Recognition of Multi-agent Interaction in Video Surveillance

Y. A. Ivanov A. F. Bobick

MIT Media Laboratory
Cambridge, MA 02139, USA

Abstract

This paper describes a probabilistic syntactic approach to the detection and recognition of temporally extended activities and interactions between multiple agents. A complete system consisting of an adaptive tracker, an event generator, and the parser performs segmentation and labeling of a surveillance video of a parking lot; the system correctly identifies activities such as `pick-up` *and* `drop-off`*, which involve person-vehicle interactions.*

The main contributions of this paper are extending the parsing algorithm to handle multi-agent interactions within a single parser, providing a general mechanism for consistency-based pruning, and developing an efficient incremental parsing algorithm.

1 Introduction

Research in the area of visual surveillance is quickly approaching the area of complex activities, which are framed by extended context. As more and more methods of identifying simple movements become available, the more the importance of the contextual methods increases. In these systems, activities and movements are not only recognized as they are detected, but their meaning is affected by the temporally extended interpretation. (e.g., see [2]). When such global context is recovered for the complete sequence, the beliefs about each component are influenced by the interpretation. For example, the class of an object may be uncertain when viewed in isolation, but if when participating in an interaction the object "plays the part" of a car then belief about its class label (e.g. `car`) is reinforced.

This paper describes a probabilistic syntactic approach to the detection and recognition of temporally extended activities and of interactions between multiple agents. The first technical contribution of this paper is the extension of previous results in the stochastic parsing of activities [2] to handle parallel input streams and to enforce class-interleaved consistency. Stochastic parsing handles noisy and uncertain classifications of the primitive events by integrating them into a coherent interpretation. Parallel input streams and consistency checking allows us to detect high level interactions between multiple objects. The second significant component is the development of a general consistency-based pruning mechanism that eliminates syntactically valid but domain-inconsistent interpretations. This is the mechanisnm that enforces, for example, temporal consistency between the events in an interpretation, or for the domain shown here, the constraint as to how far a car can move between the time when a tracker loses track of a vehicle and the time when some "new" vehicle is acquired.

We demonstrate the utility of these capabilities (and that of the stochastic parsing of activities in general) by showing results from an end-to-end, real-time surveillance system; even when multiple activities are taking place simultaneously, the system correctly identifies activities such as `PICK-UP` and `DROP-OFF`, which involve person-vehicle interactions.

1.1 Related Work

Research in the area of recognition of long term complex activities is relatively new. Courtney ([6]) developed a system, for detecting activities in a closed environment. The activities include person leaving an object in a room, or taking it out of the room. Brand ([3]) showed the results of detecting manipulations in video using a deterministic context-free grammar. Both techniques are non-probabilistic and required relatively high quality low-level detectors.

Oliver and Rosario ([10]) developed a system for detecting people interactions, which modeled interactions using Coupled Hidden Markov Model ([4]) . In the course of the latter research, a multi-agent simulation was used to produce synthetic training data to train the CHMM modeling the interaction. The relation to our work is that their representation of the multi-agent simulation can be viewed as a structured, stochastic grammar-like description of the interactions.

There has been much research done in the area of syntactic pattern recognition, which was the foundation upon which our work is built. Earley ([7]) developed the efficient parsing mechanism, which was given a probabilistic form by Stolcke([11]). Aho and Peterson ([1]) described the basis of syntactic error correction. The multi-valued string parsing was the main topic of the work by Bunke([5]).

In earlier work ([2]), we demonstrated SCFGs successfully parsing gestural input from musical conducting. A potential objection to that work was that not many domains would have grammatical structure; one goal of this paper is to show that relatively complex activities can be described compactly by such a grammar.

2 Probabilistic Parsing of Interactions

Visual surveillance requires the tracking and analysis of multiple objects that co-exist in the scene. Labeling their interactions entails formulating a model for their interactions in either object-centered or a scene-centered setting.

As a solution to this challenging problem we chose an SCFG parsing approach. Using syntactic pattern recognition in the area of activity detection is motivated by the inherent sequentiality of the domain. This is the fact which also made hidden Markov Models (HMMs) a popular solution for gesture recognition tasks. Relation between HMMs and SCFGs can be simply derived from relations between Finite State Machines and Context Free parsers. Generative capabilities of an FSM, extending to the set of regular languages, are superseeded by Context-Free Grammars and the Context-Free languages they generate. A similar relation exists between Probabilistic Finite State Machines (HMMs) and SCFGs. The descriptive and generative advantage comes, of course, with a computational cost.

