M.I.T Media Laboratory Perceptual Computing Section Technical Report No. 450

Probabilistic Parsing in Action Recognition

Y. A. Ivanov and A. F. Bobick
Room E15-383, The Media Laboratory
Massachusetts Institute of Technology
20 Ames St., Cambridge, MA 02139

Abstract

This report addresses the problem of using probabilistic formal languages to describe and understand actions with explicit structure. The paper explores a probabilistic mechanisms of parsing the uncertain input string aided by a stochastic context-free grammar. This method, originating in speech recognition, allows for combination of a statistical recognition approach with a syntactical one in a unified syntactic-semantic framework for action recognition.

The basic approach is to design the recognition system in a two-level architecture. The first level, a set of independently trained component event detectors, produces the likelihoods of each component model. The outputs of these detectors provide the input stream for a stochastic context-free parsing mechanism. Any decisions about supposed structure of the input are deferred to the parser, which attempts to combine the maximum amount of the candidate events into a most likely sequence according to a given Stochastic Context-Free Grammar (SCFG). The grammar and parser enforce longer range temporal constraints, disambiguate or correct uncertain or mis-labeled low level detections, and allow the inclusion of a priori knowledge about the structure of temporal events in a given domain.

The method takes into consideration the continuous character of the input and performs "structural rectification" of it in order to account for misalignments and ungrammatical symbols in the stream. The presented technique of such a rectification uses the structure probability maximization to drive the segmentation.

1 Introduction

1.1 Structure and Content

Our research interests lie in the area of vision where observations span extended periods of time. We often find ourselves in a situation where it is difficult to formulate and learn parameters of a model in clear statistical terms, which prevents us from using purely statistical approaches to recognition. These situations can be characterized by one or more of the following properties:

- complete data sets are not always available, but component examples are easily found;
- semantically equivalent processes possess radically different statistical properties;
- competing hypotheses can absorb different lengths of the input stream raising the need for naturally supported temporal segmentation;
- structure of the process is difficult to learn but is explicit and a priori known;
- the process' structure is too complex, and the limited Finite State model which is simple to estimate, is not always adequate for modeling the process' structure.

Many applications exist where purely statistical representational capabilities are limited and surpassed by the capabilities of the structural approach.

Syntactic Recognition

Structural methods are based on the fact that often the significant information in a pattern is not merely in the presence, absence or numerical value of some feature set. Rather, the interrelations or interconnections of features yield most important information, which facilitates structural description and classification. One area, which is of particular interest is the domain of syntactic pattern recognition. Typically, the syntactic approach formulates hierarchical descriptions of complex patterns built up from simpler sub-patterns. At the lowest level, the primitives of the pattern are extracted from the input data. The choice of the primitives distinguishes the syntactical approach from any other. The main difference is that the features are just any available measurement which characterizes the input data. The primitives, on the other hand, are sub-patterns or building blocks of the structure.

In this work we refrain from answering the question why Syntactic Pattern Recognition techniques fell out of favor with image processing community. We believe that in the domain of action recognition, where the character of the input is clearly sequential, the solution yielded by a sequential machines is called for and well justified.

We motivate Syntactic Recognition approach by the apparent granularity of the processes, recognition of which we are attempting to address in our ongoing research. Indeed, in cognitive vision we can often easily determine the components of the complex behavior–based signal. The assumption is perhaps more valid in the context of conscious directed activity since the nature of the signal is such that it is generated by a task-driven system, and the decomposition of the input to the vision system is as valid as the decomposition of the task, being performed by the observed source, into a set of simpler primitive subtasks (eg. [5, 14, 7]).

AI view

AI approaches to recognition were traditionally based on the idea of incorporating the knowledge from a variety of knowledge sources and bringing them together to bear on the problem at hand. For instance, the AI approach to

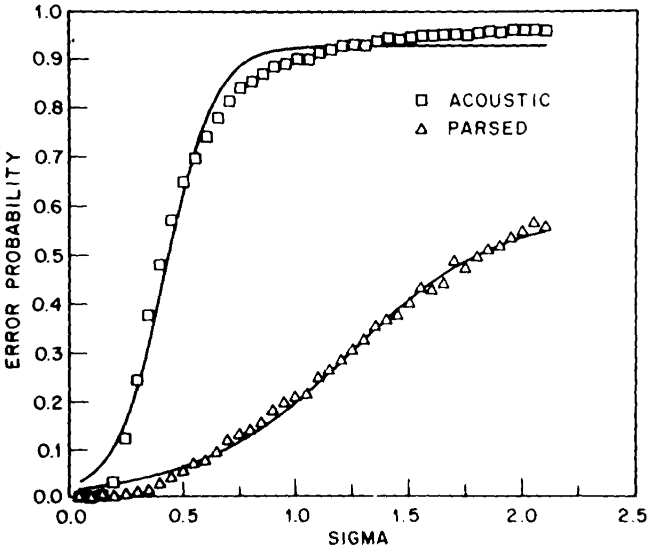


Figure 1: Error correcting capabilities of syntactic constraints

segmentation and labeling of the speech signal is to augment the generally used acoustic knowledge with phonemic knowledge, lexical knowledge, syntactic knowledge, semantic knowledge and even pragmatic knowledge.

Rabiner and Juang ([17]) show that in tasks of automatic speech recognition, the word correcting capability of higher-level knowledge sources is extremely powerful. In particular, they illustrate this statement by comparing performance of a recognizer both with and without syntactic constraints (figure[1] 1). As the deviation (noise) parameter $SIGMA$ gets larger, the word error probability increases in both cases. In the case without syntactic constraints, the error probability quickly leads to 1, but with the syntactical constraints enforced, it increases gradually with an increase in noise.

In this report we present a framework in which syntactic constraints can be enforced in tasks of machine vision. This framework makes it simple to utilize the heterogeneous set of the knowledge sources at appropriate levels of abstraction, while retaining the simplicity and elegance of probabilistic Finite State models, without the necessity to correspondingly limit the model's complexity.

2 Related Work

Probabilistic aspects of syntactic pattern recognition for speech processing were presented in many publications, for instance in [13, 10]. The latter demonstrates some key approaches to parsing sentences of natural language and shows advantages of use of probabilistic CFGs. The text shows natural progression from HMM-based methods to probabilistic CFGs, demonstrating the techniques of computing the sequence probability characteristics, familiar from HMMs, such as forward and backward probabilities in the chart parsing framework.

An efficient probabilistic version of Earley parsing algorithm was developed by Stolcke in his dissertation ([20]). The author develops techniques of embedding the probability computation and maximization into the Earley algorithm. He also describes grammar structure learning strategies and the rule probability learning technique, justifying usage of Stochastic Context-Free Grammars for natural language processing and learning.

