  $x$ ,  $\Pi(x, d)$  is 1, if  $d = 0$  and 0 otherwise.

(b) For all elements of  $V_N$ ,  $F_1$  is given by:

$$\forall y_{LHS} \in V_N : y_{LHS} \rightarrow \Pi(y_{LHS}, d)$$

$$\forall y_{RHS} \in V_N : y_{RHS} \rightarrow \Pi(y_{RHS}, r_i)$$

For a nonterminal symbol  $y$  and  $d < 1$ ,  $\Pi$  is undefined. Whenever  $\Pi$  is undefined, it takes the value 0. For  $r_i$  see 3.

(c)

$$:= \rightarrow =$$

(d)

$$| \rightarrow +$$

(e)

$$\star \rightarrow \circ$$

In the BNF notation catenation  $x_1 \star x_2$  is usually formed by writing the symbols immediately together:  $x_1 x_2$ . Each  $k$ -symbol string  $y^k$  is therefore replaced by the following expression:

$$\sum_{i=1}^k \prod_{r_i=d-1, r_i \geq 0, d > 0} \Pi(y_i, r_i) \quad (3)$$

Expression 3 seems to 'come out of the blue'. However, here is an explanation: Let us return to the two symbol string  $x_1 x_2$  with  $c$  derivations available. How many different words can we derive in  $c$  derivations from the start string  $x_1 x_2$ ? The answer is found after having analysed the following four cases:

- i. Both symbols  $x_1$  and  $x_2$  are nonterminal symbols. In order to derive a string of terminal symbols we have to assign to  $x_1$ , say  $r_1$  derivations. For this partition of  $c$  into 2 parts (2-partition) the number of words derivable is the product of the number of words derivable from  $x_1$  in  $r_1$  derivations with the number of words derivable from  $x_2$  in  $r_2$  derivations. To find all words, we have to sum over all 2-partitions. This means, all partitions generated by pairs  $r_1, r_2$  such that  $r_1 + r_2 = d$  and  $r_1, r_2 \neq r_2, r_1$  hold:

$$\circ(\Pi(x_1, c), \Pi(x_2, c), c) = \sum_{r_1+r_2=c, r_1 \geq 0, r_2 \geq 0, c \geq 0} \Pi(x_1, r_1) \cdot \Pi(x_2, r_2) \quad (4)$$

If  $r_i = 0$ ,  $\Pi(x_i, r_i)$  is undefined, because we cannot derive a string of terminal symbols and this implies that  $\Pi(x_i, r_i) = 0$ .

- ii.  $x_1$  is a terminal symbol and  $x_2$  a nonterminal symbol. With expression 4 we obtain again our result. Convince yourself, that this is the case, because  $\Pi(x_1, r_1)$  is 0 for all values of  $r_1$  except for  $r_1 = 0$ .

- iii.  $x_1$  is a nonterminal symbol and  $x_2$  a terminal symbol. This is similar to the above case.
- iv. Both symbols  $x_1$  and  $x_2$  are terminal symbols. Expression 4 is only defined, if  $c$ ,  $r_1$  and  $r_2$  are 0. In this case, its result is 1.

In order to obtain expression 3, we only have to generalize expression 4 to the  $n$ -partition case, taking terminal symbols into account. Obviously, by doing this we have generalized the 2 + 1-ary operation  $\circ$  to a  $n + 1$ -ary operation.

Finally, we can characterize the search space size  $card(S_{L,d})$  of  $L$  for all  $d \in N$  with the power series  $A(x)$ , whose coefficients are given by

$$card(S_{L,d}) = \sum_{i=1}^d \Pi(\langle startsymbol \rangle, i) .$$

The series is monotone increasing with coefficients in  $N_0$ .

2. We define the depth-bound function  $\phi(d)$  in such a way that the image  $\Phi(w)$  of each word  $w$  which can be derived with at most  $d$  derivations in  $L_1$  is derivable in at most  $\phi(d)$  derivations in  $L_2$  and that  $\phi(d)$  is the smallest number for which this holds. For convenience we split the construction of  $\phi(d)$  in two steps. In the first step we construct a recursive depth-counting function for  $L_1$ , in the second step we modify this function in order to take  $\Phi$  into account. In detail:

- (a) We construct the partial depth-counting function  $\varphi : (V_T \times N_0) \cup (V_N \times N) \rightarrow N_0$  from the production rules of  $G_1$  of  $L_1$  by the signature mapping  $F_2 : \sum_{BNF} \rightarrow \sum_{DCF}$ . For computer implementations we complete the partial function by adjoining *undefined*. This mapping is completely general.

