Recognizing Multitasked Activities from Video using Stochastic Context-Free Grammar

Darnell Moore
Texas Instruments
Video & Imaging Processing / DSP R&D Center
Dallas, TX 75243, USA

Irfan Essa
Georgia Institute of Technology
GVU Center / College of Computing
Atlanta, GA 30332-0280, USA

Abstract

In this paper, we present techniques for recognizing complex, multitasked activities from video. Visual information like image features and motion appearances, combined with domain-specific information, like object context is used initially to label events. Each action event is represented with a unique symbol, allowing for a sequence of interactions to be described as an ordered symbolic string. Then, a model of stochastic context-free grammar (SCFG), which is developed using underlying rules of an activity, is used to provide the structure for recognizing semantically meaningful behavior over extended periods. Symbolic strings are parsed using the Earley-Stolcke algorithm to determine the most likely semantic derivation for recognition. Parsing substrings allows us to recognize patterns that describe high-level, complex events taking place over segments of the video sequence. We introduce new parsing strategies to enable error detection and recovery in stochastic context-free grammar and methods of quantifying group and individual behavior in activities with separable roles. We show through experiments, with a popular card game, the recognition of high-level narratives of multi-player games and the identification of player strategies and behavior using computer vision.

Introduction & Related Work

Computer vision research has made significant progress in recent years at recognizing what people are doing. Most of the work in recognition of human activity has relied on recognizing a sequence of states using stochastic model-based approaches. For example, hidden Markov models (HMMs) have become very popular for recognizing gestures (Bobick & Wilson 1997; Schlenzig, Hunter, & Jain 1994), sign-language (Vogler & Metaxas 2001; Starner, Weaver, & Pentland 1998), and actions (Moore, Essa, & Hayes 1999a; Yamato, Ohya, & Ishii 1994; Brand, Oliver, & Pentland 1997).

However, when it comes to recognizing activities with some predefined context or inherent semantics, purely probabilistic methods can be limiting unless they are augmented by additional structure. These activities include parking cars or dropping off people at the curb, (Ivanov and Bobick (2000) recognized using stochastic parsing), longer term office and cooking activities (Moore and Essa (1999b), recognized using contextual information), airborne surveillance tasks (Bremond and Medioni (1998), recognized using deterministic structure), observing simple repair tasks (Brand (1997), recognized using interactions as a metric), and American sign language recognition (Starner and Pentland (1998), recognized using a grammar).

Our goal in this paper is to recognize separable, multitasked activities. We define multitasked activities as *the intermittent co-occurrence of events involving multiple people, objects, and actions, typically over an extended period of time*. By definition, a multitasked activity is composed of several single-tasked activities. To exploit syntax present in many interactions that are based on rules or well-defined context, we concentrate on a class of multitasked activities that possesses group separability. Grammar is then used to explain dependencies between interactions occurring within separable groups. By using syntactic models, we can more easily deal with variation in the occurrence, distribution and transition between events.

As an example, we highlight our approach using a casino card game that has multiple participants who interact with each other both independently and dependently over the duration of the game. Although the rules of play are well-defined, a great deal of variation is permitted. To recognize these and other such activities, we leverage stochastic models based on grammar to interpret syntactic structure and use computer vision to image-, detect visual primitives.

Ivanov and Bobick have accomplished complex activity recognition by combining syntactic pattern recognition with statistical approaches to recognize activity taking place over extended sequences (Ivanov & Bobick 2000). In their work, HMMs are used to propose candidates of low-level temporal features. These outputs provide the input stream for a Stochastic Context-Free Grammar (SCFG) parsing mechanism that enforces longer range temporal constraints, corrects uncertain or mislabelled low-level detections, and allows the inclusion of prior knowledge about the structure of temporal events in a given domain.

The application of syntactic information is not new to AI and computer vision and having been employed for natural language processing and for pattern/object recognition in still images for many years now. The use of SCFG instead of CFG is motivated by the fact that the stochastic-based approach provides a representation to attach a probability to a particular sequence of events. We extend the work of Ivanov and Bobick by introducing new techniques for error detection and recovery as well as strategies for quantifying participant behavior over time. We also experiment with a very challenging domain of card playing to validate our approach for complex multitasked activities.

Representation using SCFG

To characterize multitasked activities, we need models that can identify the regularities associated with complex tasks while also tolerating the dynamics associated with multiple participants and interactions scattered through time. Grammar is a mechanism that uses a system of rules to generate semantically meaningful *expressions*. Its ability to accommodate variation makes it a compelling choice for modeling complex activities, particularly those that are *rule-based, event-driven*.

Grammar is not susceptible to some of the limitations of probabilistic finite state machine representations, like the HMMs. Finite state machines are appropriate for modeling a single hypothesis or a series of them in parallel. However, as variation becomes more pronounced, it becomes exceedingly difficult to collapse additional hypotheses into a single finite state model. In contrast, the generative process associated with grammar is non-deterministic, allowing the derivation of a much longer and elaborate sequence of events (Taylor 1998). Grammar allows us to use a single, compact representation for well-understood interactive events that also accommodates the natural generalizations that occur during their performance. However, because grammar is very domain dependent and very difficult to learn without supervision, we consider manually specified grammar.

