1990

Reasoning with visual knowledge in an object recognition system

Christine Wojnowski

Rochester Institute of Technology
School of Computer Science and Technology

Reasoning with Visual Knowledge in an Object Recognition System

by
Christine Wojnowski

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by:

Professor J. A. Biles

Dr. H. Rhody

Dr. P. Anderson

July 1990

Title of Thesis: Reasoning with Visual Knowledge in an Object Recognition System

I, _____, prefer to be contacted each time a request for a reproduction is made. I can be reached at the following address:

P.O. Box 92273
Rochester, N.Y. 14692

Date: August 20, 1990

ABSTRACT

The impact of artificial intelligence on computer vision has provided various perspectives and approaches to solving problems of the human visual system. Some of the symbolic processing and knowledge-based techniques implemented in vision systems represent a meaningful extension to the low-level, algorithmic processing which has been emphasized since the advent of the computer vision field. The higher-level processes attempt to capture the essence of visual cognition, specifically by encompassing a model of the visual world and the reasoning processes that manipulate this stored visual knowledge and environmental cues. This thesis includes a discussion of existing computer vision systems surveyed from a high-level perspective. The goal of this thesis is to develop a high-level inference system that implements reasoning processes and utilizes a visual memory model to achieve object recognition in a specific domain. The focus is on symbolically representing and reasoning with high-level knowledge using a frame-based approach. The organization and structuring of domain knowledge, reasoning processes and control and search strategies are emphasized. The implementation utilizes a frame package written in Prolog.

Computing Review Codes:

I.2	Artificial Intelligence
I.2.4	Knowledge Representation Formalisms and Methods
I.2.8	Problem Solving Control Methods and Search
I.2.10	Vision and Scene Understanding
I.5.4	Computer Vision

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The integration of artificial intelligence and computer vision has evolved in the past few decades, resulting in significant advances in the computer vision field and, consequently, more capable vision systems. The impact of artificial intelligence on computer vision has provided various perspectives and approaches to solving problems of the human visual system. Some of the symbolic processing and knowledge-based techniques implemented in vision systems represent a meaningful extension of the low-level, algorithmic processing, which has been emphasized since the advent of the computer vision field. The higher-level processes attempt to capture the essence of human visual cognition, specifically by encompassing a model of the visual world and the reasoning processes that manipulate this stored visual knowledge and environmental cues.

The goal of this thesis is to develop a high-level inference system with a focus on symbolically representing and reasoning with a visual memory model. The intent is to develop the high-level portion of a system, whose goal is object recognition in a specific domain, by emphasizing the organization and structuring of domain knowledge, reasoning processes, and control and search strategies. The basic premise of the problem-solving approach used in this thesis is that perception is a process whereby hypotheses are created about what is visualized and information is gathered to support or refute these hypotheses.

Chapter Two of this thesis provides a general overview of the aspects of human vision that relate to high-level computer vision and briefly reviews early computer vision work. Also, in Chapter Two, existing computer vision systems and related work in the area are surveyed from

a high-level perspective. In Chapter Three, the conceptual issues of reasoning with visual knowledge are explored in terms of the systems surveyed in Chapter Two with the objective of establishing the foundation for the implementation presented in this thesis. A conceptual overview of the method presented in this thesis and the details of the implementation are elaborated in Chapter Four. An analysis of the results of the implementation, an example session with the high-level inference system and test plan are discussed in Chapter Five, followed by the Conclusion. Appendix A consists of the files created during processing of the example session with the high-level inference system that is discussed in Chapter Five. A summary of the identification accuracy of the test cases and a summary of results are included in Appendix B. Appendix C consists of a guide to using the high-level inference system, and Appendix D includes the input and output files of the test cases referenced in Appendix B, plus the knowledge base and Prolog code that comprises the system developed in this thesis.

CHAPTER 2

BACKGROUND

2.1 Computer Vision and the Human Visual System

Evidence for the existence of the capacity for the human visual process to reason about the world, apparently subconsciously, can be found when the human visual system is presented with a visual illusion, an occluded object within a particular context, or an unfamiliar scene [Fisc87]. Unknowingly, the complex human visual system generates multiple, competing object or scene hypotheses and effortlessly selects the most appropriate one after having knowledgeably eliminated the alternatives, thus demonstrating the ability to "visually perceive." In sequencing the steps the human visual system undergoes when it attempts to interpret an unfamiliar scene, the visual system may alternate between more than one possible interpretation with the objective of arriving at a consistent one, thus suggesting the existence of an ability to reason about what is being observed. For instance, in describing the picture depicted in Figure 2-1, the human visual system may conclude that a white square is occluding four dark circles [Fisc87, 229]. This, in turn, contrasts with the interpretation that the white square does not actually exist, but instead is an illusion (since it is the same intensity as the background).

Reasoning about the context of a scene that is distorted or provides only partial information to the human visual system suggests a dependency on a prior familiarity with the specific context in order to identify an object or interpret a scene accurately. In addition, the context-free cues obtained from an image are usually not sufficient to interpret the image, and consequently, domain-specific knowledge associated with a scene is necessary to recognize objects

or interpret a scene. This vast amount of knowledge is acquired through experiences and apparently is stored compactly in the brain to be accessed ultimately by the human visual system. As Neisser asserts, "*Visual cognition*, then, deals with the processes by which a perceived, remembered, and thought-about world is brought into being from as unpromising a beginning as the retinal patterns" [Neis67, 4].

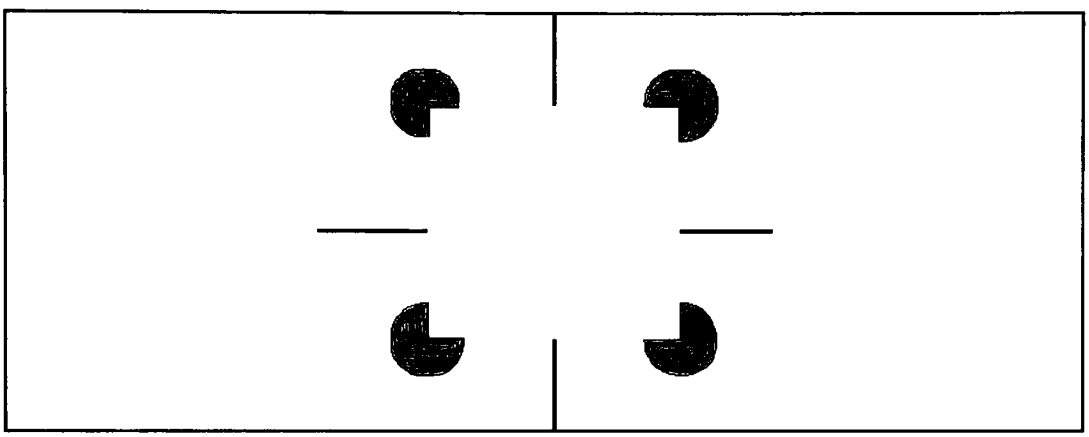

Figure 2-1 Subjective Contours (Fischler, M. A. and Firschein, O., Intelligence The Eye, the Brain and the Computer. (c) 1987 by Addison-Wesley Publishing Co. Reprinted by permission of Addison-Wesley Publishing Co., Inc., Reading, MA)

Understanding the underlying complexity of the human visual system can be crucial in providing insight into successfully developing a computer vision system. However, the emphasis placed on such an understanding does not reflect a general consensus among computer vision researchers [Lowe85, 7]. The human visual system normally operates with some degree of error, taking in varying amounts of evidence. It is a finely tuned system that processes much information and takes into consideration information derived from what is being perceived, however incomplete, along with the expectations of the observer [Barr81]. Unfortunately, it is difficult to reflect on one's own visual system, and humans often exhibit an inherent trust in their ability to perceive. This in turn sometimes contributes to a limited understanding of the human visual system.

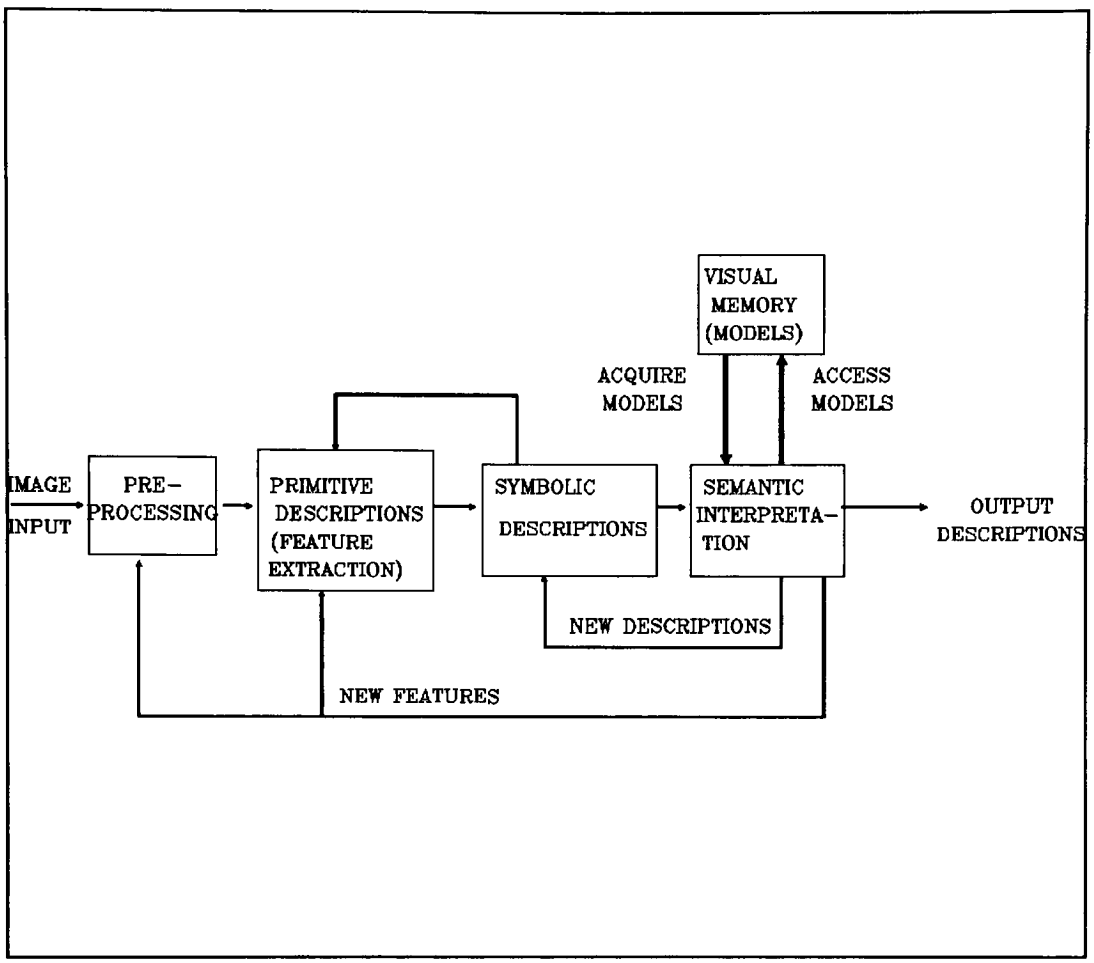

Figure 2-2 Image Understanding System ("Characterization and Requirements of Computer Vision Systems," R. Nevatia, 1978, In Computer Vision Systems, Hanson A. R., Riseman, E. M., eds. (c) 1978 by Academic Press. Reprinted by permission.)

An artificial vision system attempts to model human visual perception and cognition, and it is concerned primarily with providing a computer with the capabilities and intelligence of the human visual system. Modeling the human visual system by developing a general-purpose, domain-independent computer vision system has proved to be a difficult task [Hans78a]. Knowledge-based vision systems, as a result, have been developed that employ knowledge-based techniques, operate within a particular domain of knowledge, and are designed for a specific task. The typical input into a computer vision system is a digitized image of a scene represented as

arrays of values or a line drawing. The output can be a symbolic description, a semantic interpretation, or some other system-specific representation.

A computer vision system can be partitioned naturally into sequential levels that reflect the level of processing and representation of knowledge (see Figure 2-2) [Barr81]. Low-level (early) processing includes the processes that operate on the numerical arrays of sensory data, and is concerned with extracting information directly from the image. It is typified by data-driven or bottom-up processing. High-level (late or cognitive) processing provides the symbolic description or semantic interpretation of a scene, given the information generated by the lower levels, and focuses on modeling one's expectations and specific domain knowledge. This is accomplished by the selection and appropriate utilization of suitable knowledge representations, effective object recognition strategies, and the implementation of control strategies to direct the process of obtaining a symbolic description of a scene. High-level vision is characterized by goal-directed (top-down, predictive) analysis and the use of semantic models and relationships. Knowledge about the domain in which a scene is obtained is used extensively throughout this stage. Intermediate-level processing seeks to bridge the gap between the low and high levels and is responsible for converting the results obtained from the low level into relatively more abstract representations that are required by subsequent levels (see Figure 2-3).

2.2 Early Computer Vision Systems

The development of computer vision systems dates back to the 1960's when L. G. Roberts's program took an image of a polyhedral block scene, converted it into a line drawing, and recognized the image based on three-dimensional models of a cube, wedge and hexagonal prism [Robe65, Barr81]. Processing in Roberts's program was sequential, with segmentation followed by interpretation, and it was bottom-up in nature [Cohe82] [Barr81, 575]. The goal of

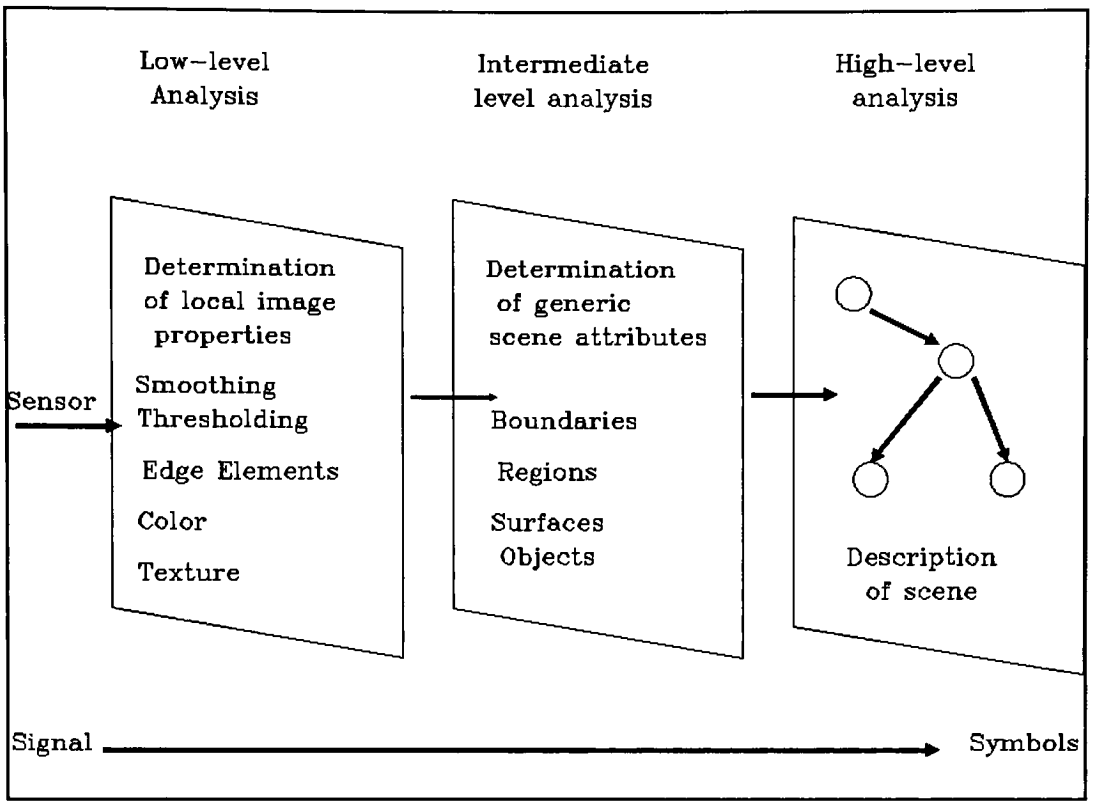

Figure 2-3 Signals-to-Symbols Paradigm (Fischler, M. A. and Firschein, O. Intelligence The Eye, the Brain and the Computer (c) 1987 by Addison-Wesley Publishing Co. Reprinted by permission of Addison-Wesley Publishing Co., Inc. Reading, MA)

the segmentation process is to partition an image into meaningful components such as regions. Unfortunately, segmentation is not always entirely reliable and therefore, errors are introduced early in processing an image [Barr81, 576]. These errors ultimately propagate to the interpretation phase, resulting in an incorrect explanation of the image. Another approach that arose from the difficulties encountered in early attempts to interpret an image entailed combining the segmentation and interpretation processes [Barr81, 576]. The segmentation phase was directed further by the interpretation phase, and the strict sense of sequential processing was abandoned.

Falk's program, INTERPRET, handled imperfect input or line drawings by employing models of objects to assist in the interpretation phase [Falk72]. Falk's program proceeds by

segmenting the line drawing, recognizing objects as instances of models, predicting a line drawing from the recognized objects, and undergoing verification, which compares the original input to the predicted line drawing. Falk employed a hypothesize-and-test strategy along with a large set of heuristics to recognize objects. Similarly, Guzman relied on heuristics to achieve the objectives of his program, SEE, which used a symbolic approach [Guzm68]. The goal of SEE was to partition a line drawing into three-dimensional elements using heuristic knowledge of vertices to combine regions [Guzm68]. The input was a symbolic representation of points, lines and surfaces, and it assumed preprocessing was completed. The program generated output in the form of lists identifying the bodies in a scene, but it did not have the capability of handling missing lines. Guzman accomplished his goal without the use of stored object models.

Early computer vision systems were characterized by simple, restricted scene domains in which models of objects could be constructed easily [Barr81, 574]. According to Barrow and Tenenbaum, the blocks world illustrates many aspects of visual perception [Barr81, 578]. However, a more difficult task involves working with real world scenes that convey meaning. More recent computer vision systems deal with domains of this context, and as a result, knowledge representation and reasoning processes have become more relevant issues in developing such systems.

In most early computer vision work there was a predominance of numerical and algorithmic methods used and an emphasis on low-level vision. As the field of computer vision evolved and more sophisticated artificial intelligence techniques were developed, more capable and comprehensive vision systems came into existence. Specifically, there was an emergence of symbolic processing, explicit reasoning techniques, attempts to model domain knowledge efficiently, processes dealing with uncertainty, and efforts to solve problems by employing empirical rules to confine the search for a suitable solution. Consequently, efforts in high-level

computer vision expanded and related high-level issues were given more attention and consideration.

2.3 Existing Computer Vision Systems from a High-Level Perspective

The following computer vision systems and research efforts will be discussed in terms of high-level vision issues, which include knowledge base design or how visual information is represented; the method of reasoning employed--top-down, bottom-up, or a combination of both; the control strategy used; the search strategy as required by the recognition process; and the uncertainty handling mechanism, if any. In other words, the structure of the system's interpretation scheme and problem-solving approach will be examined. The purpose of the following sections is to provide a sampling of various commonly used and some not-so-common approaches to high-level vision. It is not intended to be comprehensive in scope but, instead, is intended to illustrate and exemplify the high-level aspects of developing a computer vision system. Computer vision encompasses systems whose goals include object recognition, image understanding and scene interpretation.

2.3.1 VISIONS

The goal of VISIONS, a University of Massachusetts project under the development of Allen Hanson and Edward Riseman since the mid 1970's, is to generate a symbolic representation of a three-dimensional scene as portrayed in a two-dimensional image [Hans78, Hans88]. The system employs knowledge-based techniques for the interpretation of natural images (house and road scenes) and supports three levels of representation and processing. The representation and control strategies are designed to be modular and hierarchical, and the overall structure of the system allows feedback from high levels to low levels (see Figure 2-4).

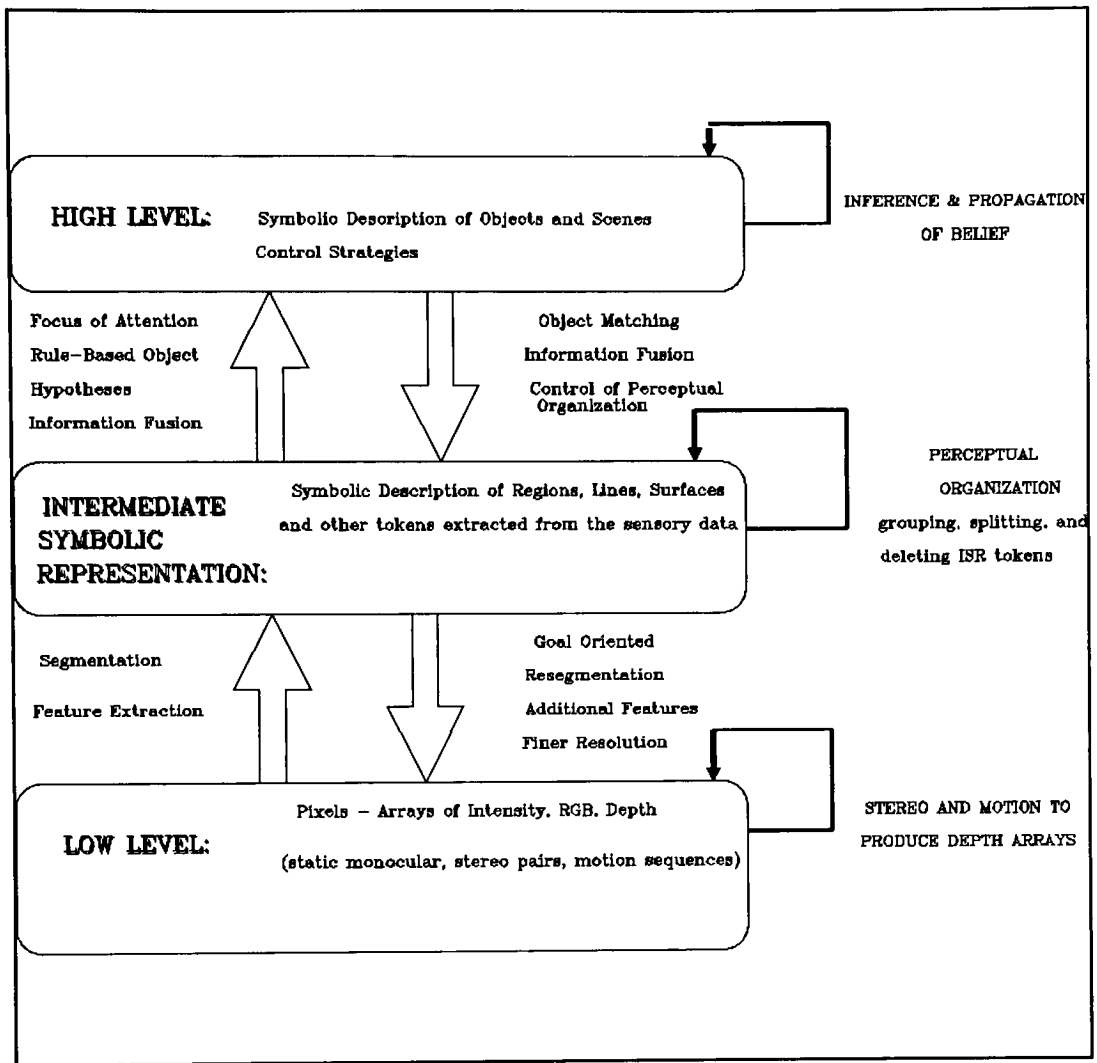

Figure 2-4 Overview of VISIONS ("The VISIONS Image Understanding System", by Hanson, A. R. and Riseman, E. M. In Advances in Computer Vision, Volume 1, Brown, Christopher, ed. (c) 1988 Lawrence Erlbaum Associates. Reprinted by permission.)

At the lowest level of representation, operations are performed on numerical arrays of sensory data. At the intermediate level, symbolic tokens representing regions, lines and surfaces are constructed from the features extracted at the low level. An intermediate symbolic representation of the image is constructed using a segmentation procedure, and an incomplete interpretation is then built by labeling intermediate tokens. Object labels activate procedural portions associated with the object that is hypothesized. The procedural components further guide the processes at the intermediate level in order to obtain a more accurate object description. The

knowledge base represented at the high level is referred to as long-term memory and is composed of a semantic network of hierarchically organized schemas, which have a procedural component incorporated into their structure. The procedural components, which are comprised of the interpretation strategies, allow instances of what is represented in long-term memory to be constructed as a network that is specifically referred to as short-term memory.

The interpretation process in VISIONS consists of four primary components: representation of knowledge, the processes that produce the internal model, the control strategy that governs these processes, and the search strategy through long-term memory [Hans78b]. Interpretation of an image in VISIONS centers on building an internal model that consists of a description of meaningful entities and their relationships in a scene.

Various types of information, which include declarative, relational and procedural, are represented in a hierarchical structure as a semantic network of schema nodes in VISIONS. A schema describes an object and the relations between parts of an object. In addition to this descriptive knowledge, schemas also encode a procedural portion that details the interpretation strategies, or object-recognition schemes comprised of hypothesis and verification processes. Long-term memory reflects the a priori knowledge of the world as general classes of objects, and models of stereotypical scenes and one's expectations. Long-term memory consists of specialization and composition hierarchies built by using IS-A and PART-OF relations, respectively.

There are three ways a concept can be represented in the knowledge base: as an object that has attributes associated with it, which are independent of its parts or the context of which it may be a part of; as a part of a schema, where information about its relationships to the larger context is expressed; and as a schema itself, where its parts and relationships between parts are made explicit [Hans78b]. The dependencies between parts of an object determine whether the

parts are represented as additional schemas or within the schema for the object [Hans88, 81]. If there are complicated relationships between parts of an object, then the object recognition strategy generates the hypotheses for the parts, otherwise, an object part is represented as a separate schema, if recognizing the part is not highly dependent on the relationships between the parts of an object.

The object-dependent interpretation strategies attempt to match the expected entities in a scene to the data in short-term memory and also, to generate an instance of a schema. Short-term memory, therefore, represents the instances of certain nodes in long-term memory and is also referred to as the blackboard. Short-term memory can also be viewed as a set of hypotheses that constitutes a partial model.

The interpretation processes are implemented using a blackboard and can be run in parallel on a multiprocessor with each schema independently accessing the blackboard [Hans88]. Knowledge sources consist of interpretation processes that exercise simple control and cannot directly access the blackboard, instead, the interpretation strategies of the schemas invoke the knowledge sources. Specifically, the interpretation strategies can verify or reject hypotheses, evaluate the validity of hypotheses, extend hypotheses by combining similar data and hypotheses, instantiate schemas, and activate schemas. It is the intention of the developers of VISIONS to devise knowledge sources as the bases for interpretation strategies, and consequently, schemas would simply invoke the appropriate knowledge source.

The interpretation strategies in VISIONS may proceed in a top-down or bottom-up manner. The various categories of interpretation methods include a hypothesis generation and extension strategy, a strategy utilizing geometric data and another strategy that entails the detection and correction of errors in interpretation.

Direct and indirect communication between schemas is possible within VISIONS, and depends on how closely objects are related. If objects are highly related, then direct communication is preferable. For instance, the interpretation strategy for a house halts processing while it waits for its requested roof and wall (its parts) hypotheses, which were directly invoked. If objects are not highly related, then their interpretation strategies proceed with their processing and indirectly communicate with other schemas through the blackboard. While a schema is active, simultaneous polling of the blackboard for hypotheses can occur. As the schema system is being revised by the developers of VISIONS, parallel distributed control of schemas is being examined.

Another major issue in the development of a computer vision system that has been explored by the designers of VISIONS is inferencing in the context of uncertain information. Due to the massive amounts of information that need to be processed in constructing an interpretation of an image, the ambiguities that exist in image data, and the fallible low- and intermediate-level processes, it is inevitable that errors will propagate insidiously to the high level [Hans88, Cohe82, Barr81]. Therefore, the issues that need to be dealt with in order to allow inferences to be made include "the representation of uncertainty within the system, the mechanisms by which uncertain information is combined, and the manner in which uncertain evidence may be extended through inference" [Hans88, 83]. To this end, the ability to merge uncertain evidence from various sources and the ability to propagate confidences through the schema structure are being explored more thoroughly in VISIONS [Hans88],

In an early version of VISIONS, estimates of conditional probabilities between a schema and each part were stored in the schemas, and used in the process of hypothesis formation [Hans78b]. The estimates were based on expectations regarding the presence of objects in certain contexts. For example, given the existence of a stop sign, it is very likely that the context of a

road scene exists. However, given a road scene, it is not as likely that a stop sign necessarily will be seen.

A dependency graph, or evidential-based model, has been examined as a basis for reasoning about images and representing knowledge in VISIONS [Wesl82]. The model, referred to as DGMES (dependency graph models of evidential support), and the method of reasoning are concerned with combining evidence for the purpose of drawing consistent inferences. An objective of this approach is to manage uncertain and inaccurate information that results in ambiguous interpretations.

The approach to reasoning using the DGMES model entails gathering positive or negative evidence about the hypotheses represented in the dependency graph, combining the evidence, and propagating the consequence of combined evidence to dependent hypotheses [Wesl82, 15]. There are distinct processes for combining evidence and drawing inferences. The model is capable of making goal-directed and data-directed inferences over a body of knowledge. Once evidence has been obtained regarding certain domain knowledge, the inference engine will arrive at a consistent interpretation.

The fundamental concepts underlying the high-level processing in VISIONS, as well as the following systems, have contributed to the conceptual foundation of and, therefore, are integral to the development of the high-level inference system presented in this thesis.

2.3.2 A Query-Based System

Dana Ballard, Christopher Brown and Jerome Feldman at the University of Rochester developed a computer vision system that specifically responds to a query presented to the system [Ball78]. The application domains includes locating ribs in a chest X-ray and finding ships in a dock scene. The system employs a distributed control strategy, and a semantic network that

encodes two-dimensional domain knowledge. The extent to which the system processes information and conducts its search is determined by the user's query. A sketchmap, which consists of instantiations of the semantic network model, is constructed in the process of answering a query. The sketchmap serves as the middle layer of information between the image data structure and the model of domain knowledge.

The image data structure contains information about the original image. Mapping procedures effectively create the mapping between the image data structures and the sketchmap by generating the appropriate associations or links. Executive procedures allow for the links formed between the model and the sketchmap, analogous, respectively, to long-term and short-term memory in VISIONS. In addition to declarative knowledge, the nodes in the model also contain procedural knowledge; the mapping and executive procedures are attached to the nodes in the semantic network.

Constraint relations between nodes are encoded in the form of links in the semantic network model. The probability that a relationship is true, along with the value of the relationship, can be encoded in links [Ball78, 273]. The developers of this query-based system differentiate between the type of knowledge represented in the semantic network that ultimately can be instantiated, versus the type of knowledge that assists in constructing a sketchmap. The former type of knowledge is represented in the network as template nodes. These nodes and the geometric relations between them form a constraint network. The nodes in the constraint network can have location descriptors which encode knowledge of where to find an object in an image.

The queries for locating a particular object are embodied in the executive procedures, which in turn choose the most appropriate mapping procedure that is associated with a node in the model. A mapping procedure has a precondition, a cost estimate and an accuracy estimate associated with it. The least expensive mapping procedure that has met its preconditions is

selected and a node is then searched for. Basically, it is the executive procedure that incorporates the control strategy for attaining the goal of constructing a sketchmap, or answering a query.

2.3.3 Schema-Based Systems

A prominent feature of a vision system involves the approach toward designing a control strategy to guide the interpretation process. A problem that some approaches do not deal with adequately is derived from having to hypothesize a particular schema before all available knowledge can be used to direct the search for that schema [Have83a, Rao88]. The underlying mechanisms of the control strategy in Jay Glicksman's system, MISSEE (Multiple Information Source SEE), is the cycle of perception [Glic83]. Basically, the three step cycle of perception, which is illustrated in Figure 2-5, is a process where schemas direct exploration in an image in which objects are sampled, and these objects in an image, in turn, modify the original schema [Neis76, 21].

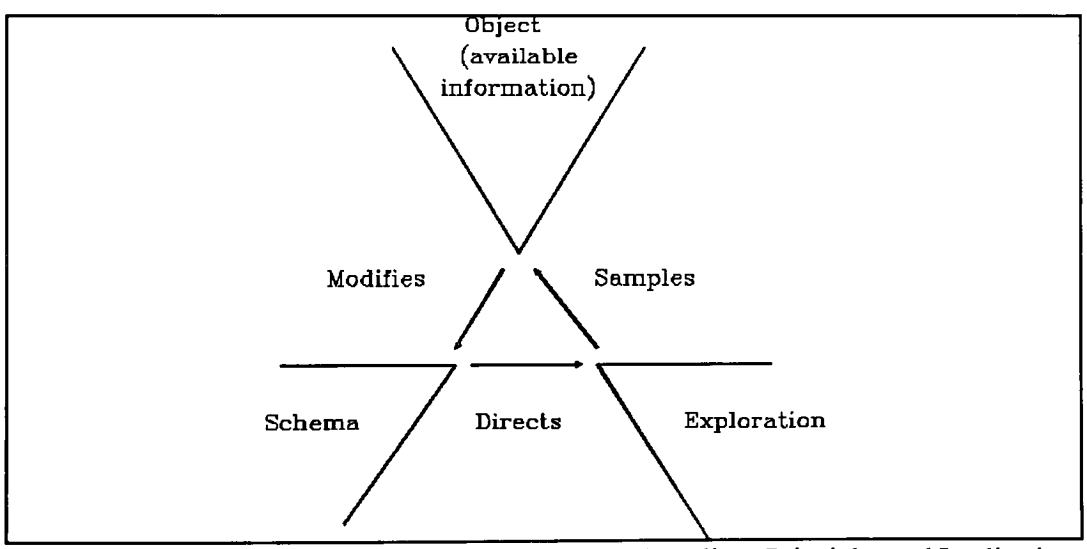

Figure 2-5 The Perceptual Cycle (From Cognition and Reality: Principles and Implications of Cognitive Psychology, by Neisser, Ulric, 1976, San Francisco, CA: W. H. Freeman and Co. (c) 1976 by W. H. Freeman and Co. Reprinted by permission.)

MISSEE is a schema-based system that exploits various sources of information and whose goal is to interpret aerial photographs of urban scenes. The purpose of integrating multiple knowledge sources as input is to resolve ambiguities that surface during the interpretation process. The sources of input in MISSEE include the digitized image, a sketchmap and information provided by the user. MISSEE also contains a semantic network of schema nodes. The schemas have procedural components, which are responsible for instantiating schemas, top-down invocation of schemas, and bottom-up processing of data. Schemas communicate through message passing, control the interpretation process and construct the final representation.

The schemas are generated and used by MAIDS, a schema manipulation system [Glic83, Glic82]. The four types of attributes associated with schemas are LINK, VALUE, PROCEDURE and CONFIDENCE. Relationships between schemas are denoted by LINKs, which can form a specialization hierarchy, a decomposition hierarchy, and instances of general classes. A VALUE attribute allows a place for a value to be stored, and if necessary, it can accommodate default values. When the value is added, removed, needed or changed, attached functions can be activated. The PROCEDURE attribute allows an attached procedure to be invoked during message passing. The CONFIDENCE attribute assists in the evaluation of interpretations. This value can be modified by an attached procedure when additional evidence is found for the object.

A schema-based system, according to Glicksman, could be a remedy to the difficulties a vision system faces from ambiguity, which occurs when data can have multiple interpretations; incomplete data, or insufficient support for an interpretation; and inconsistency, which is a consequence of existing favorable and unfavorable evidence for an interpretation [Glic82, 33].

Some of the efforts in developing computer vision systems have been directed toward and concerned with how visual knowledge is effectively organized and represented [Mack81,

Have83b]. A major impetus behind these efforts has been the inadequacies of former systems and the difficulties that arise in interpreting an image due to incomplete and inconsistent data.

The objective of Mapsee2, a revision of Mapsee1 developed by Alan Mackworth and William Havens, is to interpret sketch maps [Mack81]. It is a schema-based system that employs data-driven (bottom-up) and model-driven (top-down) interpretation strategies, and emphasizes decomposition and specialization relationships in its network of schema nodes. A schema model represents a general class of objects with attributes of class members and relationships between the class and other schemas encoded within it. An instance of a class of objects is created when an object is hypothesized. An object is recognized by recursively recognizing its components, thereby satisfying its constraints [Have83b, 93]. Recognition, therefore, is basically a search of the composition hierarchy which is controlled by the procedures attached to the schemas. After a schema has been instantiated, it can be used as a cue for hypothesizing a schema above it (bottom-up search) or as a cue for invoking a schema below it (top-down) in the composition hierarchy.

2.3.4 A Constraint Propagation Approach

In addition to model-driven and data-driven methods that are employed during the process of object recognition to obtain a suitable symbolic description of the input, a constraint-based approach has also been pursued. Havens proposes a constraint propagation approach that employs network consistency techniques and utilizes hierarchically structured schemas as a knowledge representation formalism [Have85].

A motivating factor behind Havens's efforts was organizing knowledge so that it can be effectively represented and efficiently applied to recognition [Have85, 127]. Havens emphasizes an object-centered knowledge representation in his theory of schema labeling. The goal of

schema labeling is to generate a network consistency graph which consists of schema instantiations and relevant constraints that represent the objects identified in the input. The knowledge base consists of schemas that represent general classes of objects and explicitly include relationships with other classes. Composition and specialization relationships between schemas are used to form constraints between associated instantiations in the network consistency graph. The valid combinations of parts for a class of objects are captured in composition rules which are associated with a schema. An object can be recognized by recognizing its components so that constraints for the class are satisfied.