However, we argue that it is not the computational complexity or the generative power that is of importance. Rather, it is the idea that our approach presumes some structure *a priori*: the component states of the activties are defined and their recovery is part of the interpretation task. In our work we address the recognition of activities of which we have the explicit knowledge which we can simply encode as a (typically stochastic) grammar. The advantage of such encoding is that it makes possible the recovery of intermediate action structures. For example, a `DROP-OFF` contains a component state of the car to coming to a stop. Our method of detecting a `DROP-OFF` is guaranteed to label some event as `CAR-STOP`.

The general idea of our approach, which we are presently describing is as follows:

1. we observe the scene and track moving blobs, allowing the tracker to make *probabilistic* conclusions about object identities; This labeling is performed based on object's appearance and properties of its track;
2. we probabilistically map these tracks onto a set of predetermined *events*;
3. every time an *event* occurs, we attempt to explain this event in context of others by tracking several concurrent hypotheses until one of them is confirmed with higher probability than the others.

In this list, (1) is performed by a tracker. Full resposnsibility for the track labeling is delegated to the tracker since all the data necessary to analyze object trajectories and appearance is readily available at that level.

(2) is implemented as an event generator which takes an object track and produces a (generally multi-valued) pair of events, corresponding to the track's end points. These events can be both concurrent and uncertain (see section 3.2).

The SCFG parser performs the task described by (3). It accepts events and parses them in accordance with the provided SCFG. The grammar encodes rules describing the expected structure of the activities.

2.1 Parsing Algorithm

Before we proceed with the discussion of the contribution of this work, let us briefly review the parsing algorithm used here. The general intuition for the algorithm can be developed by considering the following: before reading a symbol from the input, the parser generates a set of hypotheses for the next possible symbol (*prediction* step). After the symbol is read (*scanning*), some of the hypotheses get confirmed, but the others get discarded (*completion*). The process repeats until some final condition is met.

The remainder of this section, as well as section 2.2 is a re-iteration of the algorithm available elsewhere and can be skipped by the reader familiar with the algorithm ([11]) and our earlier work ([2]).

For each input symbol, the parser keeps a set of states that collectively describe the current pending derivations. A state is a production rule, augmented with two additional markers i and k.

Marker k indicates the starting position in the input string where the rule is currently applied, and marker i shows the current position of the parser in the string. The position of the parser inside the rule that corresponds to i is shown by the symbol "." We denote a state as:

$$i : X_k \rightarrow \lambda.Y\mu \ \ [\alpha, \gamma] \tag{1}$$

where X and Y are non-terminals and and are arbitrary sequences of terminals and non-terminals. In the SCFG parsing algorithm ([11]), each state also carries *forward* and *inner* probabilities denoted in by α and γ, respectively. α, also called a *prefix probability*, is the probability of the parsed string up to position i, and γ is a probability of the sub-string starting at k and ending at i.

Parsing begins with initializing the first state set with an *initial state*, which is generated from the root production rule of the garmmar.

2.1.1 Prediction

The prediction step is used to hypothesize the possible continuation of the input based on current position in the parse: Given the state $i : X_k \rightarrow \lambda.Z\mu$ and a production rule $Y \rightarrow \nu$, such that Y and Z are in *Left Corner* relationship (i.e. there exists a rule $Z \rightarrow Y\eta^1$), prediction produces the state:

$$\begin{cases} i : X_k \rightarrow \lambda.Z\mu \;\; [\alpha, \gamma] \\ Y \rightarrow \nu \end{cases} \Rightarrow \;\; i : Y_i \rightarrow .\nu \;\; [\alpha', \gamma']$$

where α' and γ' are computed as:

$$\begin{aligned} \alpha' &= \sum_{\forall \lambda, \mu} \alpha(i : X_k \rightarrow \lambda.Z\mu)\mathbf{R}_l(Z, Y)P(Y \rightarrow \nu) \\ \gamma' &= P(Y \rightarrow \nu) \end{aligned}$$

where $\mathbf{R}_l(Z, Y)$ is a *Reflexive Transitive Closure* matrix of the *left corner relation* between Y and Z. Complete details can be found in [11] and [9].

2.1.2 Scanning

The scanning step reads an input symbol and matches it against all pending states for the next iteration. If the next input symbol is a then the pending derivations that expect a are advanced to the next state set:

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

Those that do not expect a are eliminated.