The signature of the recursive depth counting function language is

$$\sum_{DCF} = \langle \varphi(x, d), \varphi(y, d), = s(w), \max, \bullet \rangle \tag{5}$$

$\varphi(x, d)$  is the set of all invocations and definitions of the recursive depth counting function  $\varphi$  with  $x \in V_T$  and  $d \in N_0$  as argument.  $\varphi(y, d)$  is the set of all invocations and definitions of the recursive depth counting function  $\varphi$  with  $y \in V_N$  and  $d \in N$  as argument.  $= s(w)$  denotes *is defined by 1 plus "the rest"*.  $\max$  denotes the maximum of two numbers and  $\bullet$  is a kind of a ternary function composition operation which returns the number of derivations on the longest derivation path.

$F_2 : \sum_{BNF} \rightarrow \sum_{DCF}$  is now given by:

- i. For all elements of  $V_T$ ,  $F_2$  is defined by:

$$\forall x \in V_T : x \rightarrow \varphi(x, d - 1)$$

We add for all  $x \in V_T$  a clause of the following kind to the recursive depth counting function  $\varphi$ :

$$\varphi(x, d) = \begin{cases} 0 & \text{if } d = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- ii. For all elements of  $V_N$ ,  $F_2$  is given by:

$$\forall y_{LHS} \in V_N : y_{LHS} \rightarrow \varphi(y_{LHS}, d)$$

$$\forall y_{RHS} \in V_N : y_{RHS} \rightarrow \varphi(y_{RHS}, r_i)$$

- iii. For a nonterminal symbol  $y$  with  $d < 1$ ,  $\varphi$  is undefined. For  $r_i$  see 6.

$$:= \rightarrow = s(z)$$

where  $z$  is the whole right hand side of the production.  $s(n)$  is the successor function which is defined by  $s(n) = n + 1$ ,  $s(0) = 1$  and  $s(\text{undefined}) = \text{undefined}$ .

iv.

$|\rightarrow \max$

We have completed  $\max$  for undefined values as follows,  $\max(\text{undefined}, \text{undefined}) = \text{undefined}$ ,  $\max(n_1, \text{undefined}) = n_1$  and  $\max(\text{undefined}, n_2) = n_2$ .

v.

$\star \rightarrow \bullet$

Each  $k$ -symbol string  $y^k$  on the right hand side of a production is replaced by a function call  $\bullet(y^k, d - 1)$ . The  $k + 1$ -ary operation  $\bullet$  is defined by:

$$\bullet(y^k, d) = \sum_{i=1}^k \max_{r_i=d, r_i \geq 0, d \geq 0} \varphi(y_i, r_i) \quad (6)$$

This expression is structurally identical to expression 3. We have changed what we count by simply changing the operations of the underlying algebraic system.

$\varphi$  is a partial function, which returns its argument  $d$  as result, if a word in  $L_1$  can be derived in  $d$  derivation steps and which returns the result *undefined*, if no word can be derived in  $d$  derivations in  $L_1$ .

(b) We adjust  $\varphi$  in order to take  $\Phi$  into account. Of course, how this has to be done, depends on the bounding relation  $\Phi$ .

3. Finally, we try to establish  $\text{card}(S_{L_1, d}) < \text{card}(S_{L_2, \phi(d)})$  almost everywhere by a suitable technique. We refer the reader to a black belt generatingfunctionologist for applicable techniques [Wilf, 1990].

