Analyzing Perspective Line Drawings using Hypothesis Based Reasoning

by

Prasanna Govind Mulgaonkar

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science and Applications

APPROVED:

_____

Linda G. Shapiro

_____ _____

Robert M. Haralick John W. Roach

_____ _____

Roger W. Ehrich Ezra A. Brown

August, 1984

Blacksburg, Virginia

ANALYSIS OF PERSPECTIVE LINE DRAWINGS USING HYPOTHESIS BASED REASONING

by

Prasanna Govind Mulgaonkar

Committee Chairman: Dr. Linda G. Shapiro

Computer Science

(ABSTRACT)

One of the important issues in the middle levels of computer vision is how world knowledge should be gradually inserted into the reasoning process. In this dissertation, we develop a technique which uses hypothesis based reasoning to reason about perspective line drawings using only the constraints supplied by the equations of perspective geometry. We show that the problem is NP complete, and that it can be solved using modular inference engines for propagating constraints over the set of world level entities. We also show that theorem proving techniques, with their attendant complexity, are not necessary because the real valued attributes of the world can be computed in closed form based only on the spatial relationships between world entities and measurements from the given image.

ACKNOWLEDGEMENTS

..... For they were the best of times
and they were the worst of times

A dissertation is not created in isolation. It is the product of the interaction between the people surrounding the author, and the times and the places involved. I would like to thank everyone who in one way or the other contributed to making my time as a graduate student worthwhile and in some direct or indirect way helped make this dissertation what it is.

Dr. Linda Shapiro, and Dr. Robert Haralick guided me throughout my graduate career. All that I have learnt about computer vision, is directly attributable to their time and patience. Dr. Shapiro supported my graduate education by continuous funding on her NSF grant which is what enabled me to survive. I also want to thank Dr. Roger Ehrich for serving on both my masters and my doctoral advisory committees. Dr. John Roach, and Dr. Ezra Brown also provided valuable input to this work.

I am deeply indebted to my wife Anjali to whom this work is dedicated. Without her constant encouragement and support, I would not have been able to go through all the heartaches and headaches that a PhD. requires. Amit, our three

month old son did his share by making sure that I stayed awake on the nights that I needed to. Indeed this is a family PhD. rather than an individual accomplishment.

To all the friends at the SDA Lab, who provided a friendly and stimulating working environment, I wish to say "Thank You". Especially, Ting-Chuen (King of Threshold) Pong, Greg (IBM Spy) Bushman, Greg (Mr. GIPSY) Fabregas, Tom Laffey, Trinh Ho, Einar Ostevold, and all the other folks helped make the last five years something I can look back on. Especially Hyonam Joo, who helped write, debug and often fix on very short notice, the PROLOG interpreter which was used for all the experiments described in this dissertation.

Life has facets other than just academic, and we owe a great deal to all our other friends in Blacksburg. Jyothi and Ramesh, and Nileema and Rajendra made extra-curricular life enjoyable and fun.

Finally I want to thank my parents for instilling in me the spirit of hard work and striving for academic excellence. Their preparatory work led up to this moment. I would also like to acknowledge the help that my in-laws gave. They looked after our baby and our house in the crucial months leading up to my defense so that I could be free to concentrate on my work and my wife could finish her master's degree without any distractions.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter I

INTRODUCTION

The central problem in computer vision, is one of producing an automatic system capable of analyzing unconstrained pictorial information of the same complexity and with the same accuracy as the human visual system. This is currently an unsolved problem. However, we do know that the solution of this problem must involve many diverse knowledge sources and applications of techniques from a wide variety of disciplines. Raw picture data consists of brightness or color values measured on a discrete grid of data points, which is generated by some sort of camera or other electronic imaging device. This grid of data values is called the image. It is a projection of the actual scene presented to the camera, which is corrupted by noise, and affected by other irregularities in the imaging process. The task of low level image processing is to extract from this noisy image, the actual signal and to decompose the signal into meaningful entities. Computer vision systems draw heavily from the signal processing and related fields to perform this task. The descriptions of the data extracted from the image could consist of primitive topographic information, structures such as lines, arcs and regions, and information about the spa-

tial arrangement of these entities. The kind of knowledge required to efficiently perform this task, involves knowledge about the physics of the sensors involved, characteristics of the noise patterns, and the types of entities that the higher processing levels require.

At the very top level in the computer vision hierarchy, the task is to put together all the information culled from the image by the lower levels in an attempt to identify and name the components of the scene. At this stage, the systems must use knowledge about the world, possibly stored as models of things that are likely to be present in the scene along with knowledge relating the various objects possible in the world. For example, what does a telephone look like? What is the usual spatial relationship between a telephone and a desk? Would one expect to find a telephone in an office environment or in an outdoor picture of a road? Techniques such as organizing models for rapid matching, determining which of the candidate models are likely to be the ones visible, and generating criteria for defining closeness of matches are required at the top level. In addition, artificial intelligence techniques for reasoning about the world are necessary for propagating the effects of analyzing one part of the image to the others. Techniques are necessary for incorporating fuzzy and partially defined relationships

between the entities of the world and to include expectations about the contents of the scene from spatial and temporal context.

Considerable work has been done by the various subspecialties of computer vision at both these ends of the spectrum, and there exists a good understanding of the issues involved. It is the middle grey area that researchers are just beginning to tackle. It seems clear now that as the information flows from the lower levels to the higher levels, two things happen. First the quantity of data decreases as more and more information gets abstracted from it. Secondly, analysis of the data takes into account greater and greater amounts of knowledge of the world being imaged. What is not clear is exactly how much world information is necessary or indeed should be used at each stage, and what this knowledge should consist of. Is it necessary to have object models to be able to do effective image processing such as noise removal? Or is it sufficient to know only the nonlinearities in the imaging circuits of the camera?

We feel that the ideal situation is one where world knowledge is introduced into the system in gradual steps. As we move higher in the processing hierarchy, the scope of the information used should widen. Coupled with this should be feedback from the higher levels to control the reasoning ac-

tions of the lower levels of processing. This dissertation deals with that part of processing in the middle levels of computer vision, where the knowledge about the perspective projection process is first introduced. We assume that we are given an input consisting of entities extracted from the image, and we show how the knowledge of the projection process can be introduced into the analysis.

We do not need to know much about the world at this stage. We do not have models of the structures in the world. All we know is that there are certain physical entities in the world which are mapped by the process of perspective projection onto entities in the image. On the basis of our knowledge of the projection process, we can constrain how the world entities could be organized in the scene. We make use of the fact that not all arrangements are physically possible, and of those that are possible, not all arrangements can be such that when the projection process is applied to them, the given image properties would result.

Entities in the world are normally not related by discrete relationships. There are real valued properties that can be used to describe the configurations of entities possible in space. We show that since the projection process is a well defined physical system, it is often possible to compute the values of the attributes based on the discrete com-

ponents of the relationships between them and on the measurements made by lower level processes on the image. Consequently, it is not necessary to regard the reasoning process as proving theorems over real valued attributes. The process can be regarded as a controlled combinatorial search over the discrete components of the world relationships.

We feel that human beings formulate interpretations for what they see by hypothesizing the relationships that could exist between the entities in the world. Hypotheses are formulated based on knowledge of the types of relationships that we know from experience, exist in the environment. The hypotheses that are inconsistent with the sensory data are ruled out. Those that remain, are retained as possible interpretations.

The first chapter of this dissertation discusses the essence of hypothesis based reasoning in domains where the relationships between the world entities have real valued attributes. We show that the process of hypothesis generation and testing is NP complete in cases where the attribute values can be calculated rather than having to be searched for. We formalize the meaning of consistency and introduce the concept of modular inference engines whose task it is to analyze possible hypotheses. We discuss the search space of all hypotheses and show how it may be pruned for efficient reasoning.

The second chapter deals with the task of controlling the inference engines so as to minimize the overhead involved in executing their tasks in a sequential programming environment. The third chapter deals with the knowledge about perspective projection that goes into the inference engines. We show that although the inverse of the perspective projection is not unique, hypotheses made about the spatial arrangement of entities in space can be used to compute the inverse in various cases. The next chapter discusses the actual implementation aspects of the system, and shows several examples of the system in action.

We finish with a review of the work of other researchers in the field of knowledge based vision. That chapter shows how other researchers view the problems of the field and discusses their efforts to incorporate world knowledge into the computer vision task.

Chapter II

HYPOTHESIS BASED REASONING FOR VISION

2.1 INTRODUCTION

One of the central aspects of intelligence is the ability to reason about and operate in a real world environment using sensory information. We have the ability to use various types of visual, auditory, and tactile information to understand the contents of the world around us. The process by which such information is generated from the world is a deterministic physical process. However information is lost in the process, since there is not a unique combination of objects in the world which could give rise to a particular combination of sensory patterns.

The reason for our capacity to compute the inverse mapping for any sensory process seems to be that the three-dimensional world has some regularity and that the physical system which produces the stimulus strongly constrains the possible arrangement of the three-dimensional world. Of all the possible ways in which the stimulus could be interpreted, only a few are actually consistent with the physical process involved. For example, consider the problem of interpreting the contents of a perspective image. Mathematically the inverse projection is not unique. However, humans

have no trouble in recognizing the contents of even fairly noisy pictures.

We believe that the process of interpretation can be considered to be one of formulating "educated guesses" as to the nature of the three-dimensional objects in the world and utilizing the constraints of three-dimensional coherence and world knowledge about the physical processes involved to select the reasonable arrangements from possible guessed arrangements. In essence, we hypothesize what the world could consist of and retain only the hypotheses which are consistent with the data.

In this chapter we present a computational framework for hypothesis generation and testing. We formally define the notion of a hypothesis and develop the space of all hypotheses over which the interpretation process searches for the interpretation. The search problem is NP complete. However the search space is structured by the constraints of the physical sensory system and this structure can be utilized to reduce the size of the search involved. Examples of the computational paradigm are presented where appropriate, based on the problem of analyzing line drawings of perspective projections of three-dimensional objects.

2.2 HYPOTHESIS GENERATION AND TESTING

The basic idea behind the concept of hypothesis generation and testing is simple. There is some physical world which consists of several entities and various ways in which these entities relate to each other. The actual relationships between the entities of the physical system are unknown. What is known is the types of relationships that they can participate in. The actual physical system is not accessible for the purpose of determining the relationships between the entities. However, through some well-defined process, it is possible to make some measurements which depend on the relationships. For example, the mass-spectrogram of a molecule depends on the physical properties of the molecule. A projected image of a three-dimensional object depends on the spatial relationships between the parts of the object. An interpretation of the observations consists of determining the unknown relationships between the entities. If these interpreted relationships are correct, they would give rise to the observations. (This was what the Greeks termed "saving the phenomena".)

What we know is the types of entities that can exist in the world and the semantics of the attributed relations that can exist between the entities. What we do not know is which relationships hold between which of the entities that are

present in the observed instance of the world, nor do we know the attributes of the relationships. What we observe depends not only on the specific instance of the world being observed, but also on the physical process which translates the relationships between the world entities into the sensory data. Using the terminology of database systems, we know the domains from which the entities are drawn. We know the names of the relations, the order of the relations, and the domains for each column of each relation. We also know the semantics associated with the relations. This information is termed the "intention" of a relation in the field of database systems (Dat81). What we want to determine is the "extension" of the relations, that is, the instances of the tuples which belong in each of the relations. For example, we may know that the world consists of several different types of atoms, and the relations that these atoms can participate in. The observations consist of the mass spectrogram which is produced by a physical process and which depends on the relationships between the atoms. The aim of the molecular structure determination is to determine the exact relationships between the atoms in a given molecule.

