Encoding Images into Constraint Expressions

Kazuo Hiraki, John Gennari, Yoshinobu Yamamoto, Yuichiro Anzai
Department of Computer Science, Keio University
3-14-1, Hiyoshi Kohoku-ku Yokohama 223 Japan
E-Mail: {hiraki,gennari,yoshinov, anzai}@aa.cs.keio.ac.jp

ABSTRACT

This paper presents a method, **generalization to interval**, that can encode images into symbolic expressions. This method generalizes over instances of spatial patterns, and outputs a *constraint program* that can be used declaratively as a learned concept about spatial patterns, and procedurally as a method for reasoning about spatial relations. Thus our method transforms numeric spatial patterns to symbolic declarative/procedural representations. We have implemented **generalization to interval** with ACORN,[1] a system that acquires knowledge about spatial relations by observing 2-D raster images. We have applied this system to some layout problems to demonstrate the ability of the system and the flexibility of constraint programs for knowledge representation.

1. Introduction

Representation of spatial knowledge is an important task for intelligent agents. This task can arise in many domains: visual scene understanding, problem solving, robot navigation and so on. One important aspect of spatial knowledge is the use of a set of symbolic predicates that define spatial relations among objects in a scene. Classic system such as STRIPS [Fikes 71] or GPS [Newell 63] use symbolic predicates for representing spatial knowledge. For example, in the blocks world domain, such systems usually use primitive predicates such as: **on(block1, table)**, **right-of(block1,block2)**, and **top-of(block3)**. These primitives are an abstraction of the actual scene – they give only approximate information about the location of objects – but they are important abstractions for reasoning about objects in the environment.

However, in order to apply such system to real-world domains, one would need a perceptual system to provide the appropriate symbolic primitives for reasoning (see Figure 1a). Winston's ARCH system [Winston 75] and Connell's system [Connell 87] are designed in this fashion: a vision system translated images into a set of symbolic facts, which were then used by a concept learning system. This approach leads to some difficult questions for the perceptual system: the number of primitives is not known, nor methods for mapping continuous image information into symbolic primitives. Ideally, a perceptual system should be general-purpose, and not restricted to a particular set of symbolic predicates.

Figure 2 demonstrates one problem with using *a priori* primitives for concept learning. Suppose that a vision system has six primitives to represent the distance between two objects: **very very near, very near, near, far, very far, very very far**. Given two scenes of two objects **A** and **B**,their distances are **11** and **19**, as examples for target concept. The vision system encodes these values to predicates: {**very-near(A,B), near(A,B)**} and { **far(A,B)** }. Some learning systems simply apply a kind of *dropping condition rule* to these predicates. This would mean that the system could not describe both examples with a single concept (all conditions would be dropped), even if this may be an appropriate decision.

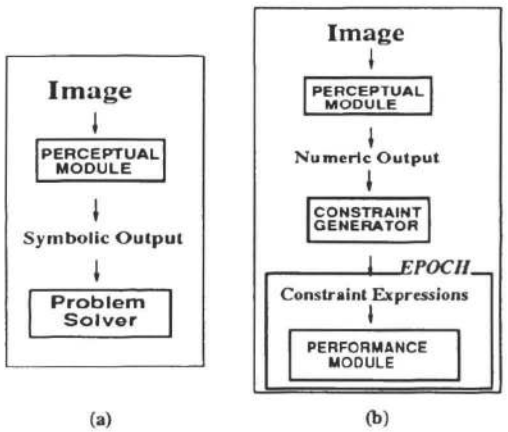


Figure 1: Datastream for classic problem solver (a) and ACORN (b).

[1] ACquisition Of Relations from Numeric data

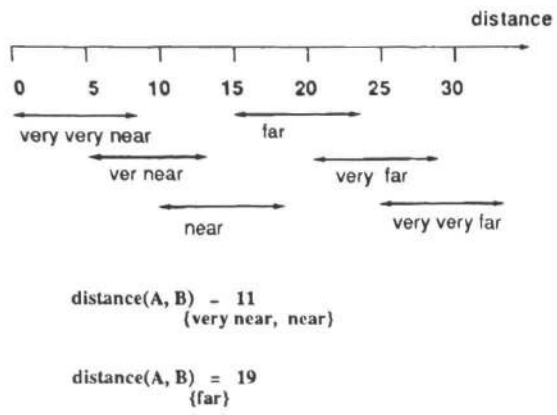


Figure 2: An inappropriateness of using *a priori* primitives

In contrast with defining *a priori* primitives, we assume all symbolic expressions are grounded in the environment. Primitive expressions should be learned from spatial patterns. According to this assumption, we have developed ACORN, a system that addresses the task of learning such expressions directly from visual scenes. In particular, our learning task can be described as follows:

Given: A set of positive training instances, where instances are 2-dimensional raster images of a scene of labeled objects.