## V. The Boston Consulting Group's Rule Language

In this section we prove that a fuzzy classifier system with the fuzzy rule language *FRL* used in the Boston Consulting Group example given in [Geyer-Schulz, 1993; Bandemer, 1993] learns a rule-base faster than a classifier system with the crisp rule language *RL*. In contrast to the language *FL* presented in [Geyer-Schulz, 1992; Lowen, 1992] the labels of a linguistic variable of *FRL* are defined by a context-free grammar, not by enumeration. The infinite language *FRL* is defined by a context-free grammar which allows the formulation of quite complex rules. An immediate consequence is that the construction of the depth-bound function  $\phi_I$  from the bounding relation  $\Phi_I$  is not as easy and as obvious as in the example presented in [Geyer-Schulz, 1992]. Obviously, the complexity of learning algorithms depends on the search space size. Therefore, let us formulate our theorem in terms of search space size.

With the help of the representation theorem of [Negoita and Ralescu, 1975] we are able to derive a crisp rule base in *RL* which approximates the fuzzy rule base in *FRL* for each fuzzy rule base of *FRL*. However, if we require an exact translation  $\Phi$  of a fuzzy rule base in *FRL* to a crisp rule base in *RL*, we get:

**Theorem 2** *Translation of a rule in FRL by  $\Phi$  generates non-denumerably many rules in RL.*

**Proof** By application of the representation theorem a fuzzy subset can be expressed by a family of non-denumerably many classical subsets. (See [Negoita and Ralescu, 1975].) ■

Of course,  $\Phi$  is not suitable for computer implementation, because it generates an infinite number of rules in *RL* for each rule in *FRL*. This means, that a fuzzy rule language ideally provides an infinite speed-up when compared with a crisp rule language. In computer implementations of fuzzy rule languages we approximate the unit interval by a finite chain  $I$ . Nevertheless, fuzzy rule languages still give a considerable speed-up over crisp rule languages.

**Theorem 3**  *$\text{card}(S_{FRL, d}) < \text{card}(S_{RL, \phi_I(d)})$ , for all  $d \geq k$ .*

**Proof** Because of space restrictions, we refer the reader to [Geyer-Schulz, 1993] for a more complete version of the proof. To get the reader started we provide in the following the definitions of *FRL*, *RL* and  $\Phi(I)$ .

1. The grammar of *FRL* is given by the following production rules:

$\langle \text{rule base} \rangle := \langle \text{rule} \rangle \mid \langle \text{rule} \rangle \langle \text{rule base} \rangle$   
 $\langle \text{rule} \rangle := \langle \text{frame} \rangle \text{ "USED" "IF" } \langle \text{truth value} \rangle$   
 $\langle \text{truth value} \rangle := \langle \text{truth value} \rangle \langle \text{connective} \rangle \langle \text{truth value} \rangle \mid \text{ "(" } \langle \text{noun} \rangle \text{ "IS" } \langle \text{verbal expression} \rangle \text{ ")"}$   
 $\langle \text{verbal expression} \rangle := \langle \text{adjective} \rangle \mid \langle \text{adverb} \rangle \langle \text{verbal expression} \rangle \mid \text{ "(" } \langle \text{verbal expression} \rangle \text{ ")"}$   
 $\langle \text{connective} \rangle \langle \text{verbal expression} \rangle$   
 $\langle \text{adjective} \rangle := \text{ "HIGH" } \mid \text{ "LOW" } \mid \text{ "MEDIUM" } \mid \text{ "UNDEFINED" } \mid \text{ "UNKNOWN"}$   
 $\langle \text{adverb} \rangle := \text{ "ABOVE" } \mid \text{ "BELOW" } \mid \text{ "AROUND" } \mid \text{ "UPPER" } \mid \text{ "LOWER" } \mid \text{ "MORE_OR_LESS"}$   
 $\mid \text{ "RATHER" } \mid \text{ "VERY" } \mid \text{ "NOT" } \mid \text{ "NEITHER" } \mid \text{ "POSSIBLY" } \mid \text{ "TRULY" } \mid \text{ "FUZZILY"}$   
 $\langle \text{connective} \rangle := \text{ "AND" } \mid \text{ "OR" } \mid \text{ "BUT" } \mid \text{ "NOR" } \mid \text{ "TO" } \mid \text{ "EXCEPT"}$   
 $\langle \text{noun} \rangle := \text{ "MARKETGROWTH" } \mid \text{ "RELATIVE_SHARE" } \mid \text{ "SALES"}$   
 $\langle \text{frame} \rangle := \text{ "STAR" } \mid \text{ "QUESTION\_MARK" } \mid \text{ "CASH\_COW" } \mid \text{ "DOG" } \mid \text{ "BUILD" } \mid \text{ "HARVEST" } \mid$   
 $\text{ "HOLD" } \mid \text{ "DIVEST"}$