Stochastic context-free grammar (SCFG) is an extension of context-free grammar (CFG) where a probability p is added to every production rule, *i.e.,* $X \to \lambda$ We can also express the rule probability p as $P(X \to \lambda)$, which essentially gives the conditional likelihood of the production $X \to \lambda$. We estimate rule probabilities by calculating the average production likelihood. The average simply divides the *count*, the number of times a rule is applied, denoted $c(X \to \lambda)$ for production $X \to \lambda$, by the count of all rules with X on the left hand side, *i.e.,*

$$P(X \to \lambda) = \frac{c(X \to \lambda)}{\sum_{\mu} c(X \to \mu)}, \tag{1}$$

where μ represents all nonterminals generated by nonterminal X.

SCFGs are superior to non-stochastic context-free grammar because the probability attached to each rule provides a quantitative basis for ranking and pruning parses as well as for exploiting dependencies in a language model. While SCFGs are computationally more demanding than simpler language models that employ finite-state and n-gram approaches, they typically perform well using a modest amount of training data and can also be extended in a straightforward fashion by adding production rules and their associated probabilities for each new primitive event (Stolcke & Segal 1994). Rule-based activities, in particular, make good candidates for use with a SCFG because they can be described using a relatively small lexicon of primitive events.

In a SCFG, the probability of the complete derivation of a string is determined by the product of the rule probabilities in the derivation. The notion of context-freeness is extended to include probabilistic conditional independence of the expansion of a nonterminal from its surrounding context (Stolcke 1994). Our motivation for using stochastic context-free grammar is to aggregate low-level events detected so that we can construct higher-level models of interaction.

Symbol Generation (Event Detection): To facilitate event detection, we need to manage prior and newly discovered information about people objects, and their interactions. Such information in the form of image-, object-, or action-based evidence that is collected into object-oriented "containers" (Moore, Essa, & Hayes 1999a). In the specific example of BlackJack, a simple vision system is used which allows for object segmentation and tracking as well as template matching so that hands can be followed or newly dealt cards can be detected. Hands are also tracked making it easy to associate the new card with the person that placed it (Figure 1). By adding domain-specific heuristics, we construct detectors for determining when betting chips are added or removed from the scene. Where applicable, we also consider an articles's location relative to the entire scene as well as in respect to other objects or people.

When a particular event is observed, its corresponding symbolic terminal is appended to the end of the activity string x, which is parsed by the Earley-Stolcke algorithm. Again, in general terms, for some domain C, we let $\mathbf{D}_C = \{D_1, D_2, \ldots\}$ represent the set of detectors for generating the set of terminal symbols V_T. For convenience, the likelihood of a detected event, *i.e.,* the likelihood of generating the terminal symbol x_i corresponding to detector D_i, is given by $P_{\mathbf{D}}(x_i) = P(D_i)$, where $P(D_i)$ is defined on a case by case basis. By processing an activity sequence in domain C, we use $\mathbf{D}_C$ to generate symbolic string $x = x_1x_2, \ldots, x_l$, with length, $l = |x|$.

When parsing is discussed in the next section, we will show that it is possible to compute the syntactical likelihood of a sequence of events $P(x)$. Such a likelihood offers a measure of how much semantic merit a sequence has. Using the independence assumption guaranteed by the use of context-free grammar, we can also describe the likelihood of string formation based on detection alone, *i.e.,*

$$P_{\mathbf{D}}(x) = \prod_{i=1}^{l} P_{\mathbf{D}}(x_i). \tag{2}$$

Unfortunately, as the length l increases, the overall likelihood of the string decreases from repeated multiplication of values less than unity. A better measure of confidence normalizes the likelihood according to l, which we describe by simply calculating the sample mean likelihood of the string, *i.e.,*

$$\tilde{P}_{\mathbf{D}}(x) = \frac{1}{l} \sum_{i=1}^{l} P_{\mathbf{D}}(x_i). \tag{3}$$

The Earley-Stolcke Parsing

For parsing input strings, we employ the Earley-Stolcke algorithm (Stolcke 1994; Earley 1968). The Earley-Stolcke algorithm uses a top-down parsing approach and context-free productions to build strings that are derivable from left to right. It maintains multiple hypotheses of all possible derivations that are consistent with the input string up to a certain point. Scanning input from left to right, the number of hypotheses increases as new options become available or decrease as ambiguities are resolved.

A set of states, determined by the length of the string, is created for each symbol in the input. Each state describes all candidate derivations. The entire set of states forms the *Earley chart*. We preserve notation[1] introduced by Earley to represent a state, which is given by

$$i:\ {}_kX \rightarrow \lambda.\mu\ , \tag{4}$$

where i is the index of the current position in the input stream and k is the *starting* index of the substring given by nonterminal X. Nonterminal X contains substring $x_k \ldots x_i \ldots x_l$, where x_l is the last terminal of the substring μ. When we are in position i, the substring $x_0 \ldots x_{i-1}$ has already been processed by the parser. *State set i* represents all of the states that describe the parsing at this position. There is always one more state set than input symbols, *i.e.*, set 0 describes the parser before any input is processed while set l depicts the parser after all processing.

Parsing Stages: Parsing proceeds iteratively through three steps: *prediction*, *scanning*, and *completion*. The prediction step is used to hypothesize the possible continuation of the input based on the current position in the derived string. We expand one branch of the derivation tree down to the set of its leftmost term to predict the next possible input terminal.

During prediction, we create a list of all the states that are syntactically possible based on prior input. These states provide the candidate terminal symbols that we can anticipate at the next position in the input string. The scanning step is where we read the next input symbol and match it against all states under consideration. Scanning ensures that the terminal symbols produced in a derivation match the input string. The scanning step promotes the states for the next iteration.

