Context Free Grammar for the Generation of a One Time Authentication Identity

Abhishek Singh, Andre L M dos Santos
Georgia Tech. Information Security Center (GTISC)
Center for Experimental Research in Computer Science (CERCS)
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{abhi, andre}@cc.gatech.edu

Abstract

An authentication protocol was proposed in [11] for the generation of one time authentication identity, which can be used as one time passwords, generation of disposal credit card numbers. The proposed protocol was designed using a context free grammar and was in the learning theory domain. The protocol required that the context free grammar used for the authentication procedure be difficult to learn. The paper discusses the relationship between the learning theory and the number theory. Then based upon the experimental limitations to learn a context free grammar, we present an algorithm for the generation of context free grammar which is difficult to learn. There exists no theoretical study which states given a set of strings from a language L, how difficult it is to generate another string which belongs to the same language. Experiments were conducted to determine this empirically. A context free grammar from the proposed algorithm was used to generate the string.

1.0 Introduction

A *representation class* is a class of objects that can be represented by strings over some alphabet. The set of all context free grammar is a representation class. A *learning algorithm* is the one that tries to learn a representation class from positive and negative instances.

Let *c* and *h* be the two instances of some representation class, where *c* is the target class or the class to be learnt, while *h* is the output class. Let *A* be the learning algorithm, which takes positive and negative instances as an input and outputs class *h.* $e_c^+(h_A)$ denotes that algorithm *A* errors on a positive instance of c and classifies them as a negative instance of h. Similarly $e_c^-(h_A)$ represents the probability that *A* errors on negative instances of *c*, by classifying then as positive instances of *h*. If , $e_c^+(h_A) = e_c^-(h_A) = 0$, then the classes *c* and *h* are identical. This is the ideal case of complete learning or the case where errors are zero can be classified as strong learning. There can be another case where the error is not zero, but it is small negligible value. This case can be defined as weak learnable. Formally defining a representation class c is weakly learnable if there exists a probabilistic polynomial time algorithm *A* that on access to a set of positive and negative instances of a target representation $c \varepsilon C$, generates an output representation *h*, so that $e_c^+(h_A) < ½ - 1/O(|c|^K)$ and $e_c^-(h_A) < ½ - 1/O(|c|^K)$ and for some constant k. |c| is usually taken as some polynomial in n, the length of the sample instance

For context free grammar, if A_{CFG} is a learning algorithm for context free grammars, then for any given context free grammar G_1, A_{CFG} on having access to a small subset of positive and negative instances (i.e., strings that belong to G_1 and are outside of G_1 respectively) can implicitly construct a grammar G_2 which is *almost identical* to G_1. Almost identical can either be a case of strong learn ability or it can be a case of weak learn ability. Strong learnability lies in un decidable domain. It can be proved as follows

Given a set of strings *S1, S2, S3…Sn, Sn+1… Sm* there can be infinite number of grammars that can generate these strings. The exact context free grammar is a grammar, which can be generated by using *S1, S2, S3…. Sn* and which successfully accepts *Sn+1, Sn+2… Sm.* Let's assume that there exist an learning algorithm *A* which on an input set of string *S1, S2, … Sn* can predict the exact context free Grammar *G* which generated it. Now let us pick any context free grammar *G1* and generate strings *S1, S2.. Sn* by randomly selecting rules from the grammar. These strings are passed through the learning algorithm *A*, which gives grammar *G2* as the exact context free grammar. However, it is known that given two context free grammar *G1* and *G2*, *L(G1) = L(G2)* is an undecidable problem [1]. Therefore there exist no way to verify if the language generated by *G1* and *G2* are equivalent. Hence there cannot be any algorithm, which can give an exact context free grammar. So the strong learn ability of context free grammar lies in the undecidable domain.

Strong learn ability implies learning the whole CFG. Since strong learnability requires false positives and false negatives to be zero, target class is same as the output class. Hence it classifies the instances with probability one. This makes strong learning is a very strong assumption. Either the output class is same as target class or it is different from the output class. Hence a random guessing would classify the output class correctly with probability ½. For cryptographic purposes it needs to be defined how much better a learning algorithm can learn a grammar compared to random guessing. To this end comes the notion of weak learnability. A representation class is weakly *learnable* if there exists an algorithm that can do a little better as a classifier than random guessing. Weak learnability expects an algorithm to do just *slightly* better than random guessing.