2. We define the formal language *RL* by presenting the production rules of its grammar below:

$\langle \text{rule base} \rangle := \langle \text{rule} \rangle \mid \langle \text{rule} \rangle \langle \text{rule base} \rangle$   
 $\langle \text{rule} \rangle := \langle \text{frame} \rangle \text{ "USED" "IF" } \langle \text{truth value} \rangle \text{ "WITH" } \langle \alpha \rangle$   
 $\langle \text{truth value} \rangle := \langle \text{truth value} \rangle \langle \text{connective} \rangle \langle \text{truth value} \rangle \mid \text{ "(" } \langle \text{noun} \rangle \text{ "IN" } \langle \text{interval list} \rangle \text{ ")"}$   
 $\langle \text{connective} \rangle := \text{ "AND" } \mid \text{ "OR" } \mid \text{ "BUT" } \mid \text{ "NOR" } \mid \text{ "TO" } \mid \text{ "EXCEPT"}$   
 $\langle \text{noun} \rangle := \text{ "MARKETGROWTH" } \mid \text{ "RELATIVE_SHARE" } \mid \text{ "SALES"}$   
 $\langle \text{frame} \rangle := \text{ "STAR" } \mid \text{ "QUESTION\_MARK" } \mid \text{ "CASH\_COW" } \mid \text{ "DOG" } \mid \text{ "BUILD" } \mid \text{ "HARVEST" } \mid$   
 $\text{ "HOLD" } \mid \text{ "DIVEST"}$   
 $\langle \text{interval list} \rangle := \langle \text{interval} \rangle \mid \langle \text{interval} \rangle \langle \text{interval list} \rangle$   
 $\langle \text{interval} \rangle := \text{ "[" } \langle \text{real} \rangle \text{ "," } \langle \text{real} \rangle \text{ "]"}$   
 $\langle \text{real} \rangle$  denotes the *m*-bit representation of a floating point number. For the rest of the example we assume 8 byte reals.  
 $\langle \alpha \rangle$  is a  $\langle \text{real} \rangle$  in  $[0, 1]$ .

3. With the help of the representation theorem of [Negoiita and Ralescu, 1975] we can construct a mapping  $\Phi : FRL \hookrightarrow RL$ , which translates each fuzzy rule to an (infinite) indexed family over the real unit interval  $[0, 1]$ . By choosing a finite chain  $I$  with elements in  $[0, 1]$  we approximate  $\Phi$  by  $\Phi_I : FRL \hookrightarrow RL$  defined by the following set of meta production rules [Geyer-Schulz, 1992; Kruse and Meyer, 1987]:

$$\forall \alpha \in I : \langle \text{frame} \rangle \text{ "USED" "IF" } \langle \text{truth value} \rangle \implies \langle \text{frame} \rangle \text{ "USED" "IF" } \langle \text{truth value} \rangle \text{ "WITH" } \langle \alpha \rangle \quad (7)$$

$$\text{ "(" } \langle \text{noun} \rangle \text{ "IS" } \langle \text{verbal expression} \rangle \text{ ")" } \implies \text{ "(" } \langle \text{noun} \rangle \text{ "IN" } \langle \text{interval list} \rangle \text{ ")" } \quad (8)$$

For the calculation of table 1 we have used a chain  $I$  with  $\text{card}(I) = 5$ .

■

We still do not know the order of magnitude of the difference of the search space sizes of *FRL* and *RL*. In order to answer this questions, we let the computer do its best and evaluated the first 20 coefficients of the resulting power series. Because of the fact that bringing difference equation schemes into a nice closed form is cumbersome, we used brute force and evaluated the difference equations directly. In table 1 we have tabulated our results. The question marks in the last column show the limits of the brute force approach: we ran out of storage on a 12 MB virtual machine. Nevertheless, the coefficients which we could obtain show that the search space size of *FRL* is considerably smaller than the search space size of *RL*.

## References

- [Aho and Ullman, 1972] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling, Volume I: Parsing*, volume 1. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1972.