Given a set of states which have just been confirmed by scanning, the completion step updates the positions in all pending derivations throughout the derivation tree. Each completion corresponds to the end of a particular nonterminal expansion which was initiated by an earlier prediction step.

States produced by each of these steps are called *predicted*, *scanned*, and *completed*, respectively. A state is called *complete* (not to be confused with *completed*) if the substring $x_j \ldots x_i$ has been fully expanded and is syntactically correct (which is written with the dot located in the rightmost position of the state, *i.e.*, $i:\ {}_jY \rightarrow \nu.$). To determine the likelihood of a string at the current index i, the *forward probability* α gives the likelihood of selecting the next state at step i, along with probability of selecting previous states, *i.e.*, $x_1 \ldots x_{i-1}$. The *inner probability* γ measures the likelihood of generating a substring of the input from a given nonterminal using a particular production. Unlike the forward probability, which starts from the beginning of the string, the inner probability starts at position k in the string.

[1]Earley notation uses **only one** "." (index point) when defining states. The reader is encouraged to pay close attention to the location of the index, which is easily confused with a period.

Parsing in Uncertainty: Human error, *i.e.*, violating rules of the activity, and mistakes made during symbol generation can produce activity sentences that are semantically meaningless. Recall $P_{\mathbf{D}}(x_i)$, which is the probability of the detectors producing symbol x_i. We factor in the likelihood of the input into the parsing mechanism by multiplying $P_{\mathbf{D}}(x_i)$ by the forward and inner probabilities during the scanning step, *i.e.*,

$$\begin{aligned} \alpha' &= \alpha(i:\ {}_kX \rightarrow \lambda.a\mu)P_{\mathbf{D}}(a), \\ \gamma' &= \gamma(i:\ {}_kX \rightarrow \lambda.a\mu)P_{\mathbf{D}}(a), \end{aligned} \tag{5}$$

where a is the terminal sampled from the input in state set i. The revised values for α' and γ' reflect the weight of the likelihood of competing derivations as well as the certainty associated with the scanned input symbol.

Selecting the ML Parse: With uncertainty present in the input, we are not guaranteed to recover a unique parse. Motivated by the use of the Viterbi parsing in the HMM, we apply a generalization of the Viterbi method for parsing a string x to retrieve the most likely probability among all possible derivations for x (in a manner similar to the one proposed by Ivanov & Bobick (2000)). In our case, this will give the most likely interactive summary of events over the duration of the sequence. To compute the Viterbi parse, each state set must maintain the maximal path probability leading to it as well as the predecessor states associated with that maximum likelihood path. Path probabilities are recursively multiplied during completion steps using the inner probabilities as accumulators (Stolcke 1994). By familiar backtracking along the maximal predecessor states, the maximum probability parse can be recovered.

To implement this Viterbi computation, we modify the parser such that each state computes its *Viterbi probability* v. Note that v is propagated similarly to γ, except that during completion the summation is replaced by maximization, such that $v_i({}_kX \rightarrow \lambda Y.\mu)$ is the maximum of all products $v_i({}_jY \rightarrow \nu.)v_j({}_kX \rightarrow \lambda.Y\mu)$ along paths that lead to the completed state ${}_kX \rightarrow \lambda Y.\mu$, *i.e.*,

$$v_i({}_kX \rightarrow \lambda Y.\mu) = \max_{\lambda,\mu}\{v_i({}_jY \rightarrow \nu.)v_j({}_kX \rightarrow \lambda.Y\mu)\}.$$

The state associated with the maximum is listed as the Viterbi path predecessor of ${}_kX \rightarrow \lambda Y.\mu$, *i.e.*,

$${}_kX \rightarrow \lambda.Y\mu = \arg\max_{\lambda,\mu}\{v_i({}_jY \rightarrow \nu.)v_j({}_kX \rightarrow \lambda.Y\mu)\}.$$

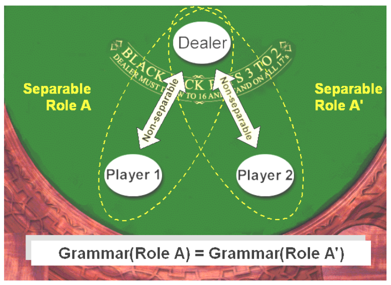
A familiar recursive procedure is required to recover the maximum likelihood (ML) derivation tree associated with the Viterbi parse. During the normal parsing operation described earlier, state ${}_kX \rightarrow \lambda.Y\mu$ maintains a pointer to the state ${}_jY \rightarrow \nu.$ that completes it, providing a path for backtracking. After arriving in the final state, the ML tree is reproduced by visiting predecessor states as identified by pointers.