2.1.3 Completion

Given the set of scanned states, completion updates the parser positions in all the pending derivations:

$$\begin{cases} j : X_k \rightarrow \lambda.Z\mu \;\; [\alpha, \gamma] \\ i : Y_j \rightarrow \nu. \;\; [\alpha'', \gamma''] \end{cases} \Rightarrow \;\; i : X_k \rightarrow \lambda Z.\mu \;\; [\alpha', \gamma']$$

where α' and γ' are computed as:

$$\begin{aligned} \alpha' &= \sum_{\forall \nu} \alpha(i : X_k \rightarrow \lambda.Z\mu)\mathbf{R}_u(Z, Y)\gamma''(i : Y_j \rightarrow \nu.) \\ \gamma' &= \sum_{\forall \nu} \gamma(i : X_k \rightarrow \lambda.Z\mu)\mathbf{R}_u(Z, Y)\gamma''(i : Y_j \rightarrow \nu.) \end{aligned}$$

and $\mathbf{R}_u(Z, Y)$ is a *Reflective Transitive Closure* matrix of the *Unit Production* relation.([9], [11]).

[1]To be precise, Y has to be reachable from Z by a chain of such rules.

2.2 Handling Uncertainty

Uncertainty in the input stream is handled by the parser in the manner described in [2]. An input symbol is presented as a group of events, each posted with its likelihood. At the *scanning* step, for each event in the group of events, and its likelihood $P(a)$, we produce the new state:

$$i : X_k \rightarrow \lambda.a\mu \;\; [\alpha, \gamma] \;\; \Rightarrow \;\; i + 1 : X_k \rightarrow \lambda a.\mu \;\; [\alpha', \gamma']$$

The likelihoods are incorporated in the parse by multiplying forward and inner probabilities of the corresponding states

$$\begin{aligned} \alpha' &= \alpha(i : X_k \rightarrow \lambda.a\mu)P(a) \\ \gamma' &= \gamma(i : X_k \rightarrow \lambda.a\mu)P(a) \end{aligned}$$

After the final state of the parse is reached, Viterbi maximization will retreive only one event of the group which had yielded the maximum likelihood parse. As will be decsribed, when an object enters the scene both `car-enter` and `person-enter` events are generated simultaneously. After the parse is completed, the one which was found to be part of the most likely parse will be selected. If the object *behaved* as a car, the `car-enter` event will be selected.

Note that the above technique successfully handles substitution errors. In order to account for deletion and insertion errors, we transform the SCFG into a robust form, which can be done automatically. First, every terminal in the grammar is replaced with a *pre-terminal*, which is a production rule rewriting a pre-terminal with the terminal or a terminal preceeded by noise: $A \rightarrow a \mid SKIP\ a$. Next, a "noise" rule is introduced, which accounts for any terminal which is "out of place": $SKIP \rightarrow a \mid b \mid \ldots \mid SKIP\ a \mid \ldots$. These rules effectively "ignore" any syntactic noise in the input while preserving the symbols in the stream.

2.3 Enforcing Track Consistency

Associated with each event is a track representing the temporally consecutive detections of an object in the scene. For a parse to be valid, not only does the sequence of events need to be grammatical, but also the tracks associated with those events must be temporally and spatially consistent.

The consistency of the sequence of tracks contained in a given parse is determined by the compatibility of their endpoints. Track inconsistency incrementally prunes many incorrect yet grammatical sequences.

The temporal consistency algorithm described in [2] presents a convenient pruning mechanism. Following that approach, we introduce two consistency attributes, which contain all tracking data pertinent to the beginning and ending samples of a parser state S_k^i (kth and ith events of the input) .

$$\mathbf{l} = \begin{pmatrix} f_l \\ t_l \\ x_l \\ y_l \\ dx_l \\ dy_l \end{pmatrix} \qquad \mathbf{h} = \begin{pmatrix} f_h \\ t_h \\ x_h \\ y_h \\ dx_h \\ dy_h \end{pmatrix}$$

where $\mathbf{l}$ is a "low mark" attribute group and $\mathbf{h}$ is the "high mark" group. The attributes contain the frame number, the time-stamp, the position of the object in the image, and its velocity.

Each event in the input stream contains tracking data associated with it since the moment this object was first found. Its low mark is the start of that track; the high mark, the end.

$\mathbf{l}$ and $\mathbf{h}$ are propagated within prediction scanning and completion as shown below.

