A Grammar for Hierarchical Object Descriptions in Logic Programs

Toufiq Parag*
Janelia Farm Research Campus-HHMI
Ashburn, VA 20147
paragt@janelia.hhmi.org

Claus Bahlmann, Vinay Shet, Maneesh Singh
Siemens Corporate Research
Princeton, NJ 08540
{claus.bahlmann, vinay.shet, maneesh.singh}@siemens.com

Abstract

Modeling objects using formal grammars has recently regained much attention in computer vision. Probabilistic logic programming, such as Bilattice based Logical Reasoning (BLR), is shown to produce impressive results in object detection/recognition. Although hierarchical object descriptions are preferred in high-level vision tasks for several reasons, BLR has been applied to non-hierarchical object grammars (compositional descriptions of object class). To better align logic programs (esp. BLR) with compositional object hierarchies, we provide a formal grammar, which can guide domain experts to describe objects. That is, we introduce a context-sensitive specification grammar or a meta-grammar, the language of which is the set of all possible object grammars. We show the practicality of the approach by an automatic compiler that translates example object grammars into a BLR logic program and applied it for detecting Graphical User Interface (GUI) components.

1. Introduction

Stochastic rule models for images have recently regained much interest in computer vision community. Several works [1, 3, 7] have proposed methods to model objects or scenes using stochastic rules for decision making purposes (e.g., recognition). One important characteristics of these studies is that they describe an object/scene in a hierarchical fashion where each entity is recursively decomposed into smaller constituent parts. Parts at the same level of hierarchy generally have some sort of semantic (geometric) relations among them.

Intuitively, a stochastic hierarchical model for objects seems more appealing than both holistic models and independent part-based models such as bag of features approach [2]. Hierarchical models are less likely to miss an object than the holistic models due to occlusion or misdetection. At the same time, a hierarchical description is difficult to be deceived by a meaningless organization of parts. The agglomeration of smaller parts into a composition in a hierarchical model also reduces the complexity for any search to be conducted on them. Similar compositional models for recognition can also be found in [8, 4]. Although the compositional tree structure described in these models are closely tied to some grammar, none of [8, 4] employ logic programs for inference.

Recent probabilistic logical model Bilattice based Logical Reasoning (BLR) [6] exploits a mathematical structure, namely bilattice framework, to incorporate uncertainties into first order logic description of an event/object. Each rule describing an event is associated with a quantitative measure of degree of belief. Given several such rules, each of which would ideally define components of an event/entity, the BLR framework combines them to make a decision (with a certain degree of belief) regarding the whole event/entity. Unlike other logic based frameworks, BLR is able to utilize stochastic rules describing the absence of a certain event or entity, i.e., they allow implications such as $\neg A \leftarrow B$. Such implications facilitate BLR to act as a discriminative model rather than a generative model (e.g. Bayesian Networks [5]).

However, the BLR framework itself does not enforce the user to adopt a modular definition of the object of interest. Therefore, the framework by itself does not enjoy the merits of a hierarchical model. In this paper, we propose a principled way, i.e., a meta-grammar, to write rules intended for general extended logic programs for object detection/recognition. The proposed meta-grammar guides (and constrains) users to define the rules hierarchically. In many ways, the resulting object description rules orients itself towards the AND-OR structure of [9] and [4] , while extending this concept by a few notable aspects (e.g., a formal integration of negation and function definitions, as will be explained). It should be noted here that, although we describe the proposed grammar to be primarily applicable to BLR, it is not specific to a certain implementation or language. In general, a grammar instance can be generically translated into a variety of different logic programs and lan-

*Work performed at Siemens Coroprate Research guage of choice.

In their survey on stochastic grammar of images, Zhu and Mumford [9] describe several studies that represent the knowledge about an object structure in a so called AND-OR graph. These studies introduce a hierarchical description of an entity. An object or entity is considered as a combination or conjunction of several components (children in the graph). For example, a wall clock is made up of a dial and hands to indicate hour, minute and second. Therefore, a clock is a conjunction of dial and hands and is represented by an AND node in this AND-OR graph.