Parsing Separable Activities

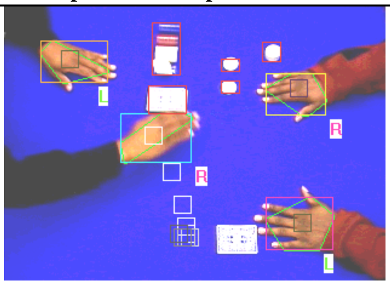
We recognize that individuals can have roles that influence how they interact with objects and other people in a process. Activities with *separable groups* are characterized by wholly independent interactive relationships between two or

Production Rules			Description
S	$\rightarrow AB$	[1.0]	Blackjack → "play game" "determine winner"
A	$\rightarrow CD$	[1.0]	play game → "setup game" "implement strategy"
B	$\rightarrow EF$	[1.0]	determine winner → "eval. strategy" "cleanup"
C	$\rightarrow HI$	[1.0]	setup game → "place bets" "deal card pairs"
D	$\rightarrow GK$	[1.0]	implement strategy → "player strategy"
E	$\rightarrow LKM$	[0.6]	eval. strategy → "dealer down-card" "dealer hits" "player down-card"
	$\rightarrow LM$	[0.4]	eval. strategy → "dealer down-card" "player down-card"
F	$\rightarrow NO$	[0.5]	cleanup → "settle bet" "recover card"
	$\rightarrow ON$	[0.5]	→ "recover card" "settle bet"
G	$\rightarrow J$	[0.8]	player strategy → "Basic Strategy"
	$\rightarrow Hf$	[0.1]	→ "Splitting Pair"
	$\rightarrow bfffH$	[0.1]	→ "Doubling Down"
H	$\rightarrow l$	[0.5]	place bets
	$\rightarrow lH$	[0.5]	
I	$\rightarrow ffI$	[0.5]	deal card pairs
	$\rightarrow ee$	[0.5]	
J	$\rightarrow f$	[0.8]	Basic strategy
	$\rightarrow fJ$	[0.2]	
K	$\rightarrow e$	[0.6]	house hits
	$\rightarrow eK$	[0.4]	
L	$\rightarrow ae$	[1.0]	Dealer downcard
M	$\rightarrow dh$	[1.0]	Player downcard
N	$\rightarrow k$	[0.16]	settle bet
	$\rightarrow kN$	[0.16]	
	$\rightarrow j$	[0.16]	
	$\rightarrow jN$	[0.16]	
	$\rightarrow i$	[0.18]	
	$\rightarrow iN$	[0.18]	
O	$\rightarrow a$	[0.25]	recover card
	$\rightarrow aO$	[0.25]	
	$\rightarrow b$	[0.25]	
	$\rightarrow bO$	[0.25]	

Symbol	Domain-Specific Events (Terminals)
a	dealer removed card from house
b	dealer removed card from player
c	player removed card from house
d	player removed card from player
e	dealer added card to house
f	dealer dealt card to player
g	player added card to house
h	player added card to player
i	dealer removed chip
j	player removed chip
k	dealer pays player chip
l	player bets chip



Seperable, Independent roles



Vision system tracking the game.

Figure 1: (left) Table showing SCFG V_{BJ} for Blackjack card game: Production rules, probabilities, and descriptions. Detectable domain-specific events make up the terminal alphabet V_T of V_{BJ}. This grammar generates a language that can describe the role between any deal-player couple. (right-top) Each dealer-player group represents separable (*independent*) roles. Within each group, individual roles are non-separable (*dependent*) and share the same grammar. (right-bottom) Cards and betting chips recognized using image feature templates. The minimum square distance between hand and object centroid is calculated to determine the last person to touch an object. The location of the object can also be used as valuable context information. This heuristic enables us to label events like, "Dealer dealt car to player."

more agents, *i.e.*, multiple speakers, but separate, independent conversations. Conversely, *non-separable roles* occur when multiple agents take on collaborative, inter-dependent behavior, *i.e.*, an argument between speakers that talk at the same time concerning the same topic.

To assess overall task interactions while preserving individual behaviors, our approach divides activities into separable groups, then develops grammars that describe the non-separable interactions in each. In the card game of Blackjack, for example, a player's conduct is motivated by how the dealer is expected to behave. While there can be many players in a single game, each bets against the dealer, independent of any other player. Since there is rarely any correlation between player interactions, each player-dealer pair represents a separable group. Interactions between player and dealer are non-separable. Consequently, we only need one simple grammar to describe interactions in a game with multiple players. The production rules for this grammar are listed in Figure 1. Terminal symbols used in alphabet V_{BJ} are based on primitive events detected.

While monitoring interactions in the game, we maintain a separate symbolic string for each person m, where $\mathbf{p} = \{p_1, p_2, \ldots p_m\}$ represents all participating players including the dealer. In our case, the relation between any event and person is established by two measures: (a) the person in contact with an article, and (b) the "owner" of the article. These tags are important in Blackjack because they help us associate an article with a respective player. Moreover, the tags remove potential ambiguity that can confuse the parser. In practice, the first measure is easily established when we detect an overlap between the image regions bounding the hand and the object. The second measure is largely determined by a proximity zone, $\mathbf{z}_m = [x_l \; y_t \; x_r \; y_b]^T$, for each person which is defined manually when the scene is initialized, then labeled with the same ID as the respective person. These tags are attached during the scanning stage when the next state is added, such that

$$i+1: \; {}_kX \rightarrow \lambda a.\mu \, [\alpha, \gamma, p_j, o(\mathbf{z}_m)],$$

where $o(\mathbf{z}_m)$ returns the ID p_j corresponding to the zone defined by $o(\mathbf{z}_m)$. During play, we track hand position to establish ownership of objects (see Figure 1).

By tagging interactions and leveraging separability, we provide an elegant treatment for evaluating concurrency using context from both exemplars and models when multiple participants and objects are involved. Ivanov and Bobick (2000) uses a much more complicated alternative that performs interleaved consistency checks on serialized event during parsing. Since we do not have to modify grammar or tag labels during parsing, much less complexity is required by our parser.