2.3 THE OPTIMAL HYPOTHESIS PROBLEM

An interpretation is a hypothesis which is consistent and is "optimal" in some sense. Although the treatment in this section is formal, it is helpful to keep in mind the intended physical interpretations for the symbols. To that end, consider the application of the hypothesis system to the problem of determining the three-dimensional object structure which gives rise to an observed perspective image.

The world being sensed consists of a set of entities and the attributed relationships between the entities. Let E be the finite set of entities $E = \{e_1,\ldots,e_n\}$ about which we want to reason. In computer vision, these entities can be three-dimensional lines, planes, and points as well as two-dimensional entities such as projected arcs.

The true but unknown state of the world is specified by a union of named attributed relations between these entities. Each relation $\underline{r}_i$ in this union, has a name r_i, and an attribute set a_i. The number of entities which $\underline{r}_i$ relates is the order of $\underline{r}_i$ and we denote it by o_i. Thus each $\underline{r}_i$ can be represented as a subset

$$\underline{r}_i \subseteq \{r_i\} \times E^{o_i} \times \mathbb{V}_i$$

where each $f \in \mathbb{V}_i$ is a single valued relation which associates a real number with each attribute in a_i. That is, $f \subseteq a_i \times \mathbb{R}$. This associated number is called the attribute value. The state of the world is then represented by

$$\underline{R} = \bigcup_{i=1,m} \underline{r}_i .$$

In the case of the perspective geometry problem, the relations are spatial relations such as parallel lines, groups of lines in a plane, or pairwise touching lines or arcs. These relations have attributes such as distance between lines, coordinates of the starting and ending points or directions of normals to planes.

The information given to the interpretation process consists of the relation names, orders of the relations, names of the relation attributes, the entities in the observed instance of the world, and measurements made by the sensing process. The interpretation process constructs the best guesses for the unknown world state $\underline{R}$. The basis for the interpretation is the observed measurements and the known form or semantics of $\underline{R}$

Let P be the set of all possible instances of tuples from any relation. Thus

$$P = \bigcup_{i=1,m} [\{r_i\} \times E^{o_i} \times \mathbb{U}_i]$$

Any state of the world is a subset of the set P. The interpretation process must effectively search all subsets of P to find the best guesses for $\underline{R}$. This is an enormous search problem.

However, by effectively using the knowledge about the semantics of the relationships between the world entities, it is possible to reduce the search space considerably. Because the process which generates the sensory data from the real world is a well defined physical process, the combinatorial search for interpretations does not have to be performed over the entire space of subsets of P defined above. The measurements and the physics of the projection process relate the attribute value relations in the tuples in $\underline{R}$ to the entities involved in the relations. It is possible to use expert knowledge encoded as inference engines to compute large portions of the attribute value relations in the last component of each tuple given only the measurements and a set of tuples complete except for the last component as being in $\underline{R}$. This means that the relational portion of the tuples (the projection over all but the last position) determines the remaining position (the attribute value relation). Thus the combinatorial search space involved is all subsets of

$$P' = \bigcup_{i=1,m} [\{r_i\} \times E^{o_i}]$$

Tuples from P which are examined during the process of searching for an interpretation, are called <u>predictions</u>.

A hypothesis H is a subset of the set of predictions $H \subseteq P$. A hypothesis may be consistent which is denoted by C or inconsistent which is denoted I. Thus there exists a function $F:H \rightarrow \{C,I\}$ which determines if a hypothesis is consistent or not. A hypothesis is consistent if upon using all inference engines, the computed attribute value relations for the tuples in H are each single valued. We will have more to say about the nature of F later.

An interpretation I is a hypothesis set $H \subseteq P$ such that $F(H) = C$. We are interested in "best" possible interpretations I. That is, we are interested in any I which maximizes some property g(I). Best possible interpretations may not be unique.

The simplest g(h) is $g_1(H) = \#H$, the number of consistent predictions about the domain. Alternatively g(H) may be defined as the number of entities whose relations are constrained by H.

$$g_2(H) = \#\{e \mid \text{there exists a tuple } (r_x, e_1, e_2, \ldots e \ldots, e_{o_x}, \mathbb{U}_x) \in H\}$$

We could also consider a weighted sum of these two elements. The only requirement on the function g(H) is that it be bounded from above. With only that restriction, any appropriate function can be used.

In the next few sections, we show that with this formalization, the problem of determining the optimal hypothesis can be shown to be an NP complete problem, provided that consistency of a hypothesis can be determined in polynomial time. The proof uses the fact that the consistent labeling problem (CLP) is a known NP complete problem (Ull81) and constructs a polynomial time transformation from the CLP to the hypothesis determination problem.

2.3.1 NP Completeness of the Optimal Hypothesis Problem

In this section, we will show that the problem of determining the optimal consistent hypothesis is in fact an NP complete problem. The solution of the problem forms two disjoint parts. As we showed in the previous section, the problem of determining the consistent relational tuples and the problem of determining the values for the unknown attributes of those tuples, can be separated out. Given a set of tuples with unknown attributes, we can use our knowledge of the physical processes involved, to compute values for the attributes. Thus, if it is possible to determine the attribute values in polynomial time, the complexity of the entire process can be no worse than the complexity of the process which determines the subsets of tuples to be examined. In this section we assume, for the moment, that we can indeed

determine consistency of a hypothesis and the values for the attributes in polynomial time. The fact that this is so, will be shown in the next section. We show that the problem of searching over the set of tuples is NP complete under that assumption.

We show that the consistent labeling problem (CLP) (Haralick & Shapiro 1979) can be transformed into the hypothesis testing problem. First we restate the CLP in a form slightly different from that presented in the paper (Ullman81). Next we show that the reformulation is identical to that in the afore mentioned paper (Ullman81) in the sense that the solution to the new formulation is also a solution to the original formulation and vice-versa. Next we show that the CLP can be transformed into the hypothesis formation framework.

2.3.1.1 Formal Definition of CLP

The formal definition of the consistent labeling problem is as follows:

CLP = {U, L, T, R} where U is a finite set of units $\{u_1,..u_n\}$, and L is a finite set of labels $\{l_1,...l_m\}$. The set $T \subseteq \{f \mid f \subseteq U\}$ is the set of mutually constrained subsets of units. T is called the unit constraint set. The set R is the allowable set of labelings for the sets of units in the unit constraint set. $R \subseteq \{g \mid g \subseteq U \times L$, g single valued and $Dom(g) \in T\}$.

A consistent labeling h is a function h:U-->L such that $\forall$ $f \in T$, $h|_f \in R$.

For complete details on the formalism and on the proof of the NP completeness of consistent labeling using this formulation, the reader is referred to Ullmann (1983) and Haralick and Shapiro (1979).

2.3.1.2 Reformulation of the CLP

We define a consistent labeling problem CLP*= $\{U,L,T,R^*\}$ where U, L and T are as above. Note that if $f \subseteq T = \{u_a,u_b,\ldots,u_j\}$, then not all possible labelings for $u_a,\ldots,u_j$ are allowed simultaneously.

Let $R^* \subseteq \{g | g \subseteq U \times L$, where g is single valued and $Dom(g) \in T\}$, be the <u>inconsistent labeling set</u>. This means that if $g = \{(u_1,l_1),\ldots,(u_k,l_k)\} \in R^*$, then $u_1,\ldots,u_k$ are distinct units and $\{u_1,\ldots,u_k\} \in T$; and the mapping $(u_1,l_1),\ldots,(u_k,l_k)$ is one of those <u>not</u> allowed.

A consistent labeling h is a function h:U --> L such that (using the notation of Ullmann 1983)

$$\forall \; f \in T, \; h|_f \notin R^*.$$

2.3.1.3 Equivalence of the Two Formulations

To show that the two formulations of the CLP are equivalent, we need to show that for all $f \in T$, and h:U->L, $h|_f \notin R^*$ iff $h|_f \in R$ as defined in (Ull83).

First, let $p = \{x | x \subseteq U \times L\}$ be the power set of U x L. It is clear that R and R^* are both subsets of p.

Let $\bar{R} = p - R$ be the set of all subsets of UxL which are not in R. Thus $h|_f \notin \bar{R}$ iff $h|_f \in R$. Clearly $R^* \subseteq \bar{R}$. However, $\bar{R}$ includes several sets that are not single valued and whose domain is not an element of T. It suffices to show that restricting R^* to be those subsets which are single valued and whose domain is an element of T does not throw out the restrictions on the mapping that we want to retain.

In other words, if $h|_f \in \bar{R}$ then $h|_f \in R^*$.

Proof:

First, if $f \in T$, then $f \neq 0$ because by the definition of T, it consists of units which mutually constrain one another. Secondly h is defined over the complete set of units u. Thus $h|_f \neq 0$.

Further because h is single valued (function) and f is a set (no duplicates), $h|_f$ is single valued as well, and the domain of $h|_f$ is $\in$ T by the definition of restriction. Thus by the definition of R^*, $h|_f \in R^*$, if $h|_f \in \bar{R}$.

2.3.1.4 The Optimal Hypothesis Problem and the Hypothesis Decision Problem

We define the optimal hypothesis problem $H_O = (E,R,F)$ to be the problem of determining the optimal hypothesis, given the set of entities E, the possible relationships between the entities $\underline{R}$ and a function F, which when given a hypothesis, tests for consistency. In this section we assume that the complexity of F is NP at worst. Further, for simplicity of notation, we will ignore the attributes associated with the relations in R, assuming for the moment that any such attributes can be computed by the consistency checking function F. How this can be done, is shown in the next section. The optimality of a hypothesis is defined to be in terms of maximizing some function g. The only requirement placed on g is that it is integer valued and bounded from above. To determine the complexity of H_O, we consider the relation decision problem H_D, defined as $H_D = (E,R,F,B)$ where E, R and F are as before. The decision problem returns a "yes" answer if there is any consistent hypothesis possible for which $g(H) \geq B$, where B is an integer numeric bound.

2.3.1.5 Transformation of the CLP to H_D

Here we show how a CLP*= (U,L,T,R^*) can be transformed into a hypothesis testing problem $H_D = (E,R,F,B)$. Note that here we use R to refer to the relations for hypothesis checking as opposed to the unit label constraint set in the previous section.

Let $E = U \times L$. Note that $\#E = \#U \times \#L$. Thus #E is polynomially related to the number of units and labels

$$E = \{\langle u_i l_j \rangle \mid u_i \in E \text{ and } l_j \in L\}$$

Let R consist of the single relation named r of order 1.

$$R = \{r\}$$

Let $B = 0$ and $P = \{(r, \langle u_i l_j \rangle) \mid \langle u_i l_j \rangle \in E\}$. Let $H \subseteq P$ be a hypothesis.

The hypothesis testing function is defined in terms of T and R^* as follows:

$$F(H) = \begin{cases} I \text{ if } \#H \neq n \text{ (\# of units)} & (1) \\ I \text{ if there exists } i,\ 1 \leq i \leq n \mid \forall_j (r, \langle u_i l_j \rangle) \notin H & (2) \\ I \text{ if there exists } i,\ 1 \leq i \leq n \mid (r, \langle u_i l_j \rangle) \in H \text{ and } (r, \langle u_i, l_k \rangle) \in H \text{ with } j \neq k & (3) \\ I \text{ if there exists } z \in Z \mid z \subseteq H \text{ where Z is defined by: } Z = \{z \mid z = \{(r, \langle u_1 l_1 \rangle) \ldots r, \langle u_j l_j \rangle)\} \text{ and } [(u_1 l_1) \ldots (u_j l_j)]^{-1} \in R^*\} & \\ C \text{ otherwise} & \end{cases}$$

Note that (1), (2), and (3) form a redundant set. They are all specified for simplicity.