The composition constraints can form a composition hierarchy while a specialization hierarchy is represented by specifying subclasses or descendants of schemas. As a result of forming the specialization hierarchy, attributes and constraints are inherited by subclasses. A labelset is associated with a class that represents its possible labels or types. The constraints are propagated ultimately through the network consistency graph by the implementation of network consistency algorithms in order to resolve ambiguities [Have85].

Schemas have a fairly uniform structure, each consisting of a set of composition rules, a set of components, a set of constraints and a labelset for the schema. The terminal nodes of the specialization hierarchy are referred to as atomic classes which have no composition and are not represented as schemas in the knowledge base. The knowledge base, also, does not contain any information about instances of object classes.

The network consistency graph is a description that is constructed from the knowledge base. Inconsistent labels that do not comply with the constraints are eliminated. As the composition rules are invoked, new schema instances are created and new constraints among the instances are established. The process of composing a network consistency graph is driven by the rules as it searches for components, applies new constraints and ultimately refines the network

description. Integrated within the composition rules are algorithms for maintaining consistency and propagating constraints in the network.

2.3.5 SIGMA

In computer vision systems that integrate both bottom-up and top-down search strategies sequentially, a decision must be made about which method to use and when to use that method. In regard to this predicament, Larry Davis and S. S. Vincent Hwang developed SIGMA with the objective of constructing a flexible reasoning process that does not rely on a vast amount of domain control knowledge [Hwan86, Davi85]. SIGMA is an image understanding system that consists of three components: a low-level vision system, a high-level vision system and a query-answering module. An image is provided to the low-level system where it undergoes segmentation, and the results of this process are stored in a database. Models of objects are provided to SIGMA by the user and the high-level vision system uses these models to interpret image structures or to guide the low-level system to locate image structures. The high-level system gradually builds an interpretation of the image by using the object models to interpret the information stored in the database. A goal is provided to the query-answering module and the process of building an interpretation continues until the goal is attained or further interpretations are not possible.

Among the many possible interpretations that may be generated, SIGMA uses the query-answering module to assist in the selection of a suitable or adequate interpretation. The high-level system activates this module to match the initial goal with the interpretation made so far. If the goal is satisfied, the high-level system continues with the interpretation process. The developers of SIGMA stress the integration of related hypotheses which underlies the processing of the high-level vision system [Hwan86]. Specifically, this process, which proceeds repeatedly,

includes the creation of hypotheses of objects, the grouping of related hypotheses, the generation of a composite hypothesis followed by the verification of hypotheses [Hwan86, 325].

The knowledge representation scheme used to build object models in SIGMA consists of frames and relations between objects encoded in rules and links, producing a graph of nodes and arcs. The slots in a frame store values of associated attributes of an object and can, also, house procedural knowledge of how to compute the value of a slot. The frames are organized into a specialization hierarchy where ancestral properties are inherited by descendants. Instances of frames are kept in the database and can be categorized as verified or hypothetical. Associated with each instance is a strength that is computed when the instantiation occurred. The value of the strength of an instance is recomputed if the instance is updated.

Relational knowledge is carefully encoded in SIGMA to be used by the high-level system to verify that the relation holds between two objects or to locate an object that is constrained by a relationship to another object. Control knowledge, which can describe a relation between two objects, is embedded within rules. In other words, a relation between a pair of objects can be expressed as a rule in the frame representing one of the objects. A rule has three components: a control condition which specifies when a rule can be used, a hypothesis that describes the object to be created when the condition is true, and an action that is invoked if the hypothesis is verified. The high-level system exploits different parts of a rule to guide its analysis.

Once image structures are extracted from the image, bottom-up analysis can take place using relational knowledge to verify constraints between objects. Top-down analysis is used to look for image structures originally missed by the segmentation process by using relations to direct the analysis as to where to look for the missing object, and to generate hypotheses.

Instantiations, or database entities, are composed of an iconic description and a symbolic description. The iconic description refers to a region in the image where an instance of the

object's frame may be found. The symbolic description specifies the values of the slot of the instantiated frame and associated constraints. If two database entities satisfy specific conditions, then they are considered consistent and consequently, these related entities can be integrated. A situation, in turn, consists of consistent entities, and a composite hypothesis takes into consideration all of the constraints on the slots of the database entities in a particular situation. It is the focus of attention mechanism within the high-level system that selects a situation that has the greatest strength and sends it to the composite hypothesis constructor. The solution generator, then, either locates an instance in the database that satisfies the composite hypothesis, or sends the descriptions of the composite hypothesis to the low-level system to perform resegmentation.

The interpretation process cycles through a number of iterations until an adequate interpretation is produced. At the end of each iteration, the query-answering module is invoked which enables descriptions of the interpretation to be displayed if a goal has been achieved.

2.3.6 Evidential Reasoning in an Object Recognition System

A major issue in high-level vision that is unavoidable and demands considerable attention is that of uncertainty handling. It is inevitable that the development of an inferencing module for a computer vision system will be influenced by how uncertainty leading to inconsistent or unreliable interpretations is handled. According to Rao and Jain: "Evidential information refers to information that is uncertain, imprecise, and inaccurate. Knowledge-based systems operating in complex environments should be able to handle evidential information" [Rao88, 77]. A computer vision system is faced with the problem of processing much information in a complex environment, some of which is incomplete, inaccurate and/or uncertain [Cohe82, Rao88]. While some systems handle uncertainty in an ad hoc manner, others emphasize explicit uncertainty handling mechanisms. The object recognition system developed by J. Kim, D. Payton and K.

Olin illustrates how evidential reasoning is applied in drawing inferences in a fairly formal manner [Kim84].

The knowledge representation scheme utilized in the object recognition system is object-oriented. The objects are represented in frames and a specialization hierarchy is formed which exhibits property inheritance. The domain knowledge is kept separate from the inference engine. Symbolic representations of regions, line, edges and vertices are extracted from the image and used in the process of hypothesis generation, confidence evaluation and confirmation. Initial hypotheses are a result of bottom-up processing of the image and subsequent hypotheses are a result of top-down processing, which can acquire additional evidence from the image. As hypotheses, which are represented as instances of frames, are generated, the inferencing process will look for evidence to support or refute these hypotheses. Additional hypotheses may be created and new evidence may refine initial hypotheses.

Specifically, the knowledge base includes domain-specific rules and heuristics for object recognition while the inference engine interprets the rules and heuristics. The classification reasoning process of the system is modeled after human intelligent classification reasoning [Kim84]. Kim et al. emphasize that this aspect of their system is based on the observation that humans describe objects in terms of how much information is available [Kim84]. Thus, an object hierarchy is well-suited to this application because objects are described at different levels of detail.

A confidence factor, represented as a likelihood of a hypothesis to be true, is associated with a hypothesis. After a hypothesis is created, it undergoes a process to confirm or refute its class membership. This entails gathering of relevant evidence and updating the confidence factor. The frames representing objects encode rules that describe conditions about attributes of the hypothesis and a confidence factor that is accumulated if the conditions are fulfilled. Evidence

in support of a hypothesis is kept separate from refuting evidence. Positive evidence for a class implies support for the ancestors of the class while negative evidence implies a lack of support for descendants of the class. The confidence of an instance of a superclass is greater than the confidence of a subclass instance. The procedural knowledge within a frame consists of procedures to look for evidence and rules to assess the level of confidence.

Kim proposes a distributed parallel model in an elaboration of a distributed inference scheme for classification where an object class hierarchy is employed along with a certainty factor formalism [Kim85]. In this model, processors corresponding to objects in the hierarchy pass messages regarding the effect of accumulated evidence to neighboring objects. A processor oversees the gathering and combining of evidence for a particular hypothesis.

Evidence for a hypothesis can be obtained from local rules, superclasses and subclasses, and integrated to obtain a global confidence value. The refuting evidence gathered from higher classes is propagated to the hypothesized object's descendants, while supportive evidence combined from the object's descendants is propagated to its superclass by the processor. The global confidence is represented by a number in the range of -1 and +1. Once the confidence of each hypothesis is known, the object can be classified by beginning at the root node of the object hierarchy and selecting the nodes of associated subclasses with the greatest confidence value.

2.3.7 A TMS Approach to Computer Vision

A truth maintenance system (TMS) represents another approach to maintaining consistency and minimizing the search process in computer vision. The object recognition system, VICTORS, developed by Gregory Provan employs such an approach [Prov88].

According to Johan de Kleer, truth maintenance systems distinguish between a problem-solving module that embodies the domain-specific rules and makes inferences, and a truth maintenance module that is responsible for recording inferences or the state of the search process [deKl86]. The functions of a TMS include maintaining a record of all inferences made, enabling nonmonotonic inferences to be made and removing contradictions [deKl86, 129]. A justification-based TMS (JTMS) records all justifications, or dependencies, and finds solutions one at a time. An assumption-based TMS (ATMS) enables solutions to be found simultaneously. The benefits derived from using a TMS include maintaining consistency and conducting more efficient searches.

VICTORS has been implemented using a scene of overlapping rectangles and has identified all incidents of a known figure within the scene [PROV88, 230]. Multiple interpretations of the figure are possible in the scene. The internal representation of dependencies is a dependency graph where antecedents and consequents of dependency relations are specified. Although VICTORS has been implemented with a justification-based and an assumption-based TMS, according to Provan, an objective in developing the system was to detect multiple interpretations at the same time, therefore, an implicit preference toward an ATMS is expressed [Prov88].

The structure of VICTORS is divided into a general constraint engine and a general reasoning engine. The constraint engine generates constraints and the reasoning engine is responsible for preserving consistency. The task of the TMS is to remove multiple or contradictory labels.

The power of an ATMS lies in its ability to locate solutions simultaneously using breadth-first search, and by refraining from exploration of portions of the search space known to be inconsistent [Prov88, 233]. In examining the reasoning aspects of the interpretation of a visual

scene, Provan illustrates that a TMS can be effectively employed in a vision system, but asserts that the performance of the ATMS slows with a greater amount of interpretations [Prov88, 235].

Although the software tools used to implement the high-level inference system in this thesis do not include a TMS utility, the basic concepts of maintaining consistency and minimizing multiple interpretations in a vision system are preserved in the implementation.

CHAPTER 3

CONCEPTUAL ISSUES IN HIGH-LEVEL VISION

Although there exists a seemingly wide range of approaches to dealing effectively with the high-level issues in a computer vision system, the unifying theme of the previously discussed systems can be reduced to modeling a human reasoning process. This, in turn, naturally leads to the question of what to reason with (in a visual domain), hence, knowledge representation becomes another major issue. Reasoning in a visual environment further constitutes mechanisms for inferencing and controlling the search strategy as required by the recognition process, and handling uncertainty which arises due to the nature of the visual environment. The surveyed systems basically are attempting to solve similar problems in ways that are restricted by the specific application domain and the objectives of the particular system--many vision systems are designed for a special purpose. The following discussion provides an analysis of the underlying conceptual basis of various approaches surveyed in this thesis and establishes the groundwork for the implementation developed in this thesis.

3.1 Visual Knowledge Representation and Organization

A computer vision system is continually confronted with processing an immense amount of information. The process of visual interpretation, specifically, is concerned with providing a symbolic description of a scene and therefore, relies on models of expectations, prototypical situations and objects as a basis to draw upon experiential knowledge. When one reasons about

a scene, for example, they are applying their expectations of the situation, along with physical evidence to their store of knowledge to reach a final conclusion.

The issues that knowledge-based vision systems are confronted with, according to Rao and Jain, include defining what knowledge is relevant to the problem, deciding how the knowledge is to be represented, and determining how and when the knowledge is to be used [Rao88, 66]. Hanson and Riseman point out a key issue in the development of a computer vision system:

> A central problem in image understanding is the representation and appropriate use of all available sources of knowledge during the interpretation process. Each of the many different kinds of knowledge that may be relevant at various points during interpretation imposes different kinds of constraints on the underlying representation. In general, the representations must be sensitive enough to capture broadly applicable "sketches" of objects and expected scenarios [Hans88, 67].

The types of knowledge that need to be represented in a vision system include declarative, or knowledge about the attributes of objects and their parts; relational, or knowledge about the relationships between parts of an object which implies the decomposition of objects into their parts; procedural, or knowledge of how to compute a value, for instance; knowledge about a specialized instance of a generic class of objects in addition to the class; and metaknowledge which is knowledge about what is known--extent and reliability, for example [Hans88, 67].

Some of the general criteria in choosing an appropriate knowledge representation formalism to build a knowledge base are that, ideally, it should facilitate inference, allow reasoning with partial information, and support various kinds of knowledge. It should also be able to support incremental additions and modifications; in other words, it should be modular and flexible. A knowledge base built using frames (or schemas) as the basic unit of representation meets these general criteria.

According to the approaches taken in the vision systems surveyed in this thesis, the popular method of choice in terms of knowledge representation formalisms is frames. Frames have been used successfully in VISIONS, SIGMA, Mapsee2, Havens's constraint propagation approach and the object recognition system which emphasizes evidential reasoning. The query-based system utilizes a semantic network and MISSEE employs a semantic network of schemas.

The natural, and also most common, way of organizing frames is hierarchically to allow for inheritance of properties and to minimize redundancies in representations. A frame is basically a data structure for representing concepts or situations and associated features of these entities [Mins75]. A useful characteristic of frames is the capability of representing exceptions and allowing for default values of slots which store the attributes of a frame. In addition to declarative knowledge, frames also can store procedural knowledge.

The concept of a schema in vision has also been supported from a cognitive psychology perspective. Neisser suggests that the cognitive structures essential for vision are anticipatory schemas [Neis76]. Neisser describes a schema as the part of the perceptual cycle that is internal to the perceiver, capable of being altered by experience, and tailored to what is being observed. Schemas enable the perceiver to accept specific information, control the activity of looking, and typically develop from general to specific [Neis76, 20].

Elaborating on the functionality of a schema, Neisser suggests that it resembles a format which specifies a particular type of information, and can act as a plan for acquiring more knowledge to fill in the format [Neis76, 55]. Perception is selective; if a format does not exist for certain information, then the information is not used. Through the cycle of perception, anticipations of specific information are constructed that allow one to accept that particular information. Anticipations guide the process of perception, but perception can also provide new information that modifies the previous perceptual schema.

Havens and Mackworth evaluate and, as a result, advocate schemas as a suitable knowledge representation in terms of their "descriptive adequacy, the ability of a representational formalism to capture the essential properties of objects and the relationships among objects in the visual world, and procedural adequacy, the capability of the representation to support efficient processes or recognition and search" [Have83b, 90].

Since frames can be utilized to adequately model one's expectations and support both declarative and procedural knowledge, they are an appropriate representation for a visual domain and are used in the implementation developed in this thesis to construct a knowledge base.

3.2 Reasoning Issues of Visual Interpretation

Reasoning within a visual domain encompasses processes for inferencing, or deriving conclusions from environmental evidence and one's expectations in a particular situation, for controlling the search through a vast amount of stored knowledge, and for controlling the application of knowledge. It also requires mechanisms for handling ambiguous data, inconsistent interpretations, and partial evidence, which are all characteristic of what the human visual system normally faces. In this section, the strategies that model the reasoning process in the surveyed systems are examined.

3.2.1 Control and Search Strategies

All of the previously described computer vision systems employ goal-directed (top-down) and/or data-directed (bottom-up) reasoning methods at various times during the recognition or interpretation process. Generally, initial hypotheses are generated in a bottom-up manner and subsequent hypotheses are generated as a result of top-down processing. Search in most of the

systems is to a large extent heuristic, applies, and is also guided by, domain-specific knowledge, and takes advantage of the structure of the composition and specialization hierarchies.

Control and search strategies are combined inextricably in solving problems that face knowledge-based vision systems. Essentially, optimizing the search process in a vision system is necessary because of the many competing and inconsistent interpretations that may be generated, and because the process operates over a large amount of knowledge. Some of the factors to consider in making the search process more efficient include eliminating inconsistencies as soon as possible, reducing the search space and exploiting constraints.

The developers of VISIONS, in an early version, emphasized a hierarchical modular control strategy that employs a hypothesize-and-test scheme at various levels of representation [Hans78b]. Also, the representation of the search space consisted of a tree whose contents reflect the history of the search process through the space of partial models [Hans78b, 304]. The root node of the tree reflects the initial state and descendant nodes consist of only variations that exist between its parent node and itself. One of the objectives in maintaining a tree of partial models was to enable a user to examine the situations in which the system makes errors to allow for subsequent error recovery. In an updated version of VISIONS, the object-recognition strategies associated with schemas provide local control information. VISIONS has progressed to where the developers' objective is to construct a network of concurrent processes that will cooperate to produce a final interpretation [Hans88, 68].

The query-based system of Ballard et al. exhibits a distributed control strategy which is globally directed by the user's query [Ball78]. Reasoning is predominantly goal-driven with the query representing a top-level goal. The process of searching the semantic network model is constrained by the level of detail of the user's query.

Both MISSEE and Mapsee2 exploit a cycle of perception as their fundamental control mechanism. In MISSEE, it is the procedural knowledge within the schemas that controls the interpretation process, and schemas communicate by passing messages [Glic83]. The recognition process within Mapsee2 is, also, driven by the procedures encoded in the schemas [Mack81]. Similarly, in the object recognition system developed by Kim et al., control is distributed over the objects represented in the knowledge base as messages are passed between objects. A process, which oversees the collection of evidence and the generation of new hypotheses, is associated with each hypothesis.

In Havens's example of schema labeling, the control strategy utilized is goal-driven with automatic backtracking [Have85]. The process of composing a network consistency graph is guided by the composition rules that reside within the schemas representing certain object classes. A global interpretation is constructed by ensuring local consistency. As constraints are added to the network consistency graph, their consistency is verified and the whole network may be refined [Have85, 129]. The ATMS in VICTORS utilizes a variation of breadth-first search as it pursues multiple solutions concurrently [Prov88]. The ATMS keeps backtracking to a minimum and explores only a section of the search process at a given time.

In SIGMA, control knowledge is embedded within rules which describe relations between two objects [Hwan86]. The high-level vision system, then, exploits different parts of a rule to guide its analysis. The global search strategy is basically sequential, as the high-level system repeatedly proceeds with its reasoning procedure of generating, integrating, abstracting and verifying hypotheses.

The shortcomings and limitations of purely, global top-down or bottom-up approaches to search render them inadequate, according to Havens, and therefore, procedural knowledge encoded within schemas offers a more favorable approach to controlling the search for instances

of a schema class [Have83a]. In a goal-driven scheme, and assuming that knowledge enabling the recognition of an object is located entirely within the schema representing that object, a schema is hypothesized as a subgoal by a higher schema before relevant knowledge about recognizing an instance of that schema class can be used to restrict the search process. This constitutes a major drawback of top-down search according to Havens, and furthermore, in a goal-driven manner, depth-first and breadth-first search methods of selecting alternative subgoals are failure-driven and therefore, inefficient [Have83a, 189].

Once an instance is hypothesized and verified in bottom-up search, higher schemas, which the instance is a component of, become supergoals of the instance and are activated to locate other parts. An inadequacy in this method results from the lack of any schema to direct recognition. In an effort to compensate for the inefficiency of top-down and bottom-up search, another approach focuses on enabling procedures that are associated with schemas to be responsible for directing the search process. With procedures associated with schemas acting as independent processes, it is the responsibility of the schema to decide which search to pursue depending on its assessment of the probability of success [Have83a, 197].

The global control strategy employed in the implementation presented in this thesis is essentially sequential, centralized and incorporates initial bottom-up analysis followed by top-down processing. The basis of control is hypothesize-and-test, although no feedback to lower levels is included.

3.3 Reasoning under Uncertainty

A computer vision system normally will incorporate some way of reasoning with incomplete data, inconsistent interpretations, ambiguous situations or errors propagated to the high level from lower levels. Since this generally reflects the nature of real world experiences,

uncertainty is inevitable in a vision system. According to Glicksman, "Handling inconsistent knowledge generally requires a flexible system that permits multiple interpretations and non-monotonic reasoning" [Glic82, 33]. The issues that warrant some consideration in order to inference in an uncertain context include the representation of uncertainty, combining uncertain evidence and propagating this uncertainty [Hans88, 83].

As the development of VISIONS has evolved, more formal methods of handling uncertainty have been pursued which include employing the Dempster-Shafer theory of evidence as a foundation for reasoning [Hans88, 83]. Wesley et al. reasoned with a hierarchical dependency graph to draw inferences from combined evidence [Wesl82]. Each proposition in the dependency graph has a confidence interval which consists of values representing support and plausibility.

The underlying basis of Glicksman's system, MISSEE, is to integrate multiple sources of knowledge to resolve ambiguities in interpretation [Glic83]. Neither MISSEE nor Mapsee2 employ explicit formal methods of handling uncertainty. Similarly, there is no formal mechanism for dealing with uncertainty in SIGMA. Instead, when an object is instantiated, a strength is computed for the instance by a procedure attached to the frame. This value can be recomputed as the values of an instance are updated during analysis. The high-level vision system then uses this value to control its focus of attention mechanism [Davi85, 22].

Consistency is ensured in Havens's illustration of schema labeling by employing network consistency procedures to propagate constraints and ultimately refine the network consistency graph [Have85, 129]. The TMS module in VICTORS directs the search for consistent interpretations by propagating constraints, ensuring global consistency and organizing the search [Prov88, 232]. In this way, ambiguous situations are appropriately handled.

The object recognition system of Kim et al. incorporates a relatively more formal and elaborate scheme of manipulating confidence factors associated with hypotheses [Kim84]. The confidence factor reflects the degree of likelihood of a hypothesis to be true. Procedures used for locating evidence and rules that contain information of how attribute values affect the hypothesis are associated with frames.

In this implementation, estimated values expressing the likelihood of compositional relationships are associated with frames. These values prove useful at a local level. In other words, they are used to resolve local ambiguities. Also, a strength value is associated with hypothesized instances to disambiguate between multiple instantiations. The value is based on the amount of constraints satisfied.

CHAPTER 4

IMPLEMENTATION

4.1 Conceptual Overview of High-Level Inference System

The following section provides a conceptual overview by describing the method used in this thesis to reason with visual knowledge at a high level in an object recognition system.

4.1.1 Knowledge Base Design

Some of the factors considered in building a knowledge base in the method used in this thesis include how and where (within a frame) to represent constraints, to what level of detail the specialization and composition hierarchies are built, what a frame represents and where (at what level in the hierarchy of frames) procedural knowledge is stored. The design of a knowledge base proves to be critical when manipulating and searching the knowledge, and therefore, requires careful consideration.

An uninstantiated frame in this thesis consists of expectations, prototypical information and object recognition procedures, whose slots are used as inherited information by the frame's corresponding instances. Frames, then, represent classes of objects and the knowledge base, then, is a hierarchically organized set of frames representing objects. An instantiated frame, or instance, represents a hypothesis. A partial hypothesis has not had all of its constraints satisfied and is awaiting or undergoing verification, while a complete hypothesis has had its constraints satisfied and consequently can assist in verifying "higher level" hypotheses. A composition

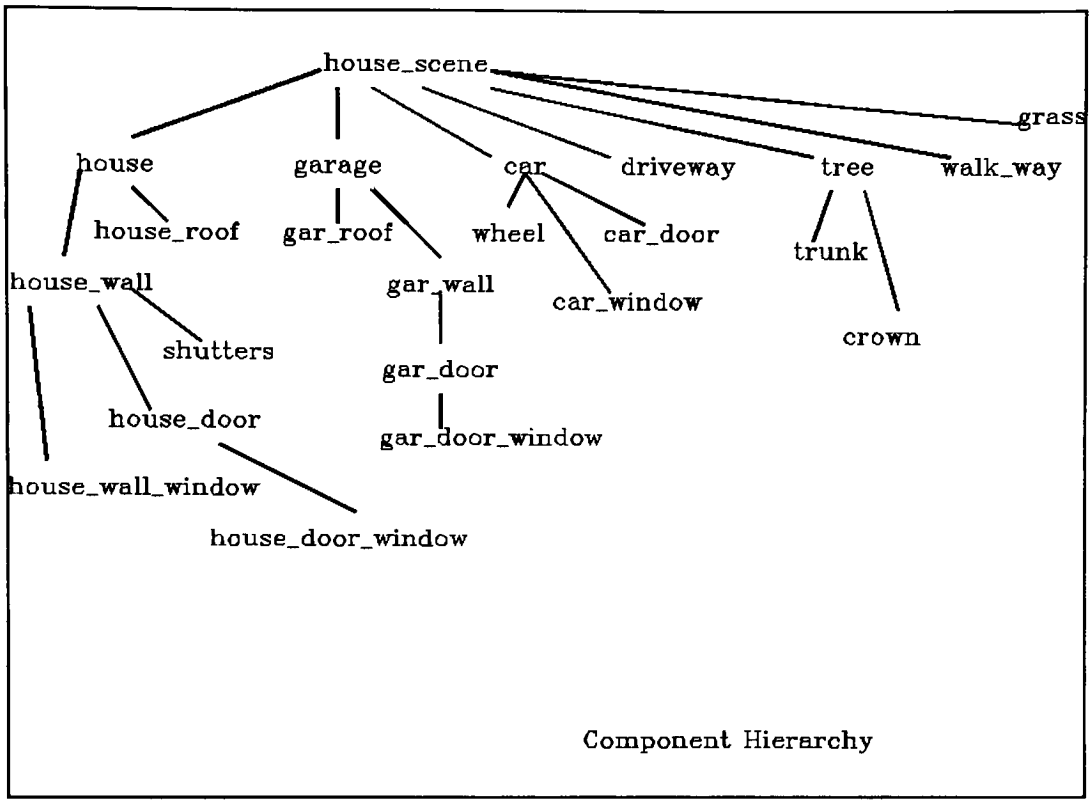

Figure 4-1 Component Hierarchy

hierarchy of objects is built using the components slots and part_of slots of frames while a specialization hierarchy is built using the is_a slots and members slots of frames (see Figure 4-1 and Figure 4-2). For example, a house_wall has components which include (from a two-dimensional perspective) windows, a house_door, and shutters, and it is part_of a house. A house_wall, also, is_a type of a wall and the class of wall objects includes the classes of garage_wall and house_wall as members.

In the method discussed in this thesis, constraints are expressed between objects at the same level in the knowledge base. For instance, given the frames: wall, garage_wall and house_wall, a next_to relationship in the garage_wall frame would have a value of house_wall rather than wall because the wall frame is found at a higher level in the specialization hierarchy. This maintains a degree of consistency when satisfying constraints of an object. The object

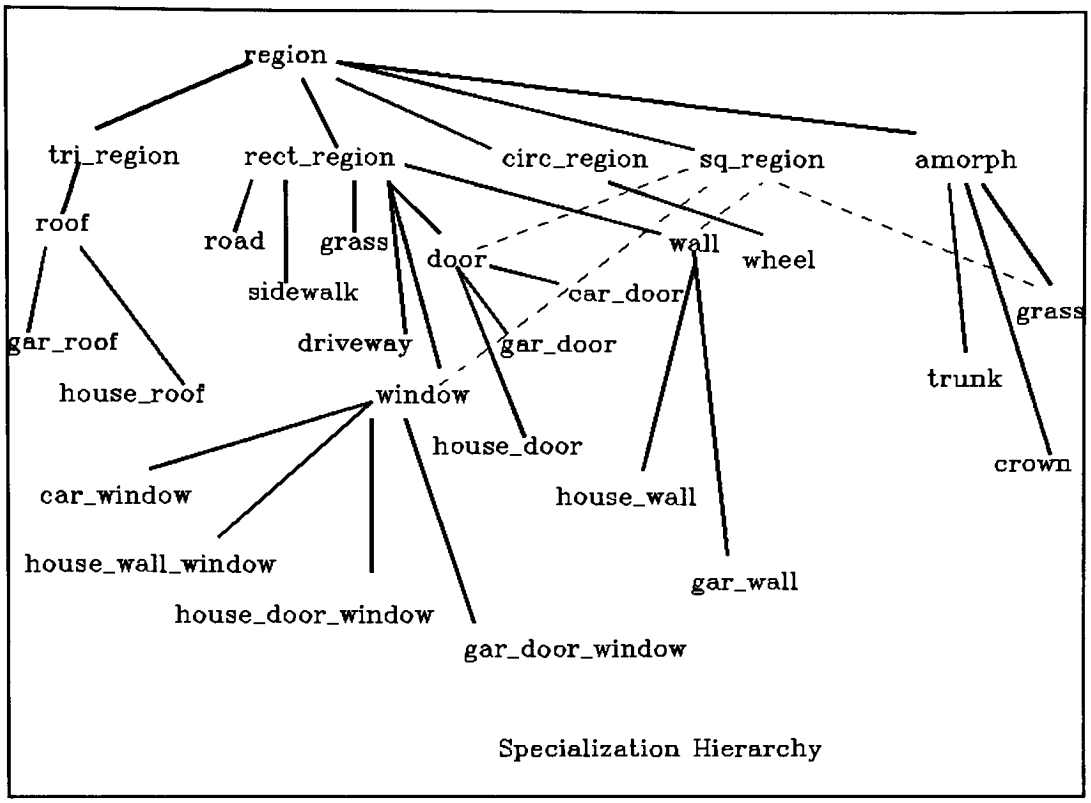

Figure 4-2 Specialization Hierarchy

recognition strategies, which are encoded as procedural knowledge, are stored in the terminal nodes of the specialization hierarchies to allow for direct inheritance by instances of those frames. The object recognition procedures are generic in format, but exploit the specific relationships and constraints of the frames to which they are attached.

4.1.2 The Process of Object Recognition

The process of object recognition is a second major issue that constitutes part of the method presented in this thesis. The input data, for instance, region 1 is square and next to region 2, is represented symbolically and initially is processed to represent more meaningful objects such as garage_wall and house_wall. Consequently, many hypotheses could be generated regarding what region 1 and region 2 represent. But, given the additional knowledge or clues,

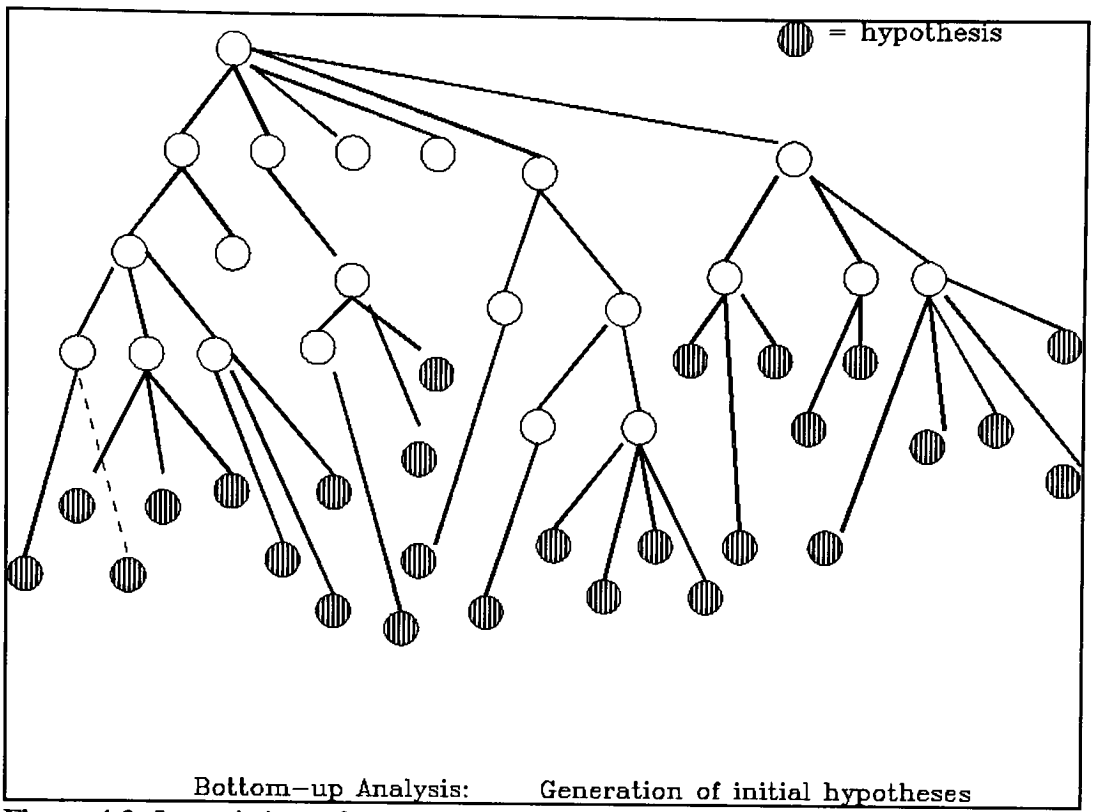

Figure 4-3 Instantiations of Frames

such as the proximal location of region 1 to region 2, their respective shapes, size and color, many hypotheses could be eliminated, thus only the most promising of which remain.

The process of object recognition begins with bottom-up analysis of the input data. This specifically entails the generation of initial hypotheses (refer to Figure 4-3). These initial hypotheses can be maintained in a priority queue or scheduler, in lieu of parallelism. Associated with an instance is a strength value that essentially reflects the amount of evidence that supports its existence. This value is used to discriminate weak hypotheses from strong hypotheses. Conceptually, if there is not enough information or cues available to recognize an object, it may be misinterpreted easily or simply unrecognizable. In any case, a measure of the available input is deemed necessary. A recognition state for each instance is also maintained that keeps track of which relationships and attributes of the instance are matched.

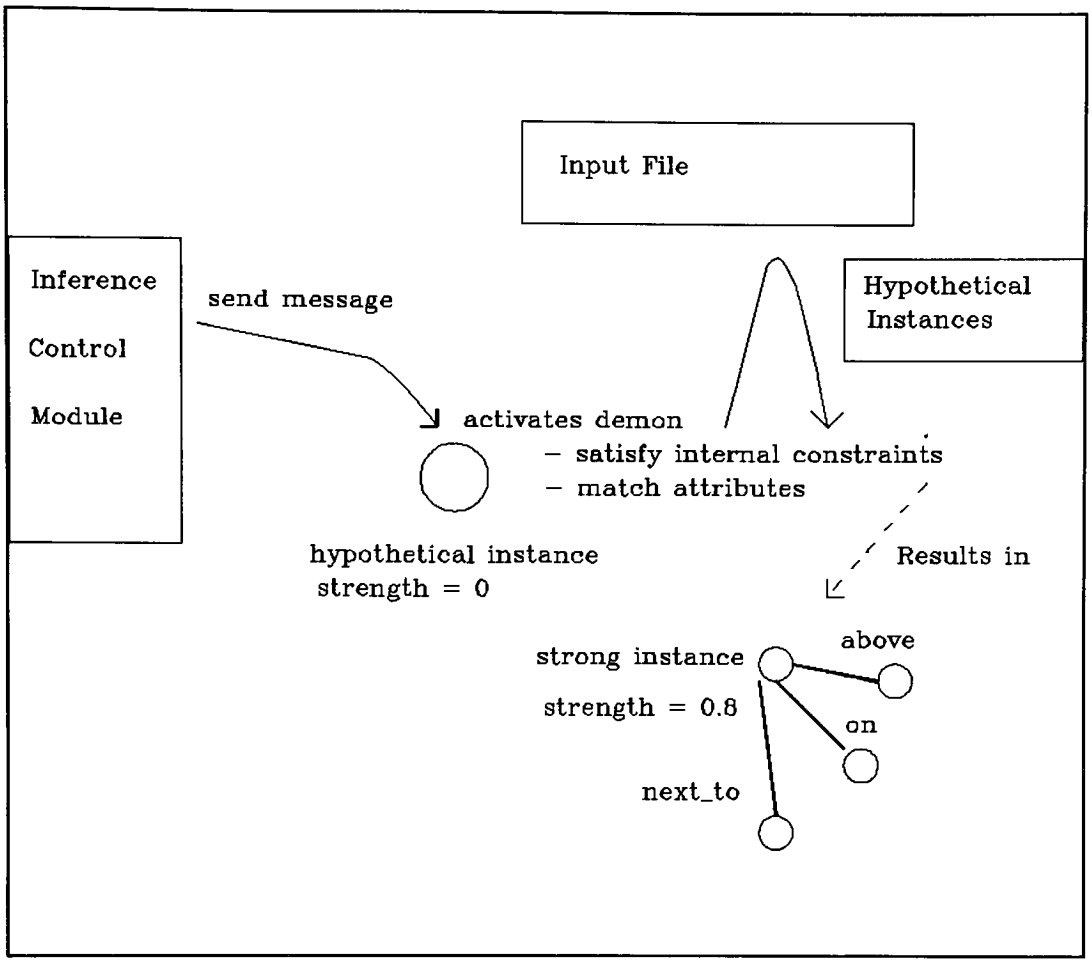

Figure 4-4 Object Recognition Process

As part of the recognition process, a message is sent to an object to activate its recognition mechanism. Knowledge about how to recognize an object is represented procedurally and is attached to the frame that represents the activated object (see Figure 4-4). As the recognition process proceeds, instances are, in effect, competing with one another for their existence.

