Stochastic grammars for intelligent software systems modeling

VASILE CRĂCIUNEAN
Department for Computer Sciences and Economic Informatics
"Lucian Blaga" University of Sibiu
Ion Rațiu Street, no. 5-7, Sibiu
ROMANIA

Abstract: - Developing a software system involves a sequence of operation classes like: gathering information from the activity field of the system; recognizing the captured information; integrating this information in the knowledgebase of the system; taking behavioral decisions that, based on his state of information, allow the system to find the solution for the given problem. Obviously software systems for problem solving must have the possibility of dynamic self information and decision correction. The stochastic generative grammar may be a useful tool for designing and developing such software systems.

Key-Words: - stochastic grammar, stochastic language, stimulation function, stimulation operator, dynamic autoinstruction process

1 Introduction

Definition 1.1. [2], [6], [7] A *Chomsky generative grammar* (shortly grammar) is a quadruple $G=(V_N,V_T,S,P)$, where V_N and V_T are alphabets of nonterminal respectively terminal symbols; $S\in V_N$ is the starting symbol or axiom and P is a finite set of pairs of words from $(V_N\cup V_T)^*$, $P=\{(u_i,v_i)\,|\,1\le i\le m\}$, so that any word u_i contains at least one nonterminal symbol. The pairs (u_i,v_i) are called *derivation rules* or production rules or simple *productions* and will be denoted by $u_i\to v_i$.

Definition 1.2. [3] Let Ω be a set. Then $\mathcal{K}\subseteq\mathcal{P}(\Omega)$ is a σ-algebra (borelian clan) who verifies the following conditions:

a) $A\in\mathcal{K}\Rightarrow C_A\in\mathcal{K}$

b) $\forall(A_1,A_2,...)\subseteq\mathcal{K}\Rightarrow\overset{\infty}{\underset{n=1}{\cup}}A_n\in\mathcal{K}$

The object from $\mathcal{K}$ are called *events*.

Definition 1.3. [3] The pair $(\Omega,\mathcal{K})$ where $\mathcal{K}$ is a σ-algebra over Ω, is called *event field*.

Definition 1.4. [3] Let $(\Omega,\mathcal{K})$ be an event field. Then the function $P:\mathcal{K}\to[0,1]$ that satisfies the following conditions

a) $A\in\mathcal{K}\Rightarrow 0\le P(A)\le 1$

b) $P(\Omega)=1$

c) $(A_n)_n\subset\mathcal{K}$, $A_n\cap A_m=\varnothing$

$$n\ne m\Rightarrow\sum_{n=1}^{\infty}P(A_n)=P(\overset{\infty}{\underset{n=1}{\cup}}A_n)$$

is called *probability measure* or *probability*.

The triplet $(\Omega,\mathcal{K},P)$ is called *probability space* ore *probability field*.

Definition 1.5. [3] Let $\mathcal{A}_n=\begin{pmatrix}A_1,...,A_n\\P_1,...,P_n\end{pmatrix}$ a finite probability filed. Then the *Shanon entropy* of filed $\mathcal{A}_n$ is $H_n(\mathcal{A}_n)=-\sum_{i=1}^{n}p_i\log_2 p_i$ with the convention $(0\cdot\log_2 0=0)$.

2 Stochastic grammars

Definition 2.1. A *stochastic generative grammar* (or shortly *stochastic grammar*) is a pair (G,f_p) where:

- $G=(V_N,V_T,S,P)$ is a Chomsky generative grammar and
- $f_p:P\to[0,1]$ is a probability function having the property $\Sigma(f_p(\alpha\to\alpha_i))=1$, where $\alpha\to\alpha_i$ are all α-productions from P.

Extending the probability functions from productions to derivations we obtain the following results: $f_p(S\overset{*}{\Rightarrow}q\to r)=f_p(S\overset{*}{\Rightarrow}q)f_p(q\to r)$, where $f_p(q\to r)=f_p(p)$, p being the set of productions applied.

Definition 2.2. The *languages* generated by the stochastic grammar (G,f_p) is

$$L(G,f_p)=\{p\,|\,p\in V_T^*\ and\ S\overset{*}{\Rightarrow}p,\ f_p(S\overset{*}{\Rightarrow}p)>0\}.$$

Definition 2.3. A *stochastic generative grammar of type* $i=0,1,2,3$ is a pair (G,f_p) where $G=(V_N,V_T,S,P)$ is a Chomsky generative grammar of type *i* and $f_p:P\to[0,1]$ is a probability function with the property $\Sigma(f_p(\alpha\to\alpha_i))=1$, where $\alpha\to\alpha_i$ are all α-productions from *P*.