We are interested in an H such that $g(H) = \#(H) \geq B = 0$ which is consistent, i.e. $F(H) = C$.

Claim: If $H = \{(r, \langle u_a l_a \rangle)(r, \langle u_b l_b \rangle) \ldots (r, \langle u_i l_i \rangle)\}$ is the largest consistent hypothesis set for H_D, then $h = \{(u_a, l_a), (u_b, l_b) \ldots (u_i, l_i)\}$ is a consistent labeling of the CLP*.

By (1) we know that #H = n,

By (2) we have the fact that dom(h) = U,

and by (3) we know that h is single valued.

It remains to be shown that there is an $f \in T$ for which $h|_f \in R^*$ iff there exists $z \in Z$ such that $z \subseteq H$.

Let $h|_f \in R^*$ for some $f \in T$ and let

$h|_f = \{(u_1 l_1) \ldots (u_i l_i)\}$. Then by construction of h from H, it implies that

$$\{(r,\langle u_1 l_1\rangle),\ldots,(r,\langle u_i l_i\rangle)\} \subseteq H.$$

But by construction of Z from R^*, we have,

$\{(r,\langle u_1 l_1\rangle),\ldots,(r,\langle u_i l_i\rangle)\} \in Z$ because $\{(U_1 l_1),\ldots,(u_i l_i)\} \in R^*$

$\therefore$ there is a $z \in Z \mid z \subseteq H$.

Let $z \in Z$ s.t. $z \subseteq H$.

Let $z = \{(r,\langle u_1 l_1\rangle),\ldots,(r,\langle u_j l_j\rangle)\}$

$\therefore$ by construction of z, $\{(u_1 l_1),\ldots,(u_j l_j)\} \in R^*$.

==> (by the definition of R^*)

There exists an $f = \{u_1,\ldots,u_j\} \in T^*$.

$\therefore$ $h|_f$ is precisely an element in R^*.

To establish NP completeness of H_D, we need to show that the transformation of CLP* --> H_D is polynomial time algor-

ithm and that we do not add any unnecessary complications in the computation of F.

Note that the size of the input of H_D, (#E) is polynomial in the size of the input of CLP*, (#U). Construction of the set Z is linear in the R^*. Similarly, the test (1), (2), and (3) in F can be performed in time proportional to n = #U. Thus the transformation can be done in polynomial time.

If H were in P instead of in NP. we would then have a polynomial time algorithm for CLP*. But CLP* is known to be NP complete. Therefore H_D is NP complete.

NP Completeness of H_O

We have shown that the hypothesis generation and testing problem framed as a decision problem H_D is NP complete. This means that unless P = NP, the optimization is at least as hard. This follows from the fact that if H_O were in P, we could answer the H_D problem by determining the optimal (maximal) interpretation I in polynomial time, computing g(I) for the optimal solution and answering H_D = true iff $g(I) \geq B$. This would be a polynomial time algorithm for H_D which has been shown to be in NP.

Thus H_O is at least as hard as H_D In this section, we show that the optimization problem itself is at least as easy as the decision problem and therefore has equivalent

complexity. We do this by giving an NP algorithm for solving the optimization problem which uses the decision problem as a "subroutine" on a non-deterministic turing machine.

Let the algorithm which solves $H_O(E,R,F)$ "guess" at the optimal solution I. To verify that it is indeed optimal, we need to show that F(I) is C, which can be done in NP time, and then verify that there does not exist any interpretation I' such that $g(I') > g(I)$.

Since g(I) can be computed in P time and since g(I) is defined as an integer, we can compute g(I) + 1 and ask the decision subroutine the question $H_D(E,R,F,g(I)+1)$. If H_D returns false, we know that I is the optimal solution.

Since $H_D(E,R,F,B)$ has been shown to be an NP this implies that $H_O(E,R,F)$ is also in NP.

Thus $H_O(E,R,F)$ is equivalent in complexity to the class of NP complete problems.

In the next section we develop ideas about consistency checking. Unless consistency checking can be performed efficiently, the problem of generating the optimal hypothesis problem will be practically intractable. We will show that for a given hypothesis, we can determine consistency without performing any search.

2.4 DETERMINING CONSISTENCY OF A HYPOTHESIS

The sensory process can have parameters. For example, the perspective image produced by a camera depends not only on the arrangement of entities in the scene but also on the location and the focal length of the camera. The parameters of the sensory process can be considered to be attributes of 0-th order relations (those which involve no world entities). Based on physical laws, the sensory process interacts with these relationships and their attributes and produces a sensory pattern consisting of sensory entities and their relationships which are also attributed. For purposes of discussion, we can call this produced pattern an "image" of the world. The attributed relationships in the image are fixed and depend on the instance of the world being sensed. These relationships between the sensory entities are termed measurements.

Formally, an image may be characterized by the following. An image consists of a set of entities $E' = \{e_1', \ldots, e_m'\}$, where each e_i' is the image of the corresponding world entity e_i. The entities e_i' are called sensory entities.

Sensory entities stand in named attributed relationships to each other. These relations are termed sensory relations and are defined as $R' = \{R_1', \ldots, R_k'\}$ where each R_i' is an attributed relation over the sensory entities. Each rela-

tion R_i' has a name r_i', and an attribute set a_i'. The relation r_i' relates o_i' sensory entities. Thus each r_i can be represented as the subset

$$R_i' \subseteq \{r_i'\} \times E'^{o_i'} \times \mathbb{U}_i'$$

where each $f' \in \mathbb{U}_i'$ is a single valued relation which assigns a value for each attribute in a_i' taking a value. We use the term measurement (denoted M) to be $M = (E', R', O', A')$ where O' is the set of all the o_i' and A' is the set of all attribute names. Note that all the information in M is known and determined unambiguously from the image.

A hypothesis is a collection of relational tuples. These tuples define relationships between entities in the real world. However these relationships cannot be verified by direct observation of the entities involved. The only access we have to the three-dimensional world is through the observed measurements which constitute a projection or a non one-one function of the world. Thus the only way to check if a hypothesis is consistent is to determine if the physics of the sensory system would give rise to the observations if the hypothesis were correct.

This means that the available inference engines must be able to compute some values which may be assigned to the attributes in the three-dimensional hypothetical world which when passed through the sensing process, would explain the

observed measurements. In general this problem is complicated by the fact that the projection process is controlled by parameters which may not be known a priori.

The search process which determines consistent hypotheses does not have to search over the space of all possible assignment of values to the attributes because of the tight constraint imposed by the mathematics of the projection process. Provided we have sufficient knowledge of the physical processes involved, if we are given a set of relational tuples with unknown attribute values and the observed measurements, it is possible to compute or tightly constrain the allowable values for the attributes.

2.5 INFERENCE ENGINES

For the purpose of efficiency, it is necessary to organize the knowledge about the inference process into compact manageable units. These units need the ability to accept as input, a specific set of relational tuples, and the observed measurements and from them to compute values for the attributes involved. Provided such units can be constructed, consistency of hypotheses can be defined in terms of their computation.

We term these computational units "inference engines". Inference engines operate on relational tuples and measured

attributes of the image, and from them, compute values for the unknown attributes of the hypothesis. That is, the inference engines extend the attribute value relations $\mathbb{U}$ associated with the relational tuples from the hypothesis. A hypothesis will be declared inconsistent if there exist two inference engines which both compute a value for the same attribute, and the two values are incompatible. In other words, whenever any attribute value relation extended by an inference engine becomes non-single valued, the hypothesis is declared inconsistent.

To be able to prove interesting properties about the process of using these computing units, we tighten the definition of inference engines as follows: inference engines are modular computation units which accept as input, a specific set of attributed relation tuples from the possible relationships between the world entities and a set of measurements taken from the sensory input. Based on the measurements and on previously computed attributes of the relational tuples, they compute values for other attributes of the input tuples. The initial hypothesis has all the attribute values unknown. No attribute has a value. The attribute value relations are empty. Attributes can only be given values based on the computations performed by the inference engines. An inference engine is defined to be in

cannonical form if it computes the value of only one attribute, that is, it extends the relation $\mathbb{U}$ for at most one tuple in its input set.

The mode in which these inference engines operate can be described in general terms as follows: The initial input consists of the relational tuples whose validity is to be determined, with all the relations $\mathbb{U}$ set to Φ. Inference engines are triggered based on their input requirements and compute values for some of the attributes. The first inference engines which get triggered are those which compute output values based only on the set of measurements. Each inference engine works on the output of the previous engine. Whenever a value gets computed for an attribute for which no previous value has been determined, that attribute assumes the newly computed value. If the attribute has a previous value, and the two values do not match, the input hypothesis is declared invalid.

Formally, an inference engine is a mapping from H X M --> H' where H and H' are hypothesis sets $H,H' \subseteq P$, and M is the set of measurements from the image. Inference engines do not add new tuples to the hypothesis sets and therefore, $\#H = \#H'$ and H' differs from H in that for any $h \in H$, where $h = (r,e_1,\ldots,e_x,\mathbb{U})$, there exists $h' = (r,e_1,\ldots,e_x,\mathbb{U}')$ with $\mathbb{U} \subseteq \mathbb{U}'$. All inference engines take M as their input. There-

fore, for notational convenience, we drop M from the input and effectively consider the engines to be mappings from H --> H'.

The questions that arise in this context deal with the termination of the stability of the computations and the termination of the process. Supposing a set of inference engine applications determine a hypothesis to be valid. Is it possible to apply a different set of engines and arrive at the conclusion that the same hypothesis is invalid? Is it possible that a different set of engine applications or a different order of engine applications will produce a different set of values for the attributes? Does the process of applying the engines to a hypothesis have to search for the proper order in which to apply them?

In predicate calculus, consistency of a set of predicates may be viewed as a conjunction of conditions that the set of predicates must jointly satisfy. Since conjunction is a commutative operation, it may seem intuitive that different orders in which the inference engines may be applied, would all yield the same results. However inference engines cannot be applied in any arbitrary order. An inference engine can only be applied if the information it requires to execute is already present. Thus there is a partial ordering which describes all the legal sequences in which the engines

may be applied. Other complications result from the fact that inference engines compute values for attributes in addition to providing a "consistent/inconsistent" response. These values cause the state of the hypothesis to change after the inference engine is invoked. Therefore some thought is required to show that the intuitive result does indeed hold and that inference engine application is a stable process whose result does not depend on which sequence in the partial ordering is chosen.

In the remainder of this section, we state and prove several lemmas leading up to a pair of theorems which show that consistency of hypotheses can indeed be defined in computationally stable terms, and that no search is needed to determine the order in which these engines need to be controlled.

We start with some definitions of the terms involved.

We say that an inference engine E is <u>applicable</u> to a subset $K \subseteq P$ iff the following two conditions are met:

a) K satisfies the input requirements of the engine E. That is K contains the relational tuples that E uses as a basis of its computation and that the attributes of these tuples which E uses in its computation, are all defined.

b) No proper subset of K satisfies the input requirements of E.

An application of an engine E to a set K is denoted as E(K) where E is applicable to K. An application of E to K may succeed or fail (with success and failure being as defined earlier). If an application succeeds, a new subset K' results which differs from K in that atmost one attribute of K which was previously "unknown", now has a value. To make the definition of E(K) complete, we provide that if the application fails, the output subset K' is the same as K. Thus E(K) = K' where if E(K) fails, K' = K. If however, the application succeeds, then K' differs from K by at most one value. That is, $\#\{x \mid x=(r,e_1,\ldots,e_x,\mathbb{U}) \in K$ and $x'=(r,e_1,\ldots,e_x,\mathbb{U}') \in K'$ and $\mathbb{U} \neq \mathbb{U}' \} \leq 1$.