Aho and Peterson addressed the problem of ill-formedness of the input stream. In [1] they described a modified Earley's parsing algorithm where substitution, insertion and deletion errors are corrected. The basic idea is to augment the original grammar by error productions for insertions, substitutions and deletions, such that any string over the terminal alphabet can be generated by the augmented grammar. Each such production has some cost associated with it. The parsing proceeds in such a manner as to make the total cost minimal. It has been shown that the error correcting Earley parser has the same time and space complexity as the original version, namely $O(N^3)$ and $O(N^2)$ respectively, where N is the length of the string. Their approach is utilized in this thesis in the framework of uncertain input and multi-valued strings.

The syntactic approach in Machine Vision has been studied for more than thirty years (Eg. [15, 3]), mostly in the context of pattern recognition in still images. The work by Bunke and Pasche ([6]) is built upon the previously mentioned development by Aho and Peterson ([1]), expanding it to multi-valued input. The resulting method is suitable for recognition of patterns in distorted input data and is shown in applications to waveform and image analysis. The work proceeds entirely in non-probabilistic context.

More recent work by Sanfeliu *et al* ([19]) is centered around two-dimensional grammars and their applications to image analysis. The authors pursue the task of automatic traffic sign detection by a technique based on Pseudo Bidimensional Augmented Regular Expressions (PSB-ARE). AREs are regular expressions augmented with a set of constraints that involve the number of instances in a string of the operands to the star operator, alleviating the limitations of the traditional FSMs and CFGs which cannot count their arguments. More theoretical treatment of the approach is given in [18]. In the latter work, the authors introduce a method of parsing AREs which describe a subclass of a context-sensitive languages, including the ones defining planar shapes with symmetry.

A very important theoretical work, signifying an emerging information theoretic trend in stochastic parsing, is demonstrated by Oomen and Kashyap in [16]. The authors present a foundational basis for optimal and information theoretic syntactic pattern recognition. They develop a rigorous model for channels which permit arbitrary distributed substitution, deletion and insertion syntactic errors. The scheme is shown to be functionally complete and stochastically consistent.

There are many examples of attempts to enforce syntactic and semantic constraints in recognition of visual data. For instance, Courtney ([8]) uses a structural approach to interpreting action in a surveillance setting. Courtney defines high level discrete events, such as "object appeared",

[1] Reprinted from [17]

"object disappeared" etc., which are extracted from the visual data. The sequences of the events are matched against a set of heuristically determined sequence templates to make decisions about higher level events in the scene, such as "object A removed from the scene by object B".

The grammatical approach to visual activity recognition was used by Brand ([4]), who used a simple non-probabilistic grammar to recognize sequences of discrete events. In his case, the events are based on blob interactions, such as "objects overlap" etc.. The technique is used to annotate a manipulation video sequence, which has an a priori known structure.

And finally, hybrid techniques of using combined probabilistic-syntactic approaches to problems of image understanding are shown in pioneering research by Fu (eg.[22, 12]).

3 Stochastic Context-Free Grammars

The probabilistic aspect is introduced into syntactic recognition tasks via Stochastic Context-Free Grammars. A *Stochastic Context-Free Grammar* (SCFG) is a probabilistic extension of a Context-Free Grammar. The extension is implemented by adding a probability measure to every production rule:

$$A \rightarrow \lambda \quad [p]$$

The rule probability p is usually written as $P(A \rightarrow \lambda)$. This probability is a conditional probability of the production being chosen, given that non-terminal A is up for expansion (in generative terms). Saying that stochastic grammar is context-free essentially means that the rules are conditionally independent and, therefore, the probability of the complete derivation of a string is just a product of the probabilities of rules participating in the derivation.

Given a SCFG G, let us list some basic definitions:

1. The *probability of partial derivation* $\nu \Rightarrow \mu \Rightarrow \ldots \lambda$ is defined in inductive manner as
   (a) $P(\nu) = 1$
   (b) $P(\nu \Rightarrow \mu \Rightarrow \ldots \lambda) = P(A \rightarrow \omega)P(\mu \Rightarrow \ldots \lambda)$, where production $A \rightarrow \omega$ is a production of G, μ is derived from ν by replacing one occurrence of A with ω, and $\nu, \mu, \ldots, \lambda \in V_G^*$.
2. The *string probability* $P(A \Rightarrow^* \lambda)$ (Probability of λ given A) is the sum of all left-most derivations $A \Rightarrow \ldots \Rightarrow \lambda$.
3. The *sentence probability* $P(S \Rightarrow^* \lambda)$ (Probability of λ given G) is the string probability given the axiom S of G. In other words, it is $P(\lambda|G)$.
4. The *prefix probability* $P(S \Rightarrow_L^* \lambda)$ is the sum of the strings having λ as a prefix,

$$P(S \Rightarrow_L^* \lambda) = \sum_{\omega \in V_G^*} P(A \Rightarrow^* \lambda\omega)$$

   In particular, $P(S \Rightarrow_L^* \epsilon) = 1$.

4 Earley-Stolcke Parsing Algorithm

The method most generally and conveniently used in stochastic parsing is based on an Earley parser ([11]), extended in such a way as to accept probabilities.

In parsing stochastic sentences we adopt a slightly modified notation of [20]. The notion of a *state* is an important part of the Earley parsing algorithm. A state is denoted as:

$$i : X_k \rightarrow \lambda.Y\mu$$

where '**.**' is the marker of the current position in the input stream, i is the index of the marker, and k is the starting index of the substring denoted by nonterminal X. Nonterminal X is said to dominate substring $w_k \ldots w_i \ldots w_l$, where, in the case of the above state, w_l is the last terminal of substring μ.

In cases where the position of the dot and structure of the state is not important, for compactness we will denote a state as:

$$S_k^i \equiv i : X_k \rightarrow \lambda.Y\mu$$

Parsing proceeds as an iterative process sequentially switching between three steps - *prediction*, *scanning* and *completion*. For each position of the input stream, an Earley parser keeps a set of *states*, which denote all pending derivations. States produced by each of the parsing steps are called, respectively, *predicted*, *scanned* and *completed*. A state is called *complete* (not to be confused with *completed*), if the dot is located in the rightmost position of the state. A complete state is the one that "passed" the grammaticality check and can now be used as a "ground truth" for further abstraction. A state "explains" a string that it dominates as a possible interpretation of symbols $w_k \ldots w_i$, "suggesting" a possible continuation of the string if the state is not complete.