The entire process of object recognition continues until all hypothetical instances are either eliminated or verified as illustrated in Figure 4-5. A basic premise of the object recognition process is to promote only promising hypotheses and eliminate weak hypotheses as soon as possible. Integral to this process is a built-in mechanism for handling multiple competing

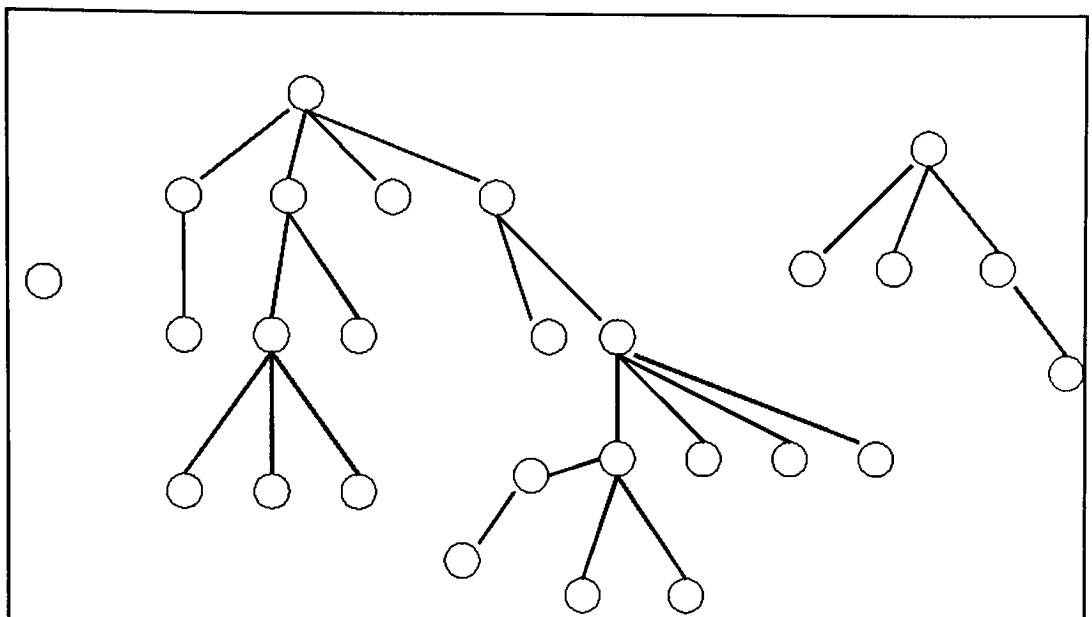

Figure 4-5 Hierarchy after Weak Hypotheses are Eliminated

hypotheses or inconsistency in generating and verifying hypotheses. In other words, at various stages of hypothesis testing and verification, those instantiated frames that are marked as most promising are tested for validity and checked for consistency with the objective of further refinement.

Top-down analysis, then is invoked to verify each hypothesis. Bottom-up search is preferable if only a few of the components of an object are recognized. Top-down analysis, on the other hand, is initiated to verify the existence of an instance after it already has been determined to be a probable hypothesis. A hypothesis is considered probable if the majority of its constraints have been matched as reflected by the value of its strength slot.

Hypothesis verification includes satisfying an object's constraints and proving the object's subgoals or compositional relationships. For example, from a top-down perspective, if it is hypothesized that a house exists in a scene, and this house is composed of house walls and a

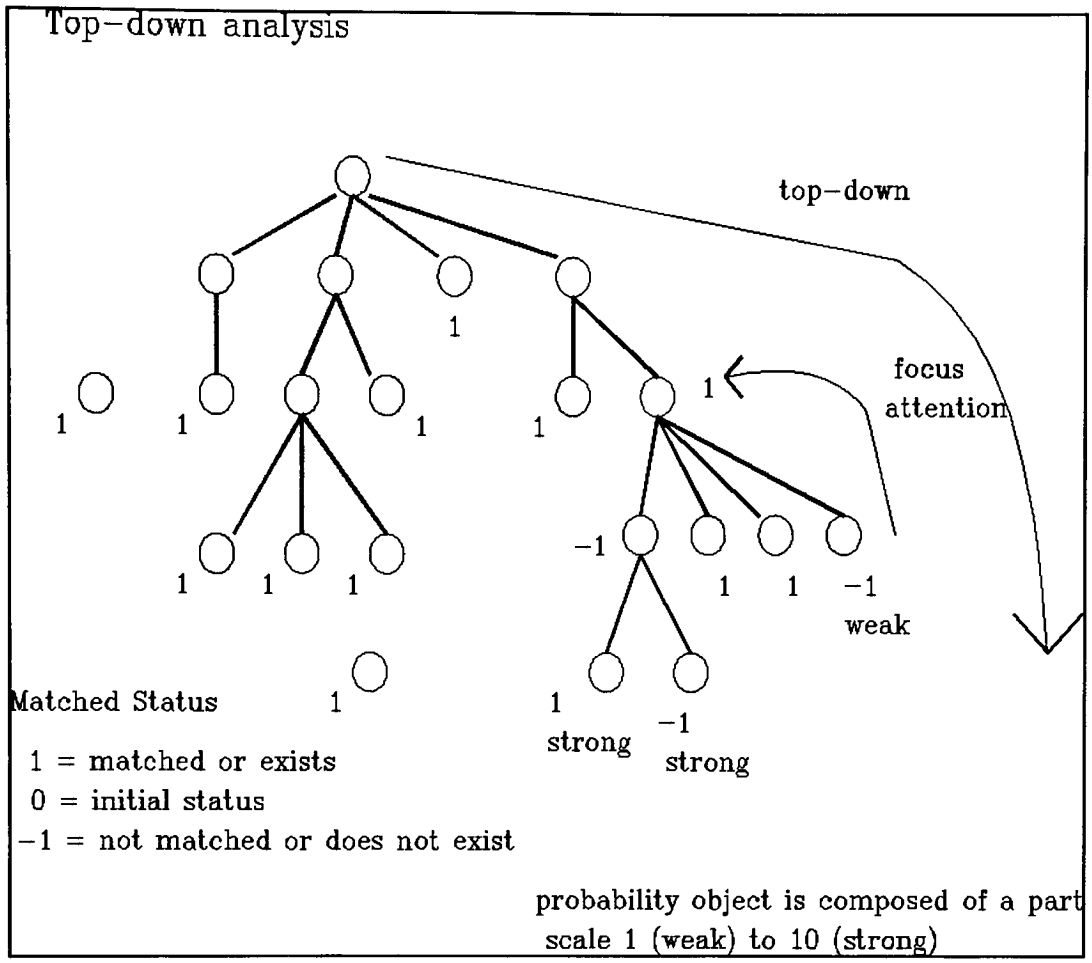

Figure 4-6 Top-down Analysis

house roof, then proving this would involve, first, satisfying the compositional constraints of the house roof object to verify its existence, which in turn may involve satisfying its compositional relationships, and similarly, applying this strategy to the house wall components. Once it is concluded that the parts of a house exist, then the compositional constraints in which a house is a part of can be satisfied, and when this processing is completed then this knowledge can be used in a bottom-up manner to infer a house scene exists. Also, following each attempt at satisfying the constraints of an instantiated frame, the object's recognition state and strength are updated to reflect the most current state of recognition, and during top-down analysis the status of components matched is updated.

The status of a hypothesis generally reflects the degree to which its compositional constraints currently have been satisfied. The initial value of all instances is 'unmatched,' and as the final composition hierarchies are generated, an instance can take on the value of 'matched' or 'non-existent.' During the process of proving an object is composed of its parts, the probability that an object is composed of a particular part is checked. If this probability is strong and it has been inferred already that the part does not exist visually, then it is very likely that the object does not exist also. On the other hand, if it is a weak probability and the part is not visually present, then this is factored into the status of the object. If the object's expected parts cannot be matched with hypothetical instances, then it is likely that the object cannot be inferred to exist as well. For example, if a house roof is missing from what is anticipated to be a house, the object in question may not be a house after all. But, if a pair of shutters is not part of a house wall, then this does not necessarily negate the existence of a house (see Figure 4-6).

It can be inferred from the previous discussion and example that there is no strict order to exploiting top-down and bottom-up analyses in the process of object recognition primarily because the search process is heuristically guided. Generally, a bottom-up form of reasoning occurs first, followed by top-down analysis. Then, an integration of both can be used depending on whatever is most appropriate at that time in the overall process. The global control mechanism governing these analyses resides external to the knowledge base and this mechanism oversees the process of object recognition and drives it in a forward direction. Ideally this entire process of object recognition could be facilitated by message passing between objects. This would involve encoding the knowledge regarding the overall state of recognition within each frame so that an object can decide intelligently which object to send a message to next. An alternative would be indirect communication which would involve an object "suggesting" which object to send a message to next by sending this information back to a central controller that

ultimately would be responsible for the final decision of where to direct the recognition process. Direct, one-way communication between the control module and each instance is the method pursued in the system developed in this thesis.

An effort to make an informed decision about which direction to pursue the recognition process reinforces the concept of focusing attention. For instance, in recognizing a house wall as part of a house, it makes sense to complete the recognition of the object the house wall is a part of, if any, rather than attempting to recognize a stop sign on a nearby road. Specifically, this can be achieved by reordering the list of partial hypotheses according to the compositional relationships of the active object. In other words, if instances exist for the objects an active object is composed of or part of, these instances or partial hypotheses are given priority over other instances in the recognition process. This would facilitate recognition of a particular object since preference is given to the processing of the object's parts. When the process of confirming the recognition of an object occurs, it in effect triggers neighboring (in the sense of a composition hierarchy) objects or raises their level of alertness.

4.2 Functional Specification

The objective of the implementation in this thesis is to model the human reasoning process that is utilized during visual recognition. Specifically, the intention is to develop a system for symbolically representing and reasoning with visual knowledge. The emphasis during the development of this high-level inference system, whose goal is object recognition, is on the organization and structuring of domain knowledge and the reasoning process that manipulates this knowledge (see Figure 4-7).

Low-level processing of an image is not an objective of this thesis, and therefore, is not treated. Instead, low-level processing is assumed to have been completed, and the input into this

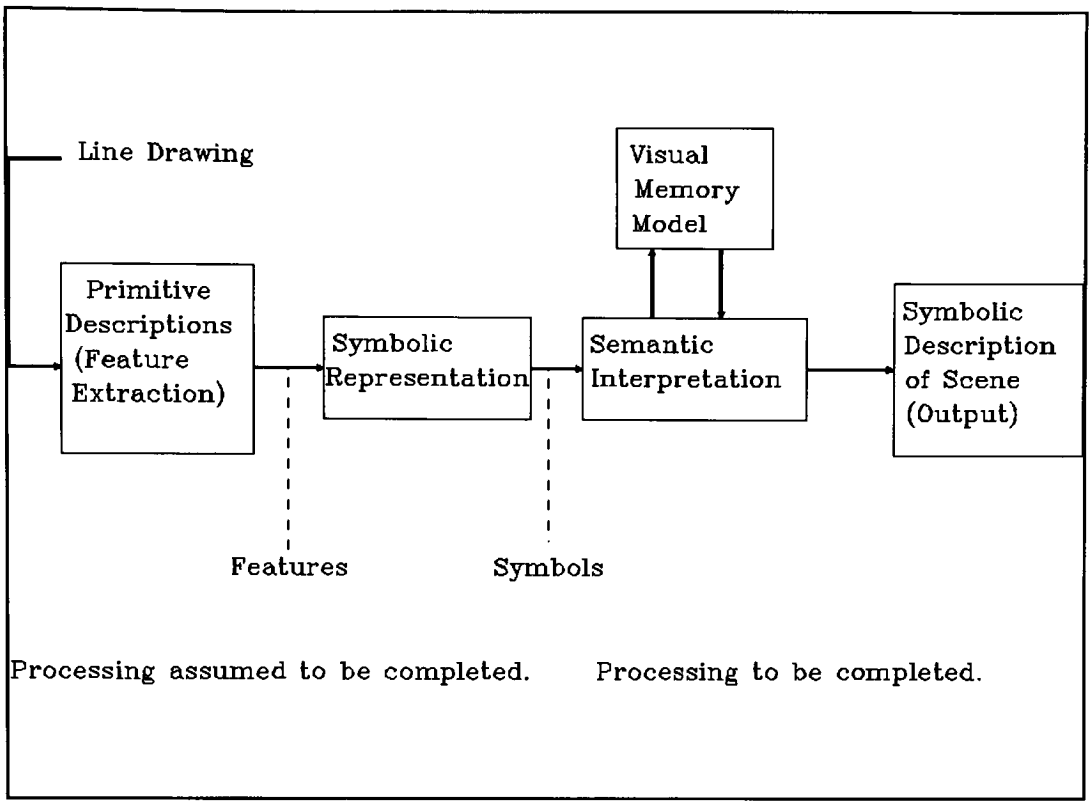

Figure 4-7 System Overview

system is in a symbolic format of generically labeled regions, which constitute the primitives. Generic scene attributes are assumed to have been determined also, and specifically include size (small, medium and large), width, and color. The shapes of the regions include triangular, rectangular, square, circular, and amorphous. A representative subset of the relationships of the primitives include: above, next_to, on, parallel_to and below. Some or all of the aforementioned are used to describe the two-dimensional line drawings that represent the original input, and they are stored in separate input files. The output consists of a symbolic description of the objects in a "scene." Recognition of objects is in a two-dimensional domain; three-dimensionality is not treated. The domain of knowledge includes house scenes at ground level.

Included in the high-level inference system is a tool for generating input. Based on a complete and accurate input file, separate input files are generated using various subsets of

information found in the original input file. Following the initial generation, some files are subject to the introduction of "noise" or "occlusion" with the objective of simulating nearly realistic visual scenes. Therefore, the amount of input, specifically the amount of primitives, constraints and attributes, among input files varies.

The evaluation of output generated from the processing of the input is based on the relative amount of information present in the input files. A symbolic interpretation is reported as output which is generated by traversing the final composition hierarchies. The final output is judged quantitatively, according to how many primitives were matched, and qualitatively, according to how correct the final interpretation is.

The system was tested with correct and complete input, and incomplete or partial information. Generally, the degree to which information can be incomplete and still have the system perform will depend on how large the knowledge base is and to what level of detail a scene is described as represented by the objects in the knowledge base. A minimal amount of accurate information is necessary for the system to attempt to analyze the input and derive an interpretation. In the case of incomplete input data, the system, if necessary, makes a best guess based on the amount of information gathered so far.

In order for the high-level processor to handle erroneous input implies the system has the capability to correct its input after having realized that the information it was provided is in fact incorrect. In other words, the system must be smart enough to adjust itself or the direction it is going. This is an area for further development and is not handled in this thesis. The system may not identify objects in a scene entirely correctly, after a sufficient amount of processing, if presented with an overwhelming amount of inconsistent or contradictory data.

4.3 System Modules

The major modules of the system include a Visual Memory Model, which constitutes the knowledge base, a Current Network Model, which describes symbolically the objects recognized in a scene, an Inference Control Module, which embodies the sequential control strategy that

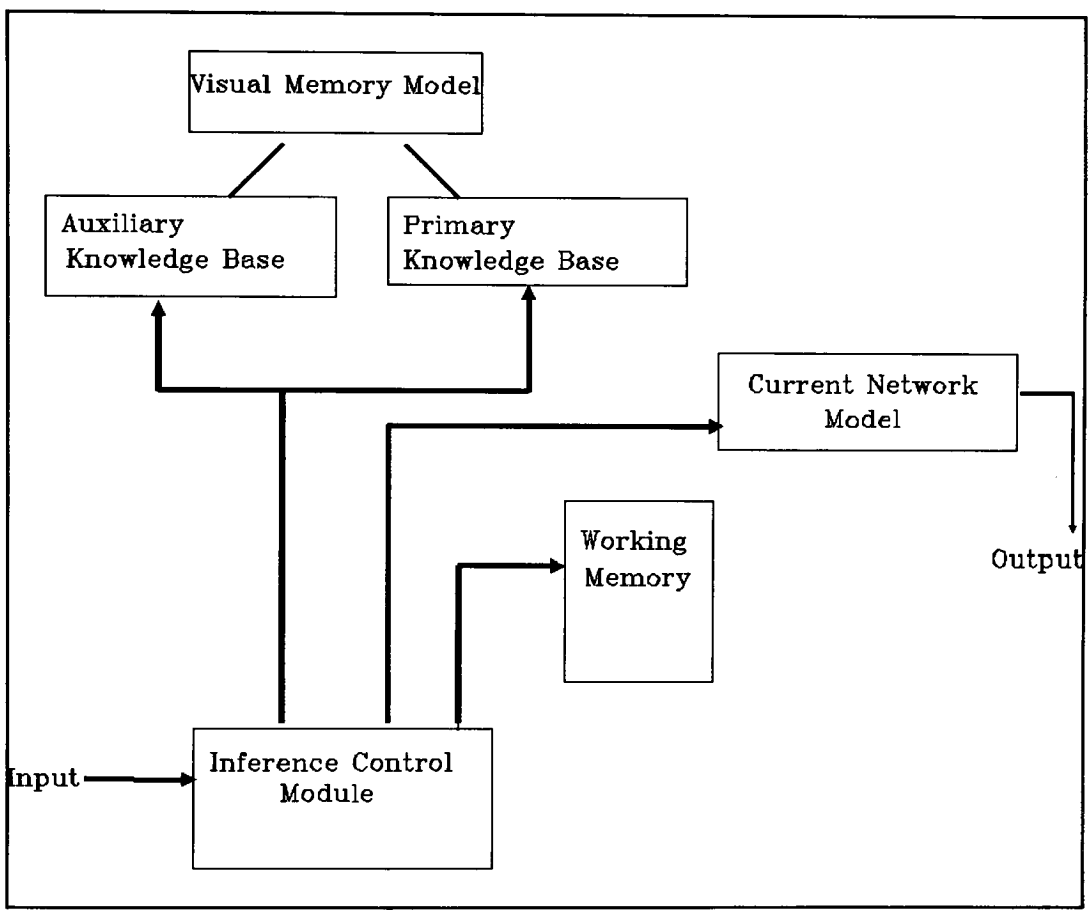

Figure 4-8 Interaction of System Modules

traverses the hierarchies in the knowledge base, and Working Memory, where partial hypotheses are listed. The intention is to develop a system that is modular and extensible (refer to Figure 4-8).

The knowledge base was designed using a frame package that was developed at Rochester Institute of Technology using Prolog [Hiss87, Bhan89]. The remainder of the high-level

inference system is implemented in Prolog. The frame package supports slots, facets and procedural knowledge in the form of demons, and ensures consistency to a certain extent. Predefined slots include is_a and in_of, and user-defined slots are allowed. Some of the facets supported by the package include value, type, minimum and maximum cardinality, and default. The values of the facets, if_needed, if_added, if_removed and if_changed are demons that activate the associated procedures when a particular set of circumstances occurs.

Specialization hierarchies of objects are built using the is_a relation provided by the frame package and a user-defined slot called members. Composition hierarchies, on the other hand, are implicitly devised by defining additional slots, part_of and components. A frame with no value for the components slot, which is constrained further by minimum and maximum cardinality values of 0, represents a terminal node of a composition hierarchy. Similarly, a frame with no value for its part_of slot, further constrained by minimum and maximum cardinality values of 0, reflects a root node of a composition hierarchy.

Instantiations of a particular frame are defined using the in_of slot, and until verified will be considered partial (incomplete) hypotheses. Integral to frames are values for the degree of likelihood that a compositional relationship is true. These values reflect the probability that a relationship is true and are not directly stored in frames, but retrieved from an auxiliary knowledge base. The main purpose of the auxiliary knowledge base is to supplement the primary knowledge base by storing relational knowledge that cannot be represented explicitly in the primary knowledge base due to the limitations of the frame package. Essentially, frames are represented in the primary knowledge base and additional information not represented in the form of frames is stored in the second knowledge base. The purpose of the probability values is to resolve local ambiguities in hypothesis formation and to assist in promoting only the most promising hypotheses.

Initially, bottom-up analysis is invoked to hypothesize an object, given the information in the input file. In other words, processing begins with bottom-up analysis and is initiated by

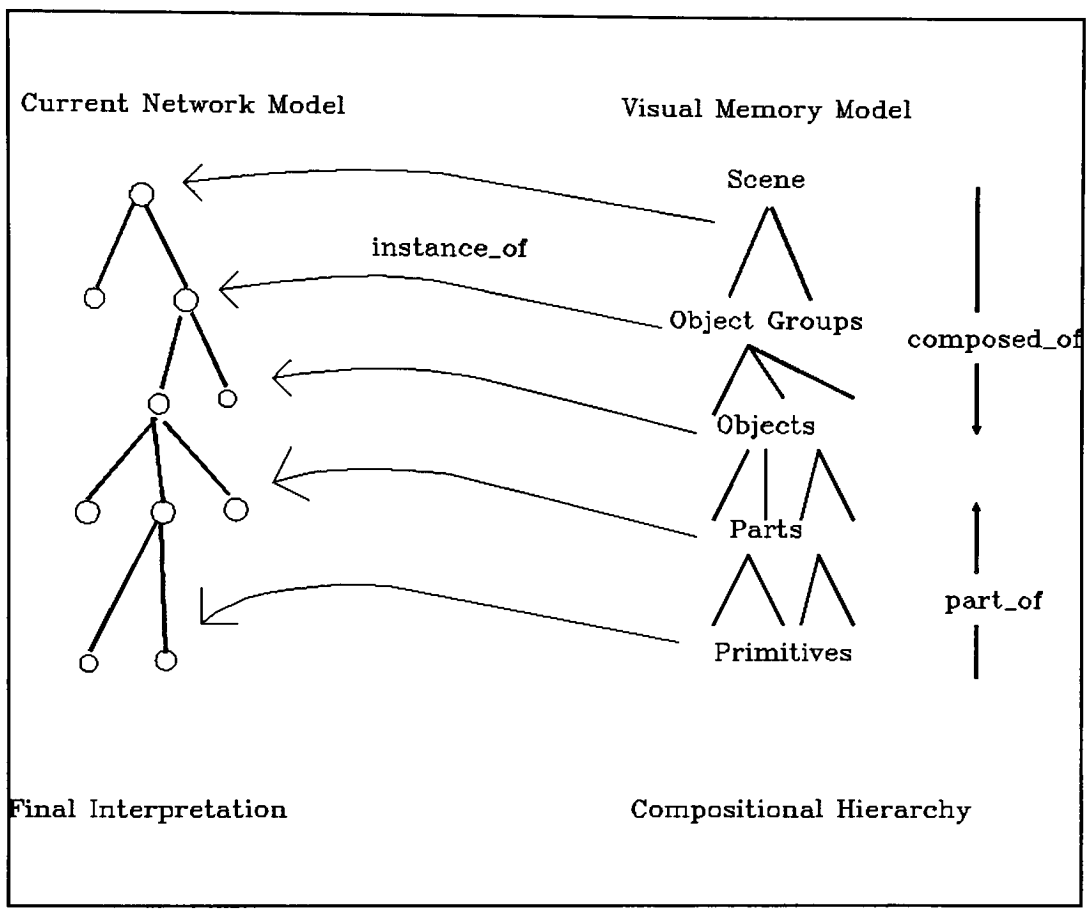

Figure 4-9 Hierarchical Representation

the global control strategy encoded in the Inference Control Module. Procedures that encode the object recognition strategies are attached to a frame and activated when the global controller sends a message to the message slot of that frame. These procedures that are associated with a frame encode the knowledge of how to recognize the object that frame represents. A frame, then, is capable of satisfying its internal constraints.

The purpose of the Current Network Model is to represent the final interpretation of the input. It consists of composition hierarchies of objects identified in the original scene (see Figure

4-9). Working Memory stores hypothetical instances of frames awaiting verification, while the Inference Control Module has access to the knowledge bases and Working Memory.

CHAPTER 5

RESULTS

A high-level inference system was developed using the method described in this thesis with a focus on reasoning with visual knowledge. A goal of the system is to generate the best solution, or description of the scene, given the available input, rather than simply producing a right or wrong answer. Since it was assumed that low-level processing was completed, the input into the system consisted of region primitives, attributes describing the primitives and basic relationships between these primitives that could be considered a reasonable outcome of the lower levels of processing. The system was tested for correctness of implementation and the output or results were evaluated both qualitatively, how accurately it generated a correct interpretation of the input, and quantitatively, how many primitives were correctly matched. The type of input varies from correct to incomplete to extraneous. Some other considerations that affected system design and performance were confining the knowledge base to that which was manageable in terms of processing time and memory consumed during processing. The tendency for the search space to expand greatly was a prevalent concern in the development of the system. Consistency checking in the knowledge base by the frame package was not enforced due to the additional time this operation consumes.

5.1 Example Session with the High-Level Inference System

This section summarizes the steps that are involved in executing the system developed in this thesis and provides an overview of a typical test run.

After the frame package is loaded and all relevant files for object recognition are loaded in the Prolog environment, a test run can proceed. At the operating system command line level, Prolog along with the frame tool are invoked.

```
> prolog frameboot
?- [loadall].              { loads knowledge bases and files }
?- run.                    { run system }
:- Enter regions in input file.   { r1   r18 are valid }
:- Enter end when done.
:- r5.
:- r6.
:- r9.
:- r7.
:- r8.
:- r1.
:- end.                    { infile is created here }
:- Would you like to subject input to noise?
:- n.
:- Would you like to subject input to occlusion?
:- n.              { processing; kbfile and objfile created }
yes.               { session is complete, outfile is created }
```

After the input file, infile, is created and the input is loaded into Prolog's internal database, processing begins. As the process of object recognition proceeds, a knowledge base file and an object file are created and saved to capture snapshots of or windows into the process. These two files are named kbfile and objfile respectively. The final interpretation is stored in a file called outfile. Refer to Appendix A for the contents of the resulting infile, kbfile, objfile, outfile and a file containing the states of recognition.

As processing begins, instantiations of the prototypical frames in the knowledge base are generated in a bottom-up fashion using the available input found in infile. Then, using the attribute and relational knowledge in infile, each of these instantiated frames goes through a constraint satisfaction and matching process where their attributes and relationships (as designated by the slots that are inherited from the frames that these hypotheses are instances of) are "filled

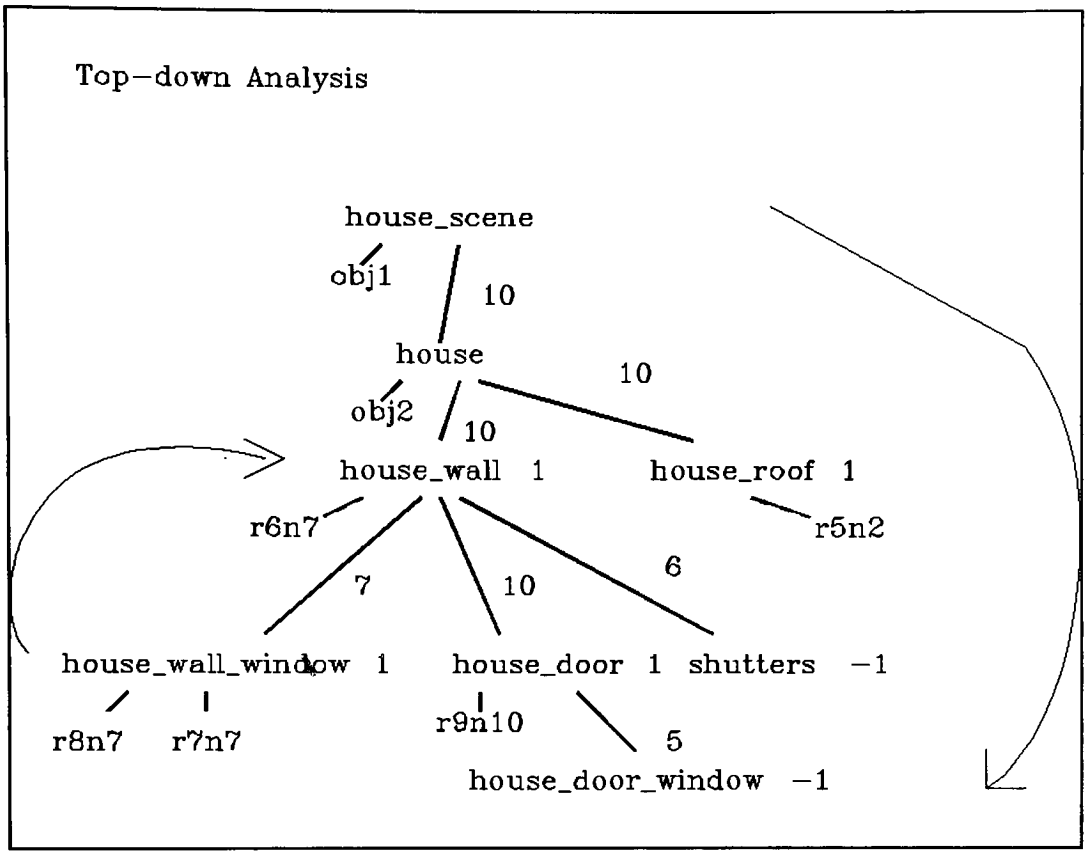

Figure 5-1 Example of Top-down Analysis

in" with available input or cues. These hypotheses will repeatedly undergo tests for validity and consistency checking so as to eliminate the weakest of hypotheses. Composition trees are then built in a top-down manner with each "lower" level node serving as a subgoal to prove or disprove. Compositional relationships are exploited at this point in the processing. For example (see Figure 5-1), in attempting to prove that a house exists in a given input file, the components of a house are checked first. The frames r8n7 and r7n7 are instantiated as house_wall_windows and r9n10 as a house_door. The combination of these two parts and their relatively strong likelihood that they are a part of a house_wall support the existence of a house_wall. However, the fact that there is no instance of the frame shutters (which is also a component of the house_wall frame) and that the likelihood a house_wall is composed of shutters is not strong, will not negate nor negatively affect the existence of a house_wall. Consequently, it is inferred that

an object representing a house_wall is present in the input. This information is then used to prove the goal above the house_wall frame in the compositional hierarchy, in other words, prove that a house exists in the input.

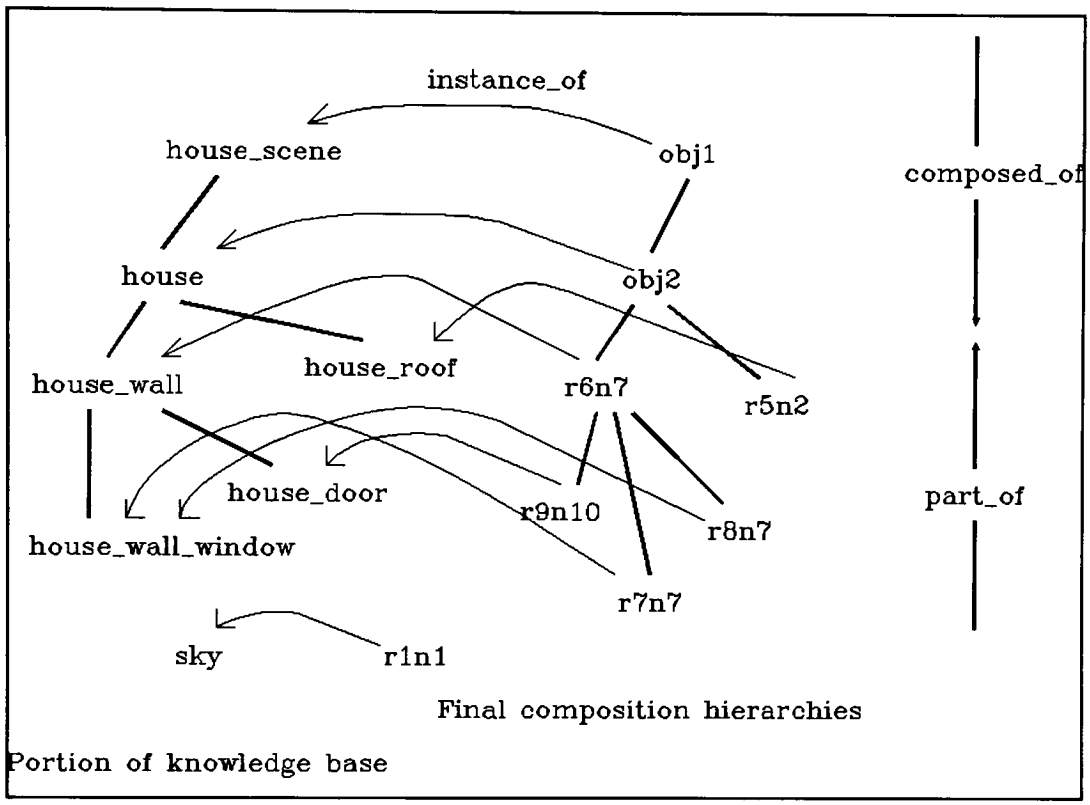

Figure 5-2 Knowledge Base and Current Network Model

The contents of kbfile include the frames that are created during processing of the input. These frames essentially are hypotheses or instantiations of the prototypical frames that are found in kb1, which is the knowledge base that is loaded initially and which represents the visual domain. The kbfile is saved at a point during processing where the most unlikely hypotheses have already been eliminated. As shown in kbfile in Appendix A, the frame r6n7, which is the unique name assigned to an instantiation of a house_wall frame, has a strength value of 0.67 as does r6n8, an instance of garage_wall, but the region r6 can only have one frame representing it. The frame r6n7 will be promoted instead of r6n8 because r6n8's strength value will be

adjusted (decreased) because the value in its 'below' slot (which is r5n1) is not a strong instance. An instance of garage_roof, r5n1, has a strength value of 0.66 while r5n2 has a strength value of 1.0 and therefore, r5n2, an instance of house_roof, will be selected over r5n1. As long as there are no other instantiated frames that are instances of house_wall (as r6n7 is) that have a larger strength value than r6n7, then all other frames that are in_of house_wall and all other frames that have the region prefix r6 are considered "weak" hypotheses, while r6n7 is marked as a strong hypothesis.

The object file, objfile, is saved at a point where the final composition hierarchies have already been created. Its contents reflect the frames that are created to represent objects such as house, garage and car, and to represent other nodes in the composition hierarchy that serve as placeholders (or unused frames) to complete the tree that is built. In the example session from above, Figure 5-2 displays a portion of what objfile represents. With the input file consisting of the regions r5, r6, r7, r8, r9 and r1, the strongest hypotheses that remain following the completion of processing which represent these primitives are r5n2, r6n7, r7n7, r8n7, r9n10, and r1n1 respectively.

5.2 Limitations of the High-Level Inference System

It would seem that regardless of the knowledge representation formalism used to build a visual memory model, there is an inherent subjectivity built into the knowledge base. In other words, those developing a knowledge base will inevitably inject their beliefs and experiences in determining how a part is represented as opposed to the object of which it is composed, to what level of detail a knowledge base is built and what the terms are in which an object is described. Representing knowledge computationally remains a difficult problem, yet its significance lies in the backbone of a computer vision system.

```
?- def_frame(house_wall:
        [is_a:
                [value:[wall], min:1, max:1],
        components:
                [value: [house_wall_window,shutters,house_door],
                min: 0, max:99],
        members:
                [value: [], min: 0, max: 0],
        status:
                [value: [0], min: 0, max: 1],
        in_of:
                [value: []],
        part_of:
                [value: [house], min: 1, max: 99],
        relations:
                [next_to, below], min: 2, max: 99],
        attributes:
                [size], min: 0, max: 99],
        next_to:
                [value: [house_wall, grass, gar_wall],
                min: 0, max: 99],
        below:
                [value: [house_roof], min: 0, max: 99],
        size:
                [value: [large], min: 0, max: 1,
                type: small @@ medium @@ large],
        strength:
                [value: [], min: 0, max: 1],
        message:
                [value: [],
                if_added: recognize_object,
                if_changed: recognize_object]
        ]).
```

Figure 5-3 Typical Frame Representation of an Object

In order for the high-level inference system developed in this thesis to execute properly, a basic format of a frame was designed and utilized. An example of a typical frame is illustrated in Figure 5-3. Implicit in the representation of an object are the necessary relationships to build the specialization and composition hierarchies. These relationships are represented by the members and is_a, and part_of and components slots respectively. Hypothetical instances are

generated from the terminal nodes of the specialization hierarchy. There is also a message slot and an if_added demon, recognize_object, present in all terminal nodes of the specialization hierarchy intended to be inherited by instantiations of these nodes.