Each component or part can be of different shapes or sizes. This diversity of configurations is captured by modeling the parts as a disjunction of several configurations. For example, continuing with the example of clock description, the dial of a clock could be rectangular, circular or even hexagonal. Therefore, the dial or the set of hands correspond to OR nodes in the graph.

These alternative configurations are further divided into smaller components and thus become an AND node of the graph. This process is continued recursively until we reach the elementary features extracted from the image. The intuitive top-down disintegration of parts facilitates an easy procedure to construct a model for an object.

In a rule-based environment, we use a formal grammar, which is referred as object grammar, to represent an object. In this grammar, the low level image features constitute the set of terminals. Contrary to monolithic models where an object is expressed with these low level features straight away, a hierarchical model defines some intermediate (and may even be hypothetical) parts/components of the object and defines the object as a semantically constrained set of these parts. These parts are the non-terminals in the object grammar.

In order to formally restrict the object grammar, we are proposing a specification grammar for object grammars. We wish to constrain the object representation to be expressed in a hierarchical manner with the options for accommodating the diversity in the construction. The resulting object grammar much similar to an intuitive graphical object representation. Each sentence of the the specification grammar is an object grammar with the aforementioned desired properties. In order to ground the proposed object grammar to an particular application case, we implement a compiler that parses and translates example object grammars into a PROLOG implementation of a BLR logic program.

The proposed technique is evaluated for software GUI (Graphical User Interface) component detection in an attempt to partially automate software testing process. Our vision based GUI parsing relies on mid- and high-level compositions of the low level features. The compositional hierarchical model is particularly suited to the GUI detection problem, as elements in a GUI are often constructed by sharing and agglomerating simple features. Neither holistic nor the bag of features model is appropriate in this scenario respectively due to large degrees of freedom in variation and substantial overlap among different GUI elements.

2. Grammar Based Object Description

The proposed syntax encourages the user to describe an object model hierarchically with AND-OR nodes. Any object is considered as a collection of its parts tied up together by semantic relationships. Each of the parts can assume any of the several alternative configurations. The proposed syntax explicitly suggests how the semantic relationships among the parts should be defined. We also have a provision for a set of relations and functions that verify and calculate properties of the parts to be utilized by the reasoning at a higher level of the graph. For this purpose, we maintain a dictionary of relations and functions which are shared among the set of rules we are writing. Once the model description is complete, it is the task of logic program to analyze and combine all the facts with the rules to evaluate a decision.

In the following, we will discuss the role of the grammar formulation on two different levels. For ease of understanding we show examples of objects of our interest in Figure 2(a). This is a screenshot of a GUI appearance – the objects we wish to detect are pushbutton, radiobutton, cehckbox etc. as shown in this figure. We first provide in Section 2.1 an example of object grammars that we intend to generate. Section 2.2 then describes the specification grammar we propose in this work.