2.3.1 Prediction

Prediction simply marks the expected beginning of the substring with the initial values $\mathbf{S}_t$:

$$\begin{cases} i : X_k \rightarrow \lambda.Y\mu \;\; [\mathbf{l}, \mathbf{h}] \\ Y \rightarrow \nu \end{cases} \Rightarrow \;\; i : Y \rightarrow .\nu \;\; [\mathbf{S}_t, \mathbf{S}_t]$$

2.3.2 Scanning

Scanning reads a terminal from the input stream and sets high marks of scanned states to the high mark of the terminal, expanding the range of the state. In addition, for all the *predicted* states expecting this terminal, it sets the low mark to the low mark of the scanned terminal:

$$\begin{aligned} i : X_k \;\; &\rightarrow \lambda.a\mu \;\; [\mathbf{l}, \mathbf{h}] \;\; \Rightarrow \\ &\begin{cases} i+1 : X_k \rightarrow \lambda a.\mu \;\; [\mathbf{l}_a, \mathbf{h}_a], \;\; if \;\; \lambda = \varepsilon \\ i+1 : X_k \rightarrow \lambda a.\mu \;\; [\mathbf{l}, \mathbf{h}_a], \;\; otherwise \end{cases} \end{aligned}$$

where $\mathbf{l}_a$ and $\mathbf{h}_a$ are low and high mark attributes of the terminal, respectively.

2.3.3 Completion

The 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:

$$\begin{cases} j : X_k \rightarrow \lambda.Y\mu \;\; [\mathbf{l}_1, \mathbf{h}_1] \\ i : Y_j \rightarrow \nu. \;\; [\mathbf{l}_2, \mathbf{h}_2] \end{cases} \Rightarrow \;\; i : X_k \rightarrow \lambda Y.\mu \;\; [\mathbf{l}_1, \mathbf{h}_2]$$

The completion is performed for all *complete* states $i : Y_j \rightarrow \nu.$, subject to consistency constrains enforced within the filtering routine.

2.3.4 Filtering

The completion step presents an opportunity to check the tracks for consistency. This check is implemented as a filtering routine that takes a completing state and the one it completes, and evaluates the cost of this operation. The cost is computed in the form of multiplicative penalty function. This computation extrapolates the position of the object, $\mathbf{r}_p$, using a constant velocity model based on the $\mathbf{h}$ attribute of the state being completed and the $\mathbf{l}$ attribute of the completing state:

$$\mathbf{r}_p = \mathbf{r}_1 + d\mathbf{r}_1(t_2 - t_1) \qquad (2)$$

where $\mathbf{r}_1 = (x_1, y_1)^T$ and $d\mathbf{r}_1 = (dx_1, dy_1)^T$ - position and velocity at the end of the first track, t_1 the time at the end of the first track, and t_2, the time at the start of the second.

The multiplicative penalty function is then computed, based on the distance between the projected position, $\mathbf{r}_p$, and the position of the object in the low mark of the completing state, $\mathbf{r}_2$:

$$f(\mathbf{r}_p, \mathbf{r}_2) = \begin{cases} 0, \;\; if \;\; (t_2 - t_1) < 0 \\ \exp\left(\frac{(\mathbf{r}_2 - \mathbf{r}_p)^T(\mathbf{r}_2 - \mathbf{r}_p)}{\theta}\right), \;\; o/w \end{cases} \qquad (3)$$

Temporal consistency is also enforced here by rejecting the states that have the time-stamp, t_1, greater than the time- stamp of the completing state, t_2. The scale parameter, θ, was found experimentally.

The penalty function is incorporated into computation of forward and inner probabilities as follows:

$$\alpha' = f(\mathbf{r}_p, \mathbf{r}_2) \sum_{\forall\nu} \alpha(i : X_k \rightarrow \lambda.Z\mu)\mathbf{R}_u(Z, Y)\gamma''(i : Y_j \rightarrow \nu.)$$
$$\gamma' = f(\mathbf{r}_p, \mathbf{r}_2) \sum_{\forall\nu} \gamma(i : X_k \rightarrow \lambda.Z\mu)\mathbf{R}_u(Z, Y)\gamma''(i : Y_j \rightarrow \nu.)$$

where $\mathbf{R}_u$ is the Unit Production Relation closure matrix ([11]).

2.4 Concurrency in a Sequential Machine

In the approach above there are three sources of concurrency for which the parser has to account:

1. *Concurrent events*, which are due to probabilistic nature of the event labels.
2. *Concurrent parses*, which occur while tracing the derivations of unrelated object tracks.
3. *Concurrent tracks*, which are a part of an interaction between multiple objects.