4.1 Prediction

In the Earley parsing algorithm the prediction step is used to hypothesize the possible continuation of the input based on current position in the parse tree. Prediction essentially expands one branch of the parsing tree down to the set of its leftmost leaf nodes to predict the next possible input terminal. Using the state notation above, for each state S_k^i and production p of a grammar $G = \{T, N, S, P\}$ of the form

$$\left\{ \begin{array}{lll} S_k^i & : & i : X_k \rightarrow \lambda.Y\mu \\ p & : & Y \rightarrow \nu \end{array} \right. \tag{1}$$

where $Y \in N$, we predict a state

$$S_i^i : \quad i : Y_i \rightarrow .\nu$$

Prediction step can take on a probabilistic form by keeping track of the probability of choosing a particular predicted state. Given the statistical independence of the nonterminals of the SCFG, we can write the probability of predicting a state S_i^i as conditioned on probability of S_k^i. This introduces a notion of forward probability, which has the interpretation similar to that of a forward probability in HMMs. In SFCG the forward probability α_i is the probability of the parser accepting the string $w_1 \ldots w_{(i-1)}$ and choosing state S at a step i. To continue the analogy with

HMMs, inner probability, γ_i, is a probability of generating a substring of the input from a given nonterminal, using a particular production. Inner probability is thus conditional on the presence of a nonterminal X with expansion started at the position k, unlike the forward probability, which includes the generation history starting with the axiom. Formally, we can integrate computation of α and γ with non-probabilistic Earley predictor as follows:

$$\left\{ \begin{array}{l} i : X_k \rightarrow \lambda.Y\mu \quad [\alpha, \gamma] \\ Y \rightarrow \nu \end{array} \right. \quad \Longrightarrow \quad i : Y \rightarrow .\nu \quad [\alpha', \gamma']$$

where α' is computed as a sum of probabilities of all the paths, leading to the state $i : X_k \rightarrow \lambda.Y\mu$ multiplied by the probability of choosing the production $Y \rightarrow \nu$, and γ' is the rule probability, seeding the future substring probability computations:

$$\begin{array}{rcl} \alpha' & = & \sum_{\forall\lambda,\mu} \alpha(i : X_k \rightarrow \lambda.Y\mu) P(Y \rightarrow \nu) \\ \gamma' & = & P(Y \rightarrow \nu) \end{array}$$

4.1.1 Recursive correction

Because of possible left recursion in the grammar the total probability of a predicted state can increase with it being added to a set infinitely many times. Indeed, if a non-terminal A is considered for expansion and given productions for A

$$\begin{array}{rcl} A & \rightarrow & Aa \\ & \rightarrow & a \end{array}$$

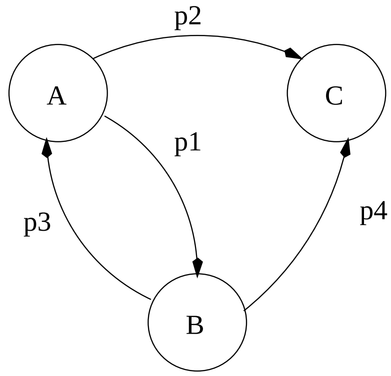
we will predict $A \rightarrow .a$ and $A \rightarrow .Aa$ at the first prediction step. This will cause the predictor to consider these newly added states for possible descent, which will produce the same two states again. In non-probabilistic Earley parser it normally means that no further expansion of the production should be made and no duplicate states should be added. In probabilistic version, however, although adding a state will not add any more information to the parse, its probability has to be factored in by adding it to the probability of the previously predicted state. In the above case that would mean an infinite sum due to left-recursive expansion.

In order to demonstrate the solution we first need to introduce the concept of Left Corner Relation.

Two nonterminals are said to be in a Left Corner Relation $X \rightarrow_L Y$ iff there exists a production for X of the form $X \rightarrow Y\lambda$.

We can compute a total recursive contribution of each left-recursive production rule where nonterminals X and Y are in Left Corner Relation. The necessary correction for non-terminals can be computed in a matrix form. The form of recursive correction matrix R_L can be derived using simple example presented in Figure 2. The graph of the relation presents direct left corner relations between nonterminals of G_1. The LC Relation matrix P_L of the grammar G_1 is essentially the adjacency matrix of this graph. If there exists an edge between two nonterminals X and Y it follows that the probability of emitting terminal a, such that

$$\left\{ \begin{array}{l} X \rightarrow Y\nu \\ Y \rightarrow a\eta \end{array} \right.$$



G_1:

$$\begin{array}{rcll} A & \rightarrow & BB & [p_1] \\ & \rightarrow & CB & [p_2] \\ B & \rightarrow & AB & [p_3] \\ & \rightarrow & C & [p_4] \\ C & \rightarrow & a & [p_5] \end{array} \qquad \left| \begin{array}{ccc} 0 & p_1 & p_2 \\ p_3 & 0 & p_4 \\ 0 & 0 & 0 \end{array} \right|$$

Figure 2: Left Corner Relation graph of the grammar G_1. Matrix P_L is shown on the right of the productions of the grammar.

G_2:

$$\begin{array}{rcll} S & \rightarrow & AB & [p_1] \\ & \rightarrow & C & [p_2] \\ & \rightarrow & d & [p_3] \\ A & \rightarrow & AB & [p_4] \\ & \rightarrow & a & [p_5] \\ B & \rightarrow & bB & [p_6] \\ & \rightarrow & b & [p_7] \\ C & \rightarrow & BC & [p_8] \\ & \rightarrow & B & [p_9] \\ & \rightarrow & c & [p_{10}] \end{array}$$

P_L	S	A	B	C
S		p_1		p_2
A		p_4		
B				
C			$p_8 + p_9$	

R_L	S	A	B	C
S	1	$\frac{-p_1}{-1+p_4}$	$p_2 p_8 + p_2 p_9$	p_2
A		$\frac{-1}{-1+p_4}$		
B			1	
C			$p_8 + p_9$	1

Table 1: Left Corner Relation (P_L) and its Reflexive Transitive Closure (R_L) matrices for a simple SCFG.

(from which follows that $X \rightarrow a\eta\nu$) is a sum of probabilities along all the paths connecting X with Y, multiplied by probability of direct emittance of a from Y, $P(Y \rightarrow a\eta)$.