Exploiting separability also allows us to assess the probabilistic behavior of individuals in the scene by isolating certain production rules that occur within a non-separable relationship. To model a particular behavior, we manually select a subset of the production rules, *i.e.*, $P_\varsigma \in P$, that provides a basis for characterizing interactions. We define $\mathbf{b}_\varsigma$ to be a vector that represents all n production probabilities in subset P_ς. We determine $\mathbf{b}_\varsigma$ from training data taken over several trials, allowing us to establish baseline models of particular behavior. For example in Blackjack, certain strategies designed to improve the odds of winning are more likely to be used by a more experienced player versus a novice. In this case, we identify $P(G \rightarrow J), P(G \rightarrow bfffH)$ and

$P(G \rightarrow Hf)$ as listed in Figure 1 as some of the metrics for determining player skill.

Each person maintains a subset of the production likelihoods $\hat{\mathbf{b}}_\varsigma$ (to reflect his/her probability of using certain rules), which is reset initially to reflect a uniform distribution. In other words, for a nonterminal X that generates n other strings of terminals and nonterminals, *i.e.,* $X \rightarrow \mu_1|\mu_2|\ldots|\mu_n$, all respective likelihoods $\mathbf{b}_X = \beta_1, \beta_2, \ldots, \beta_n$ are set identically to $\frac{1}{n}$. During separable role characterization, each individual shares the same initial set of rule likelihoods $\mathbf{b}_X$ over P_ς. Using Equation (1), rule probabilities for each individual are "tuned" *via running mean* based on observations of selected productions over the course of several trials. Comparisons between individually tuned rule probabilities $\hat{\mathbf{b}}_\varsigma$ and pre-trained models $\mathbf{b}_\varsigma$ can be made using the mean sum of the square differences or mean square error (MSE),*i.e.,*

$$err(\mathbf{b}_\varsigma - \hat{\mathbf{b}}_\varsigma) = \frac{1}{n}\sum_{i=1}^{n}(\beta_i - \hat{\beta}_i)^2. \tag{6}$$

The MSE is used to measure the pairwise distance so that the likelihood of a behavior given a model for it can be established by

$$P(\hat{\mathbf{b}}_\varsigma|\mathbf{b}_\varsigma) = 1 - \sqrt{err(\mathbf{b}_\varsigma - \hat{\mathbf{b}}_\varsigma)}. \tag{7}$$

Tuning grammars based on an individual's performance, we can assess player behavior by evaluating production probabilities. This characterization of behavior naturally improves as the number of trials is increased.

Error Detection & Recovery

Errors in the input can generate ungrammatical strings, causing the parsing algorithm to fail. We provide techniques for detecting and recovering from failures caused by certain types of errors. A *substitution error* occurs when the wrong terminal symbol is generated because the actual event is not detected as the most likely. *Insertion errors* take place when spurious symbols that do not correspond to actual events are added to the input. Finally, *deletion errors* represent failures to detect events that actually occurred.

Because we use domain-specific heuristics to detect low-level events, substitution and insertion errors are rare. However, deletion errors are more frequent because our domain-specific detectors are less robust at identifying events that deviate significantly from our heuristic models. Ivanov and Bobick handle substitution and insertion errors by modifying the grammar so that the parser accepts input that would, otherwise, generate a fatal syntax error (Ivanov & Bobick 2000). For rule-based activities, like card games, any attempt at correcting an error in this way will compromise the benefit of being able to detect rule violations. We have a vested interest in determining how, when, where, and by whom errors occur. At the same time, we wish to make parsing robust enough to tolerate erroneous input.

We attempt to recover from parsing failures by taking advantage of grammatical structure. Although the arrangement of terminals in the input is non-deterministic, it is constrained by *a priori* known rules that we leverage to anticipate future input. Parsing errors occur in the scanning stage when the symbol sampled from the input does not match any of the terminals from the prediction stage. This invariably happens during a nonterminal expansion. We revisit the prediction stage during the expansion of nonterminal X, which creates a list of productions $Y \rightarrow \nu$ that are syntactically consistent with the expansion, *i.e.,*

$$i:\ {}_kX \rightarrow \lambda.Y\mu\ [\alpha,\gamma]\ \Rightarrow\ i:\ {}_iY \rightarrow .\nu\ [\alpha',\gamma'].$$

Every nonterminal Y is also expanded until the next terminal is predicted, *i.e.,* $i:\ {}_iY \rightarrow .a\xi$. Solutions to a parsing failure are motivated by the nature of the error. We consider the following three scenarios:

If the failure is caused by an insertion error, we simply ignore the scanned terminal, and return the state of the parser to the point prior to scanning. The same pending expansions for prediction are maintained.

If the failure is caused by a substitution error, we promote each pending prediction state as if it were actually scanned, creating a new path for each hypothetical terminal. (At this point, these become hypothetical paths). We proceed with normal parsing, but instead of maintaining paths that spawn from a single, legitimately scanned terminal, we accommodate all paths from each hypothetical terminal appearing as a result of a simulated scan. A hypothetical path is terminated if another failure occurs in the next real scanning step. The actual likelihood of the event associated with the hypothetical terminal $P_{\mathbf{D}}(a)$ is recovered, then multiplied to prediction values of α and γ such that we get,

$$\begin{aligned}\alpha' &= \alpha(i:\ {}_iY \rightarrow .a\xi)P_{\mathbf{D}}(a)\\ \gamma' &= \gamma(i:\ {}_iY \rightarrow .a\xi)P_{\mathbf{D}}(a),\end{aligned} \tag{8}$$