Find: Constraint expressions among given images.

There are two important features of this task. First, it requires a translation from continuous, numeric information to more approximate symbolic knowledge.[2] The learned expressions should be invariant under translation, and should hold true over a variety of minor perturbations to the numeric coordinate input. Second, the learned knowledge should be useful for a variety of performance tasks. It should be possible to recognize examples, to generate scenes that satisfy a set of constraints, to make inferences from a set of relations, or to detect redundancy and inconsistencies in a set of relations. To this end, ACORN uses *constraint programming* [Sussman 80][Leler 88] to represent its acquired knowledge.

In this paper, we focus on layout problems to demonstrate ACORN's performance. The layout task has been studied in some fields ([Haar 82] [Yamada 90]). However, they lack the generality of ACORN's learning mechanism.

[2]Note that this is not the first system to learn symbolic information from numeric data. Aha's IBL system [Aha 91], Quinlan's C4 system [Quinlan 87] and the CART system [Breiman 84] are classifiers that work with numeric data. However, these have not been applied to spatial relations, and their learned knowledge structures have only been used for classification.

ACORN is a general-purpose system for learning symbolic relational predicates directly from numeric, image information. The next section describes our system more completely, focusing on knowledge representation and a description of the learning method. Section 3 describes the performance system and our experimentation thus far, and we conclude the paper with a discussion of future work with ACORN.

2. Description of ACORN

ACORN consists of three modules: the *perceptual module*, *the constraint generator* (or learning module) and the *performance module*. Figure 1b shows the data stream between these modules. The perceptual module generates initial, numeric scene descriptions directly from raster information. From these descriptions, the constraint generator learns a set of *constraint expressions* that describe the spatial relations between objects in the scene. Finally, the performance module can use these constraints to make predictions about a scene, make inferences with the learned relations, or generate scenes given a set of objects and relationships among those objects.

Perceptual Module

Although ACORN uses numeric input, it does not work with images directly and requires an intermediate representation of objects. Currently, we have limited our domains to two-dimensional images of rectangles.[3] For these images, the perceptual module's task is to scan a scene of dark rectangles on light paper, identify the rectangles and output a numeric representation and an arbitrary label for each rectangle. Although nontrivial, this is well within the ability of current vision systems.

In more detail, the numeric representation of each rectangle includes:

- Coordinates of each vertex of the rectangle on the global x-y coordinate system.
- Coordinates of the center of the rectangle on the global x-y coordinate system.
- Object-centered coordinate axes (major axis and minor axis) for the rectangle.
- Height of rectangle.
- Width of rectangle.

Note that only the first item need be observed from the raster image; once the four vertices are known, all other representational data can be computed. There is nothing special-purpose about this representation; indeed, we expect that the system could use other types of representations for other shapes.

[3]In Section 4, we discuss the possibility of extending ACORN to work with a richer variety of shapes.

Constraint Expression

ACORN represents learned knowledge with *constraint logic program*[Jaffar 87]. In general, constraint logic program consists of the set of *extended horn clauses* in the form:

$$Relation(\langle args \rangle) \leftarrow P_1 \wedge P_2 \wedge \ldots \wedge P_n \wedge C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

where P_i is a *term* (as in Prolog) and C_j is an algebraic constraint expression such as an equation or inequality.

One of the most remarkable features of constraint-based language is that the user does not have to write a procedure explicitly in order to solve constraints. The user has only to give *declarative* relations among variables. Therefore constraints can be used to make a variety of inferences. For example, the constraint

$$convert(C, F) \leftarrow (F = \frac{5C}{9} + 32)$$

can be used to make predictions either about F given C or C given F, without requiring separate rules. In general, constraint expressions make use of a powerful *constraint solver* that is used to make inferences or solve problems. This approach lets declarative knowledge be used procedurally, allowing ACORN to reason about its learned spatial relations in a variety of tasks.