The parsing mechanism described in [2] already handles the first two points. Indeed, *concurrent events* are handled in the usual manner as a multi-valued input string. This

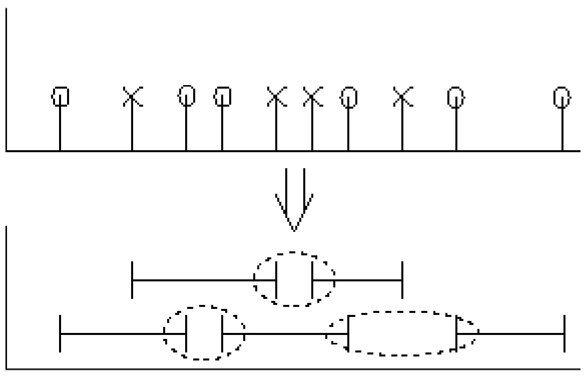

Figure 1: Performing an interleaved consistency check on the serialized events. The predecessor of the same class as the new sample is found and the check is performed on the predecessor. The dashed lines show the inter-track consistency check. Events connected by the solid lines are joined by the object identity.

problem was addressed in [5] and implemented in our earlier work ([2]). *Concurrent parses* are traced as an added benefit of the error correction with the robust grammar ([1]).

New to our framework are *Concurrent tracks*; these occur in the derivation when the grammar describes interaction between two or more objects. In our case, we need to describe interactions between cars and people. The difficulty here is presented by the fact that the events in the input stream correspond to two different entities. The entities should have consistent tracks, but since the events, related to both objects in the input stream are interleaved, the consistency check should account for that also. To accomplish that, we modify the parser in two ways:

1. Assign a class label to each production rule of the grammar. For instance, a production for `CAR-TRACK` will have a car class label[2].

2. Implement a simple search in the filtering routine (section 2.3.4), which would search the state being completed for the last child state having the class attribute the same as the completing state. The high mark is extracted from the child state and the penalty function is computed based on that attribute, instead of the high mark of the state itself. Figure 1 shows how the interleaved consistency check is performed on a string of the serialized events.

[2]It should be noted that the rule class should not be confused with the object class. The rule class is not necessarily related to the object class, although in our case it is. For instance, if one needs to handle person-person interactions with the same mechanism, different class labels should be attached to the rules corresponding to each participant.

2.5 Run-Time Incremental Parsing

At run time, the parser is presented with a potentially infinite input stream. It limits the computational complexity ($O(n^3)$) by pruning the states which have probabilities falling below certain limit and only keeping a parsing chart of a fixed but sufficient length.

To account for the fact that the string can start at any place in the input stream, we simply seed each state set with an *initial state*(see section 2.1). Whenever the final state is achieved, the Viterbi parse will automatically yield the starting position of the string having the maximum likelihood.

The parser chart does not need to be re-parsed for each sample. In fact, the algorithm can be conveniently sped up by performing the parse incrementally. At every step, the current state of the parser encodes all the history by the seed states within the window. The task is now to just perform a next iteration with the new sample, discarding the first state set of the chart. This procedure effectively prunes derivations, which are longer than the length of the chosen window. This also means that an additional condition has to be added to the filtering routine - all the states S_k^i, having $k < I_1$, where I_1 is the index of the first state set of the chart, are rejected from completion[3].

3 Recognition System

The recognition system described here consists of three main components, a tracker, a tracking event generator and a parser. The system is easily distributed and has been tested on SGI Indy R4000 running the parser and the event generator, and an SGI O2 R10000 running the tracker. The system assures real-time performance, processing data from a video camera or a VCR.

3.1 Tracker

The tracker used in the recognition system was developed by Chris Stauffer of MIT AI Lab. The description of the tracker can be found in [8]. The tracker implements an adaptive background model, which is tolerant to slow lighting changes. The tracker detects objects in the scene by presence of motion in the camera view. If this motion persists for more then some small predetermined number of frames, the object is detected and a new track container is created for it to accumulate the tracking data, related to this object. The tracker assigns a unique ID to each newly found object and tracks changes in the objects appearance, position and velocity, reporting them to the tracking event generator. Based on appearance and trajectory properties, the tracker probabilistically assigns to each object a class label (a car or a person).

[3]Note, that the length of the chart is indexed by events, not time. In our system, the value of the pruning threshold and the chart size are chosen such that cutting the chart almost never prunes active states.

3.2 Tracking Event Generator

We formulate the interactions between objects in terms of *tracker states*, rather than object trajectories, as described below. The tracker can “loose” an object and then “find” it again later, but it need not reason about the identity of the newly found object. This set of primitive tracker states, such as `object-lost`, `object-found`, forms the alphabet of the interaction, presented to the parser in form of a grammar. In this formulation, preserving the identity of an object throughout the scene is not important to the parser. The identity is preserved by the tracker where it is simple for it to do so, such as inside the consistent tracks. Then, the task of associating the disjoint pieces of the tracks falls onto the parser.