We are interested in running our inference engines on an arbitrary hypothesis set. To that end we define an inference step as E(H), where $H \subseteq P$, to be the application of E to some subset $K \subseteq H$ to which E is applicable. If the application is successful, we replace K by K' in the set H. In what follows, we will only be interested in the output of E(H) for the cases where E(H) succeeds. Therefore we can talk of E(H) = H' where H, H' $\subseteq$ P. Note that given a hypothesis H and an inference engine E, there may be more than one non-identical subset K of H to which the engine is applicable.

We define a sequence of inferences on a hypothesis H to be a sequence $E_j(E_{j-1} \ldots (H) \ldots)$ of inference steps where each inference engine E_i is applied to the output of the previous inference step and each of the inference steps 1 through j-1 is successful. Further, if for some i and k, $1 \leq i,k \leq j$, E_i is the same inference engine as E_k, we will assume that the subset of H to which they apply are different. Sequence of inferences are denoted in functional form to emphasize the fact that though they are applied sequentially, each engine operates not on the original hypothesis but on the output of the previous inference step.

A sequence of inferences $E_j(E_{j-1}..(H)..)$ is called a terminating sequence if either $E_j(.)$ is a failed inference or there does not exist any other inference engine E_{j+1} which can be applied to the result of the sequence in order to extend it.

These definitions allow us to state precisely, the operation of the engines. We can rigorously sum up the properties of the process with the following lemmas.

Lemma (1): Consider a sequence of applications of the inference engines to a hypothesis set $H \subset P$. If at the ith step, engine E_i is applicable, then E_i will be applicable at all steps $j > i$.

Proof: If E_i is applicable at step i, then there exists a subset $K_i \subseteq H$ which satisfies all of E_i's input requirements. As defined earlier, this entails:

(a) K_i contains the tuples drawn from the proper 3-D relations,

(b) The required attributes of the relational tuples in K_i have previously been assigned values.

No application of any inference engine to H removes any tuples from H and no application of any inference engine replaces a previously computed value for an attribute by "unknown". Thus at any later stage, K_i will still be a subset of H which satisfies the input requirements of E_i.

Lemma (2): If $E_i(E_{i-1}..(H)..)$ is a sequence of inferences such that application E_i fails, then for any sequence of inferences $E_i(E_{k'}(E_{k'-1}..(E_{i'}(E_{i-1}(..(H)..)$ where applications $E_{i'}$ through $E_{k'}$ succeed, E_i will still fail. In other words, postponing the application of a failing inference engine does not lead to a successful sequence.

Proof: If the initial sequence terminated in failure, it means that after the computation of E_{i-1}, E_i was applicable. By the previous lemma (Lemma 1), it will still be applicable after the engines $E_{i'}$ through $E_{k'}$ have been applied. Furth-

er none of the attributes computed up to stage E_{i-1} will change values. Since the engine E_i failed in its first position, it implies that it computed a different value for an attribute that already had a value. Thus in its new location, it will still compute the same value for the attribute in question. Therefore by definition of a failed inference step, it will still fail.

Lemma (3): Let $E_k(...E_2(E_1(H)...)$ and $E_j(...E_b(E_a(H)...)$ be two sequence of inferences. Let the second sequence be a successful sequence (i.e. the application $E_j(.)$ succeeds). Let these sequences satisfy:

(a) Every E_i, $1 \leq i \leq k-1$ in the first sequence is also in the second sequence (though not necessarily in the same order).

(b) E_k is not in the second sequence.

Then:

$E_k(E_j..(H)..)$ is a valid sequence, i.e. E_k is applicable at the end of the second sequence.

Proof: The condition of applicability of E_k demands that certain attributes of certain tuples have non-"unknown" values (the partial functions associated with the tuples be

defined for the attribute names that are needed in the computation). The application of $E_1 \ldots E_{k-1}$ forms a sufficient condition for the determination of these values. Thus at the end of the second sequence, E_k will be applicable.

Lemma (4): If $E_i(E_{i-1}\ldots(H)\ldots)$ is a sequence of inferences on the hypothesis H, then either it is a terminated sequence or it can be extended to form a terminated sequence.

Proof: Follows from the definition of a terminated sequence. If the last application (E_i) fails, then by definition, the sequence is terminated. If it succeeds, then two cases occur. Either there is no other inference engine which is applicable to the result of the given sequence which can be used to extend the sequence, or there is some engine E_{i+1} which is applicable. In the first case, the sequence is again a terminated sequence. In the second case apply the new engine to get $E_{i+1}(.)$ as a new sequence. The same arguments that applied to the original sequence, now apply to this sequence. Since the number of pairs of inference engines and the subset of the hypothesis on which they can be applied is finite in number, the process must terminate.

These lemmas will now be used to establish two key theorems about the inference process.

Theorem (1): If $E_i(E_{i-1}...(H)...)$ is a sequence of inferences on the hypothesis H which ends in failure, then all possible terminating inference sequences end in failure.

Proof of Theorem (1)

Let $E_k(..E_1(H)..)$ be a failed terminating sequence of inferences on hypothesis H. By definition, that means that $E_k(.)$ fails. Let $E_n,(...(E_i,(E_{i-1}(...(E_1(H)..)$ be a second sequence of inferences in which the applications E_1 through E_{i-1} are identical to those of the first sequence, and E_i, is not the same application as E_i. We will prove that this sequence, if extended, will also terminate in failure. Without loss of generality, (by Lemma(4)), let the second sequence be considered a terminating sequence.

Case I: If E_n, fails, the proof is complete.

Case II: If $E_{n'}$ succeeds, the second sequence is a successful terminating sequence. We will show that this leads to an inconsistency. The contradiction is by induction on the applications E_i through E_{k-1} of the first sequence.

By Lemma (3), either E_i is present in the subsequence $E_{n'}(...E_i,(.)..)$ or it is applicable after $E_{n'}$. Since the second sequence was considered a terminating sequence, It must be the case that E_i occurs somewhere between $E_{i'}$ and $E_{n'}$.

Since $E_{n'}(...(E_{i'},(.)..)$ is a successful terminating sequence, the output variable (computed attribute) of E_i will have the same value as that computed after the application of E_i in the first sequence. To see this note that the input terms for E_i are computed by the applications E_1 through E_{i-1} which remain unchanged in the second sequence.

Thus the sequence E_1 through $E_{n'}$ contains all the engines from E_1 through E_i. Consider the sequence $E_i(..(E_{i-1}(..(H)..)$, which is the subsequence of the successfully terminating sequence, $E_{n'}(...(H)..)$, and the sequence $E_i(...(H)..)$, which is a subsequence of the unsuccessfully terminating sequence $E_k(...(H)..)$. Since E_{i+1} is applicable at the end of $E_i(...(H)..)$, by Lemma 3, it must also be applicable after the E_i in the subsequence

$E_i(..(E_{i-1}(..(H)..)$ from the terminating sequence. Therefore, E_{i+1} must occur in the sequence $E_n,(...(H)..)$. A similar argument shows that the output variable of this engine too must be the same as its output in the first sequence.

This argument can be inductively extended to show that all the E_js, j=1,..,k must be in the sequence E_1 to E_n, and that all the attributes computed as the input to E_k must have the same values as they did in the first sequence.

In the first sequence, the application of E_k failed. Therefore the sequence $E_1..E_{k-1}$ must also compute a value for some attribute which is different than that computed by E_k. Let E_j $1 \leq j \leq k-1$ be the engine in the first sequence that computed the contradictory value. We have shown that it computes the same value in the second sequence as it does in the first. By the same argument E_k also computes the same value as it did in the first sequence. Since these two values were different in the first sequence, they must also be different in the second sequence leading to the inference step failing. This is in contradiction to the assumption that the second sequence was a successful sequence.

Corollary: If $E_i(E_{i-1}...(H)...)$ is a terminating sequence of inferences which ends in success, then all possible terminating sequences will end with success.

Theorem (2): Ignoring permutations, there is at most one successfully terminating sequence of inferences for a given hypothesis H.

Proof of Theorem (2): An argument exactly parallel to that of the previous proof show that if there are two successful terminating sequences of inferences for a given hypothesis H which differ by some inference engine applications, they cannot both be terminating sequences.

We use the concept of terminating sequences to define the consistency of hypotheses in the following way. Given a hypothesis, if it has an associated sequence of applications which terminate with failure, the hypothesis is inconsistent with respect to the knowledge encoded in the inference engines. If on the other hand the terminating sequence succeeds, the hypothesis is consistent. Note that by this definition, the terminating successful sequence may be of zero length if the tuples in the hypothesis are such that no in-

ference engines are applicable. The theorems tell us that the sequence of applications of inference engines to a hypothesis must terminate. Further, the result obtained does not depend on the order in which the engines are applied to the hypothesis set. The definition of applicability and inference steps tells us that adding new relational tuples into a hypothesis will not affect the inference engines that would work on the smaller hypothesis set. Further, given a consistent hypothesis, there is only one (ignoring permutations) sequence of applications which characterizes the set. There is no search involved in the determination of the consistency of a hypothesis because there is no order dependency. Thus the process of determining consistency is linear in nature.

2.6 CHARACTERIZATION OF INFERENCE ENGINES

In the previous sections, we have defined the hypothesis generation and testing problem, and shown how independent inference engines can be used to determine the consistency of a hypothesis. We showed that the process of applying the engines is a sequential process, with no search involved, and that because of this, the overall problem of finding an optimal interpretation becomes an NP complete problem.

In essence, when the problem is framed this way, it reduces from a problem of theorem proving in predicate calculus to a problem of verifying the consistency of a propositional logic statement. In this section, we characterize the nature of this transformation to understand the exact difference.

2.6.1 Predicate Calculus with Restrictions

We will use a small example to illustrate what happens when closed form expressions are used to encode the interrelationships between the numeric attributes of the tuples which constitute a hypothesis. Consider the following problem. Suppose we make the hypothesis that a point in three-dimensional space corresponds to a particular point on the image with known image coordinates X_s, Z_s. Further, we know the X coordinate of the three-dimensional point, and we know the focal length F of the imaging system. Using capital letters for atoms (or known entities) and lower case letters for variables, the problem of determining if the hypothesis is consistent, would be phrased in theorem proving as determining the proof for a theorem:

$$\vdash y \vdash z \; \mathrm{Proj}(X, y, z, F, X_s, Z_s)$$

where the predicate Proj : $\mathbb{R}^6$ -> {True,False} is true if the point defined by the space coordinates given by the first three arguments corresponds to the screen coordinates given by the last two arguments, when the camera focal length is the fourth argument. The axioms that the theorem prover would use would consist of all axioms of arithmetic and the fact that the projection of a point (x,y,z) is (xf/y,zf/y) when the camera focal length is f. We can dispense with these axioms and assume that there is some suitable representation for them as long as the predicate Proj does what is necessary.

If the hypothesis is consistent, there would be some values for the unknown variables y and z which would satisfy the theorem. If no such values could be determined, the theorem would be false and the hypothesis would be inconsistent. Consider how a theorem prover would try to prove such a predicate calculus statement. It would essentially expand the semantic tree (Cha73) and attempt to find a node in the tree that would disprove the negation of the theorem. This is precisely where the problem comes in. The semantic tree corresponding to the theorem above is infinite because the domain from which the variables y and z are drawn is the space of all real values.

Consider what happens to the same theorem, when it is recast into a framework in which the inference engines compute in closed form the inverse of the projection. The proof of the first-order logic theorem above becomes equivalent to determining the consistency of a predicate

$$\text{Proj}\ (X,\ E_y(X,F,X_s,Z_s),\ E_z(X,F,X_s,Z_s),\ F,\ X_s,\ Z_s)$$

where the functions E_y and E_z are mappings from $\mathbb{R}^4$ -> $\mathbb{R}$. Now there is no search involved. The search that the theorem prover had to do has been "compiled" into the functions E_y and E_z. By proper construction of these functions, it is possible to assert that the predicate in the second form is true iff the theorem shown before is satisfiable.