Definition 2.4. A *stimulation functions* is a mapping $f:[0,1]^n\times\{1,2,...,n\}\to[0,1]^n$,
$f(x_1,x_2,...,x_n,i)=(f_1(x_1,x_2,...,x_n,i),$
$f_2(x_1,x_2,...,x_n,i),...,f_n(x_1,x_2,...,x_n,i))$ verifying the properties:

a) $\sum_{k=1}^{n}f_k(x_1,x_2,...,x_n,i)=\sum_{k=1}^{n}x_k$ (1)

b) $f_i(x_1,x_2,...,x_n,i)>x_i$ (2)

c) $f_l(x_1,x_2,...,x_n,i)<x_l$, $(\forall)\,l\neq i$. (3)

We notice that if $\alpha\to\alpha_i$, $i=1,...,n$ are all the α-productions from *P* then we can associate to them a finite probability field

$$\mathcal{A}_n(\alpha)=\begin{pmatrix}\alpha\to\alpha_1 & \alpha\to\alpha_2 & ... & \alpha\to\alpha_n\\ p_1 & p_2 & ... & p_n\end{pmatrix} \quad (4)$$

On the set of these probability fields $\mathcal{A}$ we define now an operator called *stimulation operator*.

A *stimulation operator* is a mapping $E:\mathcal{A}\times\{1,2,...,n\}\to\mathcal{A}$ defined as

$$E(\mathcal{A}_n(\alpha),i)=\begin{pmatrix}\alpha\to\alpha_1 & \alpha\to\alpha_2 & ... & \alpha\to\alpha_n\\ q_1 & q_2 & ... & q_n\end{pmatrix}, \quad (5)$$

where $q_l=f_l(x_1,x_2,...,x_n,i)$, $l\in\{1,2,...,n\}$.

Lemma 2.1. If $\mathcal{A}_n$ is a probability field then $E(\mathcal{A}_n(\alpha),i)$ is also a probability field and $q_i>p_i$.

Proof. We have immediately

$$\sum_{k=1}^{n}f_k(x_1,x_2,...,x_n,i)=\sum_{k=1}^{n}x_k=1.$$

Example 2.1. The function
$f:[0,1]^n\times\{1,2,...,n\}\to[0,1]^n$,
$f(x_1,x_2,...,x_n,i)=(f_1(x_1,x_2,...,x_n,i),$
$f_2(x_1,x_2,...,x_n,i),...,f_n(x_1,x_2,...,x_n,i))$ where

$$f_l(x_1,x_2,...,x_n,i)=\begin{cases}(x_l+1)/2, & \text{if } l=i\\ x_l/2, & \text{otherwise}\end{cases}$$

is a stimulation function since the conditions (1), (2), (3) are all satisfied.

a) $\sum_{k=1}^{n}f_k(x_1,x_2,...,x_n,i)=(x_i+1)/2+\sum_{l\neq i}x_l/2=1$

b) $f_i(x_1,x_2,...,x_n,i)=(x_i+1)/2>2x_i/2=x_i$

c) $f_l(x_1,x_2,...,x_n,i)=x_l/2<x_l$ for $(\forall)\,l\neq i$

Let $\mathcal{A}_n(\alpha)=\begin{pmatrix}\alpha\to\alpha_1 & \alpha\to\alpha_2 & ... & \alpha\to\alpha_n\\ p_1 & p_2 & ... & p_n\end{pmatrix}$ be a finite probability field. We define now recurrently $E^m(\mathcal{A}_n(\alpha),i)$ as follows:

a) $E^0(\mathcal{A}_n(\alpha),i)=\mathcal{A}_n(\alpha)$ (6)

b) $E^{m+1}(\mathcal{A}_n(\alpha),i)=E(E^m(\mathcal{A}_n(\alpha),i),i)$. (7)

Lemma 2.2. If $\mathcal{A}_n(\alpha)$ is a probability field and $E:\mathcal{A}\times\{1,2,...,n\}\to\mathcal{A}$ is a stimulation operator, then $\lim_{m\to\infty}H_n(E^m(\mathcal{A}_n(\alpha),i))=0$.

Proof. We notice that

$$E^m(\mathcal{A}_n(\alpha),i)=\begin{pmatrix}\alpha\to\alpha_1 & \alpha\to\alpha_2 & ... & \alpha\to\alpha_n\\ r_1^m & r_2^m & ... & r_n^m\end{pmatrix},$$

where $(r_l^m)_{m\in\mathbb{N}}$, $l=\overline{1,n}$, $l\neq i$ are decreasing sequences and bounded below by 0 and therefore convergent to 0. If $l=i$ then the sequence $(r_i^m)_{m\in\mathbb{N}}$ is monotone increasing and bounded above by 1 and therefore tends to 1. Thus the entropy tends to 0 when *m* tends to infinite.