If the failure is caused by a deletion error, again we promote each pending prediction state and create a separate path for each hypothetical symbol. We proceed through the completion stage, then to prediction to generate the next state terminal. During a simulated scan, hypothetical paths that are inconsistent with the symbol that caused the first failure are terminated. When a deletion error is assumed, there is no detection likelihood to recover for the missing symbol. We approximate this likelihood, denoted as $\widetilde{P}_{\mathbf{D}}(a)$, using empirical values that we select by hand, which are influenced by historical probability values for the detection of symbol a. Modified forward and inner probabilities in the first scanning step are given as

$$\begin{aligned}\alpha' &= \alpha(i:\ {}_iY \rightarrow .a\xi)\widetilde{P}_{\mathbf{D}}(a)\\ \gamma' &= \gamma(i:\ {}_iY \rightarrow .a\xi)\widetilde{P}_{\mathbf{D}}(a),\end{aligned} \tag{9}$$

while those of the second simulated scanning step can be recovered from the original scan likelihood, *i.e.,*

$$\begin{aligned}\alpha' &= \alpha(i+1:\ {}_{i+1}Y \rightarrow .b\xi)P_{\mathbf{D}}(b)\\ \gamma' &= \gamma(i+1:\ {}_{i+1}Y \rightarrow .b\xi)P_{\mathbf{D}}(b).\end{aligned} \tag{10}$$

Using these methods, the parser is guaranteed to generate a syntactically legal interpretation but provides no warranty of its semantic legitimacy. The parser supplies the framework with the erroneous symbol and its corresponding likelihood so that records of potential failures can be attached to the appropriate person. In this way, substrings with bad

(A)

Grammar		
S	$\rightarrow$	AB
A	$\rightarrow$	aa
	$\rightarrow$	aaA
B	$\rightarrow$	bc
	$\rightarrow$	bcB

(B)

Earley Chart
predicted
$0 : {}_0 \rightarrow .S$
$0 : {}_0S \rightarrow .AB$
$0 : {}_0A \rightarrow .aa$
$0 : {}_0A \rightarrow .aaA$
scanned "a"
$1 : {}_0A \rightarrow a.a$
$1 : {}_0A \rightarrow a.aA$
none completed
predicted
$1 : {}_1A \rightarrow a.a$
$1 : {}_1A \rightarrow a.aA$

(C)

Insertion	Substitution	Deletion
scanned "b"	*scanned "b"*	*scanned "b"*
failed, expecting "*a*"	failed, expecting"*a*"	failed, expecting "*a*"
ignore "*b*"	*scanned "*a*"	*scanned "*a*"
predicted	$2 : {}_1A \rightarrow aa.$	$2 : {}_1A \rightarrow aa.$
$1 : {}_1A \rightarrow a.a$	$2 : {}_1A \rightarrow aa.A$	$2 : {}_1A \rightarrow aa.A$
$1 : {}_1A \rightarrow a.aA$	*completed*	*completed*
scanned "c"	$2 : {}_1A \rightarrow aa.$	$2 : {}_1A \rightarrow aa.$
failed, expecting "*a*"	$2 : {}_0S \rightarrow A.B$	$2 : {}_0S \rightarrow A.B$
TERMINATED	*predict*	*predict*
	$2 : {}_2A \rightarrow .aa$	$2 : {}_2A \rightarrow .aa$
	$2 : {}_2A \rightarrow .aaA$	$2 : {}_2A \rightarrow .aaA$
	$2 : {}_2B \rightarrow .bc$	$2 : {}_2A \rightarrow .bc$
	$2 : {}_2B \rightarrow .bcB$	$2 : {}_2A \rightarrow .bcB$
	scanned "c"	*scanned "*b*" (*retry*)
	failed, expecting "*b*"	$3 : {}_2A \rightarrow b.c$
	TERMINATED	$3 : {}_2A \rightarrow b.cB$
		none completed
		predicted
		$3 : {}_3A \rightarrow b.c$
		$3 : {}_3A \rightarrow b.cB$
		scanned "c"
		$3 : {}_2A \rightarrow b.c$
		$3 : {}_2A \rightarrow b.cB$
		$3 : {}_2A \rightarrow b.cB$

Figure 2: (A) Consider this simple context-free grammar, which we will use to construct the input sequence *aabc*.... A *deletion error* occurs in the detection of events such that the input only contains *abc*... (B) Shows the Earley chart after the first symbol *a*, is predicted and successfully scanned. The next scanned symbol, *b*, will cause parsing to fail under normal conditions since *a* was the only symbol anticipated during prediction. (C) Continuation of the Earley Chart shows parser recovery attempts under different error assumptions. Under the *insertion* assumption, we ignore *b* and repeat the last prediction state. Under *substitution*, we replace *b* with *a* and attempt to continue, but eventually fail when *c* is scanned (for both assumptions). Under *deletion*, we assume that we missed the detection of *a*, so we simulate its scan. This not only allows us to complete the parse, but suggests the kind of error that may have taken place. *Scan of hypothetical symbol is simulated to promote parsing step. Associated rule probabilities are ignored here for simplicity.

syntax can be more closely scrutinized to determine when an illegal action takes place.