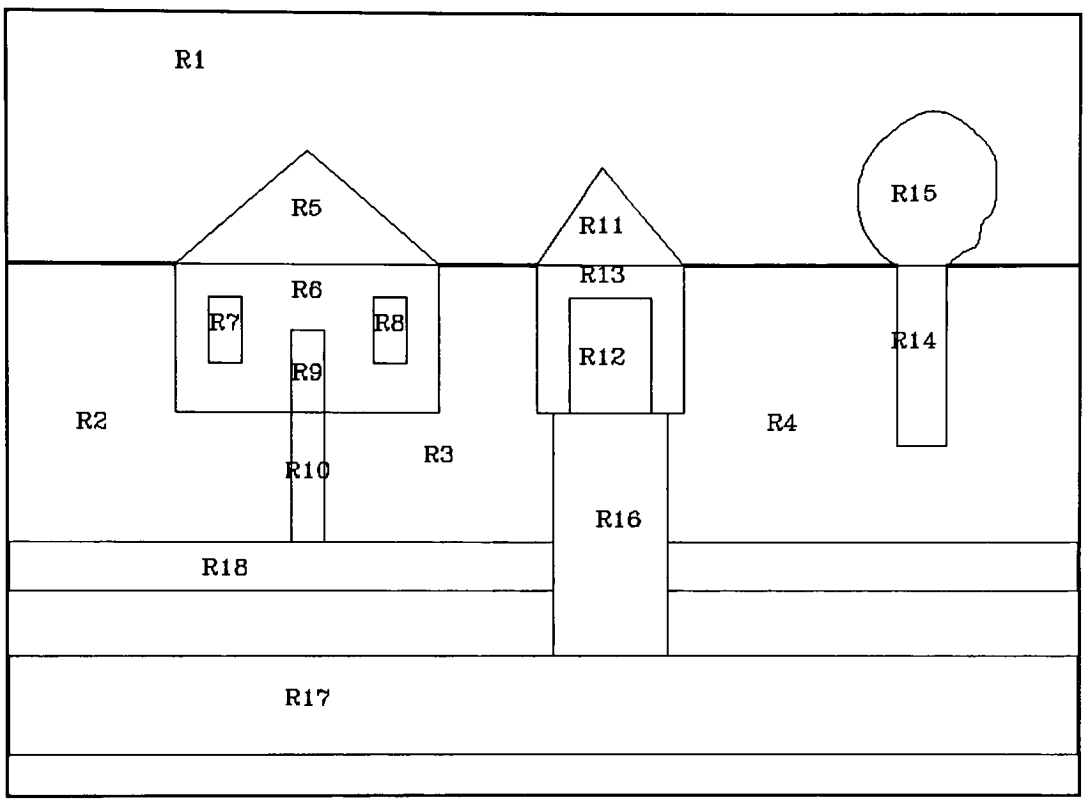

Figure 5-4 Illustration of Scene Input

Additional information about objects is represented in Prolog fact form in the auxiliary knowledge base. Included are the probability an object is composed of a part, and the object to component correspondence. For example, the first of the following Prolog facts implies that there is a probability of 10, on a scale of 1 to 10 with 10 representing the most probable, that a house is composed of a house_wall. The object-to-component correspondence reflects the amount of parts an object can potentially be composed of. For example, a house is composed of 1 roof, and a house_wall has a one-to-many correspondence with house_wall_window as indicated by the number 99.

```
prob_comp_of(house,house_wall,10).
prob_comp_of(house_door,house_door_window,5).
occ(house,house_roof,1).
occ(house_door,house_door_window,1).
occ(house_wall,house_wall_window,99).
occ(car,wheel,2).
```

Generally, these guidelines must be adhered to in order for the control mechanism to manipulate another knowledge base. The master input file, infile1, would need to be replaced, also, in conjunction with a knowledge base substitution. The master input file represents an encompassing set of primitives, relationships and attributes that would reflect a complete input "scene." An illustrative example of what the master input file, infile1, depicts is shown in Figure 5-4. Input files that were tested with the high-level inference system essentially comprise different subsets of information in the master input file.

5.3 Test Plan

The extent of testing the high-level inference system ranged from supplying complete and correct input to the lack of input to extraneous input intended to complicate existing relationships and to introduce contradictions.

The high-level inference system was designed to handle recognition of objects that are closely related, and to derive the best interpretation from presenting the system with primitives that are not closely related. For example, region primitives and relationships representing parts of a house, a walk_way, grass, and parts of a garage were supplied as input, in contrast to input such as region primitives and relationships representing a tree, a house_door, a driveway and a road. An illustrative example of input that consists of unrelated regions is depicted in Figure 5-5. The system was also tested with near realistic scenes that were subject to partial occlusion, which

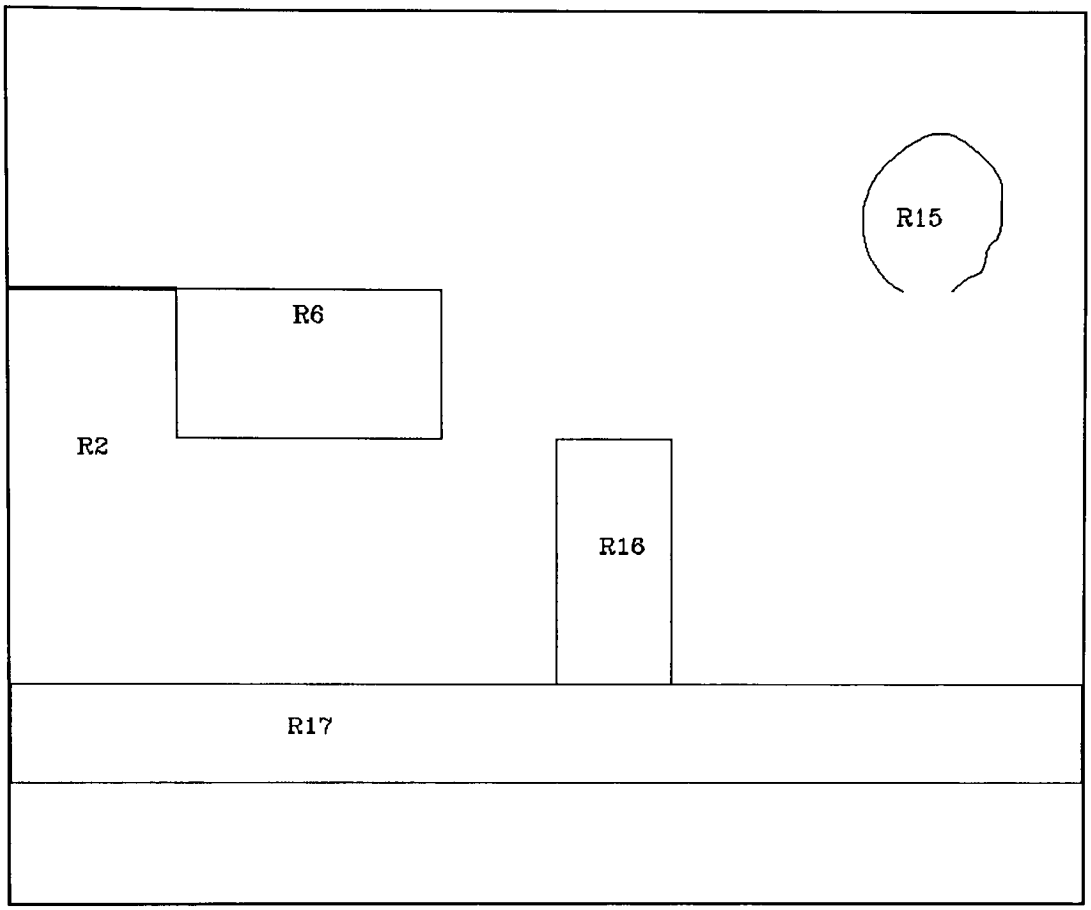

Figure 5-5 Unrelated Regions

translated at a high level into a lack of input or obscured relationships among selected primitives, and interference or noise, which translated at a high level into additional region primitives that confounded relationships with other primitives.

The input tool is an interactive module that prompts the user for the region primitives that are to be used in the current test run or session and was developed to expedite testing of the system. The available set of primitives that potentially could be used in testing the system are found in infile1. The input module also prompts the user if he or she would like to subject the input to noise or occlusion. An example session follows.

```
?- run.
:- Enter regions in input file.
```

```
:- Enter end when you are done.
:- r5.
:- r6.
:- r9.
:- r10.
:- r1.
:- r2.
:- r8.
:- r7.
:- end.
:- Would you like to subject input to noise?
:- n.
:- Would you like to subject input to occlusion?
:- y.
:- Enter primitives to subject to occlusion.
:- Enter end when you are done.
:- r10.
:- r2.
:- end.
```

The processing of the input data begins at this point, but, before processing begins, the relational knowledge of the primitives subject to occlusion are eliminated from Prolog's internal database. If the response to 'Would you like to subject input to noise?' was 'n' then the user would be prompted for 'Would you like to subject input to occlusion?'. If the answer is 'y' to the first question, then an additional input file called xinput is loaded. This file introduces new primitives overlaying the original scene that correspond to the scene illustrated in Figure 5-6.

5.4 Analysis of Results

Results of executing the high-level inference system with 50 test cases are recorded in Table 1 in Appendix B. These results yielded an 89% mean percentage of correctly identified regions with a standard deviation of 19.19. A sample correlation coefficient was computed to determine the extent to which the independent variable, the number of closely related regions, and the dependent variable, the number of correctly identified regions, are related. With an r

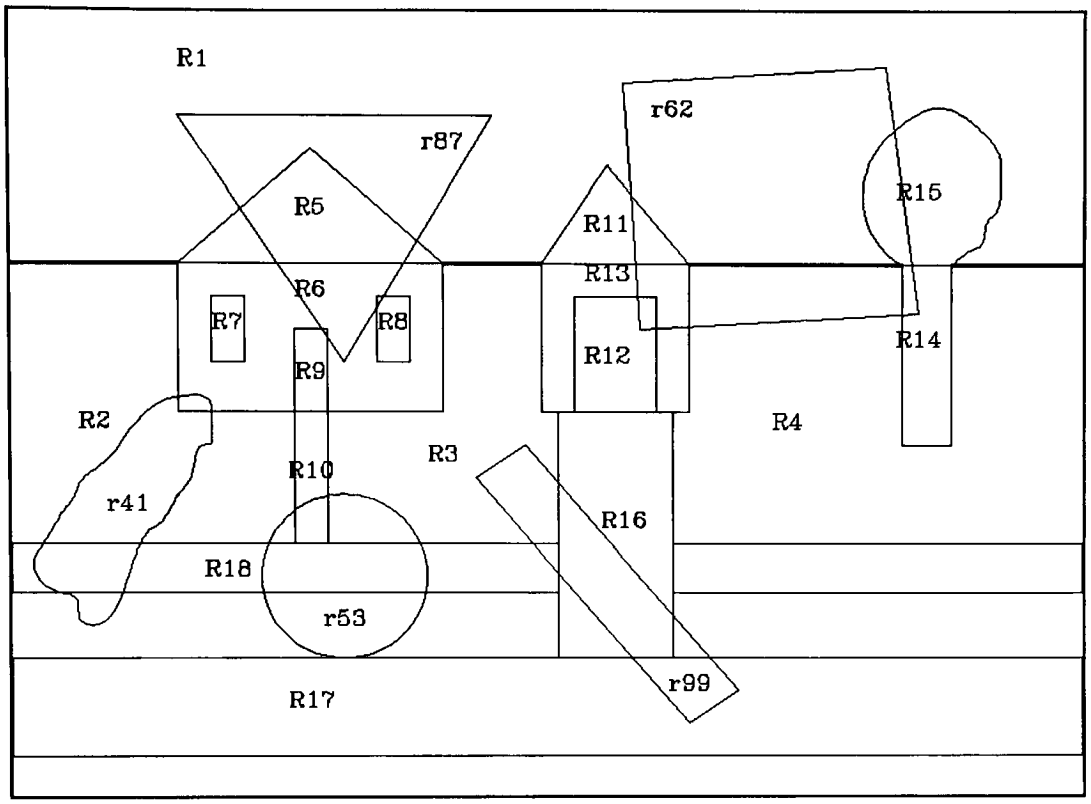

Figure 5-6 Illustration of Scene Input Subject to "Noise"

value of 0.96, the number of closely related regions and the number of correctly identified regions were tested to determine if they are related. It was concluded that these variables are linearly related, implying that the greater the number of closely related regions, the greater the number of correctly identified regions resulted.

In reference to Table 2 in Appendix B, there was a 99.2% rate of correctly identifying regions with input files that consisted of closely related regions, not subject to noise or occlusion. Input files that were subject to 'noise' and consisted of closely related regions resulted in an average 98.6% of correctly identified regions. Input files subject to 'occlusion' yielded an average 88.4%. The input files that consisted of unrelated regions resulted in the lowest mean percentage of correctly identified regions at 67.83% with the greatest standard deviation of 28.46.

The high-level inference system performed well when given complete and correct input in the cases where the primitives were closely related. When supplying incomplete input to the system, it performed reasonably well. The system generated adequate interpretations when presented with objects that were not closely related or subject to occlusion or noise. The results were consistent with how the human visual system operates. Conceptually, if there is not enough evidence or cues or there is conflicting knowledge to recognize an object or interpret a scene, either a best guess is made at the time or the visual target may be deemed unrecognizable, and an interpretation is deferred until more knowledge is acquired.

```
Final interpretation of input is
sky
house_scene
     which is composed of
           walk_way
           grass
           grass
           house
walk_way
grass
grass
house
     which is composed of
           house_wall
           house_roof
house_wall
     which is composed of
           house_door
           house_wall_window
house_door
house_wall_window
house_roof
r2 - unable to interpret accurately due to lack of available information
r16 - unable to interpret accurately due to lack of available information
r7 - unable to interpret accurately due to lack of available information
```

Figure 5-7 Example Output

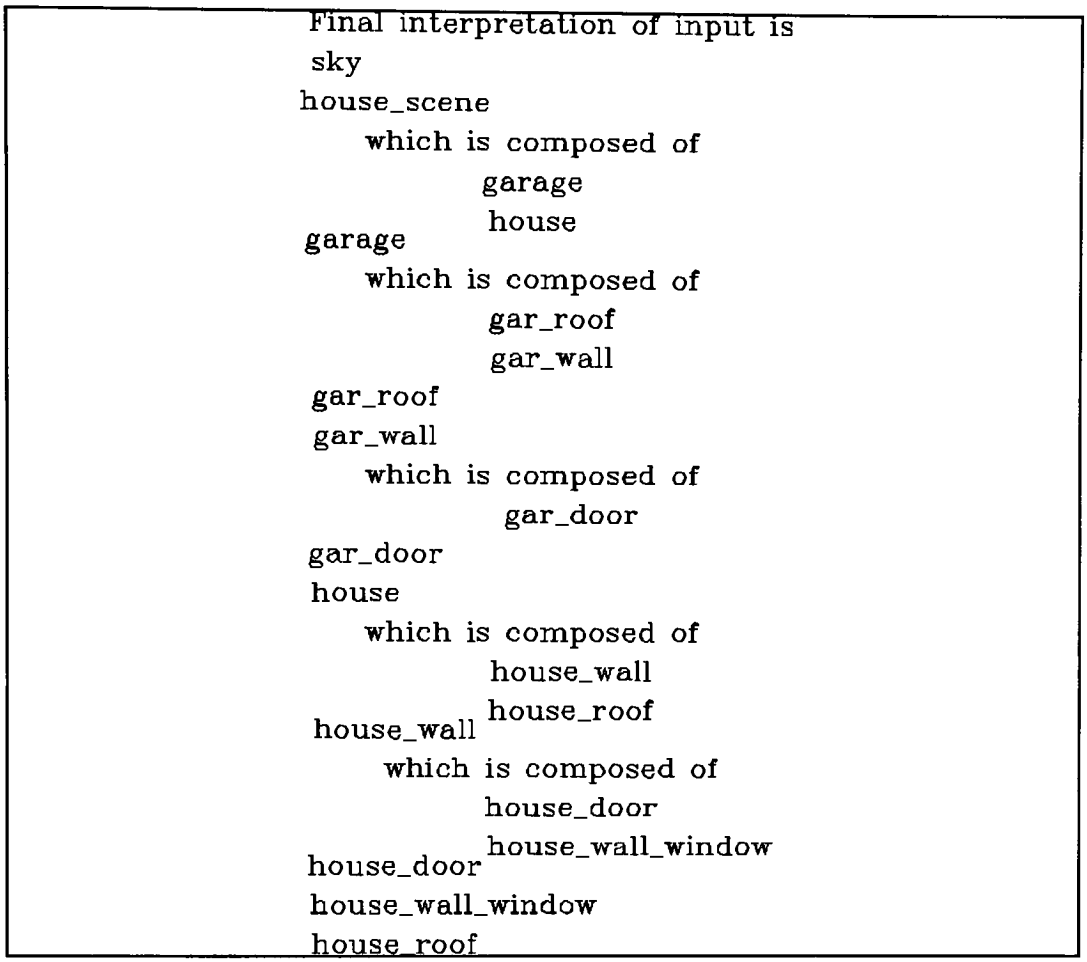

Figure 5-8 Example Output

There was more of a tendency for misinterpretation if input that reflected objects that were not closely related was supplied to the system. Also, there was a greater chance of misinterpretation if an integral component of an object was subject to occlusion due to weaker hypotheses formation.

Examples of output files are illustrated in Figures 5-7 and 5-8. The objects are listed in the farthest left-hand column and may have components, which in turn are listed under the object's name. In some cases, the system will try to generate a best match and qualify its

interpretation with a statement, such as in the case of r2 in Figure 5-7 or is unable to generate a best match and instead reports why it was not able to interpret a portion of the scene correctly.

The processing time of the high-level inference system increased with larger input files. For example, an input file with 18 primitives and corresponding information consumed approximately 60 minutes to complete when run on a personal computer (386) which was more than twice as great as an input file half the size. Elapsed time was approximately 5 minutes for an input file of 2 - 6 primitives and associated information.

CONCLUSION

This thesis addressed the issue of implementing a subset of the overall vision problem in a restricted domain, but proved to be a full scale complex problem in itself. There exist many significant issues that are critical in the design of a high-level portion of a vision system. For instance, the representation of knowledge, whether it be declarative or heuristic or any other type, can be pursued from many different perspectives, using various symbolic formalisms or knowledge representation schemes. The frame-based approach used in this thesis offered a somewhat flexible way to represent knowledge, if creating a relatively small and manageable sized knowledge base to represent a restricted domain. If an attempt is made to incorporate world knowledge, thus increasing the complexity of the knowledge base in terms of relationships and inheritance, management and manipulation of the knowledge would become unwieldy and cumbersome. On the other hand, if a more comprehensive frame tool offering more capability were used, then the issue of system performance would become a more prevalent concern due to the overhead of using a more complex tool. As indicated by the discussion of the existing systems in this thesis, efforts purposely and primarily revolved around confined scene domains.

There are limitations to modeling a reasoning process given Prolog and a visual memory using the frame-based tool previously referenced. Knowledge such as 'the crown is part of a tree and above a trunk and can be found sometimes next to a house or garage in a house scene' can be represented in a knowledge base using frames fairly easily. However, representing knowledge such as 'a rectangular region may be a sidewalk if it is above a road but not above any walkway

or any types of walkways or any members of the class of walkways or any driveway' via the frame package was awkward.

Negative knowledge was difficult to represent using the frame package unless a convention was created to handle this kind of information consistently. Generally, in Prolog, knowledge has to be specifically stated or capable of being inferred, or trying to prove the negation of a goal to satisfy all other cases is sufficient. In other words, if Prolog cannot prove a goal and returns 'yes' because this is the result of the negation of a failed goal, then this is sufficient because that case was not true anyway. This may not be the most reliable way to represent knowledge. Prolog uses a closed world assumption when trying to prove goals which implies that unless something is explicitly stated in a program, it is considered 'not true' and therefore the negation of the goal is true. For example, if the relationship 'above(road,sidewalk)' was not asserted into the database, then the goal 'not above(road,sidewalk)' would be true, but this may be incorrect because a road may be above a sidewalk in a house scene if this information was included in the scene. Consequently, information in the knowledge base in this thesis was stated in a positive sense and the recognition process tested for relationships known to exist. Generally, in representing knowledge of a visual domain there is a need to integrate a way to handle knowledge that is not yet known to the system.

Some of the obstacles encountered in developing a high-level module independent of lower-level modules of an object recognition system included supplying "real" input and the inability to refine inferences due to the lack of feedback to lower levels. Some of the issues to deal with in developing the high-level portion of an object recognition system are the large amount of information that is processed, which in turn is limited by the available hardware and software tools, the limited understanding of visual perception and cognition, and the lack of a more than suitable knowledge representation scheme.

Since the implementation presented in this thesis represents an illustration of the conceptual method previously described, bounds were placed on the development of the high-level inference system due to the limiting characteristics of hardware and software resources. Due to the restrictions imposed, there exist various areas of further development that the implementation may undergo.

One extension to the high-level inference system includes implementing parallelism as opposed to a sequential control mechanism. The processing units that would run in parallel would be an instantiated frame. As one instance is attempting to recognize itself, other instances could be performing the same operations simultaneously. This would in turn increase the system's efficiency in terms of processing time. Message passing among objects could supplement parallelism to facilitate the recognition process.

The number of objects represented in the knowledge base could be increased, which in turn would introduce more hypothetical instances which would increase the search space and potentially cause more contradictions to surface. One way to combat this would be to increase the complexity of the heuristics used to refine and eliminate hypotheses. Also, three-dimensionality could be added to the attribute knowledge which describes objects to expand the realistic dimensions of vision. Another improvement involves maintaining multiple search spaces with the objective of returning to a previous state to undo inferences made in a current state. In other words, a formal method to support nonmonotonic reasoning is needed. A truth maintenance system could be implemented to keep track of and retract dependencies.

Another extension would be to provide a lower level module that was responsible for providing the input into the high-level inference system. If a lower level module were added, feedback from the high-level portion to the low-level could be included to improve the correctness of the final interpretation. If an uncertain situation is encountered at a high-level, then control

can be transferred to a lower level, instructing it to obtain more information or refine the information already obtained.

Training a high-level module of a computer vision system to learn constitutes another possible extension to the system presented in this thesis, and probably the most challenging. In an effort to learn, the system must first recognize that an error has occurred, then it must be able to undo the mistake and correct its line of reasoning. This may require another level of knowledge that oversees the recognition process. A mechanism to register inferences and modify or update the repository of knowledge represented in the visual memory model to reflect the new knowledge would be required. Incorporating some form of feedback or reinforcement as previously mentioned would also be necessary for the system to learn.

The high-level inference system that was developed in this thesis implemented reasoning processes and utilized a visual memory model to achieve object recognition in a specific domain. Some of the conceptual issues of visual cognition were represented and implemented, for example, focusing attention, drawing inferences by combining evidence, nonmonotonic reasoning and the representation of knowledge. Overall, high-level computer vision encompasses a wide variety of problems to solve that are more well-defined at a conceptual level than at an implementation level. Perhaps it is the lack of insight into visual perception and cognition at a conceptual level that inevitably adversely affects the efforts at an implementation level. Ultimately, learning more about the human visual system--the speed with which the human visual system processes information, how the tremendous amounts of knowledge are stored in the brain, the ease with which this knowledge is accessed, and the endless capacity to reason and learn--will assist in the discovery of new approaches to modeling vision. The high-level inference system developed in this thesis attempted to deal with some of the conceptual issues.

Bibliography

[Ball78]

Ballard, D. H.; Brown, C. M.; and Feldman, J. A. "An Approach to Knowledge-directed Image Analysis." In Computer Vision Systems. Hanson, A. and Riseman, E., eds. New York, New York: Academic Press. 1978. pp. 271 - 281.

[Barr81]

Barrow, Harry G. and Tenenbaum, Jay M. "Computational Vision." *Proceedings of the IEEE*. Vol. 69. No. 5. pp 572 595. May 1981.

[Bhan89]

Bhandari, Archna. "Enhancements to the Frame Virtual Machine." Master's Thesis. Rochester Institute of Technology. Rochester, New York. 1989.

[Brat86]

Bratko, Ivan. Prolog Programming for Artificial Intelligence. Reading, Massachusetts: Addison-Wesley Publishing Company. 1986.

[Cloc81]

Clocksin, W. F. and Mellish, C. S. Programming in Prolog. New York, New York: Springer-Verlag. 1981.

[Cohe82]

Cohen, Paul R. and Feigenbaum, Edward A., eds. The Handbook of Artificial Intelligence, Volume 3. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc. 1982.

[Davi85]

Davis, Larry S. and Hwang, S. S. Vincent. "The SIGMA Image Understanding System." *IEEE Third Workshop on Computer Vision: Representation and Control.* 1985. pp. 19 - 26.

[deKl86]

deKleer, Johan. "An Assumption-Based TMS." *Artificial Intelligence.* Volume 28. No. 2. March 1986. pp. 127 - 162.

[Falk72]

Falk, Gilbert. "Interpretation of Imperfect Line Data as a Three-Dimensional Scene." *Artificial Intelligence.* Volume 3. No. 2. Summer 1972. pp. 101 144.

[Fisc87]

Fischler, Martin A. and Firschein, Oscar. Intelligence The Eye, the Brain and the Computer. Reading, Massachusetts: Addison-Wesley Publishing Company. 1987.

[Glic82]
Glicksman, Jay. "A Schemata-Based System for Utilizing Cooperating Knowledge Sources in Computer Vision." *Proceedings Fourth Biennial Conference of Canadian Society for the Computer Studies of Intelligence.* University of Saskatoon, Canada. May 1982. pp. 33 - 39.

[Glic83]
Glicksman, Jay. "Using Multiple Information Sources in a Computational Vision System." *Proceedings of Eighth International Joint Conference on Artificial Intelligence.* Volume 2. August 1983. pp. 1078 1080.

[Guzm68]
Guzman, Adolfo. "Decomposition of a Visual Scene into Three-Dimensional Bodies." *AFIPS Fall Joint Computer Conference Proceedings.* Volume 33. Part 1. 1968. pp. 291 - 304.

[Hans78a]
Hanson, Allen R. and Riseman, Edward M., eds. Computer Vision Systems. New York, New York: Academic Press. 1978.

[Hans78b]
Hanson, Allen R. and Riseman, Edward M. "VISIONS: A Computer System for Interpreting Scenes." In Computer Vision Systems. Hanson, Allen R. and Riseman, Edward M., eds. New York, New York: Academic Press. 1978.

[Hans88]
Hanson, Allen R. and Riseman, Edward M. "The VISIONS Image-Understanding System." In Advances in Computer Vision, Volume 1. Brown,Christopher, ed. Hillsdale, New Jersey: Lawrence Erlbaum Associates. 1988.

[Have83a]
Havens, William. "Recognition Mechanisms for Schema-Based Knowledge Representations." *Computer and Mathematics with Applications.* Volume 9. No. 1. 1983. pp. 185 - 199.

[Have83b]
Havens, William and Mackworth, Alan. "Representing Knowledge of the Visual World." *Computer.* Volume 16. No. 10. October 1983. pp. 90 - 96.

[Have85]
Havens, William. "A Theory of Schema Labelling." *Computational Intelligence.* Volume 1. No. 3. August 1985. pp. 127 139.

[Hiss87]
Hiss, LaMora. "A Frame Virtual Machine in C-Prolog." Master's Thesis. Rochester Institute of Technology. Rochester, New York. 1987.

[Hwan86]
Hwang, Vincent Shang-Shouq; Davis, Larry S. and Matsuyama, Takashi. "Hypothesis Integration in Image Understanding Systems." *Computer Vision, Graphics, and Image Processing*. Volume 36. 1986. pp. 321 - 371.

[Kim84]
Kim, Jin, H.; Payton, David W. and Olin, Karen E. "An Expert System for Object Recognition in Natural Scenes." *IEEE Computer Society Conference on Artificial Intelligence Applications*. 1984. pp. 170 175.

[Kim85]
Kim, Jin H. "A Distributed Computational Model of Plausible Classification Reasoning." *Second Conference on Artificial Intelligence Applications*. 1985. pp. 210 - 214.

[Lowe85]
Lowe, David G. Perceptual Organization and Visual Recognition. Boston, Massachusetts: Kluwer Academic Publishers. 1985.

[Mack81]
Mackworth, Alan K. and Havens, William S. "Structuring Domain Knowledge for Visual Perception." *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*. Volume II. August 1981. pp. 625 - 627.

[Mins75]
Minsky, Marvin. "A Framework for Representing Knowledge." In The Psychology of Computer Vision. Winston, Patrick Henry, ed. New York, New York: McGraw-Hill Book Company. 1975.

[Neis67]
Neisser, Ulric. Cognitive Psychology. New York, New York: Meredith Publishing Company. 1967.

[Neis76]
Neisser, Ulric. Cognition and Reality: Principles and Implications of Cognitive Psychology. San Francisco, California: W. H. Freeman and Company. 1976.

[Neva78]
Nevatia, Ramakant. "Characterization and Requirements of Computer Vision Systems." In Computer Vision Systems. Hanson, Allen R. and Riseman, Edward M., eds. New York, New York: Academic Press. 1978.

[Prov88]
Provan, Gregory M. "Model-Based Object Recognition: A Truth Maintenance Approach." *IEEE Fourth Conference on Artificial Intelligence Applications*. 1988. pp. 230 - 235.

[Rao88]

Rao, A. Ravishankar and Jain, Ramesh. "Knowledge Representation and Control in Computer Vision Systems." *IEEE Expert*. Volume 3. No. 1. Spring 1988. pp. 64 - 70.

[Robe65]

Roberts, L. G. "Machine Perception of Three-Dimensional Solids." In Optical and Electro-Optical Information Processing. Tippett, J. T.; Berkowitz, D. A.; Clapp, L. C.; Koester, C. J.; and Vanderburgh, Jr., A., eds. Cambridge, Massachusetts: The Massachusetts Institute of Technology Press. 1965.

[Wesl82]

Wesley, Leonard P. and Hanson, Allen R. "The Use of an Evidential-Based Model for Representing Knowledge and Reasoning about Images in the VISIONS System." *IEEE Workshop on Computer Vision: Representation and Control*. 1982. pp. 14 - 25.

[Wins75]

Winston, Patrick H., ed. The Psychology of Computer Vision. New York, New York: McGraw-Hill Book Company. 1975.

Appendix A

The following files represent a sample test run of the high-level inference system that was developed in this thesis. The files include infile, kbfile, objfile, outfile and a file that represents the states of recognition during processing.

in_127cc

input_list([amorph(r1),rect_region(r9),sq_region(r8),sq_region(r7),tri_region(r5),rect_region(r6)])
blue(r1)
large(r6)
small(r9)
medium(r7)
medium(r8)
large(r5)
below(r5,r1)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
below(r7,r5)
below(r8,r5)
above(r7,r9)
above(r8,r9)
above(r1,r5)
above(r5,r6)
on(r7,r6)
on(r8,r6)
on(r9,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)

```
?- def_frame(r1n1:
        [strength:
            [value:[1]],
        color:
            [value:[blue]],
        message:
            [value:[match_attributes]],
        above:
            [value:[r5n1,r5n2]],
        in_of:
            [value:[sky]]
        ]).

?- def_frame(r9n1:
        [strength:
            [value:[0.5]],
        size:
            [value:[small]],
        message:
            [value:[match_attributes]],
        on:
            [value:[r6n7]],
        in_of:
            [value:[shutters]]
        ]).

?- def_frame(r9n2:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.25]],
        below:
            [value:[r8n1,r7n1]],
        in_of:
            [value:[road]]
        ]).

?- def_frame(r9n3:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.666666]],
        below:
            [value:[r8n3,r7n3,r8n2,r7n2]],
        next_to:
            [value:[r8n3,r7n3,r8n2,r7n2]],
        in_of:
            [value:[grass]]
        ]).

?- def_frame(r9n4:
        [message:
            [value:[match_attributes]],
```

```
        strength:
            [value:[0.666666]],
        next_to:
            [value:[r8n1,r7n1]],
        below:
            [value:[r8n5,r7n5]],
        in_of:
            [value:[walk_way]]
        ]).

?- def_frame(r9n5:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.25]],
        below:
            [value:[r8n1,r7n1]],
        in_of:
            [value:[sidewalk]]
        ]).

?- def_frame(r9n6:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.5]],
        next_to:
            [value:[r8n1,r7n1]],
        below:
            [value:[r8n4,r7n4,r8n3,r7n3]],
        in_of:
            [value:[driveway]]
        ]).

?- def_frame(r9n7:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.666666]],
        below:
            [value:[r5n2]],
        next_to:
            [value:[r8n3,r7n3,r8n1,r7n1,r8n2,r7n2]],
        in_of:
            [value:[house_wall]]
        ]).

?- def_frame(r9n8:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.666666]],
        below:
```

```
        [value:[r5n1]],
    next_to:
        [value:[r8n2,r7n2]],
    in_of:
        [value:[gar_wall]]
    ]).

?- def_frame(r9n9:
    [message:
        [value:[match_attributes]],
    strength:
        [value:[0.5]],
    on:
        [value:[r6n8]],
    in_of:
        [value:[gar_door]]
    ]).

?- def_frame(r9n10:
    [strength:
        [value:[0.8]],
    size:
        [value:[small]],
    message:
        [value:[match_attributes]],
    below:
        [value:[r5n2]],
    on:
        [value:[r6n7]],
    next_to:
        [value:[r8n7,r7n7]],
    in_of:
        [value:[house_door]]
    ]).

?- def_frame(r9n12:
    [message:
        [value:[match_attributes]],
    strength:
        [value:[0.6]],
    next_to:
        [value:[r8n7,r7n7]],
    on:
        [value:[r6n7]],
    below:
        [value:[r5n2]],
    in_of:
        [value:[house_wall_window]]
    ]).

?- def_frame(r9n13:
    [message:
        [value:[match_attributes]],
```

```
        strength:
            [value:[0.666666]],
        on:
            [value:[r6n10]],
        below:
            [value:[r5n2]],
        in_of:
            [value:[house_door_window]]
        ]).

?- def_frame(r9n14:
        [strength:
            [value:[0.666666]],
        size:
            [value:[small]],
        message:
            [value:[match_attributes]],
        below:
            [value:[r5n1]],
        in_of:
            [value:[gar_window]]
        ]).

?- def_frame(r9n15:
        [strength:
            [value:[0.5]],
        size:
            [value:[small]],
        message:
            [value:[match_attributes]],
        in_of:
            [value:[car_window]]
        ]).

?- def_frame(r8n1:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.333333]],
        next_to:
            [value:[r7n3,r7n2]],
        in_of:
            [value:[grass]]
        ]).

?- def_frame(r8n2:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.666666]],
        below:
            [value:[r5n2]],
        next_to:
```

```
        [value:[r7n3,r7n1,r7n2]],
    in_of:
        [value:[house_wall]]
    ]).

?- def_frame(r8n3:
    [message:
        [value:[match_attributes]],
    strength:
        [value:[0.666666]],
    below:
        [value:[r5n1]],
    next_to:
        [value:[r7n2]],
    in_of:
        [value:[gar_wall]]
    ]).

?- def_frame(r8n4:
    [strength:
        [value:[1]],
    size:
        [value:[medium]],
    message:
        [value:[match_attributes]],
    on:
        [value:[r6n8]],
    in_of:
        [value:[gar_door]]
    ]).

?- def_frame(r8n5:
    [message:
        [value:[match_attributes]],
    strength:
        [value:[0.8]],
    below:
        [value:[r5n2]],
    above:
        [value:[r9n4]],
    on:
        [value:[r6n7]],
    next_to:
        [value:[r7n7]],
    in_of:
        [value:[house_door]]
    ]).

?- def_frame(r8n7:
    [strength:
        [value:[1]],
    size:
        [value:[medium]],
```

```
        message:
            [value:[match_attributes]],
        next_to:
            [value:[r7n7]],
        above:
            [value:[r9n10]],
        on:
            [value:[r6n7]],
        below:
            [value:[r5n2]],
        in_of:
            [value:[house_wall_window]]
        ]).

?- def_frame(r8n8:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.666666]],
        on:
            [value:[r6n10]],
        below:
            [value:[r5n2]],
        in_of:
            [value:[house_door_window]]
        ]).

?- def_frame(r8n9:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.333333]],
        below:
            [value:[r5n1]],
        in_of:
            [value:[gar_window]]
        ]).

?- def_frame(r7n1:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.333333]],
        next_to:
            [value:[r8n3,r8n2]],
        in_of:
            [value:[grass]]
        ]).

?- def_frame(r7n2:
        [message:
            [value:[match_attributes]],
        strength:
```

```
        [value:[0.666666]],
    below:
        [value:[r5n2]],
    next_to:
        [value:[r8n3,r8n1,r8n2]],
    in_of:
        [value:[house_wall]]
    ]).

?- def_frame(r7n3:
    [message:
        [value:[match_attributes]],
    strength:
        [value:[0.666666]],
    below:
        [value:[r5n1]],
    next_to:
        [value:[r8n2]],
    in_of:
        [value:[gar_wall]]
    ]).

?- def_frame(r7n4:
    [strength:
        [value:[1]],
    size:
        [value:[medium]],
    message:
        [value:[match_attributes]],
    on:
        [value:[r6n8]],
    in_of:
        [value:[gar_door]]
    ]).

?- def_frame(r7n5:
    [message:
        [value:[match_attributes]],
    strength:
        [value:[0.8]],
    below:
        [value:[r5n2]],
    above:
        [value:[r9n4]],
    on:
        [value:[r6n7]],
    next_to:
        [value:[r8n7]],
    in_of:
        [value:[house_door]]
    ]).

?- def_frame(r7n7:
```

```
        [strength:
            [value:[1]],
        size:
            [value:[medium]],
        message:
            [value:[match_attributes]],
        next_to:
            [value:[r8n7]],
        above:
            [value:[r9n10]],
        on:
            [value:[r6n7]],
        below:
            [value:[r5n2]],
        in_of:
            [value:[house_wall_window]]
        ]).

?- def_frame(r7n8:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.666666]],
        on:
            [value:[r6n10]],
        below:
            [value:[r5n2]],
        in_of:
            [value:[house_door_window]]
        ]).

?- def_frame(r7n9:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.333333]],
        below:
            [value:[r5n1]],
        in_of:
            [value:[gar_window]]
        ]).

?- def_frame(r5n1:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.666666]],
        below:
            [value:[r1n1]],
        above:
            [value:[r6n8]],
        in_of:
            [value:[gar_roof]]
```

```
        ]).

?- def_frame(r5n2:
        [strength:
            [value:[1]],
        size:
            [value:[large]],
        message:
            [value:[match_attributes]],
        below:
            [value:[r1n1]],
        above:
            [value:[r6n7]],
        in_of:
            [value:[house_roof]]
        ]).

?- def_frame(r6n7:
        [strength:
            [value:[0.666666]],
        size:
            [value:[large]],
        message:
            [value:[match_attributes]],
        below:
            [value:[r5n2]],
        in_of:
            [value:[house_wall]]
        ]).

?- def_frame(r6n8:
        [strength:
            [value:[0.666666]],
        size:
            [value:[large]],
        message:
            [value:[match_attributes]],
        below:
            [value:[r5n1]],
        in_of:
            [value:[gar_wall]]
        ]).

?- def_frame(r6n10:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.2]],
        below:
            [value:[r5n2]],
        in_of:
            [value:[house_door]]
        ]).
```

```
?- def_frame(r6n12:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.2]],
        below:
            [value:[r5n2]],
        in_of:
            [value:[house_wall_window]]
        ]).

?- def_frame(r6n13:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.333333]],
        below:
            [value:[r5n2]],
        in_of:
            [value:[house_door_window]]
        ]).

?- def_frame(r6n14:
        [message:
            [value:[match_attributes]],
        strength:
            [value:[0.333333]],
        below:
            [value:[r5n1]],
        in_of:
            [value:[gar_window]]
        ]).
```

```
?- def_frame(obj1:
        [status:
            [value:[1]],
        components:
            [value:[obj2]],
        in_of:
            [value:[house_scene]]
        ]).

?- def_frame(obj2:
        [status:
            [value:[1]],
        components:
            [value:[r6n7,r5n2]],
        in_of:
            [value:[house]]
        ]).

?- def_frame(obj3:
        [status:
            [value:[-1]],
        in_of:
            [value:[shutters]]
        ]).

?- def_frame(obj4:
        [status:
            [value:[-1]],
        in_of:
            [value:[house_door_window]]
        ]).

?- def_frame(obj5:
        [status:
            [value:[-1]],
        components:
            [value:[obj6]],
        in_of:
            [value:[garage]]
        ]).

?- def_frame(obj6:
        [status:
            [value:[1]],
        components:
            [value:[obj7]],
        in_of:
            [value:[gar_wall]]
        ]).

?- def_frame(obj7:
        [status:
            [value:[1]],
```

```
        in_of:
            [value:[gar_door]]
        ]).

?- def_frame(obj8:
        [status:
            [value:[-1]],
        in_of:
            [value:[gar_window]]
        ]).

?- def_frame(obj9:
        [status:
            [value:[-1]],
        in_of:
            [value:[gar_roof]]
        ]).

?- def_frame(obj10:
        [status:
            [value:[-1]],
        in_of:
            [value:[grass]]
        ]).

?- def_frame(obj11:
        [status:
            [value:[-1]],
        in_of:
            [value:[road]]
        ]).

?- def_frame(obj12:
        [status:
            [value:[-1]],
        in_of:
            [value:[car]]
        ]).

?- def_frame(obj13:
        [status:
            [value:[-1]],
        in_of:
            [value:[wheel]]
        ]).

?- def_frame(obj14:
        [status:
            [value:[-1]],
        in_of:
            [value:[car_window]]
        ]).
```

```
?- def_frame(obj15:
        [status:
            [value:[-1]],
        in_of:
            [value:[car_door]]
        ]).

?- def_frame(obj16:
        [status:
            [value:[-1]],
        in_of:
            [value:[driveway]]
        ]).

?- def_frame(obj17:
        [status:
            [value:[-1]],
        in_of:
            [value:[walk_way]]
        ]).

?- def_frame(obj18:
        [status:
            [value:[-1]],
        in_of:
            [value:[tree]]
        ]).

?- def_frame(obj19:
        [status:
            [value:[-1]],
        in_of:
            [value:[trunk]]
        ]).

?- def_frame(obj20:
        [status:
            [value:[-1]],
        in_of:
            [value:[crown]]
        ]).
```