One of the advantages of using a constraint logic programming language is that semantics can be formulated more easily than other constraint-based languages[Sussman 80][Leler 88][Borning 81], using logic programming semantics (see [Jaffar 87] for further discussion). Constraint logic programming has proved valuable in a number of domains, such as option trading[Lassez 87] and scheduling [Dincbas 88].

ACORN uses EPOCH, a constraint logic programming system developed in our laboratory.[4] EPOCH's constraint solver uses the *simplex algorithm* to solve linear programming problems. EPOCH is similar to CLP($\Re$) [Lassez 87] in its ability as a constraint solver. However, EPOCH uses two types of constraints: *necessary constraints* that must be satisfied and *preferred constraints* that should be, but do not have to be satisfied. Preferred constraints are used to choose an optimal solution when there is more than one possible answer (see [Hiraki, in press] for more details).

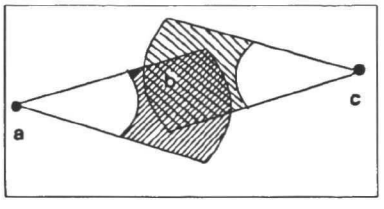
Learning Method

As described in the previous section, learned knowledge is represented with constraint expressions. ACORN uses binary constraints – it looks at information about a pair of rectangles. Note that these can be used to model n-ary spatial relations, simply by learning about all C_2^n pairs of objects in the scene. Currently, the learning module of ACORN takes the information from the perceptual module and generates constraints based on the following *relational attributes*:

- distance (γ) between corresponding vertices of the objects ($0 < \gamma$).
- direction (θ_{ij}) from $vertex_i$ to $vertex_j$ ($0 < \theta_{ij} \leq \pi$).

Certainly, other relational attributes could be used (angle between major axes, relative height or width, etc.), but thus far, these attributes have proved sufficient.

As the system observes positive instances of the relation, it builds both necessary and preferred constraints. Necessary constraints represent the necessary conditions that any instance of the relation must satisfy. They are expressed by a set of ranges defined by algebraic inequalities. For example, the necessary constraints for "point a is right-of point b" is expressed by the fan-shape region illustrated in Figure 3a. Because EPOCH's constraint solver only works with linear problems, we approximate this region with four linear inequalities.[5] EPOCH incorporates these inequalities in a constraint program that represents the fan-shaped region; Figure 3b shows the program in Prolog form. The four inequalities (the necessary constraints) correspond to four lines delimiting the region used to define "right-of" The satisfying region shrinks as the number of constraints increases (e.g. by adding "point b is right-of point c"). If the region becomes null, the constraints cannot be satisfied.



(a)

```
constraint( a([Xa,Ya]),b([Xb,Yb]) ) :-
        X = (Xb - Xa), Y = (Yb - Ya),
        Y =< 20.0*X - 100.0,
        Y >= -0.6*X, Y =< 0.4*X,
        Y >= 20.0*X - 200.0,
        pref:(X = 7.5), pref:(Y = 0.0).
```

(b)

Figure 3: (a) A region defined by some constraints. (b) A simple constraint program for the region.

On the other hand, preferred constraints represent a typical member of the learned spatial relation. In Figure 3b, *pref:* indicates a preferred constraint. ACORN uses preferred constraints for finding the optimal location of an object given a region that satisfies the necessary constraints of the learned relation. Preferred constraints are created by computing averages over the set of training instances.

[4] Both EPOCH and ACORN are implemented in Quintus Prolog.

[5] There are constraint solvers that can deal with non-linear forms, but these are more expensive [Sakai 89].

In order to learn the necessary constraints for a spatial relation, ACORN uses a simple learning rule as it observes training instances. This rule is an application to constraint programming of Michalski's "closing interval generalization" heuristic [Michalski 83]; we will refer to this as **generalization to interval:**

> For each relational attribute, given numeric values, x_1 and x_2, from two positive training instances, assume that all numeric values between x_1 and x_2 are also examples of positive instances.

This can be used directly with linear attributes such as distance, but for cyclical attributes, such as direction defined as an angle, two positive instances produce a pair of intervals. In this case ACORN adopts the heuristic of generalizing over the smaller interval.