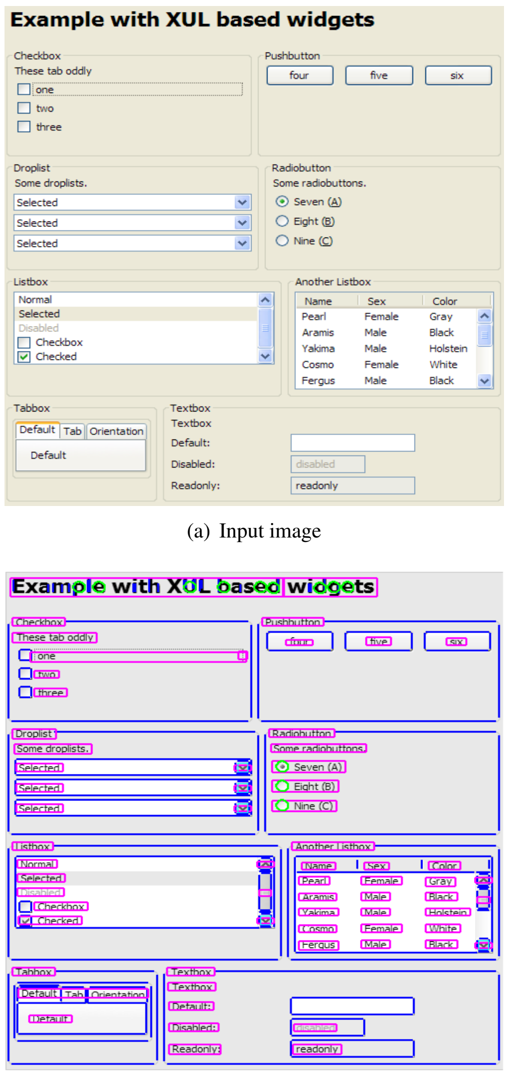
2.1. Object Grammar

The object grammar takes the form of a first order logic. In this syntax, the low level image features become the terminal symbols. For the problem of GUI recognition, we are using simple features, such as, line, text, circle, etc. as overlaid on an example GUI with colors blue, purple, green respectively in Figure 2(b). These basic features are combined and augmented by a set of geometric relations (i.e., predicates) and functions to form non-terminal symbols, such as, `Rectangle, TextCluster` etc.

The non-terminals are the intermediate parts of the object itself and are referred to as entities. The relations, such as, `is_close`, `is_inside`, etc., define the geometric relations among the features. We also use functions, such as, `compute_center, compute_intersection`, etc., to calculate necessary quantities that are used by the relations, entities and configurations. The semantic of the grammar is defined by the symbols $\lhd$, $\vee$, $\wedge$, $\neg$, and [Ef, Ea], which correspond respectively to implication, disjunction, conjunction, negation, and uncertainty values Ef and Ea as numerical evidence for and against a proposition in the logic program (see [6] for details).

```
  i. PushbuttonStart(a,b) ◁ Pushbutton(a,b).
 ii. Pushbutton(a,b) ◁ RectangularTextual(a,b)  ∨  CircularTextual(a,b)  ; [1,  0].
iii. RectangularTextual(a,b) ◁ Rectangle(a,b)  ∧  TextCluster(e,f)
        :   is_inside(a,b,e,f) ∧  is_close( compute_center(a,b), compute_center(e,f))
        :    ;  [0.9,  0].
 iv. Rectangle (a,b) ◁ CompleteRectangle(a,b).
  v. CompleteRectangle(a,b) ◁  OrthogonalLinePair(p1,r1) ∧ OrthogonalLinePair(p2,r2)
        :   is_close( compute_far_endpoint(p1,r1),compute_far_endpoint(p2,r2))
        :   compute_intersection(r1,r2,a,b)  ;  [0.9, 0].
 vi. TextCluster(a,b) ◁  TextClusterSimple(a,b).
vii. TextClusterSimple(a,b) ◁  ....
```

Figure 1. Example object grammar



(a) Input image

(b) Detected elementary features.

Figure 2. GUI component detection.

Figure 1 shows an example of an object grammar. In the specific instance of this syntax, the start symbol `PushButtonStart` infers another symbol `PushButton`, corresponding to the entity to be modeled (Rule (i)). One example of specifying several configuration is given in Rule (ii) that states `PushButton` can be a rectangular entity with some text written in it or it can be a circular entity containing texts. Each of the configurations is a collection of parts having some geometric or other relations that bind them together. These parts (e.g., `Rectangle` and `TextCluster` in Rule (iii)) become new entities to be identified next. Geometric constraints like `is_close` in Rule (iii) enforces meaningful composition between two parts: in this case, centers of two parts, `Rectangle` and `TextCluster`.