Formally,

$$\begin{array}{rcll} P(a) & = & P(Y \rightarrow a\eta) & \sum_{\forall X} P(X \Rightarrow^* Y) \\ & = & P(Y \rightarrow a\eta) & (P_0(X \Rightarrow^* Y) + \\ & & & P_1(X \Rightarrow^* Y) + \\ & & & P_2(X \Rightarrow^* Y) + \ldots \end{array}$$

where $P_k(X \Rightarrow Y)$ is probability of a path from X to Y of length $k = 1, \ldots, \infty$

In matrix form all the $k-$step path probabilities on a graph can be computed from its adjacency matrix as P_L^k. Using the matrix P_L^k, the reflexive transitive closure of the LC relation can be easily found as a matrix of infinite sums, R_L, which has a closed form solution:

$$R_L = P_L^0 + P_L^1 + P_L^2 + \ldots = \sum_{k=0}^{\infty} P_L^k = (I - P_L)^{-1}$$

Thus, the correction to the completion step should be applied by descending the chain of left corner productions, indicated by a non-zero value in R_U:

$$\left\{ \begin{array}{l} i: X_k \rightarrow \lambda.Z\mu \quad [\alpha,\gamma] \\ \forall Z \quad s.t. \quad R_L(Z,Y) \neq 0 \\ Y \rightarrow \nu \end{array} \right. \quad \Longrightarrow \quad i: Y \rightarrow .\nu \quad [\alpha',\gamma']$$

where forward and inner probabilities for the left recursive productions are corrected by an appropriate entry of the matrix R_L:

$$\begin{array}{rcl} \alpha' & = & \sum_{\forall\lambda,\mu} \alpha(i: X_k \rightarrow \lambda.Z\mu) R_L(Z,Y) P(Y \rightarrow \nu) \\ \gamma' & = & P(Y \rightarrow \nu) \end{array}$$

Matrix R_L can be computed once for the grammar and used at each iteration of the prediction step.

4.2 Scanning

Scanning step simply reads the input symbol and matches it against all pending states for the next iteration. For each state $X_k \rightarrow \lambda.a\mu$ and the input symbol a we generate a state $i+1: X_k \rightarrow \lambda a.\mu$:

$$i: X_k \rightarrow \lambda.a\mu \quad \Longrightarrow i+1: X_k \rightarrow \lambda a.\mu$$

It is readily converted to the probabilistic form. The forward and inner probabilities remain unchanged from the state being confirmed by the input, since no selections are made at this step. The probability, however, may change if there is a likelihood associated with the input terminal. This will be exploited in the next chapter dealing with uncertain input. In probabilistic form the scanning step is formalized as:

$$i: X_k \rightarrow \lambda.a\mu \quad [\alpha,\gamma] \quad \Longrightarrow i+1: X_k \rightarrow \lambda a.\mu \quad [\alpha,\gamma]$$

where α and γ are forward and inner probabilities.

Note the increase in i index. This signifies the fact that scanning step inserts the states into the new state set for the next iteration of the algorithm.

4.3 Completion

Completion step, given a set of states which just have been confirmed by scanning, updates the positions in all the pending derivations all the way up the derivation tree. The position in expansion of a pending state $j: X_k \rightarrow \lambda.Y\mu$ is advanced if there is a state, starting at position j, $i: Y_j \rightarrow \nu.$, which consumed all the input symbols related to it. Such a state can now be used to confirm other states, expecting Y as their next non-terminal. Since the index range is attached to Y, we can effectively limit the search for the pending derivations to the state set, indexed by the starting index of the completing state, j:

$$\left\{ \begin{array}{l} j: X_k \rightarrow \lambda.Y\mu \\ i: Y_j \rightarrow \nu. \end{array} \right. \Longrightarrow i: X_k \rightarrow \lambda Y.\mu$$

Again, propagation of forward and inner probabilities is simple. New values of α and γ are computed by multiplying the corresponding probabilities of the state being completed by the total probability of all paths, ending at $i: Y_j \rightarrow \nu.$. In its probabilistic form, completion step generates the following states:

G_2:

$$\begin{array}{lcll} S & \rightarrow & AB & [p_1] \\ & \rightarrow & C & [p_2] \\ & \rightarrow & d & [p_3] \\ A & \rightarrow & AB & [p_4] \\ & \rightarrow & a & [p_5] \\ B & \rightarrow & bB & [p_6] \\ & \rightarrow & b & [p_7] \\ C & \rightarrow & BC & [p_8] \\ & \rightarrow & B & [p_9] \\ & \rightarrow & c & [p_{10}] \end{array}$$

P_U	S	A	B	C
S				p_2
A				
B				
C			p_9	

R_U	S	A	B	C
S	1		p_2p_9	p_2
A		1		
B			1	
C			p_9	1

Table 2: Unit Production (P_U) and its Reflexive Transitive Closure (R_U) matrices for a simple SCFG.

$$\left\{ \begin{array}{l} j: X_k \rightarrow \lambda.Y\mu \quad [\alpha,\gamma] \\ i: Y_j \rightarrow \nu. \quad [\alpha'',\gamma''] \end{array} \right. \Longrightarrow i: X_k \rightarrow \lambda Y.\mu \quad [\alpha',\gamma']$$

$$\begin{array}{rcl} \alpha' & = & \sum_{\forall\lambda,\mu} \alpha(i: X_k \rightarrow \lambda.Y\mu)\gamma''(i: Y_j \rightarrow \nu.) \\ \gamma' & = & \sum_{\forall\lambda,\mu} \gamma(i: X_k \rightarrow \lambda.Y\mu)\gamma''(i: Y_j \rightarrow \nu.) \end{array}$$

4.3.1 Recursive correction

Here, as in prediction, we might have a potential danger of entering an infinite loop. Indeed, given the productions

$$\begin{array}{lcl} A & \rightarrow & \mathrm{B} \\ & \rightarrow & \mathrm{a} \\ B & \rightarrow & \mathrm{A} \end{array}$$

and the state $i: A_j \rightarrow a.$, we complete the state set j, containing:

$$\begin{array}{lcl} j: A_j & \rightarrow & .\ \mathrm{B} \\ j: A_j & \rightarrow & .\ \mathrm{a} \\ j: B_j & \rightarrow & .\ \mathrm{A} \end{array}$$

Among others this operation will produce another state $i: A_j \rightarrow a.$, which will cause the parse to go into infinite loop. In non-probabilistic Earley parser that would mean that we just simply do not add the newly generated states and proceed with the rest of them. However, this will introduce the truncation of the probabilities as in the case with prediction. It has been shown that this infinite loop appears due to so-called *unit productions* ([21]).

Two nonterminals are said to be in a Unit Production Relation $X \rightarrow_U Y$ iff there exists a production for X of the form $X \rightarrow Y$.

As in the case with prediction we compute the closed-form solution for a Unit Production recursive correction matrix R_U (figure 2), considering the Unit Production relation, expressed by a matrix P_U. R_U is found as $R_U = (I - P_U)^{-1}$. The resulting expanded completion algorithm accommodates for the recursive loops:

$$\left\{ \begin{array}{l} j: X_k \rightarrow \lambda.Z\mu \quad [\alpha,\gamma] \\ \forall Z \quad s.t. \quad R_U(Z,Y) \neq 0 \\ i: Y_j \rightarrow \nu. \quad [\alpha'',\gamma''] \end{array} \right. \Longrightarrow i: X_k \rightarrow \lambda Z.\mu \quad [\alpha',\gamma']$$

where computation of α' and γ' is corrected by a corresponding entry of R_U:

$$\alpha' = \sum_{\forall\lambda,\mu} \alpha(i : X_k \to \lambda.Z\mu) R_U(Z,Y) \gamma''(i : Y_j \to \nu.)$$
$$\gamma' = \sum_{\forall\lambda,\mu} \gamma(i : X_k \to \lambda.Y\mu) R_U(Z,Y) \gamma''(i : Y_j \to \nu.)$$

As R_L, unit production recursive correction matrix R_U can be computed once for each grammar.