Table 1 presents the incremental algorithm for a single linear attribute in more detail. Given the old constraint for an attribute, C_j, and the value of the attribute for the new positive instance, v_j, this algorithm returns a new constraint, C_j'. A constraint, C, is defined by the bounding interval $[\alpha, \beta]$.

Table 1: ACORN's learning algorithm for a linear relational attribute.

generalization-to-interval(C_j, v_j)
  If v_j satisfies the constraint C_j
    Then return $C_j' := C_j$
  Else If $v_j \leq \alpha$ **Then** return $C_j' := [v_j, \beta]$
  Else return $C_j' := [\alpha, v_j]$

This learning method is only an initial heuristic; it is a simple method, and has a number of drawbacks. One problem occurs with negative training instances: it is unclear how to modify constraints when the negative instance includes values between α and β. Additionally, α and β are hard thresholds: after learning, a test instance with values $\alpha - \varepsilon$, or $\beta + \varepsilon$ does not satisfy the constraint. Especially in noisy domains, one would prefer a more flexible approach. Despite its limitations, we have found that even this simple form of learning can be very useful for recognizing and reasoning with spatial relations.

3. Performance of ACORN

As we described earlier, the advantage of using constraint expressions is that the knowledge can be used in a variety of performance tasks without requiring special purpose representations or rules. If knowledge about spatial relations is represented as a set of constraint expressions, the performance system should be able to recognize examples of a given relation, generate scenes that satisfy a set of relations, make inference from a set relations, or detect redundancy and inconsistencies in a set of relations.

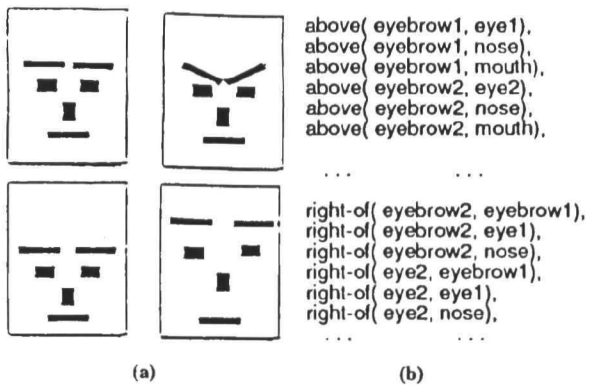


Figure 4: Face domain input for ACORN: (a) Images (b) Partial list of training relations.

In this section, we describe ACORN's ability with two performance tasks. We begin with the layout task: generating scenes from a set of relational specifications. Next, we consider a variation on this task: completing partial scenes. In this task, the system learns a set of constraints that represent the spatial relations among a set of objects in a scene. Then, the system can use these constraints to predict the location of missing objects.

The Layout Task

For the layout problem, we use the simple task of arranging a set of six rectangles into a 'face'. Figure 4a shows the training instances given to ACORN. In order to learn relations, the system needs the relation names, as well as labeled scenes. Thus, the rectangles in these figures are labeled $eye1, eye2, eyebrow1, eyebrow2, nose, mouth$ and there are two relations given: **above(X,Y)** and **right-of(X,Y)**. Figure 4b gives a partial list of the relations that were associated with each scene. ACORN uses these as positive training instances for building the necessary constraints that define the spatial relation. In particular, it builds a *constraint program* for each relation as follows:[6]

$$\begin{aligned} &constraint(Relation\text{-}name,\\ &\quad [Vertex_1^1, Vertex_2^1, Vertex_3^1, Vertex_4^1],\\ &\quad [Vertex_1^2, Vertex_2^2, Vertex_3^2, Vertex_4^2]) \leftarrow\\ &\quad necessary\_constr_1, \cdots, necessary\_constr_n,\\ &\quad preferred\_constr_1, \cdots, preferred\_constr_m. \end{aligned}$$

Note that the two arguments to this relation are represented as lists of vertices.

After building constraint programs for the relations **above** and **right-of**, ACORN can construct any arrangement of rectangles given a specification in terms of the two acquired relations. For example, given the specification:

{right-of(eye2,eye1),
above(eyebrow1,eye1),

[6] We use standard Prolog notation here.