Final interpretation of input is

sky

house_scene
  which is composed of
    house

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door
    house_wall_window
    house_wall_window

house_door

house_wall_window

house_wall_window

house_roof

NO ERRORS TO REPORT.

state(r6n7,[below],3)
state(r6n7,[size,below],3)
state(r6n8,[below],3)
state(r6n8,[size,below],3)
state(r6n10,[below],5)
state(r6n12,[below],5)
state(r6n13,[below],3)
state(r6n14,[below],3)
state(r5n1,[above],3)
state(r5n1,[below,above],3)
state(r5n2,[above],3)
state(r5n2,[below,above],3)
state(r5n2,[size,below,above],3)
state(r7n1,[next_to],3)
state(r7n1,[next_to],3)
state(r7n2,[next_to],3)
state(r7n2,[next_to],3)
state(r7n2,[next_to],3)
state(r7n2,[below,next_to],3)
state(r7n3,[next_to],3)
state(r7n3,[below,next_to],3)
state(r7n4,[on],2)
state(r7n4,[size,on],2)
state(r7n5,[next_to],5)
state(r7n5,[on,next_to],5)
state(r7n5,[above,on,next_to],5)
state(r7n5,[below,above,on,next_to],5)
state(r7n7,[below],5)
state(r7n7,[on,below],5)
state(r7n7,[above,on,below],5)
state(r7n7,[next_to,above,on,below],5)
state(r7n7,[size,next_to,above,on,below],5)
state(r7n8,[below],3)
state(r7n8,[on,below],3)
state(r7n9,[below],3)
state(r8n1,[next_to],3)
state(r8n1,[next_to],3)
state(r8n2,[next_to],3)
state(r8n2,[next_to],3)
state(r8n2,[next_to],3)
state(r8n2,[below,next_to],3)
state(r8n3,[next_to],3)
state(r8n3,[below,next_to],3)
state(r8n4,[on],2)
state(r8n4,[size,on],2)
state(r8n5,[next_to],5)
state(r8n5,[on,next_to],5)
state(r8n5,[above,on,next_to],5)
state(r8n5,[below,above,on,next_to],5)
state(r8n7,[below],5)
state(r8n7,[on,below],5)
state(r8n7,[above,on,below],5)
state(r8n7,[next_to,above,on,below],5)

state(r8n7,[size,next_to,above,on,below],5)
state(r8n8,[below],3)
state(r8n8,[on,below],3)
state(r8n9,[below],3)
state(r9n1,[on],4)
state(r9n1,[size,on],4)
state(r9n2,[below],4)
state(r9n2,[below],4)
state(r9n3,[next_to],3)
state(r9n3,[next_to],3)
state(r9n3,[next_to],3)
state(r9n3,[next_to],3)
state(r9n3,[below,next_to],3)
state(r9n3,[below,next_to],3)
state(r9n3,[below,next_to],3)
state(r9n3,[below,next_to],3)
state(r9n4,[below],3)
state(r9n4,[below],3)
state(r9n4,[next_to,below],3)
state(r9n4,[next_to,below],3)
state(r9n5,[below],4)
state(r9n5,[below],4)
state(r9n6,[below],4)
state(r9n6,[below],4)
state(r9n6,[below],4)
state(r9n6,[below],4)
state(r9n6,[next_to,below],4)
state(r9n6,[next_to,below],4)
state(r9n7,[next_to],3)
state(r9n7,[next_to],3)
state(r9n7,[next_to],3)
state(r9n7,[next_to],3)
state(r9n7,[next_to],3)
state(r9n7,[next_to],3)
state(r9n7,[below,next_to],3)
state(r9n8,[next_to],3)
state(r9n8,[next_to],3)
state(r9n8,[below,next_to],3)
state(r9n9,[on],2)
state(r9n10,[next_to],5)
state(r9n10,[next_to],5)
state(r9n10,[on,next_to],5)
state(r9n10,[below,on,next_to],5)
state(r9n10,[size,below,on,next_to],5)
state(r9n12,[below],5)
state(r9n12,[on,below],5)
state(r9n12,[next_to,on,below],5)
state(r9n12,[next_to,on,below],5)
state(r9n13,[below],3)
state(r9n13,[on,below],3)
state(r9n14,[below],3)
state(r9n14,[size,below],3)
state(r9n15,[size],2)

state(r1n1,[above],2)
state(r1n1,[above],2)
state(r1n1,[color,above],2)

Appendix B

TABLE 1

Table of Results
Summary of Identification Accuracy

Input File Name	Number of regions	Closely Related Regions	Input Subject to Noise/Occlusion	% Correctly Identified Regions
in_128b	12	Y	N / N	100
in_128c	4	Y	N / N	100
in_127f	5	N	N / N	80
in_128a	10	Y	N / N	100
in_127d	6	Y	N / N	100
in_127e	8	Y	N / N	100
in_127a	7	Y	N / N	85
in_127c	6	Y	N / N	100
in_127b	9	Y	N / N	100
in_120	9	Y	N / N	100
in_120a	9	Y	N / N	100
in_213a	6	Y	Y / N	100
in_210a	9	Y	N / Y	90
in_210b	9	Y	N / Y	78
in_210c	10	Y	N / Y	90
in_210d	10	Y	N / Y	80
in_210e	10	Y	N / Y	80
in_210f	10	Y	N / Y	90
in_114	6	Y	N / N	100
in_114a	7	Y	N / N	100
in_114c	9	Y	N / N	100
in_210g	10	Y	Y / N	90
in_401a	2	Y	N / N	100
in_401b	4	N	N / N	50

TABLE 1 - continued

Input File Name	Number of regions	Closely Related Regions	Input Subject to Noise/Occlusion	% Correctly Identified Regions
in_401	18	Y	N / N	100
in_428b	18	Y	Y / N	100
in_428a	1	N	N / N	0
in_506a	18	Y	N / Y	89
in_506b	4	Y	N / N	100
in_506c	4	Y	N / N	100
in_506d	4	Y	N / N	100
in_506e	4	Y	N / N	100
in_506f	4	Y	N / Y	100
in_506g	4	Y	N / Y	75
in_506h	4	Y	N / Y	100
in_506i	4	Y	N / Y	100
in_506j	3	N	N / N	100
in_506k	2	N	N / N	100
in_506l	4	N	N / N	75
in_506m	4	N	N / N	75
in_506n	1	N	N / N	100
in_510a	3	N	N / N	67
in_510b	3	N	N / N	67
in_510c	2	N	N / N	50
in_510d	2	N	N / N	50
in_510f	4	Y	Y / N	100
in_510g	4	Y	Y / N	100
in_510h	4	Y	Y / N	100
in_510i	4	Y	Y / N	100
in_510j	18	Y	N / Y	89

TABLE 2

Summary of Results

Type of Input	Number of Files	Mean % of Correctly Identified Regions	Standard Deviation
Closely Related Regions	19	99.2	3.44
Subject to "Noise"	7	98.6	3.78
Subject to "Occlusion"	12	88.4	8.74
Unrelated Regions	12	67.8	28.46
All types	50	89.0	19.19

Appendix B

Appendix C

Guide to Using the High-Level Inference System

In order to run the high-level inference system, the following steps must be performed.

1. Invoke Prolog environment and frame package. Ensure there is enough global stack, local stack and heap to run the system. The switches are set with recommended values and accommodate the largest input file that could be processed.

```
prolog  -G 2500 -L 1500 -H 1200  frameboot
```

2. Once in Prolog environment, load the file loadall, which will load all necessary knowledge bases and files to run the system.

```
?- [loadall].
```

3. Invoke the high-level inference system.

```
?- run.
```

4. Respond to prompts for the input tool.

```
?- Enter regions in input file.
?- Enter end when done.
:-                                        { r1 through r18 are valid }
:- end.
?- Would you like to subject input to noise?
:- n.
?- Would you like to subject input to occlusion?
:- y.
:- Enter regions to subject to occlusion.
:- Enter end when done.
:-              { a subset of the regions entered above are valid }
:- end.

or

:- Would you like to subject input to noise?
:- y.                   { xinput is loaded }
yes.                    { processing is completed }
?-                      { return to Prolog level }
```

The following files are created during a session: infile, kbfile, objfile, and outfile.

Rename these files before the next session in order to maintain them, otherwise they will be overwritten during the next test run.

Appendix C

Appendix D

The contents of this Appendix include the input files and output files that were referenced in Appendix B, the master input file, infile1, the xinput file, the knowledge base file, kb1, and portions of Prolog code.

```
input_list([amorph(r1),amorph(r4),amorph(r3),amorph(r2),amorph(r15),
amorph(r14),rect_region(r18),rect_region(r17),rect_region(r10),
rect_region(r9),tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
green(r4)
brown(r14)
green(r15)
large(r6)
small(r9)
small(r14)
small(r15)
large(r5)
narrow(r18)
wide(r17)
grey(r17)
grey(r18)
grey(r10)
below(r18,r2)
below(r18,r3)
parallel_to(r18,r17)
parallel_to(r17,r18)
below(r5,r1)
below(r4,r14)
above(r9,r10)
below(r17,r10)
below(r17,r2)
below(r17,r3)
below(r17,r4)
next_to(r10,r2)
next_to(r10,r3)
next_to(r6,r2)
next_to(r2,r6)
next_to(r3,r6)
next_to(r4,r14)
above(r1,r5)
above(r1,r15)
below(r10,r9)
below(r14,r15)
above(r5,r6)
on(r9,r6)
below(r2,r1)
above(r15,r14)
next_to(r2,r6)
below(r3,r6)
below(r9,r5)
below(r6,r5)
```

```
input_list([amorph(r1),amorph(r4),amorph(r15),amorph(r14)])
blue(r1)
green(r4)
brown(r14)
green(r15)
small(r14)
small(r15)
below(r4,r14)
next_to(r4,r14)
above(r1,r15)
below(r14,r15)
above(r15,r14)
```

input_list([amorph(r2),amorph(r15),rect_region(r17),rect_region(r16),rect_region(r6)])
green(r2)
green(r15)
large(r6)
small(r15)
wide(r17)
grey(r17)
grey(r16)
below(r17,r16)
below(r17,r2)
above(r16,r17)
next_to(r6,r2)
next_to(r2,r6)
next_to(r2,r6)

```
input_list([amorph(r1),amorph(r4),amorph(r3),amorph(r2),rect_region(r18),
rect_region(r17),rect_region(r10),rect_region(r9),tri_region(r5),
rect_region(r6)])
blue(r1)
green(r2)
green(r3)
green(r4)
large(r6)
small(r9)
large(r5)
narrow(r18)
wide(r17)
grey(r17)
grey(r18)
grey(r10)
below(r18,r2)
below(r18,r3)
parallel_to(r18,r17)
parallel_to(r17,r18)
below(r5,r1)
above(r9,r10)
below(r17,r10)
below(r17,r2)
below(r17,r3)
below(r17,r4)
next_to(r10,r2)
next_to(r10,r3)
next_to(r6,r2)
next_to(r2,r6)
next_to(r3,r6)
above(r1,r5)
below(r10,r9)
above(r5,r6)
on(r9,r6)
below(r2,r1)
next_to(r2,r6)
below(r3,r6)
below(r9,r5)
below(r6,r5)
```

in_127d

input_list([amorph(r1),amorph(r4),amorph(r15),amorph(r14),rect_region(r18),rect_region(r17)])
blue(r1)
green(r4)
brown(r14)
green(r15)
small(r14)
small(r15)
narrow(r18)
wide(r17)
grey(r17)
grey(r18)
parallel_to(r18,r17)
parallel_to(r17,r18)
below(r4,r14)
below(r17,r4)
next_to(r4,r14)
above(r1,r15)
below(r14,r15)
above(r15,r14)

input_list([amorph(r1),amorph(r4),amorph(r15),amorph(r14),
sq_region(r12),rect_region(r13),tri_region(r11),rect_region(r16)])
blue(r1)
green(r4)
brown(r14)
green(r15)
large(r13)
small(r14)
small(r15)
medium(r12)
medium(r11)
grey(r16)
below(r11,r1)
next_to(r16,r4)
below(r4,r14)
below(r13,r11)
below(r16,r12)
next_to(r4,r14)
above(r1,r11)
above(r1,r15)
below(r16,r13)
below(r14,r15)
above(r11,r13)
on(r12,r13)
above(r15,r14)

```
input_list([amorph(r1),amorph(r15),amorph(r14),rect_region(r9),
sq_region(r7),tri_region(r5),rect_region(r6)])
blue(r1)
brown(r14)
green(r15)
large(r6)
small(r9)
medium(r7)
small(r14)
small(r15)
large(r5)
below(r5,r1)
next_to(r9,r7)
below(r7,r5)
above(r7,r9)
above(r1,r5)
above(r1,r15)
below(r14,r15)
above(r5,r6)
on(r7,r6)
on(r9,r6)
above(r15,r14)
below(r9,r7)
below(r9,r5)
below(r6,r5)
```

input_list([amorph(r1),rect_region(r9),sq_region(r8),sq_region(r7),tri_region(r5),rect_region(r6)])
blue(r1)
large(r6)
small(r9)
medium(r7)
medium(r8)
large(r5)
below(r5,r1)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
below(r7,r5)
below(r8,r5)
above(r7,r9)
above(r8,r9)
above(r1,r5)
above(r5,r6)
on(r7,r6)
on(r8,r6)
on(r9,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)

```
input_list([amorph(r1),amorph(r2),amorph(r15),amorph(r14),
rect_region(r10),rect_region(r9),sq_region(r8),tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
brown(r14)
green(r15)
large(r6)
small(r9)
medium(r8)
small(r14)
small(r15)
large(r5)
grey(r10)
below(r5,r1)
above(r9,r10)
next_to(r10,r2)
next_to(r9,r8)
next_to(r6,r2)
below(r8,r5)
above(r8,r9)
next_to(r2,r6)
above(r1,r5)
above(r1,r15)
below(r10,r9)
below(r14,r15)
above(r5,r6)
on(r8,r6)
on(r9,r6)
below(r2,r1)
above(r15,r14)
next_to(r2,r6)
below(r9,r8)
below(r9,r5)
below(r6,r5)
```

input_list([amorph(r1),amorph(r3),amorph(r2),rect_region(r10),
rect_region(r9),sq_region(r8),sq_region(r7),tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
large(r6)
small(r9)
medium(r7)
medium(r8)
large(r5)
grey(r10)
below(r5,r1)
above(r9,r10)
next_to(r10,r2)
next_to(r10,r3)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
next_to(r6,r2)
below(r7,r5)
below(r8,r5)
above(r7,r9)
above(r8,r9)
next_to(r2,r6)
next_to(r3,r6)
above(r1,r5)
below(r10,r9)
above(r5,r6)
on(r7,r6)
on(r8,r6)
on(r9,r6)
below(r2,r1)
next_to(r2,r6)
below(r3,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)

```
input_list([amorph(r3),amorph(r2),rect_region(r10),rect_region(r9),
sq_region(r8),rect_region(r16),sq_region(r7),tri_region(r5),rect_region(r6)])
green(r2)
green(r3)
large(r6)
small(r9)
medium(r7)
medium(r8)
large(r5)
grey(r10)
grey(r16)
next_to(r16,r3)
above(r9,r10)
next_to(r10,r2)
next_to(r10,r3)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
next_to(r6,r2)
below(r7,r5)
below(r8,r5)
above(r7,r9)
above(r8,r9)
next_to(r2,r6)
next_to(r3,r6)
below(r10,r9)
above(r5,r6)
on(r7,r6)
on(r8,r6)
on(r9,r6)
next_to(r2,r6)
below(r3,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)
```

input_list([amorph(r1),amorph(r4),amorph(r15),amorph(r14),rect_region(r18),rect_region(r17)])
blue(r1)
green(r4)
brown(r14)
green(r15)
small(r14)
small(r15)
narrow(r18)
wide(r17)
grey(r17)
grey(r18)
parallel_to(r18,r17)
parallel_to(r17,r18)
below(r4,r14)
below(r17,r4)
next_to(r4,r14)
above(r1,r15)
below(r14,r15)
above(r15,r14)

```
input_list([amorph(r1),amorph(r4),amorph(r3),amorph(r2),amorph(r15),
amorph(r14),rect_region(r9),tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
green(r4)
brown(r14)
green(r15)
large(r6)
small(r9)
small(r14)
small(r15)
large(r5)
below(r5,r1)
below(r4,r14)
next_to(r6,r2)
next_to(r2,r6)
next_to(r3,r6)
next_to(r4,r14)
above(r1,r5)
above(r1,r15)
below(r14,r15)
above(r5,r6)
on(r9,r6)
below(r2,r1)
above(r15,r14)
next_to(r2,r6)
below(r3,r6)
below(r9,r5)
below(r6,r5)
```

input_list([amorph(r1),amorph(r4),amorph(r3),amorph(r2),amorph(r15),
amorph(r14),rect_region(r9),tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
green(r4)
brown(r14)
green(r15)
large(r6)
small(r9)
small(r14)
small(r15)
large(r5)
below(r5,r1)
below(r4,r14)
next_to(r6,r2)
next_to(r2,r6)
next_to(r3,r6)
next_to(r4,r14)
above(r1,r5)
above(r1,r15)
below(r14,r15)
above(r5,r6)
on(r9,r6)
below(r2,r1)
above(r15,r14)
next_to(r2,r6)
below(r3,r6)
below(r9,r5)
below(r6,r5)

input_list([amorph(r1),amorph(r3),amorph(r2),rect_region(r10),
rect_region(r9),sq_region(r8),rect_region(r16),sq_region(r7),
tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
large(r6)
small(r9)
medium(r7)
medium(r8)
large(r5)
grey(r10)
grey(r16)
below(r5,r1)
next_to(r16,r3)
above(r9,r10)
next_to(r10,r2)
next_to(r10,r3)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
next_to(r6,r2)
below(r7,r5)
below(r8,r5)
above(r7,r9)
above(r8,r9)
next_to(r2,r6)
next_to(r3,r6)
above(r1,r5)
below(r10,r9)
above(r5,r6)
on(r7,r6)
on(r8,r6)
on(r9,r6)
below(r2,r1)
next_to(r2,r6)
below(r3,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)

input_list([amorph(r1),amorph(r3),amorph(r2),rect_region(r10),
rect_region(r9),sq_region(r8),rect_region(r16),sq_region(r7),
tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
large(r6)
small(r9)
medium(r7)
medium(r8)
large(r5)
grey(r10)
grey(r16)
below(r5,r1)
next_to(r16,r3)
above(r9,r10)
next_to(r10,r2)
next_to(r10,r3)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
next_to(r6,r2)
below(r7,r5)
below(r8,r5)
above(r7,r9)
above(r8,r9)
next_to(r2,r6)
next_to(r3,r6)
above(r1,r5)
below(r10,r9)
above(r5,r6)
on(r7,r6)
on(r8,r6)
on(r9,r6)
below(r2,r1)
next_to(r2,r6)
below(r3,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)

```
input_list([amorph(r1),amorph(r3),amorph(r2),rect_region(r10),
rect_region(r9),sq_region(r8),rect_region(r16),sq_region(r7),
tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
large(r6)
small(r9)
medium(r7)
medium(r8)
large(r5)
grey(r10)
grey(r16)
below(r5,r1)
next_to(r16,r3)
above(r9,r10)
next_to(r10,r2)
next_to(r10,r3)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
next_to(r6,r2)
below(r7,r5)
below(r8,r5)
above(r7,r9)
above(r8,r9)
next_to(r2,r6)
next_to(r3,r6)
above(r1,r5)
below(r10,r9)
above(r5,r6)
on(r7,r6)
on(r8,r6)
on(r9,r6)
below(r2,r1)
next_to(r2,r6)
below(r3,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)
```

input_list([amorph(r1),amorph(r3),amorph(r2),rect_region(r10),
rect_region(r9),sq_region(r8),rect_region(r16),sq_region(r7),
tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
large(r6)
small(r9)
medium(r7)
medium(r8)
large(r5)
grey(r10)
grey(r16)
below(r5,r1)
next_to(r16,r3)
above(r9,r10)
next_to(r10,r2)
next_to(r10,r3)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
next_to(r6,r2)
below(r7,r5)
below(r8,r5)
above(r7,r9)
above(r8,r9)
next_to(r2,r6)
next_to(r3,r6)
above(r1,r5)
below(r10,r9)
above(r5,r6)
on(r7,r6)
on(r8,r6)
on(r9,r6)
below(r2,r1)
next_to(r2,r6)
below(r3,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)

input_list([amorph(r1),amorph(r15),amorph(r14),rect_region(r9),tri_region(r5),rect_region(r6)])
blue(r1)
brown(r14)
green(r15)
large(r6)
small(r9)
small(r14)
small(r15)
large(r5)
below(r5,r1)
above(r1,r5)
above(r1,r15)
below(r14,r15)
above(r5,r6)
on(r9,r6)
above(r15,r14)
below(r9,r5)
below(r6,r5)

input_list([amorph(r1),sq_region(r12),rect_region(r13),
tri_region(r11),rect_region(r9),tri_region(r5),rect_region(r6)])
blue(r1)
large(r6)
large(r13)
small(r9)
medium(r12)
medium(r11)
large(r5)
below(r11,r1)
below(r5,r1)
below(r13,r11)
above(r1,r5)
above(r1,r11)
above(r5,r6)
above(r11,r13)
next_to(r13,r6)
next_to(r6,r13)
on(r9,r6)
on(r12,r13)
below(r9,r5)
below(r6,r5)

input_list([amorph(r1),amorph(r3),amorph(r2),rect_region(r18),
rect_region(r17),rect_region(r10),rect_region(r9),tri_region(r5),
rect_region(r6)])
blue(r1)
green(r2)
green(r3)
large(r6)
small(r9)
large(r5)
narrow(r18)
wide(r17)
grey(r17)
grey(r18)
grey(r10)
below(r18,r2)
below(r18,r3)
parallel_to(r18,r17)
parallel_to(r17,r18)
below(r5,r1)
above(r9,r10)
below(r17,r10)
below(r17,r2)
below(r17,r3)
next_to(r10,r2)
next_to(r10,r3)
next_to(r6,r2)
next_to(r2,r6)
next_to(r3,r6)
above(r1,r5)
below(r10,r9)
above(r5,r6)
on(r9,r6)
below(r2,r1)
next_to(r2,r6)
below(r3,r6)
below(r9,r5)
below(r6,r5)

input_list([amorph(r1),amorph(r3),amorph(r2),rect_region(r10),
rect_region(r9),sq_region(r8),rect_region(r16),sq_region(r7),
tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
large(r6)
small(r9)
medium(r7)
medium(r8)
large(r5)
grey(r10)
grey(r16)
below(r5,r1)
next_to(r16,r3)
above(r9,r10)
next_to(r10,r2)
next_to(r10,r3)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
next_to(r6,r2)
below(r7,r5)
below(r8,r5)
above(r7,r9)
above(r8,r9)
next_to(r2,r6)
next_to(r3,r6)
above(r1,r5)
below(r10,r9)
above(r5,r6)
on(r7,r6)
on(r8,r6)
on(r9,r6)
below(r2,r1)
next_to(r2,r6)
below(r3,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)

```
input_list([amorph(r15),amorph(r14)])
brown(r14)
green(r15)
small(r14)
small(r15)
below(r14,r15)
above(r15,r14)
```

input_list([amorph(r1),amorph(r15),rect_region(r18),sq_region(r7)])
blue(r1)
green(r15)
medium(r7)
small(r15)
narrow(r18)
grey(r18)
above(r1,r15)

input_list([amorph(r1),amorph(r4),amorph(r3),amorph(r2),amorph(r15),
amorph(r14),rect_region(r18),rect_region(r17),sq_region(r12),
rect_region(r13),tri_region(r11),rect_region(r10),rect_region(r9),
sq_region(r8),rect_region(r16),sq_region(r7),tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
green(r4)
brown(r14)
green(r15)
large(r6)
large(r13)
small(r9)
medium(r7)
medium(r8)
small(r14)
small(r15)
medium(r12)
medium(r11)
large(r5)
narrow(r18)
wide(r17)
grey(r17)
grey(r18)
grey(r10)
grey(r16)
below(r18,r2)
below(r18,r3)
parallel_to(r18,r17)
parallel_to(r17,r18)
below(r11,r1)
below(r5,r1)
next_to(r16,r3)
next_to(r16,r4)
below(r4,r14)
above(r9,r10)
below(r13,r11)
below(r17,r10)
below(r17,r16)
below(r17,r2)
below(r17,r3)
below(r17,r4)
next_to(r10,r2)
next_to(r10,r3)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
above(r16,r17)
below(r16,r12)
next_to(r6,r2)
below(r7,r5)
below(r8,r5)

above(r7,r9)
above(r8,r9)
next_to(r2,r6)
next_to(r3,r6)
next_to(r4,r14)
above(r1,r5)
above(r1,r11)
above(r1,r15)
below(r10,r9)
below(r16,r13)
below(r14,r15)
above(r5,r6)
above(r11,r13)
next_to(r13,r6)
next_to(r6,r13)
on(r7,r6)
on(r8,r6)
on(r9,r6)
on(r12,r13)
below(r2,r1)
above(r15,r14)
next_to(r2,r6)
below(r3,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)

input_list([amorph(r1),amorph(r4),amorph(r3),amorph(r2),amorph(r15),
amorph(r14),rect_region(r18),rect_region(r17),sq_region(r12),
rect_region(r13),tri_region(r11),rect_region(r10),rect_region(r9),
sq_region(r8),rect_region(r16),sq_region(r7),tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
green(r4)
brown(r14)
green(r15)
large(r6)
large(r13)
small(r9)
medium(r7)
medium(r8)
small(r14)
small(r15)
medium(r12)
medium(r11)
large(r5)
narrow(r18)
wide(r17)
grey(r17)
grey(r18)
grey(r10)
grey(r16)
below(r18,r2)
below(r18,r3)
parallel_to(r18,r17)
parallel_to(r17,r18)
below(r11,r1)
below(r5,r1)
next_to(r16,r3)
next_to(r16,r4)
below(r4,r14)
above(r9,r10)
below(r13,r11)
below(r17,r10)
below(r17,r16)
below(r17,r2)
below(r17,r3)
below(r17,r4)
next_to(r10,r2)
next_to(r10,r3)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
above(r16,r17)
below(r16,r12)
next_to(r6,r2)
below(r7,r5)
below(r8,r5)

above(r7,r9)
above(r8,r9)
next_to(r2,r6)
next_to(r3,r6)
next_to(r4,r14)
above(r1,r5)
above(r1,r11)
above(r1,r15)
below(r10,r9)
below(r16,r13)
below(r14,r15)
above(r5,r6)
above(r11,r13)
next_to(r13,r6)
next_to(r6,r13)
on(r7,r6)
on(r8,r6)
on(r9,r6)
on(r12,r13)
below(r2,r1)
above(r15,r14)
next_to(r2,r6)
below(r3,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)

```
input_list([rect_region(r6)])
large(r6)
```

```
input_list([amorph(r1),amorph(r4),amorph(r3),amorph(r2),amorph(r15),
amorph(r14),rect_region(r18),rect_region(r17),sq_region(r12),
rect_region(r13),tri_region(r11),rect_region(r10),rect_region(r9),
sq_region(r8),rect_region(r16),sq_region(r7),tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
green(r4)
brown(r14)
green(r15)
large(r6)
large(r13)
small(r9)
medium(r7)
medium(r8)
small(r14)
small(r15)
medium(r12)
medium(r11)
large(r5)
narrow(r18)
wide(r17)
grey(r17)
grey(r18)
grey(r10)
grey(r16)
below(r18,r2)
below(r18,r3)
parallel_to(r18,r17)
parallel_to(r17,r18)
below(r11,r1)
below(r5,r1)
next_to(r16,r3)
next_to(r16,r4)
below(r4,r14)
above(r9,r10)
below(r13,r11)
below(r17,r10)
below(r17,r16)
below(r17,r2)
below(r17,r3)
below(r17,r4)
next_to(r10,r2)
next_to(r10,r3)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
above(r16,r17)
below(r16,r12)
next_to(r6,r2)
below(r7,r5)
below(r8,r5)
```

above(r7,r9)
above(r8,r9)
next_to(r2,r6)
next_to(r3,r6)
next_to(r4,r14)
above(r1,r5)
above(r1,r11)
above(r1,r15)
below(r10,r9)
below(r16,r13)
below(r14,r15)
above(r5,r6)
above(r11,r13)
next_to(r13,r6)
next_to(r6,r13)
on(r7,r6)
on(r8,r6)
on(r9,r6)
on(r12,r13)
below(r2,r1)
above(r15,r14)
next_to(r2,r6)
below(r3,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)

```
input_list([amorph(r1),sq_region(r12),rect_region(r13),tri_region(r11)])
blue(r1)
large(r13)
medium(r12)
medium(r11)
below(r11,r1)
below(r13,r11)
above(r1,r11)
above(r11,r13)
on(r12,r13)
```

input_list([rect_region(r10),rect_region(r9),tri_region(r5),rect_region(r6)])
large(r6)
small(r9)
large(r5)
grey(r10)
above(r9,r10)
below(r10,r9)
above(r5,r6)
on(r9,r6)
below(r9,r5)
below(r6,r5)

input_list([amorph(r4),amorph(r15),amorph(r14),rect_region(r18)])
green(r4)
brown(r14)
green(r15)
small(r14)
small(r15)
narrow(r18)
grey(r18)
below(r4,r14)
next_to(r4,r14)
below(r14,r15)
above(r15,r14)

input_list([amorph(r4),amorph(r15),amorph(r14),rect_region(r17)])
green(r4)
brown(r14)
green(r15)
small(r14)
small(r15)
wide(r17)
grey(r17)
below(r4,r14)
below(r17,r4)
next_to(r4,r14)
below(r14,r15)
above(r15,r14)

```
input_list([amorph(r1),sq_region(r12),rect_region(r13),tri_region(r11)])
blue(r1)
large(r13)
medium(r12)
medium(r11)
below(r11,r1)
below(r13,r11)
above(r1,r11)
above(r11,r13)
on(r12,r13)
```

input_list([rect_region(r10),rect_region(r9),tri_region(r5),rect_region(r6)])
large(r6)
small(r9)
large(r5)
grey(r10)
above(r9,r10)
below(r10,r9)
above(r5,r6)
on(r9,r6)
below(r9,r5)
below(r6,r5)

```
input_list([amorph(r4),amorph(r15),amorph(r14),rect_region(r18)])
green(r4)
brown(r14)
green(r15)
small(r14)
small(r15)
narrow(r18)
grey(r18).
below(r4,r14)
next_to(r4,r14)
below(r14,r15)
above(r15,r14)
```

input_list([amorph(r4),amorph(r15),amorph(r14),rect_region(r17)])
green(r4)
brown(r14)
green(r15)
small(r14)
small(r15)
wide(r17)
grey(r17)
below(r4,r14)
below(r17,r4)
next_to(r4,r14)
below(r14,r15)
above(r15,r14)

```
input_list([amorph(r1),amorph(r15),rect_region(r17)])
blue(r1)
green(r15)
small(r15)
wide(r17)
grey(r17)
above(r1,r15)
```

```
input_list([amorph(r1),amorph(r2)])
blue(r1)
green(r2)
below(r2,r1)
```

input_list([amorph(r15),rect_region(r18),sq_region(r12),rect_region(r9)])
green(r15)
small(r9)
small(r15)
medium(r12)
narrow(r18)
grey(r18)

input_list([amorph(r3),amorph(r14),tri_region(r11),tri_region(r5)])
green(r3)
brown(r14)
small(r14)
medium(r11)
large(r5)

```
input_list([amorph(r1)])
blue(r1)
```

```
input_list([amorph(r1),rect_region(r17),sq_region(r8)])
blue(r1)
medium(r8)
wide(r17)
grey(r17)
```

```
input_list([amorph(r3),amorph(r14),rect_region(r10)])
green(r3)
brown(r14)
small(r14)
grey(r10)
next_to(r10,r3)
```

```
input_list([rect_region(r18),rect_region(r6)])
large(r6)
narrow(r18)
grey(r18)
```

```
input_list([rect_region(r17),rect_region(r6)])
large(r6)
wide(r17)
grey(r17)
```

```
input_list([amorph(r1),sq_region(r12),rect_region(r13),tri_region(r11)])
blue(r1)
large(r13)
medium(r12)
medium(r11)
below(r11,r1)
below(r13,r11)
above(r1,r11)
above(r11,r13)
on(r12,r13)
```

input_list([rect_region(r10),rect_region(r9),tri_region(r5),rect_region(r6)])
large(r6)
small(r9)
large(r5)
grey(r10)
above(r9,r10)
below(r10,r9)
above(r5,r6)
on(r9,r6)
below(r9,r5)
below(r6,r5)

input_list([amorph(r4),amorph(r15),amorph(r14),rect_region(r18)])
green(r4)
brown(r14)
green(r15)
small(r14)
small(r15)
narrow(r18)
grey(r18)
below(r4,r14)
next_to(r4,r14)
below(r14,r15)
above(r15,r14)

```
input_list([amorph(r4),amorph(r15),amorph(r14),rect_region(r17)])
green(r4)
brown(r14)
green(r15)
small(r14)
small(r15)
wide(r17)
grey(r17)
below(r4,r14)
below(r17,r4)
next_to(r4,r14)
below(r14,r15)
above(r15,r14)
```

input_list([amorph(r1),amorph(r4),amorph(r3),amorph(r2),amorph(r15),
amorph(r14),rect_region(r18),rect_region(r17),sq_region(r12),
rect_region(r13),tri_region(r11),rect_region(r10),rect_region(r9),
sq_region(r8),rect_region(r16),sq_region(r7),tri_region(r5),rect_region(r6)])
blue(r1)
green(r2)
green(r3)
green(r4)
brown(r14)
green(r15)
large(r6)
large(r13)
small(r9)
medium(r7)
medium(r8)
small(r14)
small(r15)
medium(r12)
medium(r11)
large(r5)
narrow(r18)
wide(r17)
grey(r17)
grey(r18)
grey(r10)
grey(r16)
below(r18,r2)
below(r18,r3)
parallel_to(r18,r17)
parallel_to(r17,r18)
below(r11,r1)
below(r5,r1)
next_to(r16,r3)
next_to(r16,r4)
below(r4,r14)
above(r9,r10)
below(r13,r11)
below(r17,r10)
below(r17,r16)
below(r17,r2)
below(r17,r3)
below(r17,r4)
next_to(r10,r2)
next_to(r10,r3)
next_to(r9,r7)
next_to(r9,r8)
next_to(r7,r8)
next_to(r8,r7)
above(r16,r17)
below(r16,r12)
next_to(r6,r2)
below(r7,r5)
below(r8,r5)

above(r7,r9)
above(r8,r9)
next_to(r2,r6)
next_to(r3,r6)
next_to(r4,r14)
above(r1,r5)
above(r1,r11)
above(r1,r15)
below(r10,r9)
below(r16,r13)
below(r14,r15)
above(r5,r6)
above(r11,r13)
next_to(r13,r6)
next_to(r6,r13)
on(r7,r6)
on(r8,r6)
on(r9,r6)
on(r12,r13)
below(r2,r1)
above(r15,r14)
next_to(r2,r6)
below(r3,r6)
below(r9,r8)
below(r9,r7)
below(r9,r5)
below(r6,r5)

Final interpretation of input is

sidewalk

house_scene
  which is composed of
    tree
    walk_way
    road
    grass
    grass
    grass
    house

tree
  which is composed of
    crown
    trunk

crown

trunk

walk_way

road

grass

grass

grass

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door

house_door

house_roof

sky

NO ERRORS TO REPORT.