According to results of Kearns and Valiant [2], if $ADFA_n^{p(n)}$ denote the class of deterministic finite automata of size at most p(n) that only accepts strings of length n, and $ADFA^{p(n)} = U_{n>1} ADFA_n^{p(n)}$, then for some polynomial p(n), the problem of inverting the RSA encryption function, recognizing quadratic residues and factoring blum integers are probabilistic polynomial-time reducible to weakly learning $ADFA^{p(n)}$. They further state that any representation class whose computational power subsumes that of NC^1 is not weakly learnable. Since CFG's contains the computational complexity class NC^1, they are also not weakly learnable under the similar cryptographic assumptions as that of ADFA. From the above mentioned discussion it can be inferred that it is tough to learn CFG. Factoring RSA, recognizing quadratic residues and factoring blum integers are reducible to weakly leaning CFG's.

An algorithm to generate one time authentication identity was proposed in [11]. This algorithm can be used to generate one time passwords, disposable credit card numbers or it can be used to provide heart beat in the case of sensor networks. Detailed description of generation protocol and authentication protocol and the other detailed analysis about the protocol have been published and will not be discussed due to the space limitations.

One of the main challenges was the choice of a context free grammar for the protocol. Theoretically even though it is tough to learn the context free grammar, empirically there have been several studies to learn context free grammars. Results of these studies and the theoretical limitation to learn CFG's forms the foundation for the algorithm to generate context free grammars which are difficult to learn.

Section 2.0 discusses the experimental limitations to learn the context free grammar. In section 3.0 we present the design considerations for the automatic generation of context free grammar. In section 4.0 we present the empirical results which talks about the toughness to guess a string in a language from the given set of strings in the language. Finally in section 5.0 conclusions is presented.

2.0 Experimental Limitations to learn Context Free Grammar.

The earliest method for CFG learning was proposed by Solomonoff [3]. In this approach the learner is given a positive sample S^+ from a language L and has access to a membership oracle for L. Solomonoff in his work proposed to find repeated patterns in a string: for every string $w \in S^+$, delete sub-strings from w to create new string w' and ask the oracle if w' is in the language. If it is, then insert repetitions of the deleted sub-strings into the string and ask if the new string is also in the language. If so, then there must be a recursive rule. For example, if there are several strings of type $a^n b^n$, then we can infer that $A \rightarrow aAb$ is in the grammar. This method is inefficient and does not include all the context free grammars. It has been shown [5] that this method fails if we have sequential embedding. For the same example, if we have $A \rightarrow aAb \mid cAd$ then this method will not work. Another attempt for predicting context free grammars reported in [4]. It is assumed in [4] that there is a teacher that gives a set of strings to a learner or a program. The program first checks if the sample is already in the language. If it is, then the program decides in consultation with the teacher if it is finished or if it should continue with more samples. If the string is not in the machine's language, the program adds to the grammar a production that adds this sentence to the language, attempts to verify that this production does not lead to illegal strings, and requests the next sample from the teacher. This method heavily depends upon the order in which samples are presented. This algorithm may choose a grammar rule that is too general. At a later stage a new sample may come up and even the predicted grammar may generate text that is not in the language. An example is when predicting the language of arbitrarily long strings of *a*'s or *b*'s, but not both *a* and *b*. If samples *a* and *aa* are given the algorithm will produce a partial grammar with the following rules

$$S \rightarrow a \mid SS$$

Now if sample *b* is given it produces $S \rightarrow b$. This grammar can produce the string *ab,* which is not in the language. The algorithm has made use of only positive instances to predict the grammar. However, from Gold's theorem it is clear that both positive and negative instances are needed to predict the context free grammar. Neural networks were used to learn context free grammars in the work done by [7]. Neural networks were trained with both positive and negative instances. As in [7], the order of input was important. The network was trained first with shorter strings and then with longer instances. This scheme proved good only for small context free grammars having around four to five terminals. It failed to scale for larger context

free grammars.Genetic algorithms are used to learn stochastic context free grammars from a finite sample in [8]. Genetic algorithms [9] are a family of robust, probabilistic optimization techniques that offer advantages over specialized procedures for automated grammatical inference. A stochastic context free grammar is a variant of ordinary context free grammars in which grammar rules are associated with a probability, a real number from the range of [0,1]. A genetic algorithm was used to identify the context free grammar. This algorithm took a corpus *C* as an input. This corpus *C* for a language *L* was a finite set of strings drawn from *L* where each strings $\alpha \in C$ is associated with an integer f_α representing its frequency. As in [7] this only can be used for small grammars. The chief limitation of this approach is the cost involved in evaluating the fitness of each candidate solution, which required parsing every string in each possible way. This number of parsing operations increased exponentially with the number of non-terminals. Hence, this scheme is prohibitively costly for more than 8 non-terminals.