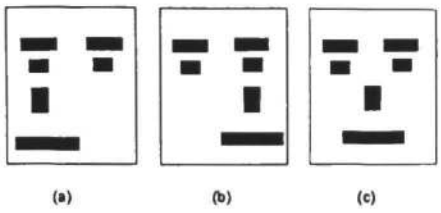


Figure 5: Faces generated from specifications.

above(eyebrow2,eye2),
above(eye1,nose), above(nose,mouth)} ,
ACORN generates the scene in Figure 5a, while
{right-of(eye2,eye1),
above(eyebrow1,eye1),
above(eyebrow2,eye2),
above(eye2,nose), above(nose,mouth)} ,
creates the scene in Figure 5b. Note that the nose and mouth are 'off-center' in these scenes because the preferred constraint for **above** is straight above. If we add the constraints **right-of(eye1,nose), right-of(nose,eye2)** to the specification, we get the 'corrected' scene as in Figure 5c.

Completing Partial Scenes

In addition to binary relations, ACORN can learn more complex n-ary spatial relations. As described in Section 2, the system simply models n-ary relations by building constraints for every pair of arguments. Although this means that there is a computational limit to the number of arguments in a relation (there are $O(n^2)$ constraints needed for an n-ary relation), we imagine that most useful relations do not have too many arguments. For example, the faces from the previous section can be used to learn a six-argument spatial relation: **face(eye1,eye2,eyebrow1,eyebrow2,mouth,nose)**.

Since there are six components of each face, there are 15 constraints that make up the constraint program for learning the **face** relation. Each constraint must include the component labels, but is otherwise similar to that presented in the previous section:

$$\begin{array}{l} constraint(\\ \quad obj\text{-}name1([Vertex_1^1, Vertex_2^1, Vertex_3^1, Vertex_4^1]), \\ \quad obj\text{-}name2([Vertex_1^2, Vertex_2^2, Vertex_3^2, Vertex_4^2])) \\ \quad \leftarrow \\ \qquad necessary\_constr_1, \ \cdots, \ necessary\_constr_n, \\ \qquad preferred\_constr_1, \ \cdots, \ preferred\_constr_m. \end{array}$$

(The relation name can be omitted since the system is learning only one relation.)

After acquiring a definition for this relation, it can be used in a number of ways. First, scenes can be 'recognized' as satisfying the constraints that define a face. Alternatively, the constraints can be used to predict the location of missing objects in a 'face' scene. That is, given a partial face, and knowledge about the **face** relation, ACORN can complete the scene. Figure 6 shows a pair of incomplete 'faces', and the resulting scenes as completed by ACORN. This task is similar to the layout problem: the system uses the preferred constraints to generate the location of the missing object.

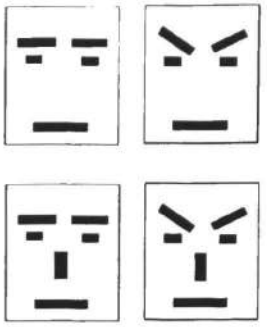


Figure 6: Two incomplete faces, and their completion by ACORN.

In general, ACORN can predict the location of any number of missing components. As more and more components are missing, the necessary constraints become looser and looser. If all components are missing, then the system uses only the preferred constraints, and this becomes a form of the layout task.

This is a simple demonstration of the generality of ACORN's acquired knowledge, and the versatility of constraint expressions. Once the system has learned a few key spatial relations, it can use this knowledge in a variety of domains and performance tasks.

4. Discussion and Future Work

As described in the above section, ACORN can perform lots of tasks with acquired constraint programs. However, ACORN includes a number of limitations that should be addressed in the future. In this section, we discuss the limitations of ACORN's learning method and indicate the directions in which the research is proceeding.

One task that seems important is to improve the learning algorithm. The system should be able to learn from negative instances and build constraints that define disjunctive spatial relations. For examples, the spatial relation **next-to(X,Y)** has to be represented by disjunctive constraint expressions. Hiraki and Anzai [Hiraki 90] describe an initial version of ACORN that was able to learn from negative instances, but only with user input, and objects that were represented as points.

A second area for improvement is the range of input images. ACORN should be able to work with triangles, circles or any concave polygon, rather than only simple rectangles. One possibility is to use 2-D generalized cylinders; these provide a consistent method for representing a variety of shapes. The potential difficulty of this approach is that it greatly increases the

number of relational attributes between each pair of objects, and thus the complexity of the constraint programs. For this problem, we should consider methods that can find the importance of relational attributes of a spatial relation and then reduce the number of given attributes.