The event generator performs a simple mapping of the tracks onto a set of events, which are passed to the parser. The events are generated based on a simple environment model in the following way:

1. If the track began in an area where objects tend to enter the scene, `car-enter` and `person-enter` events are generated. The events are marked with the corresponding likelihoods to account for errors in classification. For instance if a beginning of a person track is reported by the tracker and the likelihood of that event is 0.7, a `person-enter` event with likelihood 0.7 is posted to the parser. Along with it, a complementary event `car-enter` is posted in the same time slot, with the likelihood of 0.3.

2. If the track did not begin in one of the ”entry” areas, `car-found` and `person-found` events are generated.

3. If the track ended in one of the ”exit” areas, `car-exit` and `person-exit` events are produced.

4. If the track did not end in one of the ”exit” areas, `car-lost` and `person-lost` events are posted.

5. If an object’s velocity dropped below a certain threshold, a `stop` event is generated.

Note that each track’s endpoint is represented by a pair of concurrent events, which accounts for classification errors. The parser will select on or the other, depending on which one results in the overall parse with maximum probability. Typically, at the beginning of each track, the tracker has not observed the object long enough to be certain about its class membership. Therefore, `x-enter` and `x-found` events have likelihoods close to 0.5. In contrast, by the time the object disappears or is lost, there is enough data to make more accurate classification decision. Consequently, class likelihoods of `x-exit` and `x-lost` events are typically more committal than those of `x-enter` and `x-found`.

```
TRACK:          CAR-TRACK       [.5]
              | PERSON-TRACK    [.5]
CAR-TRACK:          CAR-THROUGH       [.25]
                  | CAR-PICKUP        [.25]
                  | CAR-OUT           [.25]
                  | CAR-DROP          [.25]
CAR-PICKUP: ENTER-CAR-B CAR-STOP PERSON-
LOST
                      B-CAR-EXIT         [1.0]
ENTER-CAR-B:   CAR-ENTER               [.5]
             | CAR-ENTER CAR-HIDDEN    [.5]
CAR-HIDDEN:   CAR-LOST CAR-
FOUND             [.5]
              | CAR-LOST CAR-FOUND CAR-
HIDDEN [.5]
B-CAR-EXIT:    CAR-EXIT                 [.5]
             | CAR-HIDDEN CAR-EXIT      [.5]
CAR-EXIT:      car-exit                 [.7]
             | SKIP car-exit            [.3]
CAR-LOST:      car-lost                 [.7]
             | SKIP car-lost            [.3]
CAR-STOP:      car-stop                 [.7]
             | SKIP car-stop            [.3]
PERSON-LOST:      person-lost           [.7]
             | SKIP person-lost         [.3]
```

Figure 2: A `DROP-OFF` branch of a simplified grammar describing interactions in a parking lot.

4 Experimental Results

Here we show results of the system run on a data collected on a parking lot at Carnegie Mellon University. The system runs in real time processing data from a live video feed or a video tape. The tracker and the event generator run on an 175 MHz R10000 SGI O2 machine. The parser runs on an 200 MHz R4400 SGI Indy.

The tracker runs at approximately 12 fps on 160x120 images. It generally exhibited unbroken tracks except in cases of occlusions and extreme lighting changes. The events were mapped using a hand-coded, probabilistic classifier for object type (e.g. car or person), which used the aspect ratio of the object.

The parser requires the interaction structure described to it in terms of Stochastic Context Free Grammar. A partial listing of the grammar employed by our system for the parking lot monitoring task is shown in figure 2. Labels in capitals are the *non-terminals* while the *terminals*, or primitives, are written in small letters. Square brackets enclose probabilities associated with each production rule. These probabilities reflect the typicality of the corresponding production rule and the sequence of primitives, which it represents.

The high-level non-terminals (CAR-THROUGH, PERSON-THROUGH, PERSON-IN, CAR-OUT, CAR-PICK and DROP-OFF) have *semantic action* blocks associated with them, which are not shown in the figure for brevity. Each such action is a simple script which outputs the corresponding label (such as DROP-OFF), and all the available data, related to the non-terminal (e.g. starting and ending video frame or time- stamp). The semantic action is invoked when the final state is reached and the resulting maximum probability parse includes the corresponding non-terminal.