Figure 2 illustrates how the parser attempts to recover from failures using the three error scenarios mentioned above. We maintain every recovered path, even if multiple tracks (each representing one of the three error scenarios) are formed from a single failure. For each of the error scenarios, we elect to tolerate only two consecutive failures before terminating the parse of that path. However, our approach can be applied iteratively so that more consecutive failures can be tolerated. A consequence of accepting more failures must be reflected by increasing the uncertainty in our approximation of $P_{\mathbf{D}}(a)$, denoted as $\widehat{P}_{\mathbf{D}}(a)$. We rely on the exponential to serve as a penalty function that is applied by multiplication to the historical mean likelihood $\widetilde{P}_{\mathbf{D}}(a)$, *i.e.,*

$$\widehat{P}_{\mathbf{D}}(a) = e^{\frac{-n}{\rho}} \widetilde{P}_{\mathbf{D}}(a), \qquad (11)$$

where n is the number of consecutive failures and ρ is empirically derived.

The algorithmic complexity of tracking multiple paths, which is a function of the production rules involved, tends to grow linearly for grammars with small terminal and nonterminal alphabets but can expand exponentially for larger grammars or for very long terminal strings. When computation and memory resources must be delicately managed, we prune recovered paths that have low overall likelihoods. Unlike the example provided in Figure 2, we can also entertain hybrid error scenarios in order to generate the most likely parse, *i.e.,* instead of treating each consecutive error by the same type of scenario, all three alternatives can be consider for each bad input symbol.

Experimental Results

We provide real examples of our approach in the domain of the card game, Blackjack. Every rule of the game was used as a production rule in the grammar with full coverage (recall Table 1). A vision system, that ran in real-time, was used for tracking activities in a controlled environment.

Experiment I: Event Detection Accuracy: Twenty-eight sequences were used to generate 700 example events, which were compiled to determine the detection rate of each detector. Each sequence consisted of a full game of Blackjack with at least one player. For example, a sequence might generate six examples of the event "player bet a chip," five examples of "dealer removed player card," etc. The overall detection rate for all events is 96.2%. The error rates for insertion, substitution, and deletion errors were 0.4%, 0.1%, and 3.4%, respectively (assessed manually). Table 1 shows the results of this examination.

Experiment II: Error Detection & Recovery: A semantically legitimate sequence was then tried with no insertion, substitution, or deletion errors, and we are able to parse the activity with 100% accuracy. To provide a more diverse sample of sequences, two testing corpa were compiled from several minutes of video where Blackjack was played, primarily with two people (one dealer and one player). Corpus A contained 28 legitimate sequences with at least one detection error per sequence (either a deletion, substitution, or insertion error). Corpus B represents a family of 10 illegitimate sequences with various errors. Sequences in Corpus B often contained illegal moves, dealer mistakes, cheating, etc. **Error recovery disabled**, only 12 of the 28 sequences (42.9%) in Corpus A could be entirely parsed without terminating in failure. Of those that could be parsed, the mean detection rate for 320 events was 70.1%. The error rates for insertion, substitution, and deletion errors were 5.8%, 14.5%, and 9.6%, respectively. None of the sequences in Corpus B could be entirely parsed.

Error recovery enabled (accepting up to 2 consecutive failures), 25 of the 28 sequences (85.7%) from Corpus A could be parsed with 93.8% of all 625 events detected accurately. Insertion, substitution, and deletion errors were *reduced* by 70.5%, 87.3%, 71.9%, respectively. Parsing im-

S	Domain-Specific Events	Detect Rate	Error Rate (%) *Ins*	*Sub*	*Del*
a	dlr removed house card	100.0	0.0	0.0	0.0
b	dlr removed plyr card	100.0	0.0	0.0	0.0
c	plyr removed house card†	100.0	0.0	0.0	0.0
d	plyr removed plyr card	100.0	0.0	0.0	0.0
e	dlr add card to house	94.6	0.0	0.0	5.4
f	dlr dealt card to plyr	92.2	0.0	0.0	7.8
g	plyr add card to house†	100.0	0.0	0.0	0.0
h	plyr add card to plyr	89.3	3.6	0.0	7.1
i	dlr removed chip	93.7	0.0	0.0	6.3
j	plyr removed chip	96.9	0.0	0.0	3.1
k	dlr pays plyr chip	96.9	0.0	0.0	3.1
l	plyr bet chip	90.5	1.1	1.1	7.4

Table 1: *Experiment I*: Detection rate of events which make up the terminal alphabet V_T of V_{BJ}. Errors are categorized as insertion, substitution, and deletion, respectively. †Denotes events with no significance to legitimate Blackjack play, but can be used to detect illegal occurrences.

S	Detect % *on*	*off*	Ins Err *on*	*off*	Sub Err *on*	*off*	Del Err *on*	*off*
a	98.8	92.5	0.0	0.0	0.0	0.0	1.2	7.5
b	97.8	90.8	0.0	0.0	0.0	0.0	2.2	9.2
c	100.0	80.0	0.0	0.0	0.0	20.0	0.0	0.0
d	100.0	91.7	0.0	0.0	0.0	0.0	0.0	8.3
e	94.0	74.9	1.2	5.0	1.2	7.5	3.6	12.5
f	95.6	70.3	0.0	2.3	0.0	9.2	4.4	18.3
g	100.0	50.0	0.0	0.0	0.0	50.0	0.0	0.0
h	80.0	41.7	4.0	8.3	8.0	33.3	8.0	16.7
i	92.9	88.9	0.0	0.0	0.0	0.0	7.1	11.1
j	96.5	92.7	0.0	0.0	0.0	0.0	3.5	7.3
k	79.0	12.5	10.5	36.5	10.5	43.8	0.0	7.3
l	90.6	55.8	4.7	17.2	2.4	9.8	2.4	17.2