$d$	$\phi_I(d)$	$\text{card}(S_{FRL,d})$	$\text{card}(S_{RL,\phi_I(d)})$
7	50	120	$3.1359 \cdot 10^{444}$
9	50	$1.6800 \cdot 10^3$	$3.1359 \cdot 10^{444}$
11	70	$2.5560 \cdot 10^4$	$1.4307 \cdot 10^{637}$
13	90	$4.4040 \cdot 10^5$	$6.5276 \cdot 10^{829}$
14	100	$4.5480 \cdot 10^5$	$7.5585 \cdot 10^{906}$
15	100	$8.0293 \cdot 10^6$	$7.5585 \cdot 10^{906}$
16	100	$8.4037 \cdot 10^6$	$7.5585 \cdot 10^{906}$
17	110	$1.5221 \cdot 10^8$	$2.5720 \cdot 10^{945}$
18	120	$1.6037 \cdot 10^8$	?
19	130	$2.9762 \cdot 10^9$	?
20	140	$3.1502 \cdot 10^9$	?

 Table 1: The Depth of the Derivation Trees and the Search Space Sizes in  $RL$  and  $FRL$ 

- [Bandemer, 1993] Hans Bandemer, editor. *Modelling Uncertain Data*, volume 68 of *Mathematical Research*, Berlin, 1993. Akademie Verlag.
- [Blum, 1967] Manuel Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the Association for Computing Machinery*, 14(2):322-336, April 1967.
- [Cutland, 1980] Nigel J. Cutland. *Computability - An Introduction to Recursive Function Theory*. Cambridge University Press, Cambridge, 1980.
- [Feferman et al., 1986] Solomon Feferman, John W. Dawson jr., Stephen C. Kleene, Gregory H. Moore, Robert M. Solovay, and Jean van Heijenoort, editors. *Kurt Gödel - Collected Works, Publications 1929 - 1936*, volume 1, New York, 1986. Oxford University Press.
- [Geyer-Schulz, 1992] Andreas Geyer-Schulz. Fuzzy classifier systems. In Lowen [1992], page ? to appear.
- [Geyer-Schulz, 1993] Andreas Geyer-Schulz. On the specification of fuzzy data in management. In Bandemer [1993], pages 105-110.
- [Gödel, 1936] Kurt Gödel. Über die Länge von Beweisen. *Beiträge zu einem mathematischen Kolloquium*, 1936. Quoted from the reprint in [Gödel, 1986].
- [Gödel, 1986] Kurt Gödel. Über die Länge von Beweisen. In Feferman et al. [1986], pages 396-398.
- [Goulden and Jackson, 1983] Ian P. Goulden and David M. Jackson. *Combinatorial Enumeration*. John Wiley & Sons, New York, 1983.
- [Kruse and Meyer, 1987] Rudolf Kruse and Klaus D. Meyer. *Statistics with Vague Data*. D. Reidel Publishing Company, Dordrecht, 1987.
- [Kuich and Salomaa, 1986] Werner Kuich and Arto Salomaa. *Scmirings, Automata, Languages*, volume 5 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin, 1986.
- [Lowen, 1992] Robert Lowen, editor. *Fuzzy Logic: State of the Art*, Dordrecht, 1992. Kluwer Academic Publisers. to appear.
- [Negoita and Ralescu, 1975] C. V. Negoita and D. A. Ralescu. Representation theorems for fuzzy concepts. *Kybernetcs*, 4:169-174, 1975.
- [Parikh, 1973] Rohit J. Parikh. Some results on the length of proofs. *Transactions of the American Mathematical Society*, 177:29-36, March 1973.
- [Salomaa, 1973] Arto Salomaa. *Formal Languages*. ACM Monograph Series. Academic Press, New York, 1973.
- [Statman, 1978] Richard Statman. Bounds for proof-search and speed-up in the predicate calculus. *Annals of Mathematical Logic*, 15:225-287, 1978.
- [Wilf, 1990] Herbert S. Wilf. *generatingfunctionology*. Academic Press, San Diego, CA, 1990.
- [Young, 1973] Paul Young. Easy constructions in complexity theory. *Proceedings of the American Mathematical Society*, 37(2):555-563, September 1973.