3.0 Design considerations for the Automatic generations of Context Free Grammar

Based upon the experimental results to learn context free grammar, discussed in section 2.0, this section presents the design consideration for the context free grammar. Let $L_{\max}$ denote the maximum length of a string generated by the grammar and L_{min} denote the minimum length. For example, for strings of size 15 – 30, L_{max} will be equal to 30 and L_{min} will be equal to 15. For a particular value of L_{max} and L_{min} the appropriate number of terminals, nonterminals and corresponding rules should be generated which should ensure that it will be tough to learn the context free grammar from the collected samples of strings. Let *n* denote the number of terminals in a language. Given L_{max}, L_{min} and *n* all the permutations of terminals to generate strings of size between L_{max} and L_{min} will be

$${}^{n}P_{L\,max} + {}^{n}P_{L\,max-1} + \ldots\ldots {}^{n}P_{L\,min}$$

With increase in *n,* the total number of permutations of all the terminals increases. Also from the experimental studies it can be seen that the probability that a learning algorithm can learn the grammar from given samples of strings decreases with the increase in the number of terminal *n*. In addition if $n > L_{max}$ then this ensures that a string of length *l* generated by randomly calling rules will never have all the terminal symbols of a language. This will ensure that at no point of time the learning algorithm has all the terminals. The value of *n* is chosen such that $L_{max} < n < k\ L_{max}$ where integer $k \in 2,3\ldots$ Since this authentication scheme is being designed by considering not only high-end workstations but also small memory devices like PDAs and smart cards, there will be a limit on the appropriate value of *n*. This is being investigated by implementing the authentication protocol on smart cards and on PDAs.

Another major design consideration is the number and format of rules in each grammar. Let us assume that the grammar rules are written in Chomsky Normal Form (CNF). In CNF (as shown in figure 1) the right side of each production is always of length two. An efficient method to automatically generate rules for the start symbol and other nonterminals is required in order to use the new cryptographic domain.

Nonterminal → Terminal Nonterminal	Rule 1
Nonterminal Terminal	Rule 2
Nonterminal Nonterminal	Rule 3
Terminal Terminal	Rule 4

Figure 1: Format of Rules for CNF

It is assumed that strings of length greater than two will be generated. So the start symbol will contain only rules (as shown in figure 1) from one to three. For the start symbol expansion, rule one, two or three is selected randomly. Terminals or non-terminals are selected to generate the specific rule. This process is repeated until all the nonterminals are used. During the random selection of rules it may happen that only the format of rule one (Non-terminal → Terminal Non-terminal) is selected for all the rules. If this happens and the start symbol is using all the terminals then a learning algorithm can easily learn the first character of strings in a CFG. Having knowledge of the first character of all the strings in a language will reduce the complexity to learn CFG. Hence it becomes essential that all the terminals should not occur in the rules for the start symbol. Another problem while generating rules for the start symbol can occur due to the more frequent appearance of some of the non-terminals as compared to others. If some non-terminals appear more frequently than others, then it may happen that while generating strings by randomly calling rules, sub strings generated by frequently occurring non-terminals will appear more often as compared to the sub strings generated by other non-terminals. Using the method discussed [3] in section 3 the CFG can be predicted from the tracking of the frequently occurring nonterminals. This can be avoided by ensuring that the probability of occurrence of all the non-terminals remains equal. To address this issue, while generating rules for the start symbol, each non-terminal is called only once as shown in the algorithm of figure 3. This makes all the combinations, which can occur due to each non-terminal, equally likely. As discussed in the previous section, with the increase in the number of non-terminals the complexity of identifying the grammar increases exponentially. If the number of non-terminals is higher than the number of terminals in a grammar, and considering the worst case of format of rule number one selected (S → Terminal Non-terminal) for all the rules in the start symbol, then all the terminals will occupy the first character of the string. This