Final interpretation of input is

house_scene
  which is composed of
    tree
    grass

tree
  which is composed of
    crown
    trunk

crown

trunk

grass

sky

NO ERRORS TO REPORT.

Final interpretation of input is

house_scene
  which is composed of
    driveway
    road
    grass

driveway

road

grass

NO ERRORS TO REPORT.

Final interpretation of input is

sidewalk

house_scene
  which is composed of
    walk_way
    road
    grass
    grass
    grass
    house

walk_way

road

grass

grass

grass

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door

house_door

house_roof

sky

r4 - unable to interpret accurately due to lack of available information

Final interpretation of input is

sidewalk

house_scene
  which is composed of
    tree
    road
    grass

tree
  which is composed of
    crown
    trunk

crown

trunk

road

grass

sky

NO ERRORS TO REPORT.

Final interpretation of input is

sky

house_scene
  which is composed of
    tree
    driveway
    grass
    garage

tree
  which is composed of
    crown
    trunk

crown

trunk

driveway

grass

garage
  which is composed of
    gar_roof
    gar_wall

gar_roof

gar_wall
  which is composed of
    gar_door

gar_door

NO ERRORS TO REPORT.

Final interpretation of input is

sky

house_scene
  which is composed of
    tree
    house

tree
  which is composed of
    crown
    trunk

crown

trunk

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door

house_door

house_roof

NO ERRORS TO REPORT.

Final interpretation of input is

sky

house_scene
  which is composed of
    house

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door
    house_wall_window
    house_wall_window

house_door

house_wall_window

house_wall_window

house_roof

NO ERRORS TO REPORT.

Final interpretation of input is

sky

house_scene
  which is composed of
    tree
    walk_way
    grass
    house

tree
  which is composed of
    crown
    trunk

crown

trunk

walk_way

grass

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door
    house_wall_window

house_door

house_wall_window

house_roof

NO ERRORS TO REPORT.

Final interpretation of input is

sky

house_scene
  which is composed of
    walk_way
    grass
    grass
    house

walk_way

grass

grass

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door
    house_wall_window
    house_wall_window

house_door

house_wall_window

house_wall_window

house_roof

NO ERRORS TO REPORT.

Final interpretation of input is

house_scene
 which is composed of
  walk_way
  grass
  grass
  house

walk_way

grass

grass

house
 which is composed of
  house_wall
  house_roof

house_wall
 which is composed of
  house_door
  house_wall_window
  house_wall_window

house_door

house_wall_window

house_wall_window

house_roof

r2 - unable to interpret accurately due to lack of available information .
r16 - unable to interpret accurately due to lack of available information .

Final interpretation of input is

sidewalk

house_scene
 which is composed of
  tree
  road
  grass
  grass

tree
 which is composed of
  crown
  trunk

crown

trunk

road

grass

grass

sky

r99 - unable to interpret accurately due to lack of available information
r87 - unable to interpret accurately due to excessive noise .
r53 - unable to interpret accurately due to excessive noise .
r41 - unable to interpret accurately due to excessive noise .

Final interpretation of input is

sky

house_scene
  which is composed of
    tree
    grass
    grass
    house

tree
  which is composed of
    crown
    trunk

crown

trunk

grass

grass

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door

house_door

house_roof

r2 - unable to interpret accurately due to lack of available information

Final interpretation of input is

sky

house_scene
 which is composed of
  tree
  grass
  grass
  house

tree
 which is composed of
  crown
  trunk

crown

trunk

grass

grass

house
 which is composed of
  house_wall
  house_roof

house_wall
 which is composed of
  house_door
  shutters

house_door

shutters

house_roof

r5 - unable to interpret accurately due to lack of available information .
r4 - unable to interpret accurately due to lack of available information .

Final interpretation of input is

sky

house_scene
 which is composed of
  walk_way
  driveway
  grass
  grass
  grass
  house

walk_way

driveway

grass

grass

grass

house
 which is composed of
  house_wall
  house_roof

house_wall
 which is composed of
  house_door
  house_wall_window
  house_wall_window

house_door

house_wall_window

house_wall_window

house_roof

r9 - unable to interpret accurately due to lack of available information .
r5 - unable to interpret accurately due to lack of available information .
r16 - unable to interpret accurately due to lack of available information

Final interpretation of input is

sky

house_scene
 which is composed of
  walk_way
  grass
  grass
  house

walk_way

grass

grass

house
 which is composed of
  house_wall
  house_roof

house_wall
 which is composed of
  house_door
  house_wall_window

house_door

house_wall_window

house_roof

r8 - unable to interpret accurately due to lack of available information .
r2 - unable to interpret accurately due to lack of available information .
r16 - unable to interpret accurately due to lack of available information
r7 - unable to interpret accurately due to lack of available information .

Final interpretation of input is

house_scene
 which is composed of
  walk_way
  grass
  grass
  house

walk_way

grass

grass

house
 which is composed of
  house_wall
  house_roof

house_wall
 which is composed of
  house_door
  house_wall_window
  house_wall_window

house_door

house_wall_window

house_wall_window

house_roof

sky

r3 - unable to interpret accurately due to lack of available information .
r16 - unable to interpret accurately due to lack of available information
r6 - unable to interpret accurately due to lack of available information .

Final interpretation of input is

sky

house_scene
 which is composed of
  walk_way
  grass
  grass
  house

walk_way

grass

grass

house
 which is composed of
  house_wall
  house_roof

house_wall
 which is composed of
  house_door
  house_wall_window
  house_wall_window

house_door

house_wall_window

house_wall_window

house_roof

r16 - unable to interpret accurately due to lack of available information

Final interpretation of input is

sky

house_scene
 which is composed of
  tree
  house

tree
 which is composed of
  crown
  trunk

crown

trunk

house
 which is composed of
  house_wall
  house_roof

house_wall
 which is composed of
  house_door

house_door

house_roof

NO ERRORS TO REPORT.

Final interpretation of input is

sky

house_scene
  which is composed of
    garage
    house

garage
  which is composed of
    gar_roof
    gar_wall

gar_roof

gar_wall
  which is composed of
    gar_door

gar_door

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door

house_door

house_roof

NO ERRORS TO REPORT.

Final interpretation of input is

house_scene
 which is composed of
  walk_way
  road
  grass
  grass
  house

walk_way

road

grass

grass

house
 which is composed of
  house_wall
  house_roof

house_wall
 which is composed of
  house_door

house_door

house_roof

sidewalk

sky

NO ERRORS TO REPORT.

Final interpretation of input is

sky

house_scene
  which is composed of
    walk_way
    grass
    grass
    grass
    house

walk_way

grass

grass

grass

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door
    house_wall_window
    house_wall_window

house_door

house_wall_window

house_wall_window

house_roof

r16 - unable to interpret accurately due to excessive noise .
r87 - unable to interpret accurately due to excessive noise .
r99 - unable to interpret accurately due to excessive noise .
r53 - unable to interpret accurately due to excessive noise .
r62 - unable to interpret accurately due to excessive noise .

Final interpretation of input is

NO ERRORS TO REPORT.

Final interpretation of input is

sidewalk

sky

NO ERRORS TO REPORT.

Final interpretation of input is

sky

sidewalk

house_scene
 which is composed of
  tree
  walk_way
  driveway
  road
  grass
  grass
  grass
  garage
  house

tree
 which is composed of
  crown
  trunk

crown

trunk

walk_way

driveway

road

grass

grass

grass

garage
 which is composed of
  gar_roof
  gar_wall

gar_roof

gar_wall
 which is composed of
  gar_door

gar_door

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door
    house_wall_window
    house_wall_window

house_door

house_wall_window

house_wall_window

house_roof

NO ERRORS TO REPORT.

Final interpretation of input is

sky

sidewalk

house_scene
  which is composed of
    tree
    walk_way
    driveway
    road
    grass
    grass
    grass
    garage
    house

tree
  which is composed of
    crown
    trunk

crown

trunk

walk_way

driveway

road

grass

grass

grass

garage
  which is composed of
    gar_roof
    gar_wall

gar_roof

gar_wall
  which is composed of
    gar_door

gar_door

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door
    house_wall_window
    house_wall_window

house_door

house_wall_window

house_wall_window

house_roof

r41 - unable to interpret accurately due to excessive noise .
r62 - unable to interpret accurately due to excessive noise .
r99 - unable to interpret accurately due to excessive noise .
r87 - unable to interpret accurately due to excessive noise .
r87 - unable to interpret accurately due to lack of available information
r99 - unable to interpret accurately due to lack of available information
r53 - unable to interpret accurately due to lack of available information
r62 - unable to interpret accurately due to lack of available information
r41 - unable to interpret accurately due to lack of available information

Final interpretation of input is

r6 - unable to interpret accurately due to lack of available information

Final interpretation of input is

sky

sidewalk

house_scene
  which is composed of
    tree
    walk_way
    driveway
    road
    grass
    garage
    house

tree
  which is composed of
    crown
    trunk

crown

trunk

walk_way

driveway

road

grass

garage
  which is composed of
    gar_roof
    gar_wall

gar_roof

gar_wall
  which is composed of
    gar_door

gar_door

house
  which is composed of
    house_wall
    house_roof

house_wall

which is composed of
house_door
house_wall_window
house_wall_window

house_door

house_wall_window

house_wall_window

house_roof

r11 - unable to interpret accurately due to lack of available information
r4 - unable to interpret accurately due to lack of available information .
r3 - unable to interpret accurately due to lack of available information .
r13 - unable to interpret accurately due to lack of available information
r6 - unable to interpret accurately due to lack of available information .

Final interpretation of input is

sky

NO ERRORS TO REPORT.

Final interpretation of input is

house_scene
  which is composed of
    walk_way
    house

walk_way

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door

house_door

house_roof

NO ERRORS TO REPORT.

Final interpretation of input is

sidewalk

house_scene
  which is composed of
    tree
    grass

tree
  which is composed of
    crown
    trunk

crown

trunk

grass

NO ERRORS TO REPORT.

Final interpretation of input is

house_scene
 which is composed of
  tree
  road
  grass

tree
 which is composed of
  crown
  trunk

crown

trunk

road

grass

NO ERRORS TO REPORT.

Final interpretation of input is

sky

NO ERRORS TO REPORT.

Final interpretation of input is

house_scene
  which is composed of
    walk_way
    house

walk_way

house
  which is composed of
    house_wall
    house_roof

house_wall
  which is composed of
    house_door

house_door

house_roof

r10 - unable to interpret accurately due to lack of available information .

Final interpretation of input is

sidewalk

house_scene
  which is composed of
    tree
    grass

tree
  which is composed of
    crown
    trunk

crown

trunk

grass

NO ERRORS TO REPORT.

Final interpretation of input is

house_scene
 which is composed of
  tree
  road
  grass

tree
 which is composed of
  crown
  trunk

crown

trunk

road

grass

r15 - unable to interpret accurately due to lack of available information
r4 - unable to interpret accurately due to lack of available information .

Final interpretation of input is

sky

NO ERRORS TO REPORT.

Final interpretation of input is

house_scene
  which is composed of
    grass

grass

sky

NO ERRORS TO REPORT.

Final interpretation of input is

sidewalk

NO ERRORS TO REPORT.

Final interpretation of input is

house_scene
 which is composed of
  tree
  grass
  garage
  house

tree
 which is composed of
  crown
  trunk

crown

trunk

grass

garage
 which is composed of
  gar_roof
  gar_wall

gar_roof

gar_wall
 which is composed of
  gar_door

gar_door

house
 which is composed of
  house_wall
  house_roof

house_wall
 which is composed of
  house_door

house_door

house_roof

r5 - unable to interpret accurately due to lack of available information .
r11 - unable to interpret accurately due to lack of available information
r3 - unable to interpret accurately due to lack of available information .

Final interpretation of input is

sky

NO ERRORS TO REPORT.

Final interpretation of input is

sky

NO ERRORS TO REPORT.

Final interpretation of input is

house_scene
  which is composed of
    tree
    walk_way
    grass

tree
  which is composed of
    crown
    trunk

crown

trunk

walk_way

grass

r3 - unable to interpret accurately due to lack of available information

Final interpretation of input is

sidewalk

r6 - unable to interpret accurately due to lack of available information

Final interpretation of input is

r6 - unable to interpret accurately due to lack of available information

Final interpretation of input is

sky

house_scene
 which is composed of
  grass
  grass
  garage

grass

grass

garage
 which is composed of
  gar_roof
  gar_wall

gar_roof

gar_wall
 which is composed of
  gar_door

gar_door

r13 - unable to interpret accurately due to lack of available information .
r12 - unable to interpret accurately due to lack of available information .
r87 - unable to interpret accurately due to excessive noise .
r53 - unable to interpret accurately due to excessive noise .
r62 - unable to interpret accurately due to excessive noise .

Final interpretation of input is

house_scene
  which is composed of
    walk_way

walk_way

r99 - unable to interpret accurately due to lack of available information
r6 - unable to interpret accurately due to lack of available information .
r87 - unable to interpret accurately due to excessive noise .
r53 - unable to interpret accurately due to excessive noise .

Final interpretation of input is

sidewalk

house_scene
 which is composed of
  tree
  grass

tree
 which is composed of
  crown
  trunk

crown

trunk

grass

r99 - unable to interpret accurately due to lack of available information
r87 - unable to interpret accurately due to excessive noise .
r53 - unable to interpret accurately due to excessive noise .
r41 - unable to interpret accurately due to excessive noise .

Final interpretation of input is

house_scene
 which is composed of
  tree
  road
  grass

tree
 which is composed of
  crown
  trunk

crown

trunk

road

grass

r99 - unable to interpret accurately due to lack of available information .
r87 - unable to interpret accurately due to excessive noise .
r53 - unable to interpret accurately due to excessive noise .
r41 - unable to interpret accurately due to excessive noise .

Final interpretation of input is

sky

sidewalk

house_scene
 which is composed of
  tree
  walk_way
  driveway
  road
  grass
  grass
  garage
  house

tree
 which is composed of
  crown
  trunk

crown

trunk

walk_way

driveway

road

grass

grass

garage
 which is composed of
  gar_roof
  gar_wall

gar_roof

gar_wall
 which is composed of
  gar_door

gar_door

house
 which is composed of
  house_wall

house_roof

house_wall
which is composed of
house_door
house_wall_window

house_door

house_wall_window

house_roof

r3 - unable to interpret accurately due to lack of available information
r8 - unable to interpret accurately due to lack of available information
r7 - unable to interpret accurately due to lack of available information

```
input_list(
 [rect_region(r6),
  tri_region(r5),
  sq_region(r7),
  rect_region(r16),
  sq_region(r8),
  rect_region(r9),
  rect_region(r10),
  tri_region(r11),
  rect_region(r13),
  sq_region(r12),
  rect_region(r17),
  rect_region(r18),
  amorph(r14),
  amorph(r15),
  amorph(r2),
  amorph(r3),
  amorph(r4),
  amorph(r1)
 ]).


blue(r1).
green(r2).
green(r3).
green(r4).
brown(r14).
green(r15).
large(r6).
large(r13).
small(r9).
medium(r7).
medium(r8).
small(r14).
small(r15).
medium(r12).
medium(r11).
large(r5).
narrow(r18).
wide(r17).
grey(r17).
grey(r18).
grey(r10).
grey(r16).

below(r18, r2).
below(r18, r3).
parallel_to(r18, r17).
parallel_to(r17, r18).

below(r11, r1).
below(r5, r1).
next_to(r16, r3).
```

```
next_to(r16, r4).
below(r4, r14).
above(r9,r10).

below(r13, r11).
below(r17, r10).
below(r17, r16).
below(r17, r2).
below(r17, r3).
below(r17, r4).
next_to(r10, r2).
next_to(r10, r3).
next_to(r9, r7).
next_to(r9, r8).
next_to(r7, r8).
next_to(r8, r7).
above(r16, r17).
below(r16, r12).
next_to(r6, r2).

below(r7, r5).
below(r8, r5).
above(r7, r9).
above(r8, r9).
next_to(r2, r6).
next_to(r3, r6).
next_to(r4, r14).

above(r1, r5).
above(r1, r11).
above(r1, r15).
below(r10, r9).
below(r16, r13).
below(r14, r15).
above(r5,r6).
above(r11,r13).
next_to(r13,r6).
next_to(r6,r13).
on(r7,r6).
on(r8,r6).
on(r9,r6).
on(r12,r13).
below(r2,r1).
above(r15,r14).
next_to(r2,r6).
below(r3,r6).
below(r9,r8).
below(r9,r7).
below(r9,r5).
below(r6,r5).
next_to(r13, r88).
next_to(r77, r13).
```

```
xinput_list([tri_region(r87),
             rect_region(r99),
             circ_region(r53),
             sq_region(r62),
             amorph(r41)
            ]).

above(r87,r10).
on(r87,r6).
on(r87,r5).
on(r87,r1).
next_to(r87,r8).
next_to(r8,r87).
below(r10, r87).

on(r41,r2).
on(r41,r6).
on(r41,r18).
above(r41,r17).
below(r41,r1).

on(r62,r11).
on(r62,r13).
on(r62,r1).
on(r62,r15).
on(r62,r14).
on(r62,r4).
below(r4,r62).

on(r99,r16).
on(r99,r3).
on(r99,r17).
next_to(r99,r18).
below(r99,r13).

on(r53,r18).
on(r53,r10).
above(r53,r17).
on(r53,r3).
on(r53,r2).
next_to(r53,r16).
next_to(r16,r53).
below(r53,r1).

next_to(r53,r41).
above(r6,r53).
above(r1,r99).
below(r99,r1).

grey(r87).
brown(r41).
large(r53).
large(r99).
```

```
large(r53).
medium(r41).
large(r62).
red(r99).
```

```
?- def_frame(very_small:
        [is_a:
            [value:[size]]
        ]).

?- def_frame(small:
        [is_a:
            [value:[size]]
        ]).

?- def_frame(medium:
        [is_a:
            [value:[size]]
        ]).

?- def_frame(large:
        [is_a:
            [value:[size]]
        ]).

?- def_frame(noise:
        []).

?- def_frame(region:
        []).

?- def_frame(tri_region:
        [is_a:
            [value:[region],
            min:1,
            max:1]
        ]).

?- def_frame(rect_region:
        [is_a:
            [value:[region],
            min:1,
            max:1]
        ]).

?- def_frame(circ_region:
        [is_a:
            [value:[region],
            min:1,
            max:1]
        ]).

?- def_frame(sq_region:
        [is_a:
            [value:[region],
            min:1,
            max:1]
        ]).
```

```
?- def_frame(amorph:
        [is_a:
            [value:[region],
            min:1,
            max:1]
        ]).

?- def_frame(blue:
        [is_a:
            [value:[color]]
        ]).

?- def_frame(yellow:
        [is_a:
            [value:[color]]
        ]).

?- def_frame(green:
        [is_a:
            [value:[color]]
        ]).

?- def_frame(white:
        [is_a:
            [value:[color]]
        ]).

?- def_frame(black:
        [is_a:
            [value:[color]]
        ]).

?- def_frame(red:
        [is_a:
            [value:[color]]
        ]).

?- def_frame(brown:
        [is_a:
            [value:[color]]
        ]).

?- def_frame(grey:
        [is_a:
            [value:[color]]
        ]).

?- def_frame(house_scene:
        [is_a:
            [value:[]],
        components:
            [value:[house,garage,grass,road,car,driveway,walk_way,tree],
            min:0,
```

```
        max:99],
    part_of:
        [value:[],
        min:0,
        max:0],
    is_a:
        [value:[]],
    members:
        [value:[],
        min:0,
        max:0],
    status:
        [value:[0],
        min:0,
        max:1],
    in_of:
        [value:[]],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(house:
    [is_a:
        [value:[]],
    components:
        [value:[house_roof,house_wall],
        min:0,
        max:99],
    part_of:
        [value:[house_scene],
        min:1,
        max:99],
    members:
        [value:[]],
    status:
        [value:[0],
        min:0,
        max:1],
    in_of:
        [value:[]],
    relations:
        [value:[next_to],
        min:1,
        max:99],
    next_to:
        [value:[garage,tree],
        min:1,
```

```
        max:99],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(wall:
    [is_a:
        [value:[rect_region,sq_region],
        min:0,
        max:2,
        type:rect_region@@sq_region],
    members:
        [value:[house_wall,gar_wall],
        min:2,
        max:99],
    in_of:
        [value:[]]
    ]).

?- def_frame(house_wall:
    [is_a:
        [value:[wall],
        min:1,
        max:1],
    components:
        [value:[house_wall_window,shutters,house_door],
        min:0,
        max:99],
    members:
        [value:[],
        min:0,
        max:0],
    status:
        [value:[0],
        min:0,
        max:1],
    in_of:
        [value:[]],
    part_of:
        [value:[house],
        min:1,
        max:99],
    relations:
        [value:[next_to,below],
        min:2,
        max:99],
    attributes:
```

```
            [value:[size],
            min:0,
            max:99],
        next_to:
            [value:[house_wall,grass,gar_wall],
            min:0,
            max:99],
        below:
            [value:[house_roof],
            min:0,
            max:99],
        size:
            [value:[large],
            min:0,
            max:1,
            type:small@@medium@@large],
        strength:
            [value:[],
            min:0,
            max:1],
        message:
            [value:[],
            if_added:recognize_object,
            if_changed:recognize_object]
        ]).

?- def_frame(gar_wall:
        [is_a:
            [value:[wall],
            min:1,
            max:1],
        components:
            [value:[gar_door],
            min:0,
            max:99],
        members:
            [value:[],
            min:0,
            max:0],
        status:
            [value:[0],
            min:0,
            max:1],
        in_of:
            [value:[]],
        part_of:
            [value:[garage],
            min:1,
            max:99],
        relations:
            [value:[next_to,below],
            min:2,
            max:99],
```

```
        attributes:
            [value:[size],
            min:1,
            max:99],
        next_to:
            [value:[house_wall],
            min:0,
            max:99],
        below:
            [value:[gar_roof],
            min:0,
            max:99],
        size:
            [value:[large],
            min:0,
            max:1,
            type:small@@medium@@large],
        strength:
            [value:[],
            min:0,
            max:1],
        message:
            [value:[],
            if_added:recognize_object,
            if_changed:recognize_object]
        ]).

?- def_frame(shutters:
        [is_a:
            [value:[rect_region],
            min:1,
            max:99],
        components:
            [value:[],
            min:0,
            max:0],
        members:
            [value:[],
            min:0,
            max:0],
        status:
            [value:[0],
            min:0,
            max:1],
        in_of:
            [value:[]],
        part_of:
            [value:[house_wall],
            min:1,
            max:99],
        relations:
            [value:[right_of,left_of,on],
            min:1,
```

```
        max:99],
    right_of:
        [value:[house_wall_window],
        min:0,
        max:99],
    left_of:
        [value:[house_wall_window],
        min:0,
        max:99],
    on:
        [value:[house_wall],
        min:0,
        max:99],
    attributes:
        [value:[size],
        min:1,
        max:99],
    size:
        [value:[small],
        min:0,
        max:1,
        type:small@@medium@@large],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(roof:
    [is_a:
        [value:[tri_region],
        min:1,
        max:99],
    members:
        [value:[house_roof,gar_roof],
        min:2,
        max:99],
    in_of:
        [value:[]],
    components:
        [value:[],
        min:0,
        max:0],
    part_of:
        [value:[],
        min:0,
        max:0],
    relations:
        [value:[above],
```

```
            min:1,
            max:99],
        attributes:
            [value:[size],
            min:1,
            max:99],
        size:
            [value:[medium],
            min:0,
            max:99,
            type:small@@medium@@large]
        ]).

?- def_frame(gar_roof:
        [is_a:
            [value:[roof],
            min:1,
            max:99],
        components:
            [value:[],
            min:0,
            max:0],
        members:
            [value:[],
            min:0,
            max:0],
        status:
            [value:[0],
            min:0,
            max:1],
        in_of:
            [value:[]],
        part_of:
            [value:[garage],
            min:1,
            max:99],
        relations:
            [value:[above,below],
            min:0,
            max:99],
        attributes:
            [value:[size],
            min:0,
            max:99],
        above:
            [value:[gar_wall],
            min:0,
            max:99],
        below:
            [value:[sky],
            min:0,
            max:99],
        size:
```

```
            [value:[medium],
            min:0,
            max:1],
        strength:
            [value:[],
            min:0,
            max:1],
        message:
            [value:[],
            if_added:recognize_object,
            if_changed:recognize_object]
        ]).

?- def_frame(house_roof:
        [is_a:
            [value:[roof],
            min:1,
            max:99],
        components:
            [value:[],
            min:0,
            max:0],
        members:
            [value:[],
            min:0,
            max:0],
        status:
            [value:[0],
            min:0,
            max:1],
        in_of:
            [value:[]],
        part_of:
            [value:[house],
            min:1,
            max:99],
        relations:
            [value:[above,below],
            min:0,
            max:99],
        attributes:
            [value:[size],
            min:0,
            max:1],
        above:
            [value:[house_wall],
            min:0,
            max:99],
        below:
            [value:[sky],
            min:0,
            max:99],
        size:
```

```
            [value:[large],
            min:0,
            max:99],
        strength:
            [value:[],
            min:0,
            max:1],
        message:
            [value:[],
            if_added:recognize_object,
            if_changed:recognize_object]
        ]).

?- def_frame(door:
        [is_a:
            [value:[rect_region,sq_region],
            min:1,
            max:2,
            type:rect_region@@sq_region],
        members:
            [value:[house_door,gar_door,car_door],
            min:0,
            max:99],
        components:
            [value:[],
            min:0,
            max:0],
        part_of:
            [value:[],
            min:0,
            max:0],
        in_of:
            [value:[]],
        relations:
            [value:[next_to],
            min:1,
            max:99],
        attributes:
            [value:[size],
            min:1,
            max:99]
        ]).

?- def_frame(gar_door:
        [is_a:
            [value:[door],
            min:1,
            max:99],
        part_of:
            [value:[gar_wall],
            min:1,
            max:99],
        components:
```

```
        [value:[gar_window],
        min:0,
        max:99],
    members:
        [value:[],
        min:0,
        max:0],
    status:
        [value:[0],
        min:0,
        max:1],
    in_of:
        [value:[]],
    relations:
        [value:[on],
        min:1,
        max:99],
    attributes:
        [value:[size],
        min:1,
        max:99],
    on:
        [value:[gar_wall],
        min:0,
        max:99],
    size:
        [value:[medium],
        min:0,
        max:1,
        type:small@@medium],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(house_door:
    [is_a:
        [value:[door],
        min:1,
        max:99],
    part_of:
        [value:[house_wall],
        min:1,
        max:99],
    components:
        [value:[house_door_window],
        min:0,
        max:99],
```

```
        members:
            [value:[],
            min:0,
            max:0],
        status:
            [value:[0],
            min:0,
            max:1],
        in_of:
            [value:[]],
        relations:
            [value:[next_to,on,above,below],
            min:1,
            max:99],
        attributes:
            [value:[size],
            min:1,
            max:99],
        next_to:
            [value:[house_wall_window],
            min:0,
            max:99],
        above:
            [value:[walk_way],
            min:0,
            max:99],
        on:
            [value:[house_wall],
            min:0,
            max:99],
        below:
            [value:[house_roof],
            min:0,
            max:99],
        size:
            [value:[small],
            min:0,
            max:1,
            type:small@@medium],
        strength:
            [value:[],
            min:0,
            max:1],
        message:
            [value:[],
            if_added:recognize_object,
            if_changed:recognize_object]
        ]).

?- def_frame(car_door:
        [is_a:
            [value:[door],
            min:1,
```

```
        max:99],
    components:
        [value:[],
        min:0,
        max:0],
    part_of:
        [value:[car],
        min:1,
        max:99],
    members:
        [value:[],
        min:0,
        max:0],
    status:
        [value:[0],
        min:0,
        max:1],
    in_of:
        [value:[]],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(sky:
    [is_a:
        [value:[amorph],
        min:1,
        max:99],
    members:
        [value:[],
        min:0,
        max:0],
    status:
        [value:[0],
        min:0,
        max:1],
    components:
        [value:[],
        min:0,
        max:0],
    part_of:
        [value:[],
        min:0,
        max:0],
    in_of:
        [value:[]],
    relations:
```

```
        [value:[above],
        min:1,
        max:99],
    attributes:
        [value:[color],
        min:1,
        max:99],
    above:
        [value:[house_roof,gar_roof,crown],
        min:0,
        max:99],
    color:
        [value:[blue],
        min:0,
        max:1],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(window:
    [is_a:
        [value:[rect_region,sq_region],
        min:1,
        max:99],
    components:
        [value:[],
        min:0,
        max:0],
    part_of:
        [value:[],
        min:0,
        max:0],
    members:
        [value:
            [house_wall_window,house_door_window,gar_window,car_window],
        min:0,
        max:0],
    in_of:
        [value:[]]
    ]).

?- def_frame(house_wall_window:
    [is_a:
        [value:[window],
        min:1,
        max:99],
    components:
```

```
    [value:[],
    min:0,
    max:0],
part_of:
    [value:[house_wall],
    min:0,
    max:99],
members:
    [value:[],
    min:0,
    max:0],
status:
    [value:[0],
    min:0,
    max:1],
in_of:
    [value:[]],
relations:
    [value:[below,on,above,next_to],
    min:1,
    max:99],
attributes:
    [value:[size],
    min:1,
    max:99],
below:
    [value:[house_roof],
    min:0,
    max:99],
on:
    [value:[house_wall],
    min:0,
    max:99],
above:
    [value:[house_door],
    min:0,
    max:99],
next_to:
    [value:[house_wall_window,shutters],
    min:0,
    max:99],
size:
    [value:[medium],
    min:0,
    max:1,
    type:small@@medium@@large],
strength:
    [value:[],
    min:0,
    max:1],
message:
    [value:[],
    if_added:recognize_object,
```

```
            if_changed:recognize_object]
        ]).

?- def_frame(house_door_window:
        [is_a:
            [value:[window],
            min:1,
            max:99],
        components:
            [value:[],
            min:0,
            max:0],
        part_of:
            [value:[house_door],
            min:0,
            max:99],
        members:
            [value:[],
            min:0,
            max:0],
        status:
            [value:[0],
            min:0,
            max:1],
        in_of:
            [value:[]],
        relations:
            [value:[below,on],
            min:1,
            max:99],
        attributes:
            [value:[size],
            min:1,
            max:99],
        below:
            [value:[house_roof],
            min:0,
            max:99],
        on:
            [value:[house_door],
            min:0,
            max:99],
        size:
            [value:[very_small],
            min:0,
            max:1,
            type:small@@medium@@large],
        strength:
            [value:[],
            min:0,
            max:1],
        message:
            [value:[],
```

```
            if_added:recognize_object,
            if_changed:recognize_object]
        ]).

?- def_frame(gar_window:
        [is_a:
            [value:[window],
            min:1,
            max:99],
        components:
            [value:[],
            min:0,
            max:0],
        part_of:
            [value:[gar_door],
            min:0,
            max:99],
        members:
            [value:[],
            min:0,
            max:0],
        status:
            [value:[0],
            min:0,
            max:1],
        in_of:
            [value:[]],
        relations:
            [value:[below,on],
            min:1,
            max:99],
        attributes:
            [value:[size],
            min:1,
            max:99],
        below:
            [value:[gar_roof],
            min:0,
            max:99],
        on:
            [value:[gar_door],
            min:0,
            max:99],
        size:
            [value:[small],
            min:0,
            max:1,
            type:small@@medium@@large],
        strength:
            [value:[],
            min:0,
            max:1],
        message:
```

```
            [value:[],
            if_added:recognize_object,
            if_changed:recognize_object]
        ]).

?- def_frame(car_window:
        [is_a:
            [value:[window],
            min:1,
            max:99],
        components:
            [value:[],
            min:0,
            max:0],
        part_of:
            [value:[car],
            min:0,
            max:99],
        members:
            [value:[],
            min:0,
            max:0],
        status:
            [value:[0],
            min:0,
            max:1],
        in_of:
            [value:[]],
        relations:
            [value:[on],
            min:1,
            max:99],
        attributes:
            [value:[size],
            min:1,
            max:99],
        on:
            [value:[car_door],
            min:0,
            max:99],
        size:
            [value:[small],
            min:0,
            max:1,
            type:small@@medium@@large],
        strength:
            [value:[],
            min:0,
            max:1],
        message:
            [value:[],
            if_added:recognize_object,
            if_changed:recognize_object]
```

```
        ]).

?- def_frame(tree:
        [is_a:
            [value:[],
            min:0,
            max:0],
        members:
            [value:[]],
        status:
            [value:[0],
            min:0,
            max:1],
        in_of:
            [value:[]],
        components:
            [value:[trunk,crown],
            min:0,
            max:99],
        part_of:
            [value:[house_scene],
            min:1,
            max:99],
        attributes:
            [value:[size],
            min:1,
            max:99],
        size:
            [value:[medium],
            min:0,
            max:1,
            type:small@@medium@@large],
        strength:
            [value:[],
            min:0,
            max:1],
        message:
            [value:[],
            if_added:recognize_object,
            if_changed:recognize_object]
        ]).

?- def_frame(trunk:
        [is_a:
            [value:[amorph],
            min:1,
            max:1],
        members:
            [value:[]],
        status:
            [value:[0],
            min:0,
            max:1],
```

```
        in_of:
            [value:[]],
        components:
            [value:[],
            min:0,
            max:0],
        part_of:
            [value:[tree],
            min:1,
            max:99],
        relations:
            [value:[below],
            min:1,
            max:99],
        below:
            [value:[crown],
            min:0,
            max:99],
        attributes:
            [value:[size,color],
            min:2,
            max:99],
        size:
            [value:[small],
            min:0,
            max:1,
            type:small@@medium@@large],
        color:
            [value:[brown],
            min:0,
            max:1],
        strength:
            [value:[],
            min:0,
            max:1],
        message:
            [value:[],
            if_added:recognize_object,
            if_changed:recognize_object]
        ]).

?- def_frame(crown:
        [is_a:
            [value:[amorph],
            min:1,
            max:1],
        members:
            [value:[]],
        status:
            [value:[0],
            min:0,
            max:1],
        in_of:
```

```
        [value:[]],
    components:
        [value:[],
        min:0,
        max:0],
    part_of:
        [value:[tree],
        min:1,
        max:99],
    relations:
        [value:[above],
        min:1,
        max:99],
    above:
        [value:[trunk],
        min:0,
        max:99],
    attributes:
        [value:[size,color],
        min:2,
        max:99],
    size:
        [value:[small],
        min:0,
        max:1,
        type:small@@medium@@large],
    color:
        [value:[green],
        min:0,
        max:1],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(road:
    [is_a:
        [value:[rect_region],
        min:1,
        max:99],
    part_of:
        [value:[house_scene],
        min:1,
        max:99],
    components:
        [value:[],
        min:0,
        max:0],
```

```
        members:
            |value:||,
            min:0,
            max:0|,
        status:
            |value:|0|,
            min:0,
            max:1|,
        in_of:
            |value:|||,
        relations:
            |value:|below,parallel_to|,
            min:1,
            max:99|,
        attributes:
            |value:|width,color|,
            min:0,
            max:99|,
        width:
            |value:|wide|,
            min:0,
            max:99|,
        color:
            |value:|grey|,
            min:0,
            max:1|,
        below:
            |value:|driveway,walk_way,grass|,
            min:0,
            max:99|,
        parallel_to:
            |value:|sidewalk|,
            min:0,
            max:99|,
        strength:
            |value:||,
            min:0,
            max:1|,
        message:
            |value:||,
            if_added:recognize_object,
            if_changed:recognize_object|
        |).

?- def_frame(car:
        |is_a:
            |value:|||,
        components:
            |value:|wheel,car_window,car_door|,
            min:0,
            max:99|,
        part_of:
            |value:|house_scene|,
```

```
        min:1,
        max:99],
    members:
        [value:[]],
    status:
        [value:[0],
        min:0,
        max:1],
    in_of:
        [value:[]],
    relations:
        [value:[next_to],
        min:1,
        max:99],
    attributes:
        [value:[size,color],
        min:2,
        max:99],
    next_to:
        [value:[gar_door],
        min:0,
        max:99],
    size:
        [value:[medium],
        min:0,
        max:1,
        type:small@@medium@@large],
    color:
        [value:[blue,red,yellow,green,white,black],
        min:0,
        max:1],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(wheel:
    [is_a:
        [value:[circ_region],
        min:1,
        max:1],
    members:
        [value:[]],
    status:
        [value:[0],
        min:0,
        max:1],
    in_of:
```

```
        [value:[]],
    components:
        [value:[],
        min:0,
        max:0],
    part_of:
        [value:[car],
        min:1,
        max:99],
    attributes:
        [value:[size],
        min:1,
        max:99],
    relations:
        [value:[below,next_to],
        min:2,
        max:99],
    below:
        [value:[car_door],
        min:0,
        max:99],
    next_to:
        [value:[wheel],
        min:0,
        max:99],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(garage:
    [is_a:
        [value:[]],
    components:
        [value:[gar_wall,gar_roof],
        min:0,
        max:99],
    part_of:
        [value:[house_scene],
        min:1,
        max:99],
    members:
        [value:[]],
    status:
        [value:[0],
        min:0,
        max:1],
    in_of:
```

```
        [value:[]],
    relations:
        [value:[next_to],
        min:1,
        max:99],
    next_to:
        [value:[house,tree],
        min:0,
        max:99],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(grass:
    [is_a:
        [value:[amorph,rect_region,sq_region],
        min:0,
        max:1],
    members:
        [value:[]],
    status:
        [value:[0],
        min:0,
        max:1],
    in_of:
        [value:[]],
    components:
        [value:[],
        min:0,
        max:0],
    part_of:
        [value:[house_scene],
        min:1,
        max:99],
    relations:
        [value:[next_to,below],
        min:2,
        max:99],
    attributes:
        [value:[color],
        min:0,
        max:99],
    next_to:
        [value:[house_wall,gar_wall,trunk],
        min:0,
        max:99],
    below:
```

```
        [value:[house_wall,gar_wall,sky,trunk],
        min:0,
        max:99],
    color:
        [value:[green],
        min:0,
        max:99],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(walk_way:
    [is_a:
        [value:[rect_region],
        min:1,
        max:99],
    components:
        [value:[],
        min:0,
        max:0],
    part_of:
        [value:[house_scene],
        min:0,
        max:99],
    members:
        [value:[],
        min:0,
        max:0],
    status:
        [value:[0],
        min:0,
        max:1],
    relations:
        [value:[below,next_to],
        min:1,
        max:99],
    attributes:
        [value:[color],
        min:1,
        max:99],
    color:
        [value:[grey],
        min:0,
        max:1],
    below:
        [value:[house_door],
        min:0,
```

```
        max:99|,
    next_to:
        |value:|grass|,
        min:0,
        max:99|,
    strength:
        |value:||,
        min:0,
        max:1|,
    message:
        |value:||,
        if_added:recognize_object,
        if_changed:recognize_object|
    |).

?- def_frame(sidewalk:
    |is_a:
        |value:|rect_region],
        min:1,
        max:99|,
    components:
        |value:||,
        min:0,
        max:0|,
    part_of:
        |value:||,
        min:0,
        max:0|,
    members:
        |value:||,
        min:0,
        max:0|,
    status:
        |value:|0|,
        min:0,
        max:1|,
    in_of:
        |value:|||,
    relations:
        |value:|below,parallel_to|,
        min:1,
        max:99|,
    attributes:
        |value:|width,color|,
        min:1,
        max:99|,
    color:
        |value:|grey|,
        min:0,
        max:1|,
    width:
        |value:[narrow|,
        min:0,
```

```
        max:99],
    below:
        [value:[grass,walk_way],
        min:0,
        max:99],
    parallel_to:
        [value:[road],
        min:0,
        max:99],
    strength:
        [value:[],
        min:0,
        max:1],
    message:
        [value:[],
        if_added:recognize_object,
        if_changed:recognize_object]
    ]).

?- def_frame(driveway:
    [is_a:
        [value:[rect_region],
        min:1,
        max:99],
    components:
        [value:[],
        min:0,
        max:0],
    part_of:
        [value:[house_scene],
        min:0,
        max:99],
    members:
        [value:[],
        min:0,
        max:0],
    status:
        [value:[0],
        min:0,
        max:1],
    in_of:
        [value:[]],
    relations:
        [value:[below,above,next_to],
        min:1,
        max:99],
    attributes:
        [value:[color],
        min:1,
        max:99],
    color:
        [value:[grey],
        min:0,
```

```
    max:1],
below:
    [value:[gar_wall,gar_door],
    min:0,
    max:99],
above:
    [value:[road],
    min:0,
    max:99],
next_to:
    [value:[grass],
    min:0,
    max:99],
strength:
    [value:[],
    min:0,
    max:1],
message:
    [value:[],
    if_added:recognize_object,
    if_changed:recognize_object]
]).
```

```
/*-----------------------------------------------------------
  Auxiliary Knowledge Base.

  The knowledge represented in this auxiliary knowledge base is intended
  to supplement the primary knowledge base.  Due to the limitations in
  representational format of the frame package used to build the primary
  knowledge base, information about the objects the frames represent is
  also found in this knowledge base.
-----------------------------------------------------------*/

/*-----------------------------------------------------------

  Probability objects are composed of their parts.

  Format:  prob_comp_of(X, Y, N).
        The probability X is composed of Y is N.
        N is based on a scale of 1 to 10, with 10 representing the
        most probable and 1 representing the least probable.

-----------------------------------------------------------*/

prob_comp_of(house,house_wall,10).
prob_comp_of(house,house_roof,10).

prob_comp_of(house_wall,house_wall_window,7).
prob_comp_of(house_wall,house_door,10).
prob_comp_of(house_wall, shutters, 6).
prob_comp_of(house_door, house_door_window, 5).

prob_comp_of(house_scene,house,10).
prob_comp_of(house_scene,garage,6).
prob_comp_of(house_scene,road,7).
prob_comp_of(house_scene,car,7).
prob_comp_of(house_scene,driveway,7).
prob_comp_of(house_scene,tree,4).
prob_comp_of(house_scene,grass,4).
prob_comp_of(house_scene,walk_way,5).

prob_comp_of(gar_wall,gar_door,10).
prob_comp_of(gar_door,gar_window,5).


prob_comp_of(tree,trunk,10).
prob_comp_of(tree,crown,9).

prob_comp_of(car,wheel,10).
prob_comp_of(car,car_window,10).
prob_comp_of(car,car_door,9).

prob_comp_of(garage,gar_wall,10).
prob_comp_of(garage,gar_roof,8).

/*-----------------------------------------------------------
```

```
Object to Component Correspondence.  occ represents how many of one
part an object is composed of (from a 2D perspective).
1 implies a one-to-one correspondence, 99 implies a one-to-many
correspondence, and any other number, N, implies a one-to-N
correspondence.

Format:  occ(X, Y, N).
        Object X is composed of N  Ys.
-------------------------------------------------------------*/

occ(house,house_wall,1).
occ(house,house_roof,1).

occ(house_wall,house_wall_window,99).
occ(house_wall,house_door,1).
occ(house_wall, shutters, 99).
occ(house_door, house_door_window, 1).

occ(house_scene,house,1).
occ(house_scene,garage,1).
occ(house_scene,road,1).
occ(house_scene,car,2).
occ(house_scene,driveway,1).
occ(house_scene,tree,2).
occ(house_scene,grass,99).
occ(house_scene,walk_way,1).

occ(tree,trunk,1).
occ(tree,crown,1).

occ(car,wheel,2).
occ(car,car_door_window,2).
occ(car,car_door,2).

occ(garage,gar_wall,1).
occ(garage,gar_roof,1).
occ(gar_wall,gar_door,1).
occ(gar_door,gar_window,1).


/*-------------------------------------------------------------

  Probability components are part of objects.

-------------------------------------------------------------*/


prob_part_of(house,house_scene,10).
prob_part_of(tree,house_scene,7).
prob_part_of(road,house_scene,6).
prob_part_of(car,house_scene,4).
prob_part_of(walk_way,house_scene,7).
```

```
prob_part_of(wheel,car,8).

prob_part_of(trunk,tree,10).
prob_part_of(crown,tree,10).

prob_part_of(wall,house,10).
prob_part_of(roof,house,10).

prob_part_of(door,house_wall,10).
prob_part_of(shutters,house_wall,10).
prob_part_of(gar_roof,garage,10).
prob_part_of(house_roof,house,10).

prob_part_of(gar_door,gar_wall,10).
prob_part_of(house_door,house_wall,10).
prob_part_of(car_door,car,10).

prob_part_of(window,house_wall,8).
prob_part_of(window,gar_door,10).
```

```
/*-----------------------------------------------------------------
  Primary and auxiliary knowledge bases are loaded.  The files
  containing the procedures to run the high-level inference system
  are consulted.
-----------------------------------------------------------------*/

:- load(kb1).
:- [xkb1].
:- [-util].
:- [prog1].
:- [prog2].
:- [prog3].
:- [prog4].
:- [prog5].
:- [prog6].
:- [prog7].


/*-----------------------------------------------------------------
  By invoking 'run,' the high-level inference system is executed.
  The procedure 'run' will call high level procedures that are
  responsible for executing portions of the system.
-----------------------------------------------------------------*/

run:-
 input_tool,
 init,
 process_input_list,
 initialize_state,
 process_top,
 refine_list,
 omit_weak,
 retract(input_list(In)),
 asserta(input_list(In)),
 (In == []
  ;
  save_file(wm_kb, kbfile)
 ),
 add_error_msgs,
 begin,
 process_rc,
 get_weak_hyp,
 test_validity,
 pass2,
 test_noise_occl,
 top_down,
 resolve_conflict,
 (In == []
  ;
  retract(objlist(Objlist)), !,
  asserta(objlist(Objlist)),
  (Objlist == []
   ;
```

```
  save_file(obj_kb, objfile)
 )
),
get_comp_top,
report_output.
```