It is precisely these functions that are encoded in the action part of the inference engines. The checking of the multiple values generated for variables of the tuples is a secondary part of the process. The theorem proving part is abstracted and pre-compiled into the predicates, reducing the complexity of the problem to a simpler domain. The search involved in systematically generating possible hypotheses works in a domain equivalent to the propositional logic. Consequently, the overall problem complexity makes the solution method tractable.

2.6.2 Inference Engines as a Reasoning Path

In addition to the characterization described above, the inference engines describe a path that the reasoning process follows in determining the consistency or inconsistency of a given hypothesis. An inference engine E:H-->H is a function which maps a hypotheses into one which may have values for some attributes which were previously unknown. H is a subset of the space of all attributed tuples P. As before, we consider the measurements to be globally accessible to all engines and therefore, do not explicitly denote M in the notation. Each inference engine has a specific input tuple set that it works on, and its result is the computation of a value for some attribute of the input set. We implicitly state that an inference engine works on a particular input set only once. That is, once the engine has "seen" a particular subset of the hypothesis, it will not apply to the same subset again. Once all applicable inference engines have worked on all their possible input sets in a given hypothesis, we define it to be a complete application sequence. In other words, a complete application sequence is one in which all attributes for which values could be computed, have been assigned values by some sequence of inference engine applications. An inconsistent hypothesis is one for which a complete application sequence assigns contradictory values for some attribute.

Let the hypothesis have m attributes associated with the tuples. Therefore at the end of any complete application sequence, at most m attributes could have values associated with them. Let n of these attributes ($n \leq m$) be the ones which get computed by the inference engines. Ignoring the actual values themselves, there are 2^n combinations in which values are assigned to attributes. Initially the hypothesis starts off with all attributes "unknown". Let the state of the hypothesis be defined by the pattern of which attribute values are known and which ones are still unknown. These states can be considered to be nodes in a network, with the inference engines defining the arcs between the nodes. If a particular inference engine can cause a transition between one state and another there is a directed arc between the two states. The process of inference engine applications results in sequential state transitions which start with the state corresponding to all attribute values unknown and end with the state in which n of the m attributes have values. Let the starting node be labeled N_0 and the final node be labeled N_n. This network has several interesting properties.

1. The digraph of the network has no cycles of length greater than one. This follows from the fact that edges in the graph are constrained by the nature of

the inference engines which define the arc. If the hypothesis is in a state in which k of the attributes have values, application of any inference engine will not reduce the number of attributes which have computed values.

2. The maximum length of any path from the start node to the final node is exactly n.
3. The graph can have cycles of length 1. Such arcs correspond to applications of an engine which re-computes the value for some attribute. In fact, if there is an arc corresponding to engine E_i from node N_i to N_j, then there is a loop also labeled E_i from node N_j to itself.
4. If there is a loop from node N_j to itself labeled E_i, then there must be an arc from some other node N_i to N_j also labeled E_i. Since we are looking for paths from the start node, we can eliminate the loops in the network.
5. In addition to the start node there may be other nodes which have in-degree zero. For example, it is not possible to compute the three-dimensional coordinates of a point based on image information alone. Thus the node which corresponds to the coordinate attributes cannot have any arcs leading into it. Howev-

er, if some coordinate information is supplied a-priori, it can be used to compute the locations of other points in the world. Thus the corresponding node can have arcs leading out. However, in a reasoning chain which starts out with the node N_0, we will not encounter any such node.

6. The graph of the network formed by deleting all loops, and removing all nodes with in-degree zero, except the start node N_0 is a simple connected digraph.

This network characterizes the path taken by the engine application process through the space of value assignments to the attributes. The point to note here is that the graph defines a very small subspace of all possible paths which could pass through all possible combinations of the m attributes being assigned values. This structure can be contrasted with what a theorem proving machine would have to contend with if the problem were phrased as a first-order logic problem.

In summary, expressing the knowledge about the projection process as closed form inverse equations, which compute values for variables based on partial knowledge, transforms the problem from the first order logic domain to that of propositional logic. The resulting reduction in complexity

makes large problems tractable. In addition, defining the consistency of the hypotheses in terms of the computations performed by the inference engines structures the space of all hypotheses and this structure can be exploited to perform the search more efficiently.

One consequence of this definition of consistency is that if a hypothesis H $\subseteq$ P is consistent, then any subset K $\subseteq$ H is also a consistent hypothesis. Thus we can replace the notion of searching for an "optimal" hypothesis by searching in P for the "maximal" hypotheses. We define a subset I $\subseteq$ P to be an interpretation if it is consistent and is maximal in the sense that for any p $\in$ P-I, the hypothesis I+{p} is inconsistent. Note that there may be more than one possible interpretation to a scene and each interpretation will correspond to one such maximal subset. We now discuss one method of finding such an interpretation.

2.7 SEARCHING FOR THE MAXIMAL HYPOTHESIS

The number of possible subsets of a set of relational tuples is 2^n where n is the number of possible tuples. This number grows exponentially in n, and as n grows, the time taken to exhaustively search the entire space, grows prohibitively. However the possible solutions are related as was pointed out in the previous section. If some subset K $\subseteq$ P is

a consistent hypothesis, then all its subsets are also consistent and can be ruled out of the search pattern. Secondly, since we are interested only in the maximal consistent subsets, we can attempt to "grow" the solutions in a sequential fashion.

In this section, we will look at two different strategies for enumerating and visiting all elements of the search space, with specific attention given to the problem of avoiding the subsets of known solutions. The first strategy involves viewing the search space as a lattice of all possible subsets of the set of tuples. This technique has the advantage that all the relevant subsets are directly linked due to the structure of the lattice. The second technique considers the space as a binary tree in which the possible solutions are the leaf nodes. We will compare the relative merits of the two schemes.

For illustrative purposes, consider a simple example. Let there be 3 tuples, labeled 1, 2 and 3 which could describe the conditions in the world. For example, tuple 1 may be (parallel linel line2), tuple 2 could be (parallel line2 line3) and tuple 3 could be (perpendicular line2 line3). For simplicity we will not consider the case where there are attributes associated with the tuples. We have shown in the

last section that the attribute values do not enter into the search and can be computed on the basis of the relational tuples. For our example, let us say that we know that in any hypothesis, tuple 2 and tuple 3 cannot co-exist. Thus, from the set of all subsets of the three tuples, two sets get eliminated, namely the sets {1,2,3} and {2,3}. The aim of the search is to select from the remaining the sets, the optimal interpretation. We will illustrate the two techniques with this simple example to bring out their differences.

2.7.1 Searching the Lattice

Let the lattice of the subsets of the set of all tuples be represented as a directed graph, with the nodes representing the subsets of the space, and an arc from node A to node B representing the fact that the subset corresponding to the node A is a proper subset of the subset corresponding to the node B.

Formally, let L be the lattice, L = (N,E), where $N=\{n_0,\ldots n_m\}$ is the set of nodes, and $E \subseteq N \times N$ is the set of directed edges over the nodes. Since there is a one to one correspondence between the nodes of L and the elements in P', we have #N = #P'. Identify as node 0, the node corresponding to the empty subset of the search space. This node is usually called the bottom (denoted $\perp$ in the termi-

nology of discrete mathematics). Further, let the node n_i be identified with an element $p_i \in P'$. We can define a measure, $S(n_i)$ as $S(n_i) = \#p_i$ with p_i related to n_i as described.

Since the network so formed defines a lattice, we can state the restrictions on the edges in E as:
$e_i = (n_{i,1}, n_{i,2}) \in E$, iff the set corresponding to $n_{i1} \subseteq$ the set corresponding to n_{i2}.

Hypotheses that can be generated about the world correspond to the nodes in the lattice. Some of these hypotheses are inconsistent, and of the remaining (consistent) hypotheses the maximal ones are called interpretations. The task of the search algorithm is to start from the empty node n_0 and traverse the structure until it finds all the interpretations.

For the example cited earlier, the associated lattice is shown in Figure 1. The search algorithm hinges on the fact that if we detect that a particular hypothesis corresponding to a node in the lattice, is a consistent interpretation, it is easy to propagate that information to other nodes in the lattice which are related to it by the subset/superset relation. If the nodes were partitioned into two disjoint subsets: N_c and N_i such that $N_c \cup N_i = N$, with N_c consisting of all the subsets of P' which are consistent, then the in-

terpretations (maximal consistent subsets) correspond to those nodes of the partial ordered set L' which is the restriction of the lattice L to the set of nodes in N_c.

Note that L' is not necessarily a lattice. By definition, for L' to be a lattice, it needs the property that all finite, non-empty subsets of L' have a node l_l and a node l_u in L', such that $l_l \subseteq$ all the nodes in the subset, and l_u is the union of all the nodes in the subset. By the construction of the partial ordered set L' from L, we know that the first property is satisfied, for if a node corresponds to a consistent hypothesis, all its subsets are also consistent and therefore in L'. However, the second property is not necessarily satisfiable. In the example under consideration, the set {2} and the set {3} are individually consistent. But the set {2,3} formed by their union is inconsistent and consequently not in L'. An interesting consequence of this fact is that there exists a node n'_0 in L' which is the greatest lower bound of all elements in L'. This node, in fact corresponds to the node n_0 in the original lattice L. Thus we can start the search from the node n_0 and be guaranteed of the fact that we will reach all the desired solutions. In fact the nodes corresponding to the interpretations are those nodes in L' with out-degree zero. Formally, node n_i is an interpretation if there does not exist any e $\in$

E' such that e = (n_i, n_j) for any j. Here E' is the set of of edges in E' given by E' = {e $\in$ E | e=(n_a, n_b), with $n_a, n_b \in N_c$}. One does not need to explicitly construct the set L' as we will show in the following algorithm.

'Algorithm for searching the Lattice'

Let us assume the existance of suitable data structures for storing the lattice and suitable operations which enable us to find for any given node in the lattice, the set of successor nodes, and the set of predecessor nodes. We define Succ(n) for any node n in N to be the set {n' | n' $\in$ N, and (n,n') $\in$ E}, and Pred(n) similarly to be {n' | n' $\in$ N, and (n',n) $\in$ E}. We also assume that we have a function Cons(n), which when given a node n in the lattice, can determine if it corresponds to a consistent hypothesis or not.

We associate with each node in the lattice, a token which represents the state of the node. The token can take on values from the set {unknown, inconsistent, consistent, maximally_consistent}. Initially all the nodes in the lattice are labeled "unknown", that is, we do not yet know the state of the hypothesis corresponding to the node.