5 Temporal Parsing of Uncertain Input

The approach described in the previous chapter can be effectively used to find the most likely syntactic groupings of the elements in a non-probabilistic input stream. The probabilistic character of the grammar comes into play when disambiguation of the string is required by the means of the probabilities attached to each rule. These probabilities reflect the "typicality" of the input string. By tracing the derivations, to each string we can assign a value, reflecting how typical the string is according to the current model.

In this chapter we to address two main problems in application of the parsing algorithm described so far to action recognition:

- Uncertainty in the input string.
  The input symbols which are formed by low level temporal feature detectors are uncertain. This implies that some modifications to the algorithm which account for this are necessary.
- Temporal consistency of the event stream.
  Each symbol of the input stream corresponds to some time interval. Since here we are dealing with a single stream input, only one input symbol can be present in the stream at a time and no overlapping is allowed, which the algorithm must enforce.

5.1 Terminal Recognition

Before we proceed to develop the temporal parsing algorithm, we need to say a few words about the formation of the input. In this thesis we are attempting to combine the detection of independent component activities in a framework of syntax-driven structure recognition. Most often these components are detected "after the fact", that is, recognition of the activity only succeeds when the whole corresponding sub-sequence is present in the causal signal. At the point when the activity is detected by the low level recognizer, the likelihood of the activity model represents the probability that this part of the signal has been generated by the corresponding model. In other words, with each activity likelihood, we will also have the sample range of the input signal, or a corresponding time interval, to which it applies. This fact will be the basis of the technique for enforcing temporal consistency of the input, developed later in this chapter. We will refer to the model activity likelihood as a *terminal*, and to the length of corresponding sub-sequence as a *terminal length*.

5.2 Uncertainty in the Input String

The previous section described the parsing algorithm, where no uncertainty about the input symbols is taken into consideration. New symbols are read off by the parser during the scanning step, which changes neither forward nor inner probabilities of the pending states. If the *likelihood of the input symbol* is also available, as in our case, it can easily be incorporated into the parse during scanning by multiplying the inner and forward probability of the state being confirmed by the input likelihood. We reformulate the scanning step as follows: having read a symbol a and its likelihood $P(a)$ off the input stream, we produce the state

$$i : X_k \to \lambda.a\mu \quad [\alpha,\gamma] \implies i+1 : X_k \to \lambda a.\mu \quad [\alpha',\gamma']$$

and compute new values of α' and γ' as:

$$\alpha' = \alpha(i : X_k \to \lambda.a\mu)P(a)$$
$$\gamma' = \gamma(i : X_k \to \lambda.a\mu)P(a)$$

The new values of forward and inner probabilities will weigh competing derivations not only by their typicality, but also by our certainty about the input at each sample.

5.3 Substitution

Introducing likelihoods of the terminals at the scanning step makes it simple to employ the *multi-valued string* parsing ([6]) where each element of the input is in fact a multi-valued vector of vocabulary model likelihoods at each time step of an event stream. Using these likelihoods, we can condition the maximization of the parse not only on frequency of occurrence of some sequence in the training corpus, but also on measurement or estimation of the likelihood of each of the sequence component in the test data.

The multi-valued string approach allows for dealing with so-called *substitution* errors which manifest themselves in replacement of a terminal in the input string with an incorrect one. It is especially relevant to the problems addressed by this thesis where sometimes the correct symbol is rejected due to a lower likelihood than that of some other one. In the proposed approach, we allow the input at discrete instance of time to include all non-zero likelihoods and to let the parser select the most likely one, based not only on the corresponding value, but also on the probability of the whole sequence, as additional reinforcement to the local model estimate.

From this point and on, the input stream will be viewed as a multi-valued string (a lattice) which has a vector of likelihoods, associated with each time step (e.g. figure 3). The length of the vector is in general equal to the number of vocabulary models. The model likelihoods are incorporated with the parsing process at the scanning step, as was shown above, by considering all non-zero likelihoods for the same state set, that is:

$$\left\{ \begin{array}{ll} i : X_k \to \lambda.a\mu & [\alpha,\gamma] \\ \forall a, \quad s.t. & P(a) > 0 \end{array} \right. \implies i+1 : X_k \to \lambda a.\mu \quad [\alpha',\gamma']$$

The parsing is performed in a parallel manner for all suggested terminals.

5.4 Insertion

It has been shown that "while significant progress has been made in the processing of correct text, a practical system must be able to handle many forms of ill-formedness gracefully" [2]. In our case, the need for this robustness is paramount.

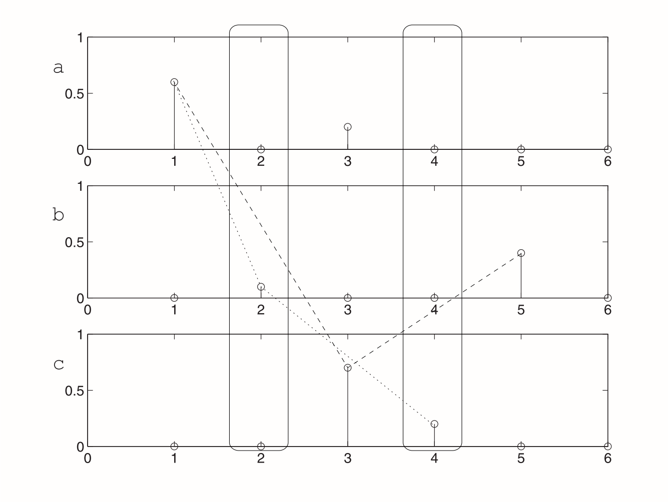


Figure 3: Example of the lattice parse input for three model vocabulary. Consider a grammar $A \rightarrow abc \mid acb$. The dashed line shows a parse acb. The two rectangles drawn around samples 2 and 4 show the "spurious symbols" for this parse which need to be ignored for the derivation to be contiguous. We can see that if the spurious symbols are simply removed from the stream, an alternative derivation for the sequence - abc, shown by a dotted line, will be interrupted. (Note the sample 3 which contains two concurrent symbols which are handled by the lattice parsing.)