Example 2.2. For the function considered in example 2.1

$$f_l(x_1,x_2,...,x_n,i)=\begin{cases}(x_l+1)/2, & \text{if } l=i\\ x_l/2, & \text{otherwise}\end{cases}$$ we have:

$r_l^m=\left(p_l+\sum_{k=0}^{m-1}2^k\right)/2^m$ if $l=i$ and

$r_l^m=p_l/2^m$ if $l\neq i$.

Obviously these sequences are converging to 1 respectively to 0 and therefore the entropy $H_n(E^m(\mathcal{A}_n(\alpha),i))$ tends to 0 when *m* tends to infinite.

As we may notice, with the stimulation operator, a stochastic grammar assimilates dynamic information by modifying the probability fields associated to the nonterminals, leading in the end to a dynamic auto instruction process. Such a tool is of great benefit in various practical applications.

For example, simulation models have a procedural character and there solutions are based on processing of experiments created in the framework of the model. Simulating a system spouses at first, a model for experiment creation, who matches best to the real evolution of the system, and second, processing procedures for this experiments that may show up the optimal decisions for a future system control.

If we denote by $V_T=\{a_1,a_2,...,a_t\}$ the set of

calculus procedures of the simulation model, then we may notice, that for solving a problem p there will be used a sequence of specific procedures for the underlying problem $\alpha = a_{p_1} a_{p_2} ... a_{p_r} \in V_T^*$. But the final sequence α is obtained after a potentially high number of tries from a set of possible sequences. Thus to our problem we may associate the language $L_p \subset V_T^*$. If we denote by π the set of all problems that can be solved by our model, then the language $L_\pi = \bigcup_{p \in \pi} L_p$ contains all the sequences of procedures corresponding to all the problems that can be solved with these model.

Certainly the development of sequences of procedures corresponding to the tries of solving a problem is made upon some model specific rules. Based on these rules there can be written the generative grammar for the language $L_p \subset V_T^*$, followed by defining the probability fields associated to the terminals and the stimulation operator.

For instance, in the framework of simulation models, the dynamic behavior over time is accomplished by procedures. The simulation starts by initializing the system with data that describe the initial state of the system. The system dynamics consists in choosing the next activities, respectively the next procedures to be executed.

An activity is a procedure composed of two parts: a test part and an action part. Every time the simulation clock advances, all the possible activities (procedures), at this moment, are explored. If the condition of an activity is satisfied than the activity is executed, else it is ignored.

For exemplification let's consider a very simple simulation model with three activities:
f – initialization of the model with data that describe the initial state of the model;
g – an specific simulation activity;
h – an special simulation activity that executes just once in the simulation process.

In this given situation the simulation process will behave as follows:

- the model is initialized (executing procedure *f*);
- the specific simulation activity *g* is executed, excepting the single moment when the special activity *h* is executed;
- the model is again initialized (procedure *f*).

Let's consider the following grammar (considering the standard notational conventions and given by productions):

1. $S \to fA$
2. $A \to gA$
3. $A \to hB$
4. $B \to gB$
5. $B \to f$

We have the probability fields: $\mathcal{A}_1(S)$, $\mathcal{A}_2(A)$, $\mathcal{A}_3(B)$ and the corresponding stimulation operators: E^S, E^A, E^B.

The simplest and intuitivist way of writing a guiding algorithm for resolving a problem based on a given stochastic grammar, is to write a function for each nonterminal symbol of the grammar.

The decision for a new alternative is made by successive attempts, in decreasing order of the probabilities from the probability field corresponding to the affected nonterminal.

The procedures from $V_T = \{f, g, h\}$ will return TRUE if they have succeeded there task and otherwise FALSE. If a procedure returns TRUE, then the corresponding stimulation operator will be applied.

If all the alternatives in a function have failed then the function returns FALSE, otherwise TRUE.

The corresponding functions for the nonterminal symbols of the above stochastic grammar are presented in the following lines.

The function corresponding to the staring symbol "*S*":

```
S()
        IF(f())
                IF A() THEN
                        RETURN (TRUE)
                ELSE
                        RETURN (FALSE)
                ENDIF
        ELSE
                RETURN (FALSE)
        ENDIF
END
```

The function corresponding to the symbol "*A*":

```
A()
        (p_k1, p_k2) = SORT_DECREASE (p_1, p_2)
        I=1;
        DO WHILE ( I <= 2 )
                IF ( k_1 == 1 )
                        IF (g())
                                IF A() THEN
                                E^A(A_2(A),1)
                                RETURN (TRUE)
                                ELSE
                                RETURN (FALSE)
```

```
                    ENDIF
                ELSE
                RETURN (FALSE)
                ENDIF
            ELSE
                IF (h())
                    IF B() THEN
                    E^A(A_2(A),2)
                    RETURN (TRUE)
                    ELSE
                    RETURN (FALSE)
                    ENDIF
                ELSE
                RETURN (FALSE)
                ENDIF
            ENDIF
        ENDDO
        RETURN (FALSE)
END
```