2.2. Specification Grammar

The specification grammar, which generates instances of an object grammar, is a context sensitive grammar $G = (T, N, R, S)$, with T and N being the set of terminal and non-terminals, R being the set of rules and S is the start symbol. The specification grammar is defined in Figure 3. In this grammar, all the instantiations of entities and configurations, their relations and functions are considered as terminals. The locations and parameters, which we refer to as identities, are also terminals. The space-holders for these elements, e.g., `entity, config, constraint` are all non-terminal symbols. We introduce a nonterminal symbol δ to imply the description of an entity or a configuration. A string with δ followed by `entity (or config)` implies the description of that particular `entity (or config)`. The symbols π and ρ denote a list of (semantic, geometric, etc.) relations and that of functions, respectively. These list of relations and functions are assumed to be stored in a separate relation and function dictionary. Finally, $\sharp$ corresponds to a 'formatting' variable implemented by a newline followed and preceded by white-space characters. We are specifying some notations used in the grammar

i. $\texttt{S} \rightarrow \texttt{compStart}(\tau) \lhd \texttt{entity}(\tau).\ \sharp\ \delta\texttt{entity}(\tau)$

ii. $\delta\texttt{entity}(\tau) \rightarrow \texttt{entity}(\tau) \lhd \gamma \mid neg\ \texttt{entity}(\tau) \lhd \gamma \mid \texttt{feature}(\tau);[\texttt{Ef, Ea}].\sharp$
$\mid neg\ \texttt{feature}(\tau)\ ;\ [\texttt{Ef, Ea}].\sharp$

iii. $\gamma \rightarrow \texttt{config}(\tau) \vee \gamma\ \sharp\ \delta\texttt{config}(\tau) \mid neg\ \texttt{config}(\tau) \vee \gamma\ \sharp\ \delta\texttt{config}(\tau)$
$\mid \texttt{config}(\tau);[\texttt{Ef, Ea}].\ \sharp\ \delta\texttt{config}(\tau) \mid neg\ \texttt{config}(\tau);[\texttt{Ef, Ea}].\ \sharp\ \delta\texttt{config}(\tau)$

iv. $\delta\texttt{config}(\tau) \rightarrow \texttt{config}(\tau) \lhd \nu \mid neg\ \texttt{config}(\tau) \lhd \nu$

v. $\nu \rightarrow \texttt{entity}(\tau) \wedge \nu\ \sharp\ \delta\ \texttt{entity}(\tau) \mid neg\ \texttt{entity}(\tau) \wedge \nu\ \sharp\ \delta\texttt{entity}(\tau)$
$\mid \texttt{entity}(\tau)\ :\ \pi\ :\ \rho\ ;[\texttt{Ef, Ea}].\ \sharp\ \delta\texttt{entity}(\tau)$
$\mid neg\ \texttt{entity}(\tau)\ :\ \pi\ :\ \rho\ ;[\texttt{Ef, Ea}].\ \sharp\ \delta\texttt{entity}(\tau)$

vi. $\pi \rightarrow \texttt{constraint}(\tau) \wedge \pi \mid \texttt{constraint}(\tau) \mid \epsilon$

vii. $\rho \rightarrow f(\iota) \wedge \rho \mid f(\iota) \mid \epsilon$

viii. $\tau \rightarrow \iota \mid \tau\ ,\ f(\iota)\ ,\ \tau \mid f(\iota)\ ,\ \tau \mid \tau\ ,\ f(\iota) \mid f(\iota)$

ix. $\iota \rightarrow \texttt{p}\ ,\ \iota \mid \texttt{p} \mid \epsilon$

x. $neg \rightarrow \neg \mid \texttt{Not}_{NBD} \mid \texttt{Not}_{NAL}$

xi. $\sharp \rightarrow$ `WhiteSpace Newline Whitespace`

xii. `compStart` $\rightarrow$ `PushButtonStart, ...`

xiii. `entity` $\rightarrow$ `PushButton` | `Rectangle,...`

xiv. `config` $\rightarrow$ `CircularIcon` | `RectangularText, ...`

xv. `feature` $\rightarrow$ `Line` | `Circle` | `Text,...`

xvi. `constraint` $\rightarrow$ `is_close` | `is_inside` | `is_perpendicular, ...`