Indeed, the parsing needs to be performed on a lattice, where the symbols which we need to consider for inclusion in the string come at random times. This results in appearance of "spurious" (i.e. ungrammatical) symbols in the input stream, a problem commonly referred to as *insertion*. We need to be able to consider these inputs separately, at different time steps, and disregard them if their appearance in the input stream for some derivation is found to be ungrammatical. At the same time, we need to preserve the symbol in the stream for considering it in other possible derivations, perhaps even of a completely different string. The problem is illustrated by figure 3. In this figure we observe two independent derivations on an uncertain input stream. While deriving the string acb, connecting samples 1, 3 and 5, we need to disregard samples 2 and 4, which would interfere with the derivation. However we do not have an a priori information about the overall validity of this string, that is, we cannot simply discard samples 2 and 4 from the stream, because combined with sample 1, they form an alternative string, abc. For this alternative derivation, sample 3 would present a problem if not dealt with.

There are at least three possible solutions to the insertion problem in our context.

- Firstly, we can employ some *ad hoc* method which "synchronizes" the input, presenting the terminals coming with a slight spread around a synchronizing signal, as one vector. That would significantly increase the danger of not finding a parse if the spread is significant.
- Secondly, we may attempt gathering all the partially valid strings, performing all the derivations we can on the input and post-processing the resulting set to extract the most consistent maximum probability set of partial derivations. Partial derivations acquired by this technique will show a distinct split at the position in the input where the supposed ungrammaticality occurred. Being robust for finding ungrammatical symbols in a single stream, in our experiments, this method did not produce the desired results. The lattice extension, extremely noisy input and a relatively shallow grammar made it less useful, producing large number of string derivations, which contained a large number of splits and were difficult to analyze.
- And, finally, we can attempt to modify the original grammar to explicitly include the ERROR symbol (e.g. [1]).

In our algorithm the last approach proved to be the most suitable as it allowed us to incorporate some additional constraints on the input stream easily, as will be shown in the next section.

We formulate simple rules of the grammar modifications and perform the parsing of the input stream using this modified grammar.

Given the grammar G, robust grammar $\hat{G}$ is formed by following rules:

1. Each terminal in productions of G is replaced by a pre-terminal in $\hat{G}$:
$$\begin{array}{lcl} G: & \Rightarrow & \hat{G}: \\ A \rightarrow bC & & A \rightarrow \hat{B}C \end{array}$$
2. For each pre-terminal of $\hat{G}$ a skip rule is formed:
$$\begin{array}{lcl} \hat{G}: & & \\ \hat{B} & \rightarrow & b \mid SKIP\ b \mid b\ SKIP \end{array}$$
This is essentially equivalent to adding a production $\hat{B} \rightarrow SKIP\ b\ SKIP$ if $SKIP$ is allowed to expand to ϵ.
3. Skip rule is added to $\hat{G}$, which includes all repetitions of all terminals:
$$\begin{array}{lcl} \hat{G}: & & \\ SKIP & \rightarrow & b \mid c \mid \ldots \\ & & \mid b\ SKIP \mid c\ SKIP \\ & & \ldots \end{array}$$

Again, if $SKIP$ is allowed to expand to ϵ, the last two steps are equivalent to adding:

$$\begin{array}{lcl} \hat{G}: & & \\ \hat{B} & \rightarrow & SKIP\ b\ SKIP \\ SKIP & \rightarrow & \epsilon \mid b\ SKIP\ \ldots \end{array}$$

This process can be performed automatically as a preprocessing step, when the grammar is read in by the parser, so that no modifications to the grammar are explicitly written.

6 Enforcing Temporal Consistency

Parsing the lattice with the robust version of the original grammar results in the parser consuming a maximum amount of the input. Indeed, the $SKIP$ production, which

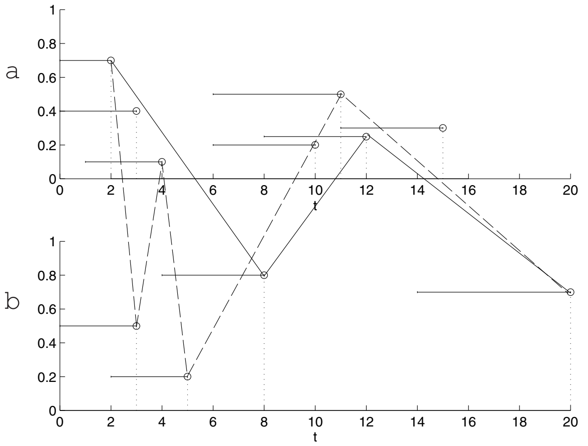


Figure 4: Example of temporally inconsistent terminals. Given a production $A \rightarrow ab \mid abA$ unconstrained parse will attempt to consume maximum amount of samples by non-$SKIP$ productions. The resulting parse, $ababab$, is shown by the dashed line. The solid line shows the correct parse, $abab$, which includes only non-overlapping terminals (horizontal lines show the sample range of each candidate terminal).

is usually assigned a low probability, serves as a function that "penalizes" the parser for each symbol considered ungrammatical. This might result in the phenomenon where some of the "noise" gets forced to take part in a derivation as a grammatical occurrence. The effect of this can be relieved if we take into account the temporal consistency of the input stream, that is, only one event happens at a given time. Since each event has a corresponding length, we can further constrain the occurrence of the terminals in the input so that no "overlapping" events take place. The problem is illustrated in figure 4.

In Earley framework, we can modify the parsing algorithm to work in incremental fashion as a filter to the completion step, while keeping track of the terminal lengths during scanning and prediction.

In order to accomplish the task we need to introduce two more state variables - h for "high mark" and l for "low mark". Each of the parsing steps has to be modified as follows:

6.1 Scanning

Scanning step is modified to include reading the time interval associated with the incoming terminal. The updates of l and h are propagated during the scanning as follows:

$$i : X_k \rightarrow \lambda.a\mu \quad [l, h] \quad \Longrightarrow \quad i+1 : X_k \rightarrow \lambda a.\mu \quad [l, h_a]$$

where h_a is a high mark of the terminal. In addition to this, we set the time-stamp of the whole new $(i+1)$-th state set to h_a: $S_t = h_a$, to be used later by prediction step.

6.2 Completion

Similarly to scanning, completion step advances the high mark of the completed state to that of the completing state, thereby extending the range of the completed non-terminal.

$$\left\{ \begin{array}{l} j : X_k \rightarrow \lambda.Y\mu \quad [l1, h1] \\ i : Y_j \rightarrow \nu. \quad [l2, h2] \end{array} \right. \quad \Longrightarrow \quad i : X_k \rightarrow \lambda Y.\mu \quad [l_1, h_2]$$

This completion is performed for all states $i : Y_j \rightarrow \nu.$ subject to constraints $l_2 \geq h_1$ and $Y, X \neq SKIP$.