The production rule probabilities have been manually set to plausible values for this domain. Learning these probabilities is an interesting problem, which is planned for future work. However, our observations showed that the grammatical and spatial consistency requirements eliminate the majority of incorrect interpretations. This results in our system being quite insensitive to the precise values of these probabilities.

The test data consisted of approximately 15 minutes of video, showing several high level events such as drop-off and pick-up. The events were staged in the real environment, where the real traffic was present concurrently with the staged events. The only reason for staging the events was to have more examples within 15 minutes of video. The drop-offs and pick-ups were performed by people unfamiliar with the system. The resulting parses were output in the real time. In figures 4 a) - e) we show a sequence of 5 consecutive detections of high level events. The sequence shown in the figure, demonstrates the capability of the system to parse concurrent activities and interactions. The main event in this sequence is the DROP-OFF. While monitoring this activity, the system also detected unrelated high level events: 2 instances of CAR-THROUGH and a PERSON-THROUGH event. The figure 4 f) shows the temporal extent of activities, shown iconically in figures 4 a)-e).

All of these parses can be traced down to the primitives, which hold the track data. Consequently, the complete track can be reconstructed, as shown by white traces in figures 4 a) - e). In the longest segment of video, the event generator produces between 150 and 200 events; the exact count depends upon the reaction of the tracker to video noise. After tuning the environment map used by the event generator to convert tracks to events, all the high level interactions were correctly detected.

5 Conclusions and Future Work

In this paper we showed our approach to labeling high-level activities with a stochastic context-free parser. Main contributions of this work include:

1. extending parsing algorithm to handle concurrent events within a single parser;
2. implementing a more efficient incremental parsing scheme;
3. providing a general mechanism for consistency-based pruning;
4. demonstration of the results on the real surveillance data.

One of the issues that we will address in our future work is more accurate modeling of the environment. Currently, tracks are mapped onto events with a non-probabilistic map of the environment. This results in a high sensitivity of the event generation to subtle changes in timing of the tracker. We are also planning on learning the rule probabilities, observing the environment for extended period of time. This will help more accurate modeling the traffic patterns as well as performance of the tracker.

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] A. Bobick and Y. Ivanov. Action recognition using probabilistic parsing. In *Proc. Comp. Vis. and Pattern Rec.*, pages 196–202, Santa Barbara, CA, 1998.

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

[4] M. Brand and N. Oliver. Coupled hidden markov models for complex action recognition. In *CVPR*, pages 994–999, Puerto Rico, 1996. IEEE.

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

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

[7] Jay Clark Earley. *An Efficient Context-Free Parsing Algorithm*. Ph.d., Carnegie-Mellon University, 1968.

[8] W. E. L. Grimson, C. Stauffer, R. Romano, and L. Lee. Using adaptive tracking to classify and monitor activities. In *Comp. Vis. and Pattern Rec.*, pages 22–29, Santa Barabara, CA, 1998. IEEE.

[9] Y. A. Ivanov and A. F. Bobick. Probabilistic parsing in action recognition. Technical Report TR 450, MIT Media Lab, Vision and Modeling Group, 1997.

[10] Nuria Oliver, Barbara Rosario, and Alex Pentland. Statistical modeling of human interactions. In *CVPR, The Interpretation of Visual Motion Workshop*, pages 39–46, Santa Barbara, CA, 1998. IEEE.

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

DROP-OFF

DRIVE-IN

Event	UID	Avg. Size	Class	P	x	y	t	frame
ENTER	724	0.122553	0	0.5	0.450094	0.938069	917907137.8	1906
ENTER	665	0.046437	1	0.5	0.6107	0.94674	917907122.5	1799
PERSON-LEAVE	665	0.045869	1	0.997846	0.648089	0.98855	917907142.7	1938
STOPPED	724		0	0.995784	0.348569	0.345513	917907146.5	1964
ENTER	780	0.034293	1	0.5	0.74188	0.980292	917907151.3	1998
ENTER	790	0.069093	0	0.5	0.814565	0.032611	917907153.4	2012
FOUND	787	0.033573	1	0.5	0.297585	0.357887	917907153.1	2010
CAR-LEAVE	790	0.061263	0	0.997285	0.975971	0.211984	917907155.3	2025
PERSON-LEAVE	780	0.038616	1	0.999923	0.974494	0.865237	917907158.6	2047
PERSON-LEAVE	787	0.032045	1	0.999997	0.296519	0.183704	917907158.7	2048
ENTER	813	0.034776	1	0.5	0.012821	0.348379	917907160.9	2063
ENTER	816	0.093513	0	0.5	0.960425	0.793899	917907161.9	2070
CAR-LEAVE	724	0.097374	0	0.993211	0.972272	0.693728	917907165.2	2091
CAR-LEAVE	816	0.089424	0	0.99023	0.693699	0.990798	917907165.2	2091

Figure 3: Results of track mapping on one of the runs of the system. Two subsets of events, outlined in the picture, correspond to `DRIVE-IN` and `DROP-OFF`. Interpretation of this data is shown in figure 4.