xvii. f $\rightarrow$ `compute_enter` | `compute_corners` | `Perimeter, ...`

xviii. `p` $\rightarrow$ `LineParam` | `XY-coord, ...`

Figure 3. Specification grammar.

in Table 1 as an easy reference for the specification grammar.

$\lhd$	implication
`feature`	elementary fetures
ν	conjunction of entities
`entity`	part of component
γ	disjunction of config
`config`	alternate configuration
$\vee$	disjunction in object grammar
$\wedge$	conjunction in object grammar
neg	negative operator in BLR
π	list of constraints
`constraint`	constraint among parts
ρ	list of functions
f	function defined on parts
τ	parameter list
ι	parameter
`Ef` and `Ea`	numerical evidence of BLR

Table 1. Explanation of symbols in the Meta-grammar

For better understanding, the explanation of meta-grammars will accompany corresponding instantiation of the variables in the context of GUI objects. The start symbol of this syntax is S. Rule (i) states the first rule of the object grammar : a component is an entity. The non-terminal δ followed by `entity` (or `config`) then provides the definition of the `entity` (or `config`), respectively. The white space character $\sharp$ acts as a delimiter between two rules of object grammar.

In terms of GUI object, `compStart` and `entity` correspond to `PushButtonStart` and `PushButton` respectively. The argument τ indicate the parameters, e.g., location coordinate, of an `entity` or `config`. Rule (ii) invokes the description of an `entity`. Each entity can assume one of the several different configurations recursively defined in Rule (iii) with non-terminal γ, or one of the basic logical features.

The *neg* operator allows to write negative implications and the symbols , Ef and Ea, within square braces in Rules (ii) and (iii) quantify the degree of belief for a rule to be true and false, respectively. These two aspects of meta-grammar are specific to BLR framework and can be removed/modified for other inference tools.

Similarly, each configuration `config` is a conjunction of parts or `entitys`, as expressed by Rule (iv) and by ν recursively in Rule (v). The instantiation of `config` in the context of GUI object are `RectangularTextual` and `CircularTextual`. The constituent `entitys` such as

`Rectangle` and `TextCluster` are supposed to abide by the constraints specified by the list π, which, in our example, are `is_close` and `is_inside`. We also allow users to compute new properties of the `config`, through the list of functions in ρ ,e.g., `compute_intersection`, to be used at different levels of the hierarchy. Note also that Rule (viii) allows the parameters τ to be functions, enabling us to define constraints on functions such as `is_close( compute_center(a,b), compute_center(e,f))`.

Rules (xii)–(xviii) state the possible instantiations of the elements depending on the specific problem at hand. In the present grammar formulation, the concrete entities, concepts, etc. are not constrained to be identical on both sides of an object grammar, e.g., in Rules (ii) and (iv). This can be formally achieved with help of an attribute grammar formulation. Having now formally defined the language of object grammars, it is straightforward to parse object grammars and translate them into an equivalent, low- level program, such as a set of BLR rules.

The constraint to alternate between AND-OR entity/configuration enforces one to produce modular, reusable grammar that is similar to an intuitive graphical representation and at the same time flexible enough to incorporate a wider formulation of the object. From the system design perspective, the meta-grammar can a basis of a graphical editing tool for object description.