6.3 Prediction

Prediction step is responsible for updating the low mark of the state to reflect the timing of the input stream.

$$\left\{ \begin{array}{l} i : X_k \rightarrow \lambda.Y\mu \\ Y \rightarrow \nu \end{array} \right. \quad \Longrightarrow \quad i : Y \rightarrow .\nu \quad [S_t, S_t]$$

Here, S_t is the time-stamp of the state set, updated by the scanning step.

6.4 Misalignment Cost

The essence of the filtering technique described in the previous section is that only the paths that are consistent in the timing of their terminal symbols are considered. This does not interrupt the parses of the sequence since filtering, combined with robust grammar, leaves the subsequences which form the parse connected via the skip states.

In the process of temporal filtering, it is important to remember that the terminal lengths are themselves uncertain. A useful extension to the temporal filtering is to implement a softer penalty function, rather than a hard cut-off of all the overlapping terminals, as described above. It is easily achieved at the completion step where the filtering is replaced by a weighting function $f(\theta)$, of a parameter $\theta = h_1 - l_2$, which is multiplied into forward and inner probabilities. For instance, $f(\theta) = e^{-\psi\theta^2}$, where ψ is a penalizing parameter, can be used to weigh "overlap" and "spread" equally. The resulting penalty is incorporated with computation of α and γ:

$$\begin{array}{lcl} \alpha' & = & f(\theta) \sum_{\forall \lambda, \mu} \alpha(i : X_k \rightarrow \lambda.Z\mu) R_U(Z, Y) \gamma''(i : Y_j \rightarrow \nu.) \\ \gamma' & = & f(\theta) \sum_{\forall \lambda, \mu} \gamma(i : X_k \rightarrow \lambda.Y\mu) R_U(Z, Y) \gamma''(i : Y_j \rightarrow \nu.) \end{array}$$

More sophisticated forms of $f(\theta)$ can be formulated to achieve arbitrary spread balancing as needed.

6.5 Viterbi Parsing

Viterbi parse is applied to the state sets in a chart parse in a manner similar to HMMs. Viterbi probabilities are propagated in the same way as inner probabilities, with the exception that instead of summing the probabilities during completion step, maximization is performed. That is, given a complete state S_j^i, we can formalize the process of computing Viterbi probabilities v_i as follows:

$$v_i(S_k^i) = max_{S_j^i}(v_i(S_j^i) v_j(S_k^j))$$

and the Viterbi path would include the state

$$S_k^i = argmax_{S_j^i}(v_i(S_j^i) v_j(S_k^j))$$

The state S_k^i keeps a back-pointer to the state S_j^i, which completes it with maximum probability, providing the path for backtracking after the parse is complete. The computation proceeds iteratively, within the normal parsing process. After the final state is reached, it will contain pointers to its immediate children, which can be traced to reproduce the maximum probability derivation tree.

6.5.1 Viterbi Parse in Presence of Unit Productions

The otherwise straight forward algorithm of updating Viterbi probabilities and the Viterbi path is complicated in the completion step by the presence of the Unit productions in the derivation chain. Computation of the forward and inner probabilities has to be performed in closed form, as described previously, which causes the unit productions to be eliminated from the Viterbi path computations. This results in producing an inconsistent parse tree with omitted unit productions, since in computing the closed form correction matrix, we collapse such unit production chains. In order to remedy this problem, we need to consider two competing requirements to the Viterbi path generation procedure.

1. On one hand, we need a batch version of the probability calculation performed, as previously described, to compute transitive recursive closures on all the probabilities, by using recursive correction matrices. In this case, Unit productions do not need to be considered by the algorithm, since their contributions to the probabilities are encapsulated in the recursive correction matrix R_U.
2. On the other hand, we need a finite number of states to be considered as completing states in a natural order so that we can preserve the path through the parse to have a continuous chain of parents for each participating production. In other words, the unit productions, which get eliminated from the Viterbi path during the parse while computing Viterbi probability, need to be included in the complete derivation tree.

We solve both problems by computing the forward, inner and Viterbi probabilities in closed form by applying the recursive correction R_U to each completion for non-unit completing states and then, considering only completing unit production states for inserting them into the production chain. Unit production states will not update their parent's probabilities, since those are already correctly computed via R_U. Now the maximization step in computing the parent's Viterbi probability needs to be modified to keep a consistent tree. To accomplish this, when using the unit production to complete a state, the algorithm inserts the Unit production state into a child slot of the completed state only if it has the same children as the state it is completing. The overlapping set of children is removed from the parent state. It extends the derivation path by the Unit production state, maintaining consistent derivation tree.

Pseudo-code of the modifications to the completion algorithm which maintains the consistent Viterbi derivation tree in presence of Unit production states is shown in Figure 5.

However, the problem stated in subsection 4.3.1 still remains. Since we cannot compute the Viterbi paths in closed form, we have to compute them iteratively, which can make the parser go into an infinite loop. Such loops should be avoided while formulating the grammar. A better solution is being sought.

```
function StateSet.Complete()
    ...
    if( State.IsUnit())
        State.Forward = 0;
        State.Inner = 0;
    else
        State.Forward = ComputeForward();
        State.Inner = ComputeInner();
    end

    NewState = FindStateToComplete();
    NewState.AddChild(State);
    StateSet.AddState(NewState);
    ...
end

function StateSet.AddState(NewState)
    ...
    if( StateSet.AlreadyExists(NewState)
        OldState = StateSet.GetExistingState(NewState);
        CompletingState = NewState.GetChild();
        if( OldState.HasSameChildren(CompletingState))
            OldState.ReplaceChildren(CompletingState);
        end
        OldState.AddProbabilities(NewState.Forward, NewState.Inner);
    else
        StateSet.Add(NewState);
    end
    ...
end
```

Figure 5: Simplified pseudo-code of the modifications to the completion algorithm.

7 On-line Processing

7.1 Pruning

As we gain descriptive advantages with using SCFGs for recognition, the computational complexity, as compared to Finite State Models, increases. The complexity of the algorithm grows linearly with increasing length of the input string. This presents us with the necessity in certain cases of pruning low probability derivation paths. In Earley framework such a pruning is easy to implement. We simply rank the states in the state set according to their *per sample* forward probabilities and remove those that fall under a certain threshold. However, performing pruning one should keep in mind that the probabilities of the remaining states will be underestimated.

7.2 Limiting the Range of Syntactic Grouping

Context-free grammars have a drawback of not being able to express countable relationships in the string by a hard bound by any means other than explicit enumeration.