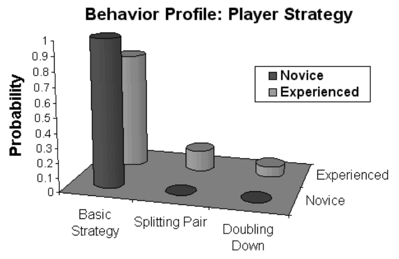
Table 2: *Experiment II*: Detection and error rates for Corpus A with error recovery turned on and off. Error recovery improves overall detection rate by 33.8%.

proved by 40% for Corpus B sequences with error recovery turned on, with an average 85.4% of high-level events recognized accurately. This improvement in the parsing rate is attributed to recovering from insertion errors, which simply skipped over rule violations encountered during the sequence. We assessed that 22.5%, 17.5%, and 60.0% of errors were caused by insertion, substitution, and deletion errors, respectively.

To measure the performance of the parser under a variety of conditions, including consecutive error burst, Corpus C was developed from 113 simulated terminal strings representing legal plays with various errors. Using this data, the probability of detection for each event $P_{\mathbf{D}}(a)$ is estimated using the average determined in Table 1. Homogeneous error types present the worst-case system complexity due to contiguous blocks of substitution and deletion errors. Heterogeneous error scenarios benefit from the treatment used for insertion errors, which only need to maintain the same set of pending states, effectively lowering overall system complexity. We also learn empirically that to recover from an error burst of length n, we must accept at least n consecutive failures to recover.

Experiment III: High-level Behavior Assessment: We examined non-separable roles between a player and the dealer to assess patterns of behavior. The conduct of the dealer is strictly regulated by the rules of Blackjack, but the player is permitted to execute a range of different strategies to improve his/her chance of winning. We define a *novice* as a player whose moves are limited to *basic strategy*[2] where *experts* employ more advanced strategies, such as "splitting pairs" and "doubling down." The profile for these two behaviors is shown in Figure 3.

[2]When no extra cards are dealt to the player after the initial card pair, basic strategy is assumed.



Behavior	Detection Accuracy
Low-risk	92%
High-risk	76%
Novice	100%
Expert	90%

Figure 3: *Experiment III:* (left) Trained behavior profiles of player strategy for novice and expert. (right) Table of Classification of behaviors.

We can also assess other behaviors, such as whether a player is a low-risk or high-risk player by evaluating betting amounts. After tuning several behaviors with actual and synthetic training data, roughly 10 trials per individual were conducted to assess behavior. Results are shown in Figure 3.

Summary

We show that SCFG is a powerful method for extracting high-level behaviors from sequences that have multiple people, objects, and tasks taking place over an extended period of time. By monitoring how frequently some production rules are used, we demonstrate a quantitative technique for assessing behaviors in non-separable roles. Using a strategy that proposes multiple hypotheses for recovering from errors in the input, our results show that parsing improves by over 40% and reduces some errors by as much as 87%. By closely examining multitasked, collaborative tasks such as card games, we develop methods that are appropriate for treating other highly complicated human activities.

References

Bobick, A. F., and Wilson, A. D. 1997. A state based approach to the representation and recognition of gesture. *PAMI* 19(12):1325–1337.

Brand, M.; Oliver, N.; and Pentland, A. 1997. Coupled hidden markov models for complex action recognition. In *CVPR*.

Brand, M. 1997. Understanding manipulation in video. In *Proceedings of Second International Conference on Face and Gesture Recognition*, 94–99.

Bremond, F., and Medioni, G. 1998. Scenario recognition in airborne video imagery. In *DARPA Image Understanding Workshop 1998*, 211–216.

Earley, J. C. 1968. *An Efficient Context-Free Parsing Algorithm*. Ph.D. Dissertation, Carnegie-Mellon University.

Ivanov, Y., and Bobick, A. 2000. Recognition of visual activities and interactions by stochastic parsing. *PAMI* 22(8):852–872.

Moore, D.; Essa, I.; and Hayes, M. 1999a. Context management for human activity recognition. In *Proceedings of Audio and Vision-based Person Authentication 1999*.

Moore, D.; Essa, I.; and Hayes, M. 1999b. Exploiting human actions and object context for recognition tasks. In *ICCV'99*, 80–86.

Schlenzig, J.; Hunter, E.; and Jain, R. 1994. Recursive identification of gesture inputs using hidden markov models. In *WACV94*, 187–194.

Starner, T.; Weaver, J.; and Pentland, A. 1998. Real-time american sign language recognition using desk and wearable computer based video. *PAMI* 20(12):1371–1375.

Stolcke, A., and Segal, J. 1994. Precise n-gram probabilities from stochastic context-free grammars. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, 74–79. Las Cruces, NM.

Stolcke, A. 1994. *Bayesian Learning of Probabilistic Language Models*. Ph.d., University of California at Berkeley.

Taylor, R. G. 1998. *Models of Computation and Formal Languages*. Oxford University Press.

Vogler, C., and Metaxas, D. 2001. A framework for recognizing the simultaneous aspects of american sign language. *CVIU* 81(3):358–384.

Yamato, J.; Ohya, J.; and Ishii, K. 1994. Recognizing human action in time-sequential images using a hidden Markov model. In *CVPR1992*, 379–385.