2.3. Practical Consideration

Detection time of an object reduces when we use a hierarchical model for the object as opposed to a monolithic model of it. Let us consider an object is composed of p basic features and hypothesize an intermediate part of this object to comprise $q < p$ features. If there are n basic features detected in the whole image, detecting this intermediate part by a logic program has a time complexity bounded by $\binom{n}{q} = O(n^q)$ and detecting the object itself from these $\frac{n}{q}$ intermediate parts has a time complexity $\binom{n/q}{p/q} = O((\frac{n}{q})^{\frac{p}{q}})$. Whereas, detecting the object from the basic features themselves requires $\binom{n}{p} = O(n^p)$ time which is larger than $O((\frac{n}{q})^{\frac{p}{q}}) + O(n^q)$.

Breaking down larger objects into a deeper object hierarchy also allows us to maintain a more generic dictionary of generic relations and functions. This substantially reduces the size of object model description, increases modularity and readability and further simplifies sharing of the relations and functions among different objects.

3. Application: Detection of GUI widgets

For GUI component detection experiment, we created object grammars for groupboxes, pushbuttons, radiobuttons, droplists, and checkboxes, as shown in Figure 2(a).

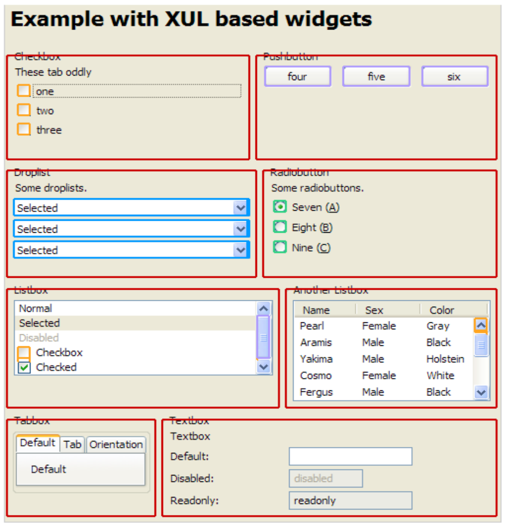


Figure 4. GUI component detection output. Red: groupbox, purple: pushbutton, green: radiobutton, blue: droplist

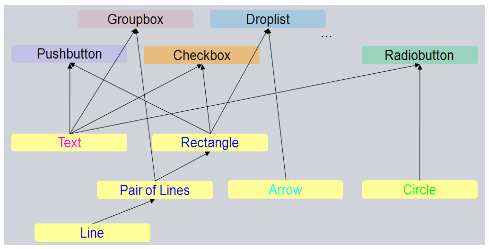


Figure 5. Dependence among parts.

A dependency graph of the main concepts in this grammar is sketched in Figure 5. Starting from the initial features, an arrow in the dependency graph implies constituent component of a part or entity. The dependency graph clearly shows how same elementary feature or intermediate parts are shared by different GUI objects. We applied BLR inference on the object grammar generated by the meta-grammar compiler.

3.1. Qualitative results

The result of the logical inference on the example image of Figure 2(a) is shown in Figure 4. It can be seen that all of the groupboxes (red), pushbuttons (purple), radiobuttons (green), and droplist (blue) are correctly detected. In this example, a single checkbox (orange) has been missed, and false alarms for checkboxes and a pushbutton can be identified. Our object grammar descriptions of these components were rather simple – we foresee a more accurate recognition with a more detailed grammar.

3.2. Quantitative results

A quantitative experimental analysis has been performed on a set of 6 images of resolution 1280×960, comprising a total of 20-40 objects per category. Table 2 shows the detection rate (true positive rate; TP rate) and the number of false alarms (FP) per image. Good results were obtained for pushbuttons, droplists, and radiobuttons, checkboxes provoked misses and false alarms. Again, a grammar containing more details is expected to improve the result.

Component	% correct	FA/Image
Pushbutton	90%	0.17
Menulist	100%	0
Radiobutton	90%	0.33
Checkbox	67%	2.67

Table 2. Quantitative results for different GUI components

3.3. Sharing subgraphs

As can be seen from Figure 5, different GUI elements share certain subgraphs in the grammar. For instance, all of pushbutton, checkbox, and droplist are implicated by the rectangle node. This property is specific to the modular and deep representation chosen, compared to a monolithic one, and specifically beneficial for robustness (in the context of parameter learning) and computational complexity.