The complete algorithm for the search in the lattice is shown in figure 2. The procedure is invoked by SEARCH (n_0). At the end of this procedure, all the nodes labeled "maximal-consistent" are the required interpretations. The key

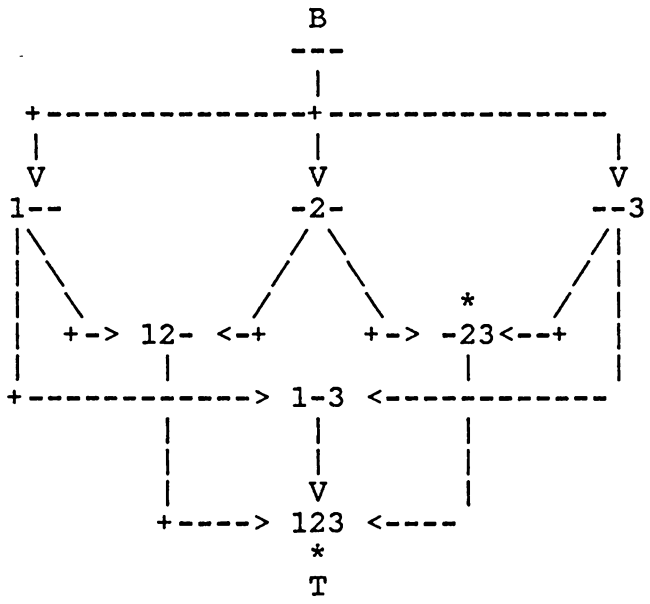


Figure 1: Lattice corresponding to the example
Showing the lattice corresponding to the example. Nodes are indicated by the subset of P' that they correspond to. Directed arcs point from a node to the corresponding supserset. The bottom of the lattice is indicated as B and the top as T. The inconsistent nodes are marked *.

```
Recursive Procedure SEARCH ( NODE )
(*****************************************************
(*
(*   Searches a given lattice for the optimal solutions
(*
(*   On entry, all nodes i have label(I) = "unknown"
(*   On exit,  all optimal solutions have
(*                 label(I) = "Maximally-consistent"
(*
(*   Calls:
(*      SUCC   Returns list of successor nodes
(*      PRED   Returns list of predecessor nodes
(*      CONS   Succeeds if node is consistent
(*      MARK   Marks subsets of consistent nodes
(*      MARKSS Marks supersets of inconsistent modes.
(*
(*  Invoked as: SEARCH ( n0 )
(*
(*****************************************************

type (NODE,I,J)      node
type (TOTRY,SI,PI)   set of nodes
type (SUCC,PRED)     function, returns set of nodes
type (MARK)          recursive procedure
type (MARKSS)        recursive procedure
begin
    TOTRY <- SUCC( NODE )
(*
(*                              For all successors of
(*                              current node
(*
    for all I in TOTRY do
       if label(I) = "unknown"
(*
(*                              If it has not been
(*                              visited
(*
          if CONS( I ) = "consistent"
(*
(*                              If it is consistent
(*                              it could be
(*                              optimally consistent
(*
              label(I) <- "maximally-consistent"
              SI <- SUCC ( I )
              for all J in SI
```

Figure 2: The Algorithm for searching a Lattice.

```
                    if label(J) =  "consistent" or
                                   "unknown"
                       label(I) <- "consistent"
                    endif
                endfor
  (*
  (*                              Mark all subsets of
  (*                              the current solution
  (*                              as being consistent
  (*
                MARK ( I )
  (*
  (*                              Recursively visit all
  (*                              nodes below current
  (*                              node
  (*
                SEARCH( I )

            else
                label( I ) <- "inconsistent"
  (*
  (*                              If the current node is
  (*                              inconsistent,
  (*                              so are all its successors.
  (*
                MARKSS( I )
            endif
          end for
      end SEARCH.



  Recursive Procedure MARKSS ( NODE )
  (************************************************
  (*
  (*   Recursively marks all the sucessor nodes as
  (*   inconsistent
  (*
  (*   Calls:
  (*      SUCC   Returns list of successor nodes
  (*
  (************************************************
  type (NODE,I)  node
  type (SUCC)    function, returns set  of nodes
  type (S)       set of nodes
  begin
      S <- SUCC( NODE )
      for all I in S
        label ( I ) <- "inconsistent"
        MARKSS( I )
```

Figure 2: The Algorithm for searching a Lattice (cont.)

```
    end for
end MARKSS



Recursive Procedure MARK ( NODE )
(************************************************
(*
(*   Recursively marks all the predecessor nodes as
(*   consistent
(*
(*   Calls:
(*      PRED   Returns list of successor nodes
(*
(************************************************

type (NODE,I)  node
type (PRED)    function, returns set of nodes
type (S)       is set of nodes
begin
    S <- SUCC( NODE )
    for all I in S
      label ( I ) <- "consistent"
      MARKSS( I )
    end for
end MARK
```

Figure 2: The algorithm for searching a lattice.

(Concl.)

idea in this algorithm is that if a particular node is marked "inconsistent", all its supersets are also inconsistent and no further checking is necessary. This follows from the idea that you cannot make some hypothesis which is already inconsistent, consistent by adding new predictions. The inconsistent "core" as it were, stays in the hypothesis. Similarly, once a consistent set is determined, that information is propagated to all its subsets which then do not need to be examined.

The utility of representing the search space as a lattice follows from the fact that all the subsets and supersets are related in a logical manner and propagating the information about consistency and inconsistency is superior to recomputing the function Cons at every node.

From the storage point of view though this representation is not as useful. Note that to be able to mark a particular node, one must have access to all the nodes in the lattice and since the number of nodes is of the order of $2^{\#P'}$, the exponential growth limits the useful size of the lattice that may be handled by the procedure.

The alternative way to view the search space is as a binary tree. We will discuss this view in the next section.

2.7.2 The Search Space as a Binary Tree

The search space can be organized as a binary tree. Suppose there are n possible relational tuples. The tree would then have n+1 levels: 0 through n. At each node of the tree (with the exception of the leaf nodes) there are two children. For a node at depth i, $0 \leq i \leq n-1$, these children correspond to the selection of and the rejection of the relational tuple i+1 respectively. By convention, we choose the left child to correspond to the inclusion of the tuple and the right child to correspond to its elimination. Thus each leaf node corresponds to a subset in which some tuples have been selected and some left out.

Although the tree in this form has twice as many nodes (leaf + internal) as the corresponding lattice, the advantage of this representation is that the entire tree does not have to be stored at any time. The tree can be searched by a backtracking tree search in which once a subtree is examined, all information about it can be discarded. However, it is not convenient to prune subsets of known solutions in this representation, because the subsets are not related in any consistent or simply denoteable fashion. We will examine this problem and indicate the solution that allows rapid pruning of the tree.

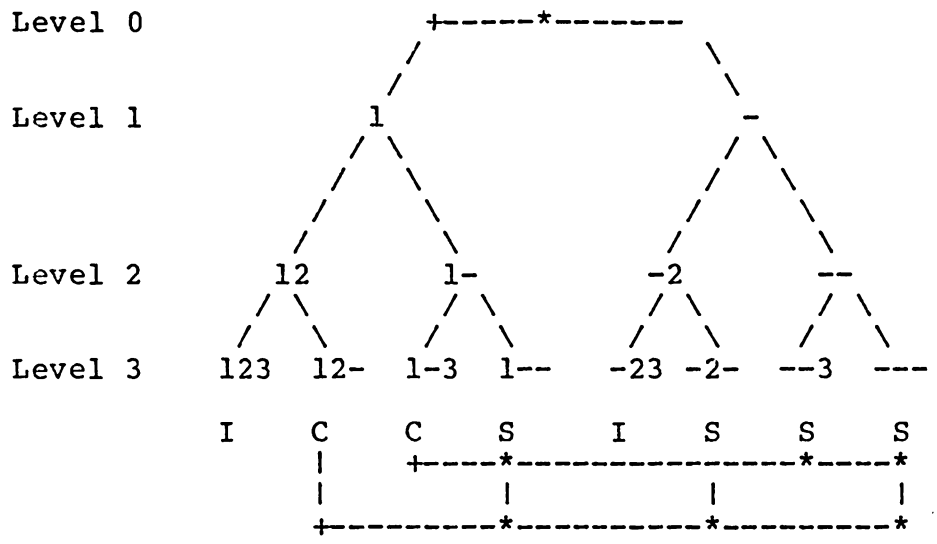


Figure 3: Tree representation for the example Space of 3 relational tuples with 123 and 23 being inconsistent (I), 12 and 13 being maximally consistent (C), and 1, 2, 3, and NUL being the subsets (S) of the maximally consistent subsets. Also note how the subsets of the maximally consistent hypotheses do not follow any regular pattern.

Let us assume some ancillary data structure is used to record solutions (or consistent leaf nodes) as they are visited in a preorder traversal of the binary tree. Associated with this storage is the required set of procedures which can quickly determine if a given subset of P is a subset of any of the solutions which have been generated. One way to avoid outputing the subsets is to wait until a "solution" is generated and then check if it is a subset of a previously generated solution. Note that since the tree is traversed in preorder, and since the tree organization is as described earlier, the larger subsets of P are visited before the smaller ones. Thus there cannot be any case where we have a subset S of P marked as a solution and then visit a leaf node which is also a solution but contains S.

However this strategy forces all the nodes of the tree to be visited. Inspection of the tree shown in the example indicates that subsets of solutions are clustered together and detection of subsets early would improve the efficiency of the search process. We can prove two important theorems which deal with this problem.

Theorem (3): At any non-leaf node in the tree, define the "leftmost solution" to be the leftmost leaf node of the subtree rooted at the non-leaf node under consideration. If the

leftmost solution in a subtree is a subset of a previously generated solution, then all the possible solutions in the subtree are subsets of previously generated solutions. This allows us to prune that entire subtree and back up.

Proof: Consider the set P of possible relational tuples to be ordered in some arbitrary way and let p_i be the ith tuple in P according to the ordering. We can associate with the set P a bit string n bits long (where n is the number of tuples in P). Any subset of P will then correspond to a string of 1s and 0s where a 1 in bit position i corresponds to the tuple p_i in the set and a 0 corresponds to the omission of the tuple. At any internal node in the tree at level i+1, we have a fixed substring of 0's and 1's which correspond to the tuples which have been selected up to that point. Since the tuples are selected sequentially, the choices that are left correspond to the bit string from position i+1 through n. The leftmost leaf node in the tree below the node under consideration, is reached by selecting the leftmost branch at all nodes under the current one. This corresponds to filling in a 1 in all the remaining bit positions from i+1 through n. All the other subsets correspond to bit strings with the same prefix (bit positions 1 through i) and a suffix bit string which is different from all 1s.

If the leftmost leaf node is a subset of a previously generated solution, it means that there is some solution which contains 1's wherever the leftmost node string contains 1's.

To show the proof, let the prefix string at a node be s_p (for example, 100101). Let the suffix of the leftmost solution be s_s (for example, 1111). If the leftmost solution is the subset of some previous solution X, it means that X has 1's wherever $s_p s_s$ (1001011111) has 1's. In our example, the solution X must look like 1xx1x11111 where x stands for zero or one. All the subsets in the subtree under consideration have bit strings of the form $s_p s_a$ where the string s_a is x^k where k = length (s_s), (10010lxxxx in the example), and the x's are either zero or one. All such strings also have 1's exactly where the solution X has 1's and therefore, each is a subset of the solution X.

Theorem (4): At any non-leaf node in the tree, define the "rightmost solution" to be the rightmost leaf node of the subtree rooted at the non-leaf node under consideration. If the rightmost solution is not a subset of any previously generated solution, then none of the possible solutions in the subtree can be subsets of previous solutions.

Proof: The proof is similar to that of Theorem (3).

This theorem says that if a node passes the test for the rightmost solution, we can do away with the checking for subsets of previous solutions while we are exploring the subtree under the node. This allows the search procedure another degree of efficiency.

2.8 FORWARD CHECKING

The efficiency of the tree search can also be improved by the incorporation of forward checking into the procedure. The speedup obtained by forward checking operators has been described by Haralick and Elliot (Har79) in great detail. In this paper we will not go into the general details of forward checking but will restrict ourselves to the way in which it is used in the hypothesis generation problem.