The function corresponding to the symbol "*B*":

```
B()
        (p_k1, p_k2) = SORT_DECREASE (p_1, p_2)
        I=1;
        DO WHILE ( I <= 2 )
            IF ( k_1 == 1 )
                IF (g())
                    IF B() THEN
                    E^B(A_2(B),1)
                    RETURN (TRUE)
                    ELSE
                    RETURN (FALSE)
                    ENDIF
                ELSE
                RETURN (FALSE)
                ENDIF
            ELSE
                IF (f()) THEN
                    E^B(A_2(B),2)
                    RETURN (TRUE)
                ELSE
                    RETURN (FALSE)
                ENDIF
            ENDIF
        ENDDO
        RETURN (FALSE)
END
```

We notice that in each function the appropriate stimulation operator is applied only on success. Hence, over time the model will learn to follow the shortest way for solving a problem.

The initial probability fields $\mathcal{A}_2(A)$, $\mathcal{A}_2(B)$ will be:

$$\mathcal{A}_2(A) = \begin{pmatrix} A \to gA & A \to hB \\ 1/2 & 1/2 \end{pmatrix}$$

$$\mathcal{A}_2(B) = \begin{pmatrix} B \to gB & B \to f \\ 1/2 & 1/2 \end{pmatrix}$$

Hence, in the considered example, the corresponding language is

$$L(G) = \{fg^m hg^n f \mid m,n > 0\}$$

and so it is to be expected that after a sufficient amount of time elapsed, the probability fields $\mathcal{A}_2(A)$, $\mathcal{A}_2(B)$ will come very close to the values

$$\mathcal{A}_2(A) = \begin{pmatrix} A \to gA & A \to hB \\ 1 & 0 \end{pmatrix}$$

$$\mathcal{A}_2(B) = \begin{pmatrix} B \to gB & B \to f \\ 1 & 0 \end{pmatrix}$$

such that the entropy may approach 0. Consequently the first activity tried out by the system will be the *g* activity.

Practically, the function calls will synchronize along the advancing moments of the simulation clock.

3 Conclusions

An intelligent system must have a collection of algorithms that enables him to treat information in such way to obtain solutions for certain problems. With these algorithms the system processes the received data (input information) exploring his knowledgebase, which in turn is enriched under the "truth" effect determined by the operations described in the logical algorithms.

The system behavior in finding the solutions implies maximizing the usefulness and minimizing the risks. A system becomes more efficient as the problems he resolves are more complex, as the time necessary for resolving the problem is shorter and as the extrapolation provided over time is further in the future. All the operations that define the way of resolving the problem considered in a programming attempt, are achieved by the system due some algorithms that describe, step by step, all decisions and there logical conditions. Therefore an intelligent system must posses a collection of algorithms that

enables him to find the solution or solutions of the problems based on his states of information.
All these algorithms can be enclosed in our model in a set of functions F that represents the set of terminals from the stochastic grammar. Hence, in nature the solution for most of the problems involve vague and uncertain information it might be preferable a model based on fuzzy set and tools from probability theory.

References:

[1] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison Wesley, 2001.

[2] Vasile Crăciunean, *Proiectarea Translatoarelor*, Sibiu, Editura Alma Mater, 2002

[3] Marius Iosifescu, *Lanţuri Markov finite şi aplicaţii*, Editura tehnică, 1977

[4] Ernest G. Manes, Michael A. Arbib. *Algebraic Approaches to program semantics*, Springer Verlag New York Berlin Heidelberg London Paris Tokyo, 1986

[5] Marvin Scheaffer, *A Mathematical Theory of Global Program Optimization*, Prentice-Hall, 1973

[6] Salomaa, A. *Formal Languages*, New York, Academic Press, 1973.

[7] Creanga I., Reischer C., Simovici D. *Introducerea algebrică in informatică*., vol. 1 şi vol. 2, 1974

[8] Paun G., *Probleme actuale in teoria limbajelor formale*, Editura Academiei, 1983.

[9] Păun G., *Gramatici contextuale*, Bucureşti, 1982

[10] Gh. Păun, *Mecanisme generative ale proceselor economice*, Ed Tehnică, Bucureşti, 1988

[11] Seymour Ginsburg, *Mathematical Theory of Context Free Languages*, McGraw - Hill, New York, 1966

[12] Jürgen Dassow, Gheorghe Păun, Regulated *Rewriting in Formal Language Theory*, Akademie Verlag Berlin, 1989

[13] Seymour Ginsburg, *Algebraic and Automata Theoretic Properties of Formal Languages*, North Holland Company, 1975

[14] John E. Hopcroft, Jeffkey D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison Wesley Company, 1979