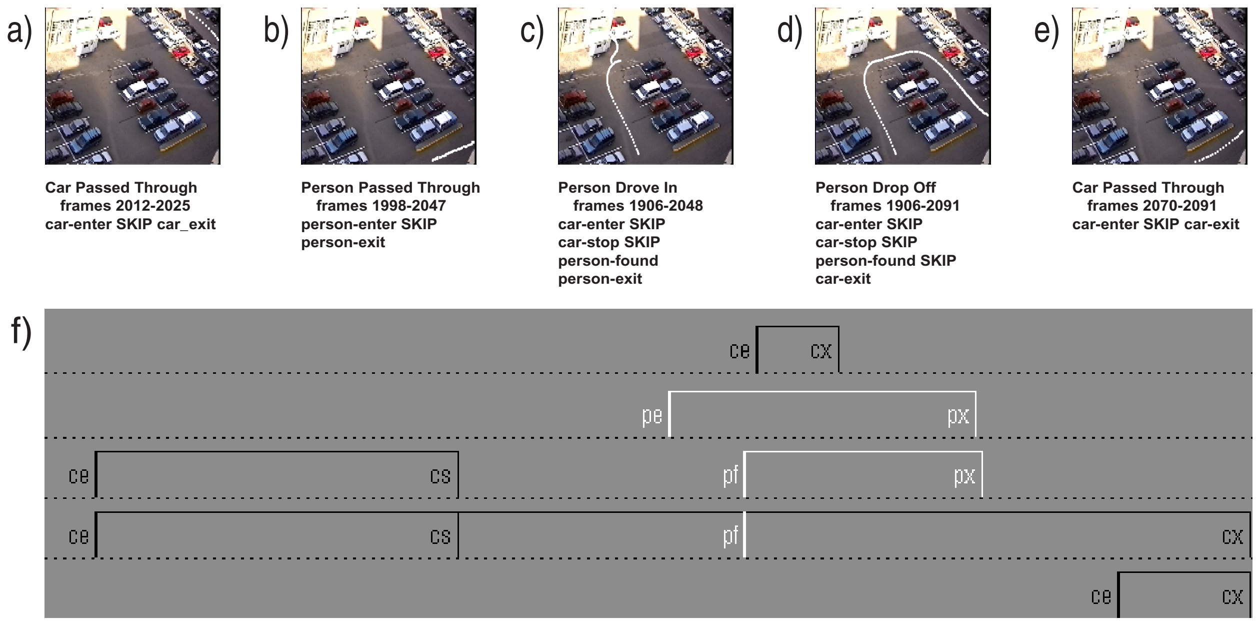
a)
b)
c)
d)
e)
f)
Car Passed Through frames 2012-2025 car-enter SKIP car_exit
Person Passed Through frames 1998-2047 person-enter SKIP person-exit
Person Drove In frames 1906-2048 car-enter SKIP car-stop SKIP person-found person-exit
Person Drop Off frames 1906-2091 car-enter SKIP car-stop SKIP person-found SKIP car-exit
Car Passed Through frames 2070-2091 car-enter SKIP car-exit
ce
cx
pe
px
ce
cs
pf
px
ce
cs
pf
cx
ce
cx

Figure 4: a) A car passed through the scene, while `DROP-OFF` was performed. Corresponding track is shown by a sequence of white pixels. b) Person passing through. c) A person left the car and exited the scene. At this moment the system has enough information to emit the `DRIVE-IN` label. d) The car leaves the scene. The conditions for `DROP-OFF` are now satisfied and the label is emitted. e) Before the car performing the `DROP-OFF` exits the scene, it yields to another car passing through, which is shown here. f) Temporal extent of the actions shown in a)-e). Actions related to people are shown in white. Top line of the picture corresponds to the label a), the bottom one - e). Car primitives are drawn in black. The figure clearly demonstrates concurrency of events. In this figure, primitive events are abbreviated as follows: **ce** - `car-enter`, **cs** - `car-stop`, **cx** - `car-exit`, **pe** - `person-enter`, **pf** - `person-found`, **px** - `person-exit`.