One implication of this drawback is that we cannot formally limit the range of the application of a $SKIP$ productions and formally specify how many $SKIP$ productions can be included in a derivation for the string to still be considered grammatical. When parsing with robust grammar is performed, some of the non-terminals being considered for expansion cover large portions of the input string consuming only a few samples by non-$SKIP$ rules. The states, corresponding to such non-terminals, have low probability and typically have no effect on the derivation, but are still considered in further derivations, increasing the computational load. This problem is especially relevant when we have a long sequence and the task is to find a smaller subsequence inside it which is modeled by an SCFG. Parsing the whole sequence might be computationally expensive if the sequence is of considerable length. We can attempt to remedy this problem by limiting the range of productions.

In our approach, when faced with the necessity to perform such a parse, we use implicit pruning by operating a parser on a relatively small sliding window. The parse performed in this manner has two important features:

- The "correct" string, if exists, *ends* at the current sample.
- The *beginning* sample of such a string is unknown, but is within the window.

From these observations, we can formalize the necessary modifications to the parsing technique to implement such a sliding window parser:

1. Robust grammar should only include the $SKIP$ productions of the form $A \rightarrow a \mid SKIP\ a$ since the end of the string is at the current sample, which means that there will not be trailing noise, which is normally accounted for by a production $A \rightarrow a\ SKIP$.
2. Each state set in the window should be seeded with a starting symbol. This will account for the unknown beginning of the string. After performing a parse for the current time step, Viterbi maximization will pick out the maximum probability path, which can be followed back to the starting sample exactly.

This technique is equivalent to a run-time version of Viterbi parsing used in HMMs ([9]). The exception is that no "backwards" training is necessary since we have an opportunity to "seed" the state set with an axiom at an arbitrary position.

8 Conclusions

The use of formal languages and syntax-based action recognition is reasonable if decomposition of the action under consideration into a set of primitives that lend themselves to automatic recognition is possible. In contrast, applying the described techniques to the problems where no apriory knowledge about the process structure can be obtained will most likely fail. Existence of such a structure in the process is a pre-requisite for using Syntactic Pattern recognition in general.

8.1 Relation to HMM

While SCFGs look very different from HMMs, they have some remarkable similarities. In order to consider the similarities in more detail, let us look at a subset of SCFGs - Stochastic Regular Grammars. SRG is a 4-tuple $G = N, T, P, S$, where the set of productions P is of the form $A \rightarrow aB$, or $A \rightarrow a$. If we consider the fact that HMM is a version of probabilistic Finite State Machine, we can apply simple non-probabilistic rules to convert an HMM into an SRG. More precisely, taking transition $A \rightarrow^a B$, with probability p_m of an HMM, we can form an equivalent production $A \rightarrow aB \quad [p_m]$. Conversion of an SCFG in its general form is not always possible. An SCFG rule $A \rightarrow aAa$, for instance cannot be converted to a sequence of transitions of an HMM. Presence of this mechanism makes richer structural derivations, such as counting and recursion, possible and simple to formalize.

Another, more subtle difference has its roots in the way HMMs and SCFGs assign probabilities. SCFGs assign probabilities to the production rules, so that for each non-terminal, the probabilities sum up to 1. It translates into probability being distributed over sentences of the language - probabilities of all the sentences (or structural entities) of the language sum up to 1. HMMs in this respect are quite different - the probability gets distributed over all the sequences of a length n ([10]).

In the HMM framework, the tractability of the parsing is afforded by the fact that the number of states remains constant throughout the derivation. In the Earley SCFG framework, the number of possible Earley states increases linearly with the length of the input string, due to the starting index of each state. This makes it necessary in some instances to employ pruning techniques which allow us to deal with increasing computational complexity when parsing long input strings.

References

[1] A. V. Aho and T. G. Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM Journal of Computing*, 1, 1972.

[2] J. Allen. Special issue on ill-formed input. *American Journal of Computational Linguistics*, 9(3–4):123–196, 1983.

[3] M. L. Baird and M. D. Kelly. A paradigm for semantic picture recognition. *PR*, 6(1):61–74, 1974.

[4] M. Brand. Understanding manipulation in video. In *AFGR96*, pages 94–99, 1996.

[5] R. A. Broks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, April 1989.

[6] H. Bunke and D. Pasche. Parsing multivalued strings and its application to image and waveform recognition. *Structural Pattern Analisys*, 1990.

[7] J. H. Connell. *A Colony Architecture for an Artificial Creature*. PhD thesis, MIT, 1989.

[8] J. D. Courtney. Automatic video indexing via object motion analysis. *PR*, 30(4):607–625, April 1997.

[9] T. J. Darrell and A. P. Pentland. Space-time gestures. *Proc. Comp. Vis. and Pattern Rec.*, pages 335–340, 1993.

[10] Charniak. E. *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts and London, England, 1993.

[11] Jay Clark Earley. *An Efficient Context-Free Parsing Algorithm*. PhD thesis, Carnegie-Mellon University, Computer Science Department, Carnegie-Mellon University, Pittsburg, PA, Aug. 1968.

[12] K. S. Fu. A step towards unification of syntactic and statistical pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(3):398–404, May 1986.

[13] F. Jelinek, J. D. Lafferty, and R. L. Mercer. Basic methods of probabilistic context free grammars. In Pietro Laface and Renato Di Mori, editors, *Speech Recognition and Understanding. Recent Advances, Trends, and Applications*, volume F75 of *NATO Advanced Study Institute*, pages 345–360. Springer Verlag, Berlin, July 1992.

[14] P. Maes. The dynamic of action selection. *AAAI Spring Symposium on AI Limited Rationality*, 1989.

[15] R. Narasimhan. Labeling schemata and syntactic descriptions of pictures. *InfoControl*, 7(2):151–179, June 1964.

[16] B. J. Oomen and R. L. Kashyap. Optimal and information theoretic syntactic pattern recognition for traditional errors. *SSPR 6th International Workshop on Advances in Structural and Syntactic Pattern Recognition.*, 1996.

[17] L. R. Rabiner and B. H. Juang. *Fundamentals of speech recognition.* Prentice Hall, Englewood Cliffs, 1993.

[18] A. Sanfeliu and R. Alquezar. Efficient recognition if a class of context-sensitive languages described by augmented regular expressions. *SSPR 6th International Workshop on Advances in Structural and Syntactic Pattern Recognition.*, 1996.

[19] A. Sanfeliu and M. Sainz. Automatic recognition of bidimensional models learned by grammatical inference in outdor scenes. *SSPR 6th International Workshop on Advances in Structural and Syntactic Pattern Recognition.*, 1996.

[20] A. Stolcke. *Bayesian Learning of Probabilistic Language Models.* PhD thesis, University of California at Berkeley, 1994.

[21] A. Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2), 1995.

[22] W. G. Tsai and K. S. Fu. Attributed grammars - a tool for combining syntatctic and statistical approaches to pattern recognition. In *IEEE Transactions on Systems, Man and Cybernetics*, volume SMC-10, number 12, pages 873–885, 1980.