To describe the forward checking algorithm used, we must first look at the basic backtracking tree search. At each non-leaf node of the tree we have a choice of either adding the next relational tuple or not adding it. First the left branch is explored. We add the tuple under consideration to the subset of P generated so far. The inference engines are then run on the resulting partial hypothesis. If they indicate that the new hypothesis is inconsistent, the right branch is taken to the next lower level and the process re-

curses. When we reach a leaf node, we have a subset of P which is consistent and is a candidate solution to be added to the ancillary data structure. This basic search process is modified with the addition of the leftmost solution and the rightmost solution checks at each non-leaf node. This guarantees that subsets of known solutions are not generated.

With forward checking, at each non-leaf node, we try not only the next tuple in the selection order, but all the future tuples which are still available. The resulting subsets determine the consistency or inconsistency of all the remaining tuples with respect to the choice of tuples which have already been "committed". Then at leaf nodes lower in the subtree, if a tuple has been marked as not consistent because of forward checking higher in the tree, the left branch is automatically ruled out.

Under the forward checking paradigm, the concept of leftmost solution needs to be modified. The leftmost solution of a non-leaf node is the leftmost possible leaf node in the subtree which is admissible under the constraints determined by the forward checking (See Figure 4). The question that needs to be answered in this context is the following. Since the remaining leaf nodes in the subtree are not subsets of the leftmost solution, is Theorem 3 still applica-

ble. The proof that it is still applicable hinges on the fact that not all the leaf nodes to the right of the leftmost subset are candidates for solutions. The forward checking criteria rule out the leaf nodes in the subtree that are not subsets of the leftmost solution.

Proof: The proof here parallels the proof of Theorem (3). We illustrate it here by example. Let 100101 be the prefix which is the same for all the solutions in the subtree. Let there be four more tuples left in the set P corresponding to the next four bit positions in the string. Thus all the leaf nodes in the subtree have strings of the form 100101xxxx where the x's are either zero or one. Of these four tuples, let the second and third tuple be ruled out because of the forward checking. Thus the only candidates for solutions correspond to bit strings of the form 100101x00x. The leftmost of these candidates corresponds to the string 1001011001. If this string is a subset of some previous solution X, then X must correspond to the bitstring 1xx1x11xx1. It can be seen that all the candidate solutions in the subtree have 1's exactly where X has 1's and are therefore also subsets of X.

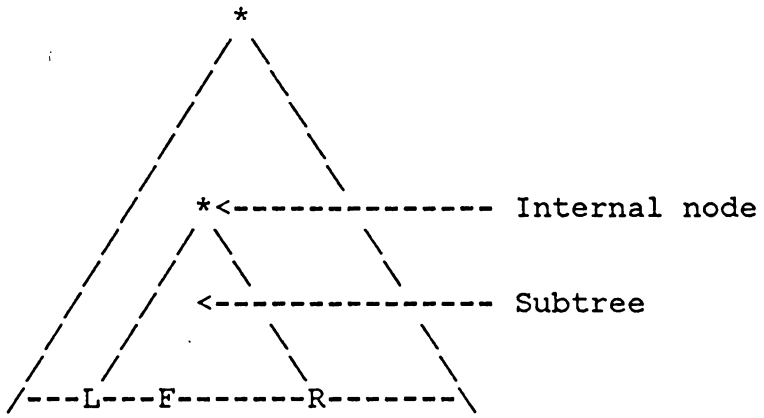


Figure 4: A subtree rooted at an internal node

The leftmost hypothesis in the subtree (L), the rightmost hypothesis (R) and a possible location for the leftmost solution under forward checking (F) are also marked.

2.9 LOGICAL INCONSISTENCY AND COMPLETENESS FOR PRUNING

In addition to the numerical consistency criteria imposed by the inference engines, another form of consistency is important in the hypothesis generation process. This is termed the logical consistency of the space. For example, two lines cannot be simultaneously parallel and perpendicular. Logical consistency arises from the world semantics and it is often not possible to define in a numerical sense. Other examples of such logical consistency are the symmetry and transitivity of relations. These constraints can also be encoded as extensions of the forward checking which rule out possible future tuples from consideration. For example, if the left branch at any internal node takes the tuple (parallel line1 line2) into the hypothesis set, forward checking can rule out (perpendicular line1 line2). This dichotomy between the logical and numeric constraints parallels the distinction between "geometric" and "relational" consistency which was reported in previous work (Mul84).

In addition to ruling out possiblities, the concept can be extended further to forcing the selection of certain tuples based on the tuples in a partially completed hypothesis. Note that if a partial consistent hypothesis is extended by adding a new tuple, it may become inconsistent. This will be tackled in the tree search by taking the right

branch at the appropriate node. However in the case where the relation semantics dictate that the tuple must be included in a hypothesis, the resulting subset of P must be ruled out (although by the numerical computations of the inference engines, it is consistent). For example, a hypothesis which consists of two tuples (parallel line1 line2) and (parallel line2 line3) alone must be ruled out because the transitive nature of the parallel relation dictates that the tuple (parallel line1 line3) should also be included in the hypothesis. The original hypothesis is in that sense "incomplete".

This fact can be utilized in the tree search as follows. At any internal node in the tree, whenever a new tuple is added (i.e. a left branch is taken), all the tuples that are logically implied by the previously accepted set of tuples and the new tuple are also put into the partial subset. If any of these logically implied tuples have been ruled out by forward checking at previous levels, the subtree below the current node is discarded and the procedure backs up.

Thus at each node of the tree, tuples are not instantiated individually but in "chunks". Thus the effective depth of the tree decreases. The actual amount by which the tree search gets pruned depends on the degree to which subsets of tuples imply the selection of other tuples. As a simple ex-

ample consider the space P with 3 tuples in it. Let these tuples be called a, b and c and let their semantics imply that tuple a and b together logically imply tuple c,and b and c imply a. The search space consists of 2^3 = 8 subsets. However because of the chunking effect of the logical relationship, the number of viable subsets that must be searched reduces. The subsets {a,b}, {b,c} are ruled out leaving only six possible subsets to consider.

2.10 APPLICATIONS TO COMPUTER VISION

In this chapter, we developed the theory of hypothesis based reasoning and indicated that it would be applicable to the problems in computer vision. We showed that the complexity of the search algorithms would be NP complete. We showed that knowledge about consistency encoded as modular inference engines, allow us to define consistency of hypotheses in terms of the applications of the engines. We showed that we do not have to search over the set of all engines for all possible ways in which they may be applied, in order to determine consistency. We also showed what the space is that the procedures have to search over.

In order to apply the technique to computer vision, we need to define the following unknowns:

1. The particular computer vision problem being tackled. As we have indicated earler, we will look at the problem of analyzing the perspective line drawings consisting both of straight lines and curves.
2. How the independent inference engine can be made to cooperate and how their applications can be controlled. This problem is addressed in Chapter III, which deals with control strategies for hypothesis based reasoning. In that chapter, we will develop a hybrid scheme which has the advantages of both the fully distributed control schemes and the agenda based systems current in the AI literature.
3. What are the specific pieces of knowledge that can be used in the inference engines? In Chapter IV, we will address this question. We will use the equations of perspective geometry, and knowledge about the interplay between spatial relationships to generate consistent solutions.
4. Comparison with existing knowledge based systems in the computer vision literature will be dealt with in Chapter V.

Chapter III

CONTROL STRUCTURES FOR REASONING

The task of analyzing perspective line drawings can be viewed as one of hypothesis based reasoning. We are given an image of the scene which consists of image entities and attributed spatial relationships between the entities. This given information is termed the measurement of the scene. In addition we know the types of entities that actually exist in the three-dimensional world and the possible relationships that may hold between the world entities. What we do not know, and what we wish to find out from our analysis is the type of the world entity that corresponds to each entity in the image, and the spatial relationships that hold between them.

In the previous chapter, we showed that this analysis task can be viewed as a hypothesis based reasoning process. The physics of the projection process which produces the image from the scene strongly constrains the possible interpretations that we may produce from the given set of measurements. Although the inverse of the perspective projection is not unique, it is a deterministic physical process, and therefore, it is possible to determine the consistency of hypotheses that we make about the organization

of the entities in the scene. In fact, given a hypothesis that some world entities corresponding to some image entities stand in some spatial relationship, we can compute some of the unknown attributes of other entities that are visible. The computations are based on the measurements made of the observable entities in the image. These computed attributes can then be used as constraints to verify the consistency of future hypotheses.

We showed that these pieces of knowledge can be implemented as modular computational units which we termed inference engines. Each inference engine has a specific set of conditions which must be satisfied by the hypothesis being worked on, and each engine updates the hypothesis in a predetermined fashion. Using this concept of modular engines, we defined consistency of a hypothesis and showed how hypothesis could be grown to form consistent interpretations. We showed that there is no order dependency in inference engine applications and that the process must terminate. Under these circumstances, finding the best interpretation is an NP complete problem.

In this chapter, we discuss the control strategy to handle this reasoning process and compare it with two of the classical control strategies in the AI literature. We will show that in a sequential, interpretive environment, our

strategy has advantages over the classical control algorithms in terms of amount of effort expended and ease of upgrading the system.

3.1 REQUIREMENTS FOR THE CONTROL

Each inference engine is a complete module. As its input, it takes the entire hypothesis that has been constructed up to that point, and examines it for specific conditions under which it can operate. For example, consider an inference engine which computes the vanishing point of parallel lines. Given the data from the image, and a hypothesis about the lines in the three-dimensional world, this inference engine computes the vanishing point of the lines which are hypothesized as being parallel. What should the hypothesis contain that enables this engine to "fire"? It must contain a prediction that there exist two lines (say line a and line b) which are parallel in the world. Only when such a prediction is present in the hypothesis, will the inference engine be able to perform its function.

Once the inference engine detects that such a prediction is present in the hypothesis, it obtains the relevant parameters from the image measurements, (say the equations of the projections of these lines) and computes the vanishing point. Now there are three choices open. First, it may be

the case that the vanishing points of the two lines were previously unknown. In this case, the action taken by the inference engine is simply to assert that the vanishing point of the two lines is the one that it just computed. However, based on inference engines that were activated before the current one, it may happen that the vanishing point of one or both of these lines may have previously been assigned some values. If the newly computed vanishing point matches the previously asserted values, everything is fine. Otherwise, the hypothesis is inconsistent because the two different reasoning paths which lead to values for the same physical attribute, do not yield the same result. In this case, the inference engine must indicate that an inconsistency has been detected.

To sum up, inference engines

1. check the hypothesis for relevant predictions,
2. use the predictions combined with measurements from the image to compute values for attributes,
3. check these values for consistency with previous predictions for the same attributes, and
4. if the hypothesis is inconsistent, indicate that backtracking must occur in the search for the interpretation.

In this chapter, we are concerned with the question of how such a group of modules can be controlled. Since inference engines accept as input hypotheses modified by other engines, and in turn modify the attributes of the hypotheses themselves, we need to have a technique for sharing all available information in a global sense. Secondly, note that if an inference engine works on some subset of the hypothesis, and if its conclusion is that the hypothesis is consistent, the hypothesis may remain unchanged. That is, all the predictions that formed a part of the hypothesis before the inference engine was invoked, are all still present at the termination of the inference process. Therefore, some care has to be taken to ensure that the same inference engine does not claim that it is still applicable, and repeat its calculations over and over again. Effective control of the inference process demands that an effective technique be developed for marking the fact that a certain subset of predictions has been examined by some particular inference engine at some stage in the process of interpretation. We will first examine the question of data sharing and sequencing the inference engines, and then look at the use of associative memory for saving the history of computations done by each engine.