makes the learning of the first character of all the strings generated by a CFG a trivial task. So in a grammar the number of non-terminals should be as high as possible, however it also should not exceed the number of terminals. Hence the number of non-terminals occurring in a grammar is chosen randomly as *kn* where 0<k< 1. For the experiments, the value of k is fixed to be 0.75. Another major consideration is the number of terminals, which should occur in rules for a non-terminal. As per the algorithm shown in figure 2, while generating a rule for a non-terminal, one format of the rule amongst the four formats of rules (as shown in fig 1) is chosen randomly. The appropriate number of terminals/nonterminals is selected to generate the rule, which follows the appropriate format. This process is repeated until the desired number of terminals is guaranteed in the rules for the non-terminal. Again considering the worst-case scenario, it may happen that while generating rules for a non-terminal, only the format of rule number one (Non-terminal → Terminal Non-terminal) is selected for all the rules. In this scenario too, if a non-terminal uses all the terminals, then a learning algorithm can easily predict a few characters of the string and hence it can make a reasonable guess about the CFG. Taking a simple example, if there are two non-terminals A, B and there are 3 terminals a,b,c, considering the worst scenario, the following rules are generated for the non-terminal A → aB | bA| cb and B→ bA|aB|ca. For such types of rules, learning the few characters is a trivial task. Two characters which will occur in a string generated by these rules can be 'ab','aa','ba',bb' which is all the combinations of the terminals 'a' and 'b'. This is because 'a' and 'b' occur in all the nonterminals (A and B) in the format of rule number one. From this discussion it can be noticed that the two conditions are necessary to cause the mentioned weakness

- All terminals should occur in every nonterminals
- These terminals should be present in the form of rule number one.

If this happens, a learning algorithm has to make a reasonable guess only for the last few characters. In order to decrease the probability of such weakness to be present, the total number of terminals, which should occur in the rules for a non-terminal, is chosen randomly between mn and pn where (0<m<p<1). For the experiments, *m* was fixed to be 0.5 and *p* was fixed to be 0.75. As discussed in the previous section, rules must be written so as to accommodate sequential embedding [10]. Sequential embedding makes it difficult to identify the grammar from the captured strings. In sequential embedding, each non-terminal should have at least two or more expansions. If r denotes the number of sequential embedding rules for a non-terminal X, then the value of r is chosen randomly

Procedure *RuleforNonterminal*()
Input: Number of Terminals, Terminal Pool, Nonterminal Pool, Nonterminal
Output : Rules for a Nonterminal
Begin
Select a Random Number *rno* such that *mn* < *rno* < *pn*.
(Where 0<*m*<*p*<1)
Select r for sequential embedding such that *qn* < *r* < *wn*
(Where 0< *q* < *w* <0.5
Begin While(*r*!=0)
Randomly select a format of rule from rule 1-3 .
Pick up nonterminal from the nonterminal pool if the selected format is of rule3.
Pickup terminal from the terminal pool if selected format is of rule 1 or 2.
Generate rule as per the selected format.
If nonterminal pool is empty reconstruct it.
If terminal pool is empty reconstruct it.
End While

Begin While(*rno*!=0)
Randomly select a format of rule from rule 1-4.
Pick up nonterminal/nonterminals from the nonterminal pool as per selected format.
Pick up terminal/terminals from the terminal pool as per the format of the rule.
If single terminal is picked then *rno* = *rno* –1
If two terminals are picked then *rno* = *rno* –2 Generate the rule as per the selected format.
If nonterminal pool is empty reconstruct the nonterminal pool.
If terminal pool is empty reconstruct it.
End while
End

Figure 2 Algorithm to generate rules for the nonterminals

between q*n < r < w*n. where (0<q<w<0.5). The algorithm presented in figure 2 is being used to generate the rules for the nonterminals. The nonterminal pool used in the algorithms shown in figures 2 and 3 is an array, which contains the nonterminals for a grammar. The same is true for the terminal pool. Selection of a nonterminal from the nonterminal pool involves choosing one of the nonterminal from the array. As soon as the element is chosen from the nonterminal pool it is deleted from it. When the nonterminal pool becomes empty all the elements are reinserted in the nonterminal pool in a random fashion. A similar operation is performed on the terminal pool, which comprises the terminals.