Ultimately, knowledge about spatial relations should be used by higher-level systems such as problem solvers, natural language system, and navigation systems. In the future, we hope to connect ACORN with such a system. For example, Yamada's SPRINT[Yamada 90] system reconstructs spatial configurations from a given natural language description, can be a good application for ACORN. SPRINT uses hard-coded constraint expressions corresponding to special words that have spatial information. ACORN may be used to learn the meaning of such words and any new spatial concepts that may be used. In general, our system may be useful as an initial phase for any of the higher-level uses of spatial knowledge: route knowledge, spatial maps, and path planning.

We believe that ACORN addresses a critical task for an intelligent learning system: acquiring symbolic relational knowledge from numeric image data. Certainly, our current system includes many limitations, but we hope that it is a good start toward a general-purpose learning system for spatial relations.

Acknowledgment

This research was supported in part by a grant from IBM Japan. The authors would also like to acknowledge Pat Langley for his comments during the initial stages of this paper.

References

[Aha 91] Aha, D.W., Kibler, D. and Albert, M.K.: Instanced-based learning algorithms, *Machine Learning*, **6**, 37-66, 1991.

[Borning 81] Borning,A.: The programming language aspects of ThingLab: A constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems*, **3**, 353-387, 1981.

[Breiman 84] Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J.:*Classification and Regression Trees*, Wadsworth International Group, 1984.

[Connell 87] Connell,J. H. and Brady,M.: Generating and generalizing models of visual objects, *Artificial Intelligence*, **31**, 159-183, 1987.

[Dincbas 88] Dincbas, M., Van Hentenryck, P., Simmonis, H., Aggoum, A., Graf, T. and Berthier, F.: The constraint logic programming language CHIP, *Proc. of the International Conference on Fifth Generation Computer Systems*, 693-702, 1988.

[Fikes 71] Fikes, R. E. and Nilson, N. J. : STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence*, **2**, 1971.

[Haar 82] Haar, R. L.: Sketching: Estimating object position from relational descriptions, *Computer Graphics and Image Processing*, **19**, 227-247, 1982.

[Hiraki 90] Hiraki, K. and Anzai, Y.: Knowledge acquisition for spatial knowledge-based system, *Proc. of 5th Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1990.

[Hiraki, in press] Hiraki, K., Nishizawa, T. and Anzai, Y.: Acquisition and use of constraint expressions for spatial knowledge, (in Japanese) *Computer Software* (Journal of the Japan Society for Software Science and Technology), *in press.*

[Jaffar 87] Jaffar,J. and Lassez,J.: From unification to constraints, In Furukawa,K., Tanaka,H. and Fujisaki,T. (eds.), *Logic Programming '87, Lecture Notes in Computer Science*, Springer, 1987.

[Lassez 87] Lassez, C.: Constraint logic programming and option trading, *IBM Technical Report*, 1987.

[Leler 88] Leler, W.: *Constraint Programming Languages: Their Specification and Generation*, Addison-Wesley, 1988.

[Michalski 83] Michalski, R.: A theory and methodology of inductive learning, In Michalski, R., Carbonell, J. and Mitchell, T. (eds.), *Machine Leaning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann, 1983.

[Newell 63] Newell, A. and Simon, H. A.: GPS, A program that simulates human thought, In Feigenbaum, E.A and Feldman, J. (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963.

[Quinlan 87] Quinlan, J. R.: Simplying decision trees, *Int. Journal Man-Machine Studies*, **27**, 221-234, 1987.

[Sakai 89] Sakai, K., and Aiba, A.: CAL: A theoretical background of constraint logic programming and its applications, *J. Symbolic Computation*, **8**, 589-603, 1989.

[Sussman 80] Sussman, G. J. and Steel, G. L.: CONSTRAINTS – A language for expressing almost-hierarchical descriptions, *Artificial Intelligence*, **14**, 1-39, 1980.

[Winston 75] Winston, P.H.: Learning structural descriptions from examples, In Winston, P.H. (ed.), *The Psychology of Computer Vision*, McGraw-Hill, 1975.

[Yamada 90] Yamada, A., Amitani, K., Hoshino, T., Nishida, T., Doshita, S.,: The analysis of the spatial description in natural language and the reconstruction of the scene, (in Japanese) *Journal of Information Processing Society of Japan*, **31**, No.5, 660-672, 1990.