/*----------

The following global lists are stored in Prolog's internal database.

---

instances_que	stores hypothetical instances that are generated from bottom-up analysis
shque	stores strong instances
whque	stores weak instances
strlist	stores strong instances and newly created instances of object frames that are generated from top-down analysis
valid_hyp	contains list of valid hypotheses
comp_match	contains lists of frames, total number of components to match and the actual number matched so far eg. [[r9n9, 5, 2]]
unmatched	contains lists of frames and the number of components that have not been literally matched eg. [[r9n9, 1]]
input_list	contains input terms such as r9(rect_region)
new_input_list	contains input terms for current session
valid_regions	contains master list of valid regions eg. r5,r4,r7
new_region_list	contains list of regions that are relevant for the current session, i.e., a subset of valid_regions

region	contains list of regions
strlist_reg	contains list of regions that are associated with the instances in strlist
memberlist	contains types of instances that are relevant for the current session, eg. [sky,grass,gar_wall]
objlist	stores list of object names, eg. [obj4,obj19]
error_codes	stores lists of a number and associated error message eg. [[1,[error,one]]]
error_list	stores lists of frames and the number that refers to a particular error message, to be read when output is reported, eg. [[r9n9, 1]]
oddlist	contains instantiated frames whose associated region is in the input_list but has not been matched and therefore this match does not exist in strlist
final_comp	contains list of root nodes of compositional hierarchies that will comprise the final output for the current session
noise_occl_list	list of regions that have been subject to noise or occlusion eg. [r9]

The following comprise the files and knowledge bases that are used during processing of the system.