4.0 Experimental Results.

There exists no theoretical result which given a set of strings from a particular language, demonstrates how difficult is it to guess another string which belongs to the

Procedure *rulefortartSymbol*()
Input : Terminal Pool, Nonterminal Pool.
Output: Rules for Start Symbol in Chomsky normal form.
Begin While (Nonterminal pool is empty){
1.Randomly select a format of a rule from rules 1-3
2. Pick up nonterminal/nonterminals from nonterminal pool as per the requirement of the selected format.
3. Pick up terminal from terminal pool if selected format is of rule 1 or 2.
4. From the selected nonterminal/terminal generate the rule which follows the selected format.
End while

Figure 3.0 Algorithm to generate Rules for the start symbol

same language. Some experimental tests were conducted to determine it empirically. A context free grammar was constructed using the algorithm discussed in section 3. The number of terminals was chosen to be 30, with 26 nonterminals used, which is a random value between ¾ n and n. The number of sequential embedding for each rule was chosen between n/10 and n/30. This resulted in a value between 1 and 3. The rules were then used to generate strings on different ranges: 3 to 25, 3 to 50, 3 to 100, 3 to 150. Four types of tests were performed on these strings based upon the starting terminal and ending terminal. These tests were named *frontbreaking*, *backbreaking, allfrontbreakimg, allbackbreaking.* The *frontbreaking* test selects a string and breaks into different combinations such that the first terminal remains same. For example, different combinations of the string "abcdefg" by the *frontbreaking* rule are abc, abcd, abcde, and abcdf. The b*ackbreaking* rule selects a string and breaks it into different combinations such that the last terminal remains the same. Different combinations for the same example by back breaking rule are efg, defg, cdefg, bdefg . These strings were passed through the grammar. The *allfrontbreaking* type of rule involves collection of all the terminals, which start a string, followed by the selection of a string and parsing it to find out if any of the start terminals appears in it. In case of appearance of any of the start terminals at any position except at the starting of the string, the string is broken such that start symbol occupies the first position and the *frontbreaking* rule is applied. The *allbackbreaking* type of rule involves collection of all the terminals, which ends the strings, followed by the selection of a string and parsing it. In case of appearance of any of the end terminals at any position except at the end, string is broken such that the end terminal occupies the last position and the *backbreaking* rule is applied. Taking a simple example if there are two strings "abcdef" and "crafgd" we collect all the terminals which start the strings and all the terminals which end the string. For starting terminals we get "a" (starting terminals for "abcdef") and "c" (starting terminal for "crafgd"). A string is chosen and it is parsed to find out if any of the start symbols appear in it. In the current example "c" appears in "abcdef" so by the allfront breaking rule the strings "cdef" and "cde" are generated. Similarly all the end terminals are collected. For the given example the end terminals are "f" and "d". Since end terminal "f" appears in "crafgd" strings "craf" and "raf" are generated using the *allbackbreaking* rule.

674 strings of length ranging from 3 – 25 were generated. By applying *frontbreaking* rule, the strings resulted in 4902 combinations. Out of these 4902 combinations 390 strings were accepted by the grammar. So for strings of length between 3 – 25, around 90% of time it can be ensured that the strings generated by applying *frontbreaking* rule will not be accepted by the grammar. 956 strings of length between 3-150, resulted in 20925 combinations by *frontbreaking* rule, out of which only 398 strings got accepted by the grammar. This gives an acceptance rate of 1.9%. For each range, a different set of strings was generated by randomly expanding the start symbol and nonterminals. This means that the 674 strings generated for the length-range 3-25 are totally different from 956 strings generated for the length-range 3-150. As the string-length increases, the *front-breaking* rule results in increasing number of output combinations. And at the same time, the number of strings accepted by the grammar decreases. By applying *backbreaking* rule, 674 strings of length between 3- 25 resulted in 4902 combinations, out of which 195 got accepted by the grammar. 956 strings of length ranging from 3 – 150 resulted in 20925 combinations, out of which 212 got accepted. For the *backbreaking* rule, string-length ranging from 3 – 25 resulted in 3.9% acceptance and of size between 3 – 150 results in 1.013% acceptance by the grammar. By applying the *allfrontbreaking* rule, 724 strings of length between 3-25, resulted in 4250 combinations out of which 169 got accepted. 1203 strings of size ranging from 3 – 150 resulted in 19991 combinations by using *allfrontbreaking* rule. Out of 19991 strings, 213 got accepted giving an acceptance rate of 1.06%. *Allbackbreaking* rule for 724 strings of length ranging from 3 – 25 resulted in 3369 combinations out of which 125 strings got accepted by the grammar. This gives an acceptance rate of 3.7%. 1023 strings of length between 3- 150, resulted in 16232 combinations by *allbackbreaking* rule. Out of 16232 combinations, 209 strings got accepted by the grammar. This gives an acceptance rate of 1.28%