For logical inference, it is beneficial to perform inference bottom-up, corresponding to a logical inference strategy called forward chaining, such that nodes are computed only once. In order to support the comments made in Section 2.3 by a concrete example, we set up a small experiment, where we infer about the rectangle by two different strategies. In a monolithic approach, a rectangle is directly implicated by a set of four lines, two of which are parallel to each other but perpendicular to the other two lines. In a finer grained approach, a rectangle is implicated by two (L-shaped) orthogonal pairs of lines, which in turn are implicated from the set of all detected lines. In both cases, we utilize neighborhood information to speed up the search. Both approaches produce the identical output, however, the monolithic one 4 times slower. This interesting result is an indication for a practice to prefer deep hierarchies over shallow ones.

4. Discussion

We have proposed a generic meta-grammar for producing hierarchical object description rules to be used with a logic program, focusing on bilattice based logical reasoning (BLR). We have shown results for parsing a GUI for the purpose of robust GUI testing. In the computer vision context, this approach is not limited to GUI parsing, but can advantageously be applied to many kinds of object detection and recognition. This particularly applies to situations where domain knowledge is available and can be formulated by domain experts. The work of [6] demonstrates the utility of such model for several different examples. For future work, we want to formulate grammars for additional elementary GUI components as well as more complex structures, such as graphs, images (2D/3D), etc.

In addition, usability aspects of the grammar can be further addressed by a graphical tool, where domain experts can visually create a grammar for objects of interest. The proposed meta-grammar can be utilized to define activities as well, facilitating activity detection from a video. The graphical tool will be constrained to conform with our meta-grammar. It would provide the user with flexibility to define the object or activity, and their parts, following the alternating conjunction-disjunction format and using the relation-function dictionary.

Overall, we would like to point out that, though the meta-grammar was designed to work with BLR logic programs, the basic structure of it is much more general. With minor modifications, it is possible to produce a specification grammar for other stochastic logic programs being utilized in computer vision.

Acknowledgements: The authors gratefully acknowledge help from Visvanathan Ramesh, Steve Masticola, Michael Gall and Roberto Suez of SCR.

References

[1] H. Chen, Z. J. Xu, Z. Q. Liu, and S. Zhu. Composite templates for cloth matching and sketching. In *CVPR*, 2006. 1

[2] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, 2004. 1

[3] F. Han and S. Zhu. Bottom-up/top-down image parsing by attribute graph grammar. In *ICCV*, 2005. 1

[4] L. Lin, T. Wu, J. Porway, and Z. Xu. A stochastic graph grammar for compositional object representation and recognition. *Pattern Recogn.*, 42(7):1297–1307, 2009. 1

[5] H. Schneiderman. Learning a restricted bayesian network for object detection. In *CVPR*, 2004. 1

[6] V. Shet, M. Singh, C. Bahlmann, V. Ramesh, J. Neumann, and L. Davis. Predicate logic based image grammars for complex pattern recognition. *IJCV*, 93(2):141–161, 2011. 1, 2, 6

[7] Z. Tu, X. R. Chen, A. Yullie, and S. Zhu. Image parsing: unifying segmentation, detection and recognition. *IJCV*, 63(2):113–140, 2005. 1

[8] W. Wang, I. Pollak, T. shing Wong, C. A. Bouman, and M. P. Harper. Hierarchical stochastic image grammars for classification and segmentation. *IEEE Trans. Image Processing*, 15:3033–3052, 2006. 1

[9] S. Zhu and D. Mumford. A stochastic grammar of images. *Foundation and Trends in Computer Graphics and Vision*, 2(4):259–362, 2006. 1, 2