---

infile	input file
kbfile	file of frames that are stored in wm_kb
objfile	file of object frames created when producing a final interpretation
outfile	output file, stores final interpretation
wm_kb	internal working memory knowledge base
obj_kb	knowledge base of object frames
kb1	knowledge base of frames representing domain of house scenes
xkb1	auxiliary knowledge base supplements kb1

```
/*-----------------------------------------------------------
  The procedure init initializes the instances_que, whque and shque
  lists.  Initialize_state initalizes a state that is asserted
  into Prolog's internal database for every instance. This term has
  the format:  state(r9n99, [xxx], N)  where xxx is a list of
  relationships and attributes that are matched for that instance
  (i.e. corresponding slot has a value) and N is the total amount of
  constraints and attributes that potentially can be matched for
  that instance. The state of an instance is maintained in order to
  determine the current strength value.  State and strength will be
  updated during processing.
-----------------------------------------------------------*/
init:-
  assert(instances_que([])),
  assert(whque([])),
  assert(shque([])).


initialize_state:-
  retract(instances_que(L)), !,
  init_recog_states(L),
  asserta(instances_que(L)), !.

init_recog_states(L):-
 (member(Head, L),
  init_state(Head),
  fail
  ;
  !).

init_state(Frame):-
  (ret_val(Frame, relations, List1)
   ;
   List1 = []
  ),
  (ret_val(Frame, attributes, List2)
   ;
   List2 = []
  ),
  conc(List1, List2, Final_list),
  length(Final_list, Total_num),
  Term =.. [state, Frame, [], Total_num],
  asserta(Term), !, Val = [0],
  add_val(Frame,strength, Val), !.

/*-----------------------------------------------------------
  The following procedures process each primitive in the input list.
-----------------------------------------------------------*/
process_input_list:-
  retract(input_list(L)), !,
  asserta(input_list(L)),
  process_terms(L).
```

```
process_terms(List):-
 (member(Term, List),
  call_once(process_low_cue(Term)),
  fail
  ;
  !).


/*-------------------------------------------------------
   An instance(s) is(are) generated and given a unique name for every
   primitive term in the input list. For example, if the current term
   is a rect_region, then get all terminal node members of the rect_region
   frame and terminal nodes of the specialization hierarchies
   (which are denoted by an empty members slot).
-------------------------------------------------------*/

process_low_cue(Term):-
  functor(Term, Functor, _),
  arg(1, Term, Ins),
  (frame_match_subset([is_a: [value: [Functor]]], Lyst1), !
   ;
   Lyst1 = []
  ),
  (frame_match_exact([members: [value: []]], Lyst2), !
   ;
   Lyst2 = []
  ),
  retract(instances_que(In)), !,
  difference(Lyst1, In, Valid_lyst1),   % eliminate already instantiated frames
  gen_desc_list(Valid_lyst1, _, Totlist), !,
  intersect(Lyst1, Lyst2, Lyst3), !,
  intersect(Lyst2, Totlist, Lyst4), !,
  conc(Lyst3, Lyst4, Lyst),
  Number is 1,
  difference(Lyst, In, Valid_list),
  generate_inst(Valid_list, Number, Ins, In, Newlist),
  asserta(instances_que(Newlist)), !.

/*-------------------------------------------------------
  Procedure process_top calls process_arg to process each instance in
  the instances_que list.
-------------------------------------------------------*/
process_top:-
  retract(instances_que(List)),
  asserta(instances_que(List)),
  process_arg(List, List, Return_list),
  asserta(instances_que(Return_list)).


/*-------------------------------------------------------
   get_pos_strength is passed a list of instances that are all instantiations
   of the same object and returns a list that contains only those instances
   which have strength values greater than 0.
```

```
-------------------------------------------------------------*/

get_pos_strength([], [], _):- !.

get_pos_strength(List1, Newlist, Base_strength):-
 List1 = [Head | Tail1],
 (ret_val(Head, strength, Strength_list), !
  ;
  Strength_list = [0]
 ),
 Strength_list = [Strength],
 (Strength > Base_strength,
  Newlist = [Head | Tail2]
  ;
  Newlist = Tail2
 ),
 get_pos_strength(Tail1, Tail2, Base_strength).


/*-------------------------------------------------------------
   find_same_regions is passed a list of instances and a region (r3, for
   example) and returns a new list of instances whose region prefix matches
   the region passed in, i.e., collect all r3's.
-------------------------------------------------------------*/

find_same_regions([], [], Region):- !.

find_same_regions(List, Newlist, Region):-
  List = [Head | Tail1],
  parse_inst_name(Head, Newregion), !,
  (Newregion == Region,
   Newlist = [Head | Tail2]
   ;
   Newlist = Tail2
  ),
  find_same_regions(Tail1, Tail2, Region).


call_once(X):-
  call(X), !.

/*-------------------------------------------------------------
   process_arg processes each item in the instances_que list, i.e. every
   instantiated frame, by sending a message to the instance's message slot
   to satisfy its constraints and match its attributes. If no constraints
   can be satisfied or attributes matched, then the instance's strength
   slot is set to 0.
   If the value of this slot is 0, then undefine the frame and remove it
   from the instances_que list, since this instance represents an unlikely
   hypothesis. If this value is not 0, then move it to the end of the
   instances_que list since it represents a weak or probable hypothesis.
   Rlist is the list of instances returned after a first pass.
-------------------------------------------------------------*/
```

```
process_arg([], Rlist, Rlist):- !.

process_arg(List, Rlist, Returnlist):-
  List = [Head | Tail],
  send_message(Head, satisfy_constraints),
  send_message(Head, match_attributes),
  ret_val(Head, strength, Val),
  Val = [Value],
  (Value == 0,
   undef_frame(Head),
   del(Head, Rlist, Newlist)
   ;
   shift(Rlist, Newlist)
  ),
  process_arg(Tail, Newlist, Returnlist).

/*-------------------------------------------------------------
  Match_attributes looks for available input or cues that were asserted
  into the internal database to fill the value facet of each attribute
  slot.  If a match is found, then the instance's recognition state is
  updated accordingly.
-------------------------------------------------------------*/

match_attributes(Frame):-
  ret_val(Frame, in_of, Parent_frames),
  parse_inst_name(Frame, Instance), !,
  process_parents(Frame, Instance, Parent_frames).


process_parents(Frame, Instance, Parent_frames):-
  (member(Parent_frame, Parent_frames),
   ret_val(Parent_frame, attributes, Attribute_list),
   call_once(process_attribute_list(Frame, Instance, Parent_frame,
          Attribute_list)),
   fail
   ;
   !).

process_attribute_list(Frame, Instance, Parent_frame, Attribute_list):-
  (member(Attribute, Attribute_list),
   ret_val(Parent_frame, Attribute, Args),
   call_once(process_attributes(Frame, Instance, Attribute, Args)),
   fail
   ;
   !).

process_attributes(Frame, Instance, Attribute, Args):-
  (member(Arg, Args),
   call_once(find_attributes(Frame, Instance, Attribute, Arg)),
   fail
   ;
   !).
```

```
find_attributes(Frame, Instance, Attribute, Arg):-
  Term1 =.. [Arg, Instance],
  ((call(Term1),
    add_val(Frame, Attribute, Arg),
    update_recognition_state(Frame, Attribute)
   )
   ;
   true
  ).

/*-----------------------------------------------------------
  Satisfy_constraints looks for available input or cues to fill the
  value facet of slots that represent relationships.  If a match is
  found, then the instance's recognition state is updated accordingly.
-----------------------------------------------------------*/

satisfy_constraints(Frame):-                    % eg Frame is r6n6
  ret_val(Frame, in_of, Parent_frames),           % get parents of this instance
  parse_inst_name(Frame, Instance), !,          % eg Instance is r6
  process_parent_frames(Frame, Instance, Parent_frames).


process_parent_frames(Frame, Instance, Parent_frames):-
  (member(Parent_frame, Parent_frames),
   ret_val(Parent_frame, relations, Relations),
   call_once(process_relations(Frame, Instance, Parent_frame, Relations)),
   fail
   ;
   !).

process_relations(Frame, Instance, Parent_frame, Relations):-
  (member(Relation, Relations),
   ret_val(Parent_frame, Relation, Prototype_args),
   call_once(process_constraints(Frame, Relation, Instance, Prototype_args)),
   fail
   ;
   !).

process_constraints(Frame, Relation, Instance, Prototype_args):-
  (member(Prototype_arg, Prototype_args),
   call_once(find_constraints(Instance, Relation, List_of_args)),
   call_once(match_constraints(Frame, Relation, List_of_args, Prototype_arg)),
   fail
   ;
   !).


match_constraints(Frame, Relation, List_of_args, Prototype_arg):-
  (member(Related_inst, List_of_args),
   call_once(frame_match_subset([in_of:[value:[Prototype_arg]]],
                                Lyst_of_instances)),
   call_once(look_for_match(Lyst_of_instances, Related_inst, Frame, Relation)),
   fail
```

```
  ;
  !).


look_for_match(Lyst_of_instances, Related_inst, Frame, Relation):-
  (member(Instance1, Lyst_of_instances),
   call_once(parse_inst_name(Instance1, Ins)),
   call_once((Ins == Related_inst,
           call_once(add_val(Frame, Relation, Instance1)),
           call_once(update_recognition_state(Frame, Relation))
           ;
           true
          )),
   fail
   ;
   !).

/*-----------------------------------------------------------
  Recognize_object is an if_added demon that represents a top level
  procedure to pass control to a specific recognition procedure.
-----------------------------------------------------------*/

recognize_object(Frame, Slot, Val):-
  Name = [Val],
   (Val == satisfy_constraints,
    satisfy_constraints(Frame)
   )
   ;
   (Val == match_attributes,
    match_attributes(Frame)
   ).

/*-----------------------------------------------------------
  Send_message is the top-level procedure that activates an instance's
  recognize_object demon.
-----------------------------------------------------------*/

send_message(Object, Msg):-
  (ret_val(Object, message, Val), !
   ;
   Val = []
  ),
  (isempty(Val),
   add_val(Object, message, Msg)
   ;
   rem_val(Object, message, Val),
   add_val(Object, message, Msg)
  ).

/*-----------------------------------------------------------
  Update_strength updates an instance's strength value. Strength
  represents how strong of a hypothesis an instance is. The scale
  is 0 - 1.0 where 1.0 represents the strongest hypothesis and 0
```

```
 reflects an unlikely or improbable hypothesis.
-----------------------------------------------------------------*/

update_strength(Frame, List, Total_constraints):-
  length(List, Constraints_matched),
  (Total_constraints = 0,
   Ratio is 0
   ;
   Ratio is Constraints_matched / Total_constraints
   ),
   Value = [Ratio],
   change_val(Frame, strength, Val, Value).

/*-----------------------------------------------------------
  Update_recognition_state is passed an instance name and a matched
  relation or attribute and updates the instances's state.
-----------------------------------------------------------------*/

update_recognition_state(Frame, Arg):-  % eg, Frame is r6n2, Arg is next_to
  Term =.. [state, Frame, List, Num],
  retract(Term), !,
  ((member(Arg, List),                  % if not a member, add to list
   asserta(Term), !,
   update_strength(Frame, List, Num))
   ;
   (List1 = [Arg | List],
   Term2 =.. [state, Frame, List1, Num],
   asserta(Term2), !,
   update_strength(Frame, List1, Num)
  )).


find_constraints(Instance, Relation, List_of_args):-
  Term1 =.. [Relation, Instance, Arg1],
  (setof(Arg1, Term1, List_of_args)
   ;
   List_of_args = []
  ),
  !.

/*-----------------------------------------------------------
  Parse_inst_name is passed an instance name (r9n99) and returns
  an instance prefix (r9).
-----------------------------------------------------------------*/

parse_inst_name(Inst_name, Instance):-
  name(Inst_name, Lyst),
  Lyst = [Arg1, Arg2, Arg3 | Tail],
  (Arg1 == 114,
   (Arg3 == 110,                        % 110 = letter n
    Inst_lyst = [Arg1, Arg2]            % one digit eg. r1
    ;
    Inst_lyst = [Arg1, Arg2, Arg3]      % two digits eg. r14
```

```
  ),
  name(Instance, Inst_lyst)          % Instance returned eg. r6
  ;
  name(Instance, Lyst)
 ).

/*-------------------------------------------------------
  Generate_inst is responsible for adding a new instance to the working
  memory knowledge base.
-------------------------------------------------------*/

generate_inst( [], N, I, Inp, Inp):- !.

generate_inst( Lyst, Num, Instance, Inp, Newlist):-
  Lyst = [Head | Tail],
  assign_inst_name(Instance, Num, Inst_name),
  Newlist = [Inst_name | Newtail],
  add_frame(wm_kb, (Inst_name: [in_of: [value: [Head]]])),
  Num1 is Num + 1,
  generate_inst( Tail, Num1, Instance, Inp, Newtail).


/*-------------------------------------------------------
  assign_inst_name assigns a unique name to an instance given Instance.
  For example, Instance is r17 and Inst_name is r17n14.
-------------------------------------------------------*/

assign_inst_name(Instance, Num, Inst_name):-
   name(Instance, First),
   name(n, X),
   conc(First, X, L1),
   name(Num, X1),
   conc(L1, X1, L2),
   name(Inst_name, L2).

/*-------------------------------------------------------
  gen_desc_list returns Final_list which is a list of all members of
  each frame in the list, L1.
-------------------------------------------------------*/

gen_desc_list(L1, Templist, Final_list):-
  isempty(L1), !,  Final_list = Templist
  ;
  L1 = [Arg | Tail],
  (ret_val(Arg, members, L), !
   ;
   L = []
  ),
  conc(L, Templist, Newlist),
  gen_desc_list(Tail, Newlist, Final_list).
```

```
begin:-
  asserta(shque([])),
  asserta(whque([])),
  asserta(error_list([])),
  asserta(oddlist([])).

/*-------------------------------------------------------
  Get_weak_hyp creates the weak hypothesis list (whque).
-------------------------------------------------------*/

get_weak_hyp:-
 retract(instances_que(List1)), !,
 asserta(instances_que(List1)),
 retract(shque(List2)), !,
 asserta(shque(List2)),
 difference(List1, List2, Differences),
 retract(whque(List3)), !,
 asserta(whque(Differences)).

/*-------------------------------------------------------
  The purpose of process_rc is to process each instance in the instances_que
  list to obtain the strongest hypotheses which are stored in shque. The
  remaining hypotheses are stored in whque. Process_rc takes each instance
  and compares the strength values of the instances that are in_of the same
  type and compares the strength values of the instances that are hypothesized
  to match the same primitive region, i.e. all r9's.
  eg.      r1n1 x      r4n5       r3n8 x       all are in_of sky
           r1n2        r4n8 x     r3n4
           r1n5        r4n2 x
                       r4n6 x

  The instance(s) with the greatest strength value(s) of the first row
  are set aside as the their corresponding columns are checked. If the
  instance still has the greatest strength value among its column counterparts
  then it is marked as being the instance with the greatest strength, i.e.,
  the instance has the most input supporting its existence. The instances
  that are not set aside in the first row will be put on the weak
  hypothesis list as will the instances in the specific column(s) of the
  instance that has the greatest strength.
  For example, the instances above that are all marked with an x beside them
  are marked for the weak list because r4n5 was found to be the instance
  with the greatest strength value. This represents a means of eliminating
  weak or improbable hypotheses from the analysis process.
-------------------------------------------------------*/

process_rc:-
 retract(instances_que(List)), !,
 asserta(instances_que(List)),
 process_rows(List, List),
 retract(shque(N)), !,
 asserta(shque(N)).

process_rows(List1, List):-
```

```
 (member(Instance, List1),
  call_once(process_row(List, Instance)),
  fail
  ;
  !
  ).

process_row(List, Instance):-
 get_row(Instance, Row),
 find_largest(Row, New_rowlist),
 process_column(New_rowlist, List).

/*-------------------------------------------------
  add_to_list adds an instance to the list, shque.
-------------------------------------------------*/

add_to_list(L1):-
 (member(X, L1),
  call_once(retract(shque(L2))),
  call_once((call_once(member(X, L2)),
             call_once(asserta(shque(L2)))
             ;
             Newlist = [X | L2],
             call_once(asserta(shque(Newlist)))
            )),
  fail
 ;
 !
 ).

process_column(New_rowlist, List):-
  (member(Frame1, New_rowlist),
   call_once(parse_inst_name(Frame1, Region)),
   call_once(find_same_regions(List, Column, Region)),
   call_once(find_largest(Column, New_columnlist)),
   call_once(add_to_list(New_columnlist)),
   fail
   ;
   !
   ).

/*-------------------------------------------------
  get_row gets all instances of Instance and returns Row.
-------------------------------------------------*/

get_row(Instance, Row):-
  (ret_val(Instance, in_of, Val), !
   ;
   Val = []
  ),
  Val = [Value],
  (frame_match_subset([in_of: [value: [Value]]], Row), !
   ;
```

```
  Row = []
 ).


/*-------------------------------------------------------
 find_largest finds the Instance(s) with the greatest strength value
 and returns Returnlist.
-------------------------------------------------------*/

find_largest(List1, Returnlist):-
 find_strengths(List1, Newlist),
 maxlist(Newlist, Max),
 get_largest(List1, Max, Returnlist).


/*-------------------------------------------------------
 find_strengths finds the strength values of List1 and returns
 Newlist which is the list of strength values.
-------------------------------------------------------*/

find_strengths([], []):- !.

find_strengths(List1, Newlist):-
 List1 = [Head | Tail1],
 (ret_val(Head, strength, Strength_list), !
  ;
  Strength_list = [0]
 ),
 Strength_list = [Strength],
 Newlist = [Strength | Tail2],
 find_strengths(Tail1, Tail2).

/*-------------------------------------------------------
 get_largest gets the largest strength value (Max) among a list of
 instances (List1) and returns Returnlist.
-------------------------------------------------------*/

get_largest([], Max, []):- !.

get_largest(List1, Max, Returnlist):-
 List1 = [Instance | Tail1],
 (ret_val(Instance, strength, Strength_list), !
  ;
  Strength_list = [0]
 ),
 Strength_list = [Strength],
 (Strength == Max,
  Returnlist = [Instance | Tail2]
  ;
  Returnlist = Tail2
 ),
 get_largest(Tail1, Max, Tail2).
```

```
/*-------------------------------------------------------
 The purpose of omit_weak is to eliminate all hypotheses from the
 instances_que list that have a strength value of 0.  It is called
 after refine_list is called.
-------------------------------------------------------*/

omit_weak:-
 retract(instances_que(List)), !,
 get_pos_strength(List, Newlist, 0),
 !,
 asserta(instances_que(Newlist)).

/*-------------------------------------------------------
 The purpose of refine_list is to test each relationship of each instance
 in the instances_que list for its strength.  If the related instance
 whose frame name is found in a relationship slot of an instance does not
 exist or has a strength value of 0, then it is removed from the slot
 and the strength and state of the instance are updated to reflect this
 change.
-------------------------------------------------------*/

refine_list:-
 retract(instances_que(Ilist)), !,
 asserta(instances_que(Ilist)),
 (member(Arg, Ilist),
  call_once(test_relationships(Arg, Ilist)),
  fail
  ;
  !).

test_relationships(Frame, Ilist):-
 (ret_val(Frame, relations, Relations), !
  ;
  Relations = []
 ),
 test_relations(Frame, Relations, Ilist).


test_relations(Frame, Relations, Ilist):-
 (member(Relation, Relations),
  ret_val(Frame, Relation, Related_instances),
  call_once(test_related_inst(Frame, Relation, Related_instances)),
  fail
  ;
  !).

/*-------------------------------------------------------
 Test_related_inst checks if an instance exists prior to obtaining
 its strength value.
-------------------------------------------------------*/

test_related_inst(Frame, Relation, Related_instances):-
 (member(Related_inst, Related_instances),
```

```
  call_once((call_once(frame_subsumes(X, Related_inst)),
          call_once((ret_val(Related_inst, strength, Strength_list), !
                  ;
                  Strength_list = [0]
                 ))
          ;
          Strength_list = [0]    % if frame does not exist, strength = 0
          )),
  Strength_list = [Strength],
  call_once(test_strength(Frame, Relation, Related_inst, Strength)),
  fail
  ;
  !).


test_strength(Frame, Relation, Related_inst, Strength):-
  (Strength == 0,
   (ret_val(Frame, Relation, List), !
    ;
    List = []
   ),
    rem_val(Frame, Relation, Related_inst),
    length(List, Length),
    (Length == 1,
     adjust_state(Frame, Relation)
     ;
     true
    )
   ;
   true
   ).

/*-----------------------------------------------------------
  Adjust_state is called when the slot value of a relationship is to be
  deleted and the strength value then updated accordingly.
-----------------------------------------------------------*/

adjust_state(Frame, Relation):-
  Term1 =.. [state, Frame, List, Num],
  retract(Term1), !,
  del(Relation, List, Newlist),
  Term2 =.. [state, Frame, Newlist, Num],
  asserta(Term2),
  update_strength(Frame, Newlist, Num).
```

```
/*-----------------------------------------------------------
  test_validity tests all relationships for validity. These constraints
  must be based on relationships with strong instances, otherwise, frame
  is not as strong as indicated. Two passes are made through the list
  of strong instances.

-----------------------------------------------------------*/

test_validity:-
  asserta(valid_hyp([])),
  retract(instances_que(Ilist)), !,
  asserta(instances_que(Ilist)),
  retract(shque(L)), !,
  asserta(shque(L)),
  test_valid_rel(L, Ilist),
  check_validity(L, Ilist),
  retract(valid_hyp(Valid_list)), !,
  asserta(valid_hyp(Valid_list)),
  asserta(shque(Valid_list)),

  conc(Valid_list, L, L1), !,
  asserta(valid_hyp([])),
  test_valid_rel(Valid_list, Ilist),
  check_validity(Valid_list, L1),
  difference(L, Valid_list, Result), !,
  (Result == []
   ;
   retract(valid_hyp(Valid_list2)), !,
   asserta(valid_hyp([])),
   test_valid_rel(Valid_list2, Ilist),
   check_validity(Valid_list2, L1)
  ),
  retract(valid_hyp(Valid)), !,
  asserta(valid_hyp(Valid)),

  difference(L, Valid, Weak_hyp), !,
  asserta(shque(Valid)),
  retract(whque(Old_weak)), !,
  conc(Old_weak, Weak_hyp, New_weak),
  asserta(whque(New_weak)).


 check_validity(List1, List):-
  (member(Instance, List1),
   call_once(check_row(List, Instance)),
   fail
   ;
   !
   ).

 check_row(List, Instance):-
  get_row(Instance, Row),
  intersect(Row, List, Newrow),
```

```
 find_largest(Newrow, New_rowlist),
 check_column(New_rowlist, List).


check_column(New_rowlist, List):-
  (member(Frame, New_rowlist),
   call_once(parse_inst_name(Frame, Region)),
   call_once(find_same_regions(List, Column, Region)),
   call_once(find_largest(Column, New_column)),
   call_once(add_valid(New_column)),
   fail
   ;
   !
   ).

add_valid(List):-   % if not already a member of the valid_hyp list then add
  (member(Arg, List),
   call_once(retract(valid_hyp(List2))),
   call_once((call_once(member(Arg, List2)),
              call_once(asserta(valid_hyp(List2)))
              ;
              Newlist = [Arg | List2],
              call_once(asserta(valid_hyp(Newlist)))
             )),
   fail
   ;
   !
   ).

test_valid_rel(Framelist, Ilist):-
  (member(Frame, Framelist),
   call_once((ret_val(Frame, relations, Relations), !
              ;
              Relations = []
             )),
   call_once(test_rel(Frame, Relations, Ilist)),
   fail
   ;
   !
   ).

test_rel(Frame, Relations, Ilist):-
  (member(Relation, Relations),
   call_once((ret_val(Frame, Relation, Related_instances), !
              ;
              Related_instances = [])),
   call_once(intersect(Related_instances, Ilist, Rel_insts)),
   call_once(test_valid_inst(Frame, Relation, Rel_insts)),
   fail
   ;
   !
   ).
```

```
/*-----------------------------------------------------
  If each related instance of a relation in a frame is not found in
  the strong list of instances, then remove that related instance and
  if this is the only related instance, then adjust the state of the
  frame which will also update its strength value.
-----------------------------------------------------*/

test_valid_inst(Frame, Relation, Related_instances):-
 (member(Related_inst, Related_instances),
  call_once(retract(shque(List))),
  call_once(asserta(shque(List))),
  call_once((member(Related_inst, List)
          ;
          call_once(ret_val(Frame, Relation, Rel_list)),
          call_once((member(Related_inst, Rel_list),
                  call_once(rem_val(Frame, Relation, Related_inst))
                  ;
                  true
                 )),
          call_once(length(Rel_list, Length)),
          call_once((Length == 1,
                  call_once(adjust_state(Frame, Relation))
                  ;
                  true
                 ))
        )),
  fail
  ;
  !
  ).

/*-----------------------------------------------------
  assign_obj_name assigns a unique name to an instantiation of an
  object frame.
-----------------------------------------------------*/

assign_obj_name(Num, Objname):-
 name(obj, First),
 name(Num, Second),
 conc(First, Second, Name),
 name(Objname, Name).

/*-----------------------------------------------------
  get_nodes gets a list of instances of Element.
  If Element is already an instance, do not add to kb as an object,
  else add object to kb.
-----------------------------------------------------*/

get_nodes(Element, Nodes):-
  (ret_val(Element, in_of, Parents), !
  ;
  Parents = []
  ),
```

```
 (Parents == [|,
  (frame_match_subset([in_of: [value: [Element|||, Elementlist), !
   ;
   Elementlist = [|
  ),
  retract(strlist(Strlist)), !,
  asserta(strlist(Strlist)),
  intersect(Elementlist, Strlist, Oldnodes),
  (Oldnodes == [|,
   add_object(Element, Objname, Nlist),
   Nodes = [Objname | Oldnodes[
   ;
   Nodes = Oldnodes
  )
 ;
 Nodes = [Element]
 ).

match_instance(Element, Component):-
 ret_val(Element, components, Arglist),
 (member(Component, Arglist)     % if not a member, then add value
  ;
  add_val(Element, components, Component),
  update_match_status(Element)
 ).

/*-------------------------------------------------------
  If Node has no components and is not a strong instance, then
  mark this node as non-existent by placing value of -1 in its
  status slot.
-------------------------------------------------------*/

test_node(Node, New_status):-
 get_components(Node, Complist),
 (Complist == [|,
  retract(shque(Stronglist)), !,
  asserta(shque(Stronglist)),
  (not(member(Node, Stronglist)),
   New_status = -1,
   change_val(Node,status, _, New_status)
   ;
   ret_val(Node, status, New_status_list),
   New_status_list = [New_status[
  )
  ;
  ret_val(Node, status, New_status_list),
  New_status_list = [New_status[
 ).


/*-------------------------------------------------------
 Process_components processes the components of each node while
 building the final composition hierarchy.  Upon first calling this
```

```
procedure, only those nodes that have instances in Strlist and
have no components are marked as matched (i.e. their status slot = 1).
As each node in the tree matches its components, its status slot is
updated.  The tree is built in a bottom-up fashion since an object
cannot be marked as existing until all of its components are either
existing or are non-existent but have weak associations.
Each new node is tested for existence before it is added as a component
to a higher level node.  If it does not exist, then its object-component
probability is tested.  If this is a strong probability, then objects
that this component is a part of are marked as nonexisting also.  If this
is a weak probability, then the objects this component is a part of are
updated accordingly, i.e. the component is not added to the higher level
object but the object's status is updated as if it had been added.

-------------------------------------------------------------*/

process_components(Strlist, Element, []):- !.

process_components(Strlist, Element, Complist):-
  Complist = [Component | Tail1],
  get_components(Component, Clist),
  retract(strlist(Stronglist)), !,
  asserta(strlist(Stronglist)),
  get_nodes(Component, Nodes), !,
  Nodes = [Node | Tail2],
  get_nodes(Element, Element_nodes), !,
  Element_nodes = [Element_node | Tail3],
  (ret_val(Node, status, Status_val), !
   ;
   Status_val = []
  ),
  Status_val = [Status],
  (Status == 1,
   test_node(Node, Newstatus),
   (Newstatus == -1,        % node does not exist as part of final tree
    test_prob(Element_node, Node, Newstatus)
    ;
    check_occ(Element, Component, Element_node, Nodes)
   )
   ;
   process_components(Stronglist, Component, Clist), % move down tree
   (ret_val(Node, status, Status_list), !    % test top-level relat again
    ;
    Status_list = []
   ),
   Status_list = [Node_status],
   (Node_status == 1,
    test_node(Node, New_status),
    (New_status == -1,        % node does not exist as part of final tree
     test_prob(Element_node, Node, New_status)
     ;
     check_occ(Element, Component, Element_node, Nodes)
    )
```

```
   ;
    test_prob(Element_node, Node, Node_status)
   )
  ),
  process_components(Stronglist, Element, Tail1).   % move laterally across
                                                    % tree

/*-------------------------------------------------------
  get_partofs returns a list of objects that Object is part_of
-------------------------------------------------------*/

get_partofs(Object, Partof_list):-
  (ret_val(Object, part_of, Partof_list), !
   ;
   Partof_list = []
  ).

/*-------------------------------------------------------
  get_components returns a list of objects that compose Object
-------------------------------------------------------*/

get_components(Object, Component_list):-
  (ret_val(Object, components, Component_list), !
   ;
   Component_list = []
  ).

/*-------------------------------------------------------
  find_top_node is passed an Instance and a Node and returns the
  highest (in a compositional hierarchy) node that Instance is
  part of
-------------------------------------------------------*/

find_top_node(Instance, Node, Topnode):-
  (ret_val(Instance, part_of, Node), !
   ;
   Node = []
  ),
  (Node == [],
   Topnode = Instance, !
   ;
   Node = [Parent],
   find_top_node(Parent, Nextnode, Topnode)
  ).

/*-------------------------------------------------------
  add_object is passed a Component and returns the name of a new object,
  Objname, and a new list of strong instances, Newerlist.
-------------------------------------------------------*/

add_object(Component, Objname, Newerlist):-
  retract(counter(Number)),
  Newnum is Number + 1,
```

```
 asserta(counter(Newnum)),
 assign_obj_name(Newnum, Objname),
 add_frame(obj_kb, (Objname:[in_of: [value: [Component]]])),
 init_match([Objname]),
 retract(strlist(Oldlist)), !,
 Newlist = [Objname | Oldlist],
 retract(objlist(Obj)), !,
 Newobj = [Objname | Obj],
 asserta(objlist(Newobj)),
 shift(Newlist, Newerlist),
 asserta(strlist(Newerlist)).

/*-------------------------------------------------
 top_down is a high-level procedure that calls procedures to
 initialize the match status of strong instances and to build
 the final composition hierarchies.
-------------------------------------------------*/

top_down:-
 asserta(counter(0)),
 retract(shque(List)),
 asserta(shque(List)),
 asserta(comp_match([])),
 asserta(unmatched([])),
 init_match(List),
 asserta(strlist(List)),
 asserta(matchlist([])),
 asserta(objlist([])),
 build_tree(List).

/*-------------------------------------------------
 build_tree is passed a list of strong instances and calls
 process_components to process the components of each instance
 in order to build the final composition hierarchies.
-------------------------------------------------*/

build_tree(List1):-
  (member(Instance, List1),
   call_once(retract(strlist(List2))),
   call_once(asserta(strlist(List2))),
   call_once(find_top_node(Instance, _, Top)),
   call_once(get_nodes(Top, Topnodes)),
   call_once(Topnodes = [Topnode | Tail]),
   call_once((ret_val(Topnode, status, Status_val), !
            ;
            Status_val = []
           )),
   call_once(Status_val = [Status]),
   call_once((Status == 1
            ;
            call_once(get_components(Top, Complist)),
            call_once(process_components(List2, Topnode, Complist))
           )),
```

```
  fail
  ;
  !
  ).

/*-------------------------------------------------------
  init_match initializes the status slot of each instance in the list that
  is passed to it. If an instance has no components according to its
  prototype frame, then a value of 1 is assigned to status slot.
  Otherwise, the status slot is initialized to 0.
-------------------------------------------------------*/

init_match(List):-
  (member(Arg, List),
   call_once(get_components(Arg, Complist)),
   call_once(length(Complist, Length)),
   call_once(retract(comp_match(Oldlist))),
   call_once(retract(unmatched(Old))),
   call_once((Length == 0,
              Newlist = [[Arg, 0, 0] | Oldlist],
              call_once(add_val(Arg, status, 1))
              ;
              Newlist = [[Arg, Length, 0] | Oldlist],
              call_once(add_val(Arg, status, 0))
             )),
   New = [[Arg, 0] | Old],
   call_once(asserta(unmatched(New))),
   call_once(asserta(comp_match(Newlist))),
   fail
   ;
   !
   ).

/*-------------------------------------------------------
  update_match_status will update the status slot of each instance that
  is passed to it. The value is determined according to how many
  components are matched so far.
-------------------------------------------------------*/

update_match_status(Instance):-
  (ret_val(Instance, components, Complist), !
   ;
   Complist = []
  ),
  length(Complist, Instancelength),
  retract(comp_match(L)), !,
  retract(unmatched(L2)), !,
  asserta(unmatched(L2)),
  Term = [Instance, Total, Num_matched],
  Term2 = [Instance, Number],
  (member(Term, L),
   (Total == 0,
    Ratio is 1,
```

```
  change_val(Instance, status, _, Ratio),
  asserta(comp_match(L))
  ;
  (member(Term2, L2),
   Newlength is Instancelength + Number,
   Newterm = [Instance, Total, Newlength],
   del(Term, L, Templist),
   Newtemp = [Newterm | Templist],
   asserta(comp_match(Newtemp)),
   Ratio is Newlength / Total,
   (Ratio > 1,
    Newratio is 1
    ;
    Newratio is Ratio
   ),
   change_val(Instance, status, _, Newratio)
  ;
   asserta(comp_match(L))
  )
 )
;
asserta(comp_match(L))
).
```

```
/*-------------------------------------------------------
  add_error_msgs asserts a list of error_codes and messages into
  the internal database.
-------------------------------------------------------*/

add_error_msgs:-
  asserta(error_list([])),
  asserta(error_codes([[1, [-,unable,to,interpret,accurately,due,to,lack,of,
                            available,information,.]],
                       [2, [-,unable,to,interpret,accurately,due,to,
                            excessive,noise,.]]
                      ])
         ).

/*-------------------------------------------------------
  resolve_conflict calls procedures to get a member list and to check
  conflict.  The objective is to check that the number of occurrences
  of each instance in the list of strong instances is consistent with
  the object to component correspondence (occ) values.
-------------------------------------------------------*/

resolve_conflict:-
 asserta(memberlist([])),
 retract(shque(Stronglist)), !,
 asserta(shque(Stronglist)),
 retract(strlist(Strlist)), !,
 asserta(strlist(Strlist)),
 get_memberlist(Stronglist),
 retract(memberlist(Memberlist)), !,
 asserta(memberlist(Memberlist)),
 check_conflict(Stronglist, Strlist, Memberlist).

/*-------------------------------------------------------
  get_memberlist will add the frames that each instance in Stronglist
  is an instance of to memberlist.
-------------------------------------------------------*/

get_memberlist(Stronglist):-
 (member(Instance, Stronglist),
  call_once(ret_val(Instance, in_of, Typelist)),
  Typelist = [Type],
  call_once(retract(memberlist(Old))),
  New = [Type | Old],
  call_once(asserta(memberlist(New))),
  fail
  ;
  !
  ).

get_occurrences(Arg, List, Num):-
   Arglist = [Arg],
   call_once(intersect(List, Arglist, Result)),
   call_once(length(Result, Num)).
```

```
/*-------------------------------------------------------------
  Check_conflict checks each instance on the strong list and stores
  the type of instance in memberlist.  If the type of instance already
  exists in memberlist, then its occ is checked and if this number is
  1, then the instance is put on the error list.
-------------------------------------------------------------*/

check_conflict(Stronglist, Strlist, Memberlist):-
 (member(Type,Memberlist),
  call_once(get_occurrences(Type, Memberlist, Num)),
  call_once((Num =< 1
         ;
        call_once((frame_match_subset([in_of:[value:[Type]]], Matches), !
                ;
                Matches = []
               )),
        call_once(intersect(Matches, Stronglist, Result)),
        Result = [Instance | Tail],
              % Instances in Result are all of same type
        call_once(get_partofs(Instance, Partoflist)),
        call_once((Partoflist == []
                ;
                call_once(get_all_occ(Instance, Partoflist, Number)),

 % at this point it is known that # occurrences in Strlist is > 1

                call_once((Number == 1,
                        call_once(retract(error_list(Old))),
                        call_once(parse_inst_name(Instance, Region)),
                        Sublist = [ Region,  1 ],
                        call_once((member(Sublist, Old),
                                call_once(asserta(error_list(Old)))
                                ;
                                New = [Sublist | Old],
                                call_once(asserta(error_list(New)))
                               ))
                        ;
                        true
                        ))
                 ))
         )),
   fail
   ;
   !
   ).


   % get total list of all results and then get all occ

   get_all_occ(Instance, [], 0).

   get_all_occ(Instance, Result, Total):-
    Result = [Frame | Tail],
```

```
 get_occ(Frame, Instance, Num),
 get_all_occ(Instance, Tail, Newnum),
 Total is Num + Newnum.

/*-------------------------------------------------------
  The purpose of module pass2 is to resolve any primitive regions that
  were not matched following the first attempt at matching instances
  in the strong que to prototypes in the knowledge base. If there is
  still not enough information to match an instance to a prototypical
  frame, then the strongest instance is chosen and put on an error
  list. The message associated with the error will be reported
  to the user when processing is completed in the output module.
-------------------------------------------------------*/

pass2:-
  asserta(oddlist([])),
  retract(instances_que(Ilist)), !,
  asserta(instances_que(Ilist)),
  retract(whque(Weaklist)), !,
  asserta(whque(Weaklist)),
  retract(shque(Stronglist)), !,
  asserta(shque(Stronglist)),
  get_input_regions,
  get_unmatched(Odd),        % eg. Odd = [r12]
  retract(xinput_list(Xlist)), !,  % if Xlist is not empty then it is known
  asserta(xinput_list(Xlist)),     % that input was subject to noise. Report
  (Xlist == [],                    % appropriate error message in test_oddlist
   Num = 1
   ;
   Num = 2
  ),
  (Odd == [],
   test_oddlist(Stronglist, Num)
   ;
   add_odd_to_error(Odd, Num),
   get_oddlist(Weaklist, Odd),
   retract(oddlist(Oddlist)), !,          % eg. oddlist = [r12n5,r12n7]
   asserta(oddlist(Oddlist)),
   process_rows(Oddlist, Ilist),
   test_oddlist(Oddlist, Num),
   get_weak_hyp,
   retract(shque(N)), !,
   asserta(shque(N))
  ).

 add_odd_to_error(Odd, Msgnum):-        % Odd eg. [r9] add to error list
  (member(Term, Odd),
   call_once(retract(error_list(Old))),
   Sublist = [Term, Msgnum],
   call_once((member(Sublist,Old),
            call_once(asserta(error_list(Old)))
            ;
            New = [Sublist | Old],
```

```
          call_once(asserta(error_list(New)))
         )),
  fail
  ;
  !
  ).

/*-------------------------------------------------------
  get_oddlist will add an instance to oddlist.
-------------------------------------------------------*/

get_oddlist(Weaklist, Odd):-
  (member(Arg, Weaklist),
   call_once(parse_inst_name(Arg, Region)),
   call_once((member(Region, Odd),
          call_once(retract(oddlist(Old))),
          New = [Arg | Old],        % add instance to oddlist eg. r12n7
          call_once(asserta(oddlist(New)))
          ;
          true
         )),
  fail
  ;
  !
  ).

/*-------------------------------------------------------
  test_oddlist will test each instance in Oddlist, if the instance's
  strength value is less than 0.5 then it is added to the error_list.
-------------------------------------------------------*/

test_oddlist(Oddlist, Msgnum):-
  (member(Frame, Oddlist),
   call_once((ret_val(Frame, strength, Strength_val), !
          ;
          Strength_val = []
         )),
   Strength_val = [Strength],      % since there is not much info available,
   call_once((Strength < 0.5,     % put instance on error list
          call_once(retract(error_list(Old))),
          call_once(parse_inst_name(Frame, Region)),
          Sublist = [Region, Msgnum],
          call_once((member(Sublist,Old),
                call_once(asserta(error_list(Old)))
                ;
                New = [Sublist | Old],
                call_once(asserta(error_list(New)))
               ))
          ;
          true
         )),
  fail
  ;
```

```
  !
  ).

test_noise_occl:-
  retract(noise_occl_list(Noise_occl_list)), !,
  asserta(noise_occl_list(Noise_occl_list)),
  retract(shque(Stronglist)), !,
  asserta(shque(Stronglist)),
  (Noise_occl_list == []      % if noise_occl_list is not empty, then
  ;                           % test strength of each instance in strlist
  retract(xinput_list(Xlist)), !,  % if Xlist is not empty then it is known
  asserta(xinput_list(Xlist)),  % that input was subject to noise. Report
  (Xlist == [],               % appropriate error message in test_oddlst.
   Num = 1
   ;
   Num = 2
  ),
  test_oddlist(Stronglist, Num)
 ).


/*-----------------------------------------
  Odd reflects those primitives that have not been matched.
-----------------------------------------*/

get_input_regions:-
  asserta(strlist_reg([])),   % list of regions matched in strong list
  asserta(region([])),        % list of regions in input list
  retract(input_list(L)), !,
  asserta(input_list(L)),
  get_regions(L).

get_regions(List):-
  (member(Term, List),
   call_once(add_region(Term)),
   fail
   ;
   !
  ).

add_region(Term):-
  functor(Term, Functor, _),
  arg(1, Term, Region),
  retract(region(Reglist)), !,
  Newlist = [Region | Reglist],
  asserta(region(Newlist)).


/*-----------------------------------------------
  get_unmatched will return a list of regions that were not matched
  in Oddlist.
-----------------------------------------------*/
```

```
get_unmatched(Oddlist):-
  retract(shque(Strlist)), !,
  asserta(shque(Strlist)),
  retract(region(Reglist)), !,
  asserta(region(Reglist)),
  get_reg_name(Strlist),
  retract(strlist_reg(Strlist_reg)), !,
  asserta(strlist_reg(Strlist_reg)),
  difference(Reglist, Strlist_reg, Oddlist).   % Oddlist contains what is in
                                               % Reglist and NOT in Strlist_reg
get_reg_name(List):-
  (member(Frame, List),
   call_once(parse_inst_name(Frame, Region)),
   call_once(retract(strlist_reg(Old))),
   New = [Region | Old],
   call_once(asserta(strlist_reg(New))),
   fail
   ;
   !
   ).
```

```
/*-------------------------------------------------------
  Get_occ is passed in an Element and Component and returns a Number
  which reflects the object-to-component correspondence that is found
  in the facts loaded into the internal database from the file xkb1.
-------------------------------------------------------*/

get_occ(Element, Component, Number):-
  (ret_val(Element, in_of, Element_type), !
   ;
   Element_type = []
  ),
   (Element_type == [],
    Etype = Element
    ;
    Element_type = [Etype]
   ),
  (ret_val(Component, in_of, Component_type), !
   ;
   Component_type = []
  ),
  (Component_type == [],
    Ctype = Component
    ;
    Component_type = [Ctype]
   ),
  Term =.. [occ, Etype, Ctype, Number],
  call(Term).

/*-------------------------------------------------------
  Check_occ is passed an Element, Component, Element_node and a list of
  Nodes.  It checks the occ of the Element and Component and if it is
  greater than 1, then it adds all instances in Nodes to the component
  slot of Element_node in the procedure match_all and then updates the
  status of Element_node by calling update_match_status.  If the occ
  value is 1, then match_instance is called to add the instance Node to
  the component slot of Element_node.
-------------------------------------------------------*/

check_occ(Element, Component, Element_node, Nodes):-
  get_occ(Element, Component, Number),
  (Number > 1,
   match_all(Element_node, Nodes),
   update_match_status(Element_node)
   ;
   Nodes = [Node | Tail],
   match_instance(Element_node, Node)
  ).

/*-------------------------------------------------------
  Match_all takes Element and a list of Nodes and adds each Node in the
  list to the components slot of Element.
-------------------------------------------------------*/
```

```
match_all(Element, Nodes):-
 (member(Node, Nodes),
  call_once(ret_val(Element, components, Arglist)),
  call_once((member(Node, Arglist)
           ;
           call_once(add_val(Element, components, Node))
          )),
  fail
  ;
  !
 ).

/*-------------------------------------------------
 Get_comp_prob obtains the probability that X is composed of Y by
 utilizing the facts that are loaded into the internal database
 from the file xkb1, the auxiliary knowledge base.
 Facts are in the form of prob_comp_of(X, Y, N). Prob is returned
 to the calling procedure, while Element and Component are passed
 into to this procedure.
-------------------------------------------------*/

get_comp_prob(Element, Component, Prob):-
 (ret_val(Element, in_of, Element_type), !
  ;
  Element_type = []
 ),
  (Element_type == [],
   Etype = Element
   ;
   Element_type = [Etype]
  ),
 (ret_val(Component, in_of, Component_type), !
  ;
  Component_type = []
 ),
 (Component_type == [],
   Ctype = Component
   ;
   Component_type = [Ctype]
 ),
 Term =.. [prob_comp_of,Etype,Ctype, Prob],
 call(Term).


/*-------------------------------------------------
  If there is a strong probability that Component is a component of
  Element and Component is non-existent, then mark Element as
  non-existent also, by changing its status value to -1.
-------------------------------------------------*/

test_prob(Element, Component, Newstatus):-
 get_comp_prob(Element, Component, Prob), !,
 (Prob >= 8,
```

```
 (Newstatus == -1,
  change_val(Element, status, _, Newstatus)
  ;
  true
 )
 ;
 ret_val(Element, status, Element_status),
 Element_status = [Elstatus],
 (Elstatus == -1
  ;
  update_status(Element)
 )
).


/*-------------------------------------------------------
  Update status of Element by bypassing an existing weak component,
  i.e. do not add instance of component to instance of Element,
  but do account for its weak presence in the status of Element.
  Add instance to list of unmatched components for that Element.
  Unmatched is a list of lists that have the format [Arg, Num] where
  Arg is an instance and Num is the number of unmatched components.
  Unmatched = [[Arg1, Num1], [Arg2, Num2], ... , [Argn, Numx]].
-------------------------------------------------------*/

update_status(Element):-
 retract(comp_match(List)), !,
 Term = [Element, Total, Num_matched],
 (member(Term, List),
  (Total == 0,   % if there are no components to match, then Element is
   Ratio is 1,   % a terminal node of a composition hierarchy
   change_val(Element, status, _, Ratio),
   asserta(comp_match(List))
   ;
   Newlength is Num_matched + 1,
   del(Term, List, Templist),
   Newterm = [Element, Total, Newlength],
   Newtemp = [Newterm | Templist],
   asserta(comp_match(Newtemp)),
   Ratio is Newlength / Total,
   (Ratio > 1,
    Newratio is 1
    ;
    Newratio is Ratio
    ),
   change_val(Element, status, _, Newratio),
   retract(unmatched(Old)), !,
   Term2 = [Element, Number],   % add unmatched component to list
   (member(Term2, Old),
    del(Term2, Old, Temp),
    Newnum is Number + 1,
    Term3 = [Element, Newnum],
    New = [Term3 | Temp],
```

```
  asserta(unmatched(New))
  ;
  asserta(unmatched(Old))
 )
)
;
asserta(comp_match(List))    % if term is not a member of list
).
```

```
/*-------------------------------------------------------
  This file contains the modules necessary to report the output.
  The output is a symbolic interpretation of the input.
  These modules essentially traverse the composition hierarchies
  that represent the final interpretation.
-------------------------------------------------------*/

get_comp_top:-
  retract(strlist(List)), !,
  asserta(strlist(List)),
  asserta(final_comp([])),
  get_compnodes(List).

/*-------------------------------------------------------
  get_compnodes takes each instance in List and finds the topmost
  node in the composition hierarchy and adds it to final_comp list.
-------------------------------------------------------*/

get_compnodes(List):-
 (member(Arg, List),
  call_once(find_top_node(Arg, _, Top)),
  call_once(get_nodes(Top, Topnodes)),
  call_once(Topnodes = [Topnode | Tail]),
  call_once((ret_val(Topnode, status, Status_val), !; Status_val = [])),
  call_once(Status_val = [Status]),
  call_once((Status >= 0.8,
          call_once(retract(final_comp(Old))),
          call_once((member(Topnode, Old),
                    New = Old
                    ;
                    New = [Topnode | Old]
                   )),
          call_once(asserta(final_comp(New)))
         ;
         true
        )),
  fail
  ;
  !
  ).

/*-------------------------------------------------------
  check_errorlist checks the error list and reports the corresponding
  error message according to the error code associated with the instance
  in the error list.
-------------------------------------------------------*/

check_errorlist:-
  retract(error_codes(Codes)), !,   % error msgs tagged with a number
  asserta(error_codes(Codes)),
  retract(error_list(Err)), !,      % actual errors encountered during this
  asserta(error_list(Err)),         % session   eg. [r9n9, 1].
  (Err == [],
```

```
  nl, write('NO ERRORS TO REPORT.'), nl
  ;
  nl,
  report_errors(Codes, Err)
 ).

/*-------------------------------------------------------
  report_errors writes any error messages to the appropriate output file.
-------------------------------------------------------*/

report_errors(Codes, Err):-   % write error message
  (member(Term, Err),
   Term = [Frame, Num],
   call_once(find_err_msg(Codes, Num, Sentence)),
   write(Frame),tab(2),write_sent(Sentence),
   fail
   ;
   !
  ).

/*-------------------------------------------------------
  find_err_msg searches the list of error codes to find the error
  message to report to the user.
-------------------------------------------------------*/

find_err_msg(Codes, Num, Sentence):-
(member(Term, Codes),
  Term = [Number, Msg],
  call_once((Num == Number,
             Sentence = Msg
             ;
             fail
           ))
  ).

/*-------------------------------------------------------
  report_output is responsible for reporting the final symbolic
  interpretation of the input to the user.
-------------------------------------------------------*/

report_output:-
  Outfile = outfile,
  tell(Outfile),
  write_sent([final,interpretation,of,input,is]),
  nl,
  retract(final_comp(List)), !,
  asserta(final_comp(List)),
  retract(strlist(Strlist)), !,
  asserta(strlist(Strlist)),
  write_comp(List, Strlist),
  check_errorlist,
  tell(user).
```

```
/*-------------------------------------------------------------
  write_comp writes the current frame name unless the instance has
  components, in which case the routine is called recursively.
-------------------------------------------------------------*/

write_comp([], Strlist):- !.

write_comp(List, Strlist):-
  List = [Arg | Tail],
  nl,
  write_frame([Arg], 0),
  get_components(Arg, Complist),
    (Complist == []
     ;
     intersect(Complist, Strlist, Result),
     (Result == []
      ;
      write(' which is composed of '), nl,
      write_frame(Complist, 5),
      write_comp(Complist, Strlist)
     )
    ),
  write_comp(Tail, Strlist).

/*-------------------------------------------------------------
  write_frame writes the name of a frame to the current output file.
-------------------------------------------------------------*/

write_frame(List, Indent):-
 (member(Arg, List),
  ret_val(Arg, status, Status_val), Status_val = [Status],
  (Status >= 0.8,
    ret_val(Arg, in_of, Frametypes),
    Frametypes = [Frametype],
    tab(Indent), write(Frametype), nl
   ;
   true
  ),
  fail
  ;
  !
  ).
```

```
/*-----------------------------------------------------
  Input Generator Tool.

  The purpose of the modules contained in this file is to generate
  separate input files to be processed by the high-level inference
  system.

  The user is prompted to enter the regions to be included in the
  input file.  Once each region is checked for validity, it is
  stored in the list called new_region_list.
  After valid regions are gathered, then all pertinent information,
  i.e., all relational and attribute knowledge, regarding those
  selected regions are placed in the new input file also.
-----------------------------------------------------*/

input_tool:-
  write('Enter regions in input file.'),nl,
  write('Enter end when you are done.'), nl,
  get_valid_regions,
  asserta(new_region_list([])),
  retract(valid_regions(Regions)), !,
  asserta(valid_regions(Regions)),
  call_once(get_input(Regions)),
  Infile = infile,
  call_once(get_region_input(Infile)),
  gather_input(Infile),

  asserta(noise_occl_list([])),
  check_response(Infile).

/*-----------------------------------------------------------
  The following modules comprise the code that allows the user to subject
  the input to noise or occlusion.  Initially, the user is prompted to
  subject input to noise or occlusion.  The response is error checked and
  the appropriate module is then called where the user is asked to enter
  the primitives that are to be subject to noise or occlusion.  These
  primitives are stored in the list, noise_occl_list([]).
-----------------------------------------------------------*/

check_response(Infile):-
  write('Would you like to subject input to noise? '), nl,
  error_check_response(Valid_response),
  Prompt1 = noise,
  (Valid_response == y,
   call_input_module(Prompt1, Infile)
  ;
   asserta(xinput_list([])),
   write('Would you like to subject input to occlusion? '), nl,
   Prompt2 = occlusion,
   error_check_response(Valid_resp),
   (Valid_resp == y,
    call_input_module(Prompt2, Infile)
    ;
```

```
    true
   )
  ).

error_check_response(Valid_response):-
  read(Response),
  (Response == y,
   Response = Valid_response
   ;
   (Response == n,
    Response = Valid_response
    ;
    write('Invalid response.  Enter again.'), nl,
    error_check_response(Valid_response)
   )
  ).

call_input_module(Prompt, Infile):-
  (Prompt == noise,
   noise_module
   ;
   (Prompt == occlusion,
    occlusion_module,
    search_infile(Infile)
    ;
    true
   )
  ).

noise_module:-        % load extraneous input into internal database
  [xinput],
  add_xinput.

add_xinput:-
  retract(input_list(L1)), !,
  retract(xinput_list(L2)), !,
  asserta(xinput_list(L2)),
  conc(L1, L2, L3),
  asserta(input_list(L3)).


occlusion_module:-
  write('Enter primitives to subject to occlusion.'), nl,
  write('Enter end when done.'), nl,
  retract(new_region_list(New_region_list)), !,
  asserta(new_region_list(New_region_list)),
  get_second_input(New_region_list).

get_second_input(List):-
  repeat,
  read(Term),
  call_once((Term == end,
          !
```

```
        ;
        call_once(check_term(Term, List)),
        fail
       )).

check_term(Term, List):-
  (member(Term, List),
   retract(noise_occl_list(Old)), !,
   (member(Term, Old),
    write(Term), write(' is in list already.'), nl,
    asserta(noise_occl_list(Old))
    ;
    New = [Term | Old],
    asserta(noise_occl_list(New))
   )
   ;
   write(Term), write(' is an invalid region.'), nl
  ).

/*-------------------------------------------------------
  The following modules take each primitive in the noise_occl_list and
  search the primary input file, infile1, for corresponding
  relationships.  If any are found, then they are deleted from Prolog's
  internal database, but they remain in the newly created input file
  for reference purposes.

-------------------------------------------------------------------*/

search_infile(Infile):-
  retract(noise_occl_list(Noise_occl_list)), !,
  asserta(noise_occl_list(Noise_occl_list)),
  see(infile1),                 % open infile1 for reading
  read(Term),                     % read past first term in input file
  search_for_cues(Infile, Noise_occl_list),
  seen,                         % close file for reading
  see(user).                     % redirect input to screen

search_for_cues(Infile, Noise_occl_list):-
  repeat,
  read(Term),                       % read each term in infile1
  call_once((Term == end_of_file,
        !
        ;
        call_once(delete_cues(Term, Infile, Noise_occl_list)),
        fail
       )).


delete_cues(Term, Infile, Noise_occl_list):-
  Term =.. [Functor | Args],       % check if a term has a primitive that
  length(Args, Args_length),       % is found in the noise_occl_list
  (Args_length == 1                % attribute eg. blue(r9)
   ;
```

```
  Args = [Arg1, Arg2],          % relationship eg. above(r9,r5)
  (member(Arg1, Noise_occl_list), % if primitive is in noise_occl_list
   retract(Term)                % then delete term from database
   ;
   true
  ),
  (member(Arg2, Noise_occl_list),
   retract(Term)
   ;
   true
  )
 ).

/*-------------------------------------------------------
  get_valid_regions will create a master list of valid regions
  according to the input list in Infile1.
-------------------------------------------------------*/

get_valid_regions:-
  asserta(valid_regions([])),
  see(infile1),
  read(T),
  seen,
  T =.. [ input_list, Args | Tail],
  build_region_list(Args).

build_region_list(Args):-
 (member(Term, Args),
  call_once(build_regions(Term)),
  fail
  ;
  !
  ).

build_regions(Term):-
  functor(Term, Functor, _),
  arg(1, Term, Region),
  retract(valid_regions(Old)), !,
  New = [ Region | Old],
  asserta(valid_regions(New)).

/*-------------------------------------------------------
  get_input will allow a user to enter a region primitive and
  then calls check_region to check the validity of the entry.
-------------------------------------------------------*/

get_input(Regions):-
  repeat,
  read(Term),
  call_once((Term == end,
          !
          ;
          call_once(check_region(Term, Regions)),
```

```
          fail
         )).

check_region(Newregion, Regions):-
  (member(Newregion, Regions),
   retract(new_region_list(Oldlist)),!,
   (member(Newregion, Oldlist),
    write(Newregion), write(' is in list already.'), nl,
    asserta(new_region_list(Oldlist))
    ;
    Newlist = [Newregion | Oldlist],
    asserta(new_region_list(Newlist))
   )
   ;
   write(Newregion), write(' is an invalid region.'), nl
  ).

/*-----------------------------------------------------------------
  get_region_input will store the new input list in the file Infile.
  Input_list is the original input_list.
-----------------------------------------------------------------*/

get_region_input(Infile):-
   see(infile1),
   read(T),
   seen,
   T =.. [input_list, Input_list | Tail],
   retract(new_region_list(Newregion_list)),!,
   asserta(new_region_list(Newregion_list)),
   asserta(new_input_list([])),
   call_once(check_terms(Input_list, Newregion_list)),
   retract(new_input_list(Inlist)), !,
   asserta(new_input_list(Inlist)),
   asserta(input_list(Inlist)),
   tell(Infile),
   write(input_list(Inlist)), nl,
   tell(user).

check_terms(Input_list, Newregion_list):-
  (member(Term, Input_list),
   call_once(Term =.. [Reg_type, Region]),
   call_once((member(Region, Newregion_list),
              call_once(retract(new_input_list(Temp))),
              New_input = [Term | Temp],
              call_once(asserta(new_input_list(New_input)))
              ;
              true
             )),
   fail
   ;
   !
   ).
```

```
/*-----------------------------------------------------------------
 gather input will gather the corresponding attributes and relationships
 associated with each region primitive in the new input list.
-----------------------------------------------------------------*/

gather input(Infile):-
 retract(new region list(Newregion list)), !,
 asserta(new region list(Newregion list)),
 see(infile1),
 read(Term),          % read past first Term in input file - input list([]).
 get rel attr(Infile, Newregion list),
 seen,
 told,
 see(user),           % redirect input and output to screen
 tell(user).

get rel attr(Infile, Newregion list):-
 repeat,
 read(Term),          % read remaining Terms
 call once((Term == end of file,
          !
          ;
          call once(check input(Term, Infile, Newregion list)),
          fail
         )).


check input(Term, Infile, Newregion list):-
  Term =.. [Functor | Args],
  length(Args, Args length),
  (Args length == 1,          % attribute: eg. blue(r24).
   Args = [Region | Rest],
   (member(Region, Newregion list),
    asserta(Term),
    tell(Infile),
    write(Term), nl
    ;
    true
   )
   ;
   Args = [Arg1, Arg2 | Tail],     % relationship:  eg. above(r11, r4).
   (member(Arg1, Newregion list),
    (member(Arg2, Newregion list),
     asserta(Term),
     tell(Infile),
     write(Term), nl
     ;
     true
    )
    ;
    true
   )
  )
```

```
/*-----------------------------------------------------------------

  Utility programs

-----------------------------------------------------------------*/


max(X, Y, X):-          % find the maximum of two numbers
 X >= Y.

max(X, Y, Y):-
 X < Y.

maxlist([X], X). % find the maximum of a list of items

maxlist([X, Y | Rest], Max):-
  maxlist([Y | Rest], Maxrest),
  max(X, Maxrest, Max).


collect_list(Arg, Tval):-
  frame_subsumes(Arg, Val), !,
  Tval = [Val | Rest], write(Tval),
  collect_list(Arg, Rest).


difference([], _, []).        % return L, the difference between the first
                              % two argument lists
difference([X | L1], L2, L):-
  member(X, L2), !,
  difference(L1, L2, L).

difference([X | L1], L2, [X | L]):-
  difference(L1, L2, L).


intersect(X, [], []).                 % the third argument is returned as
                                      % a list of elements common to the first
intersect([], Y, []).                 % two argument lists

intersect([X | L1], L2, L):-
  (member(X, L2),
  L = [X | Rest],
  intersect(L1, L2, Rest)
  )
  ;
  intersect(L1, L2, L).


list_length([], 0).           % return N, the length of the first
                              % argument list

list_length([ _ | Tail], N):-
  length(Tail, N1),
```

```
N is 1 + N1.


conc(L, [], L).              % concatenate the first two argument lists and
                             % return the third argument list
conc([], L, L).

conc([X|L1], L2, [X|L3]):-
  conc(L1, L2, L3).


del(X, [X | Tail], Tail).    % delete X from the second argument list
                                    % and return the third argument list
del(X, [Y | Tail], [Y | Tail1]):-
   del(X, Tail, Tail1).


shift([First | Rest], Shifted):-     % shift the first element of the
  conc(Rest, [First], Shifted).      % first argument list and return
                                           % the list 'Shifted'

fmdall(X, Goal, Xlist):-
      call(Goal),
      assertz(queue(X)),
      fail
      ;
      assertz(queue(bottom)),
      collect(Xlist).

collect(L):-
   retract(queue(X)), !,
   ( X == bottom, !, L = []
     ;
     L = [X | Rest], collect(Rest)
   ).


isempty([]).

add(X,L,[X | L]).


member(X, [X | Tail]).       % goal succeeds if X is a member of the
                             % second argument list
member(X, [Head | Tail]) :-
   member(X, Tail).
```