From the experiments, it can be concluded that the chance of acceptance of a string generated by breaking the strings of size ranging from 3 to 150 is only 1%. For 99% of the cases it can be ensured that the strings generated by breaking the strings will not belong to the language. It was also concluded that if the difference between maximum length and minimum length is very large then the acceptance probability of a string generated after breaking

the string reduces. In all these tests the maximum length of strings generated was 150.

5.0 Conclusion.

A context free grammar has been used for the first time for the design of an [11] authentication protocol. Many details about the protocol have been omitted, since the protocol has already been published. The main contribution of this paper is to present the theoretical and experimental limitations to learning context free grammars, and, based upon these limitations, to present the design consideration for the context free grammar and an algorithm to generate context free grammars based upon the experimental limitations. Tests were then conducted to determine given a set of string from a language how difficult it is to generate another string which belong to the same language. For the best case chances of acceptance of strings generated after breaking the string is 1%. As the size of the string increases percentage of accepted strings generated after breaking the strings decreases. Hence if the output of an authentication protocol comprises only the strings belonging to the shared secret language between Alice and Bob, then the length of the strings should be large enough to make it tough for Eve to guess the next string from the given set of strings. The output of [11] the proposed algorithm does not comprise the strings belonging to the shared language. The chances of guessing the output of the protocol forms the focus of current investigation. Theoretical results to determine the hardness of guessing a string belonging to language from the given set of strings will be an interesting study

Acknowledgement

The authors are grateful to the anonymous reviewers for their constructive comments. The authors also wish to acknowledge valuable discussions with Arnab Paul, Aranyak S Mehta and Prof. H. Venkateswaran.

References

[1] Lewis H.R., Papadimitriou C. H., *Elements of the Theory of Computation,* Prentice – Hall, 1998.

[2] Kearns Michael and Valiant Leslie, "Cryptographic limitations on learning Boolean formulae and finite automata", *Journal of ACM*, 41(1): 67 – 95, January 1994.

[3] Solomonoff R. J., "A method for discovering the grammars of phase structure language", *Information processing,* New York: UNESCO, 1959.

[4] Knobe Bruce and Knobe Kathleen, "A method for inferring context free grammars", Information Control 2(2), pp 129

[5] Gold E Mark, "language Identification in a Limit", Information and Control, 10(5), pp 447 – 474, 1967.

[6] Angulin Dana, "Negative Results for equivalence queries , Machine Learning 2(2), pp. 121 – 150, 1990.

[7] Das Sreeupa, Giles C. Lee, Sun Guo- Zheng, " Learning context free grammars : Capabilities and Limitations of a Recurrent Neural Networks with an External Stack Memory" , Fourteen Annual Conference of the Cognitive Science Society, Morgan Kaufmann, San Mateo, CA, P 791 – 795, 1992.

[8] Keller, B. and Lutz R., "Learning Stochastic Context free grammars from examples from corpora using a genetic algorithm", in ICANNGA 97.

[9] Holland, J.H, "Adaptation in Natural and Artificial Systems", University of Michigan Press, Ann Arbor, MI, 1975.

[10] Fu King-Sun and Booth Taylor R., " Grammatical Inference: Introduction and survey", Parts I and II, IEEE Transaction Systems, Man and Cybernetics, SMC- 5(1) and (4), pp. 95 – 111 and pp. 409-423, 1975.

[11] Abhishek Singh, Andre L M dos Santos, "Grammar based offline generation of Credit card numbers", Proceedings of 17th ACM Symposium on Applied Computing SAC 2002, (Computer Security Track), March 10th – 14th , 2002, Madrid , Spain.