3.2 CONTROLLING THE INFERENCE ENGINES

It should be clear from the previous section, that the inference engines can run in an entirely distributed mode of operation as long as there is some common area where the hypothesis can be stored for access and update by each module. A common data area for such purposes is called a blackboard in the AI literature. The first use of this term in a large expert system was in the HEARSAY II expert system for the speech analysis (Eng77). Basically, a blackboard can be defined as a structured global database which can be queried and updated by independent processes which use the data. Access mechanisms have to be provided for the access and update procedures to maintain consistency of the database. Computational processes in HEARSAY II were organized as independent units (demons) which would be triggered by specific processing conditions noted in the blackboard. For example, if the sentence currently being parsed had a noun phrase isolated, the adjective expert would recognize that fact by examining the blackboard and get triggered.

Let us examine how such a completely distributed system would actually work in a programming environment which is sequential. Each self-contained expert would need to have the capability of analyzing the contents of the blackboard to determine if it could use the available data and contri-

bute new information to the system. On determining that it was in fact capable of performing its specified task, it would then need to compute its results and place them in the appropriate place in the global database.

Such a control strategy is conceptually simple. However, in sequential computing machines, it needs modification. The usual modification is the addition of a supervisory task whose function it is to poll each module to determine if it could run, and if the module could run, to allow it to complete its task. The simplest supervisory strategy is to sequentially poll each module in turn until one full sweep of all the modules yielded no new information on the blackboard.

In addition, each module needs to remember the particular set of inputs that it has examined so that when it is polled again, it will not repeat calculations that it has already performed. Otherwise the system will loop forever, constantly repeating the same task over and over again.

To summarize the distributed processing control strategy with processes communicating over a common shared database,

1. a supervisory process sequentially invokes each computational module;
2. each module examines the available data and determines if it can work;

3. if it can perform its duties, it remembers the exact condition under which it was triggered and updates the database with its conclusions; and
4. the process terminates when no module can add any new information to the blackboard.

Although this control strategy works well in many situations, it is easy to see that it has shortcomings. It takes a certain amount of effort for each module to determine if its conditions have been met by previous computations and if it can compute new results using input it had not used before. Sequentially invoking each module is computationally very expensive. Most of the time, the majority of the modules will simply not be applicable. As the number of modules increase, the time spent in the supervisory process rises and becomes comparable to the time spent by the modules actually computing new results.

The solution to this problem is to recognize the fact that the modules are not actually independent. Most modules depend on others for their input. Consider the example given before, the adjective detection module depends on the fact that a noun phrase has been detected by a previous module. Consequently, it is possible to effectively disregard the existence of the adjective module until an appropriate noun phrase has been detected. This can be done by providing the

supervisory process with the appropriate "meta" knowledge about the task and instead of a simple sequential scan of the modules, it performs the task of invoking the computations in a more intelligent fashion.

However, if the supervisory process needs to perform checks of complexity similar to that of the modules themselves, nothing is gained. On the other hand, recognizing the fact that the modules are related, the task of checking the pre-conditions of a module can be delegated to the other modules which are related. In the simplest case, if module A uses the output of B as its input, module B can check its output at the time it is generated and inform module A when its output is of the correct form.

Systems using this strategy are termed agenda based systems in the AI literature.

3.3 AGENDA BASED CONTROL STRATEGY

The most complex agenda based system in the AI literature is the AM expert system (Len82) whose field of expertise is in the area of arithmetic proofs. In essence, agenda based systems also consist of independent processes communicating over a common shared information source. However, each module knows about the other modules in the system, and knows the conditions which are required by the other modules to

perform their tasks. Once a module computes a particular piece of information, it knows which of the other modules could use what it has just computed. It then checks to see if the preconditions of the other modules are satisfied by the state of the blackboard updated with the information it just computed. If it detects that some module has all the information it needs, it then triggers that module.

On sequential systems, the process of triggering a module is implemented by placing the module to be executed on a first in first out queue. The task of the supervisory process is then reduced to removing the first task on the queue and executing the appropriate module. The process terminates when the queue is emptied. This simple strategy may be further enhanced by treating the queue as a set and allowing the supervisory process the freedom to choose the next module to be tried. This allows some meta level knowledge about the problem domain to be placed in the control process.

Under such a control paradigm, it is not necessary for each module to check the blackboard for its pre-conditions. The fact that someone placed it on the agenda queue is sufficient information that its preconditions are satisfied. From the standpoint of execution speed, this situation is excellent. Unless some module can potentially run, its pre-

conditions are never checked, and for all practical purposes, it does not exist. Thus, unlike the distributed processing case described earlier, there is no overhead involved with checking if all the modules can execute.

In summary, the agenda based systems are characterized by

1. Each module knows which other modules depend on it for input
2. Each module checks all the pre-conditions for those modules which depend on it.
3. If some other module's pre-conditions are satisfied, the module is placed on a processing queue for future execution.

However the drawback with this scheme is the fact that the checking of preconditions, is far too distributed. Consider what happens if a new module must be added to the system. If the new module takes n pieces of data as its input, and if each of these pieces is updated by m possible modules each, in the worst case mn modules must be changed to recognize the existence of the new module. In each of those mn cases, code corresponding to the full check on the preconditions of the new module must be added and checked. This becomes worse as more and more modules are added. So although we gain execution speed, it is at the expense of readability and maintainability of the system as a whole. Contrast this

situation with the demon based control strategy. Demons are totally uncoupled, and consequently whenever a new module must be added to the system, the only change that must be made is to inform the supervisory process that the number of modules to be polled has increased.

In the next section we present a hybrid technique which retains the best of both worlds. It has the advantage of the ease of maintenance of the distributed control strategy, along with the speed of the agenda system. Although it is not as fast as the agenda system, or as uncoupled as the distributed system, we find that it to be a suitable blend of the two.

3.3.1 The Network Model for Module Control

All inference engines take a particular set of tuples and their attributes from the hypothesis as their input and produce new values for some of the attributes as their output. These new values update the hypothesis and are in turn accepted as inputs by other inference engines in the system. This fact can be compactly denoted, by representing the engines as nodes in a network. The nodes in a network may have some arcs leading in to the node, and some arcs leading out of the node. Every node leading in corresponds to an input attribute of some tuple which must have a previously comput-

ed value, while outgoing arcs represent values for attributes that are computed by the inference engine located at the node.

In addition to these data paths, the inference engines have access to the measurements that are computed from the image. All the inference engines have equal access to the entire set of image information. Since the image information is given a-priori, there is no need to explicitly denote it as inputs to the inference engines.

In addition, the tuples in the hypothesis must be of the right form. For example, consider the input requirements of the inference engine that computes the focal length of the camera (See Chapter IV for details). It needs to have two lines in the image, which are not mutually parallel. Besides, it needs the computed vanishing points for each of the lines. The vanishing points must be nonsingular. Only under these conditions can the inference engine perform its computation. If this engine were to be controlled using an agenda based paradigm, all these checks would need to be performed by each inference engine which computes vanishing points of a line. Thus the inference engines which compute vanishing points for a line, need to know not only all the information about the line, but also all information about other lines in the image and their current status, to be

able to determine if the focal length expert should be invoked.

From the point of view of modularity, it is desirable that the amount of information that any given module needs to know should be restricted to the minimum necessary to achieve the desired efficiency. If the focal length computing module were to be updated to require some additional information as input, the amount of updating necessary to keep the system consistent would be minimized.

In the network approach, the only information that is provided to the modules which compute values for attributes, is the names of the inference engines which could potentially use the computed value. Thus all that the vanishing point calculation engines need to know is that there exists a module for the computation of focal length, which uses vanishing point that it calculates.

Once an inference engine has computed its values and updated the global database to reflect the changes, it then triggers the appropriate inference engine which could use the results of its computation. It is the job of the triggered module to check the conditions and determine if it indeed has all the required information. If a module gets triggered, and figures out that it cannot do its task, it turns itself off. It then stays off until some other module determines that it may be able use its output.

The key to the success of such a scheme hinges on the fact that once values are computed for any attribute, these values remain in the global database and are not lost. Thus if at a later time, some module can use them, they are still available.

In the case of the focal length module, consider what happens when some module computes the vanishing point of some line. Assume for the moment that this is the only line for which a vanishing point has been calculated. The vanishing point module turns on the focal length module. However, all the input conditions for the focal length module have not yet been satisfied. Consequently the focal length module turns itself off. It will stay off until a vanishing point gets calculated for some other line in the image. Once again it will get invoked, and will examine the available information to see if it can perform its task.

Initially all the modules are deactivated. The process that generates the hypotheses, turns on appropriate modules depending on the tuples being hypothesized. Not all these modules may actually be applicable. Each module then gets a chance to see the input and determine their own conditions. Those that are not applicable, turn themselves off. Others compute new values and in turn, mark other modules for activation.

Thus this scheme is decoupled in the sense that the conditions for activating each module is concentrated in the module itself. It has the flavor of an agenda based system. Modules know of the existence of other modules to the point that they know which modules could, under some circumstances, use the output that they generate. The advantage of this system over the pure demon based system is that most of the time, the majority of modules are deactivated and therefore do not have to be considered by the supervisory process. Thus there is no overhead involved in trying to invoke modules which are obviously inapplicable. The advantage over the pure agenda based system is the fact that the amount of information that other modules need to have about any given module is absolutely minimal.

The overall control algorithm consists of a supervisor, which sequentially polls every module that is not deactivated. If the module determines that it is not applicable, it deactivates itself. If on the other hand, the module is applicable, it performs its actions, and if it computes any new values for attributes, it in turn activates other inference engines. Note that if an active engine determines that it can perform its task, it does not turn itself off, because there may be more than one subset of the hypothesis to which it may be applicable. It turns itself off only after

it has performed its task on all the applicable subsets of the hypothesis.

We have shown in the previous chapter, that the order in which the engines are actually invoked does not affect the result. This means that the simple sequential scheduling policy described above is appropriate and no wrong conclusions will be reached.

3.4 CONTROL OF INFERENCE ENGINES USING ASSOCIATIVE TABLES

As was shown in the previous sections, one aspect of effective control is to be able to selectively enable and disable the inference modules. In addition to this, the inference engines need the capability to determine with minimum computational overhead, whether they are in the "on" state or the "off" state, and to find the subset of the data on the blackboard which they need to analyze, if they are "on". If they find a subset of the data to which they are applicable, they must then determine if that subset has been examined in the past. In this section we show that the use of associative tables greatly simplifies this task.

Consider the actions that an inference module may be called upon to perform. Since these actions all affect the global database, we may consider the generic actions that can be used on any data structure. These actions are: a) ad-

dition of new information, b) deletion of old information, or c) updating and changing existing information. In addition, an inference engine may determine that any information it could add, or update, already existed in the database, and consequently, would take no action. In general, these actions would be data dependent. That is, some combination of input conditions would be satisfied by the global data; based on this, the inference actions would be performed.

If the action of deleting information or updating existing information changed the preconditions for the inference engine, there would be no problem. Otherwise, the input conditions would still hold after the application of the inference task. In this case, if the module were to be polled again, it would still find the same conditions as before, and recompute the previously computed values.

There are several ways in which this problem could be tackled. One technique could be to construct all engines in such a way that either all changes to the database were such that the input conditions would be no longer applicable after the task. This could be done by making sure that one of the input condition determines whether the output variables of the engine have already been computed. Thus the same engine, if called on to test the blackboard, would not work with the same data. However, this is an undesirable situa-

tion in the context in which this model of inference engines was developed.

By our definition of consistency, we declare a hypothesis inconsistent precisely when an inference engine recomputes the value for a variable which was given a value by another engine, and finds that the two values do not agree. It hinges on the fact that values get computed more than once, and obviously, testing the output variable of the inference engine is not the solution we seek.

Another possible solution could be to define some kind of a search pattern in which an inference engine searches for the data on the blackboard for applicable input sets. Thus it would never return to a particular input without having scanned the entire possible sets of inputs on which it could perform its task.

This too is an undesirable situation, because for it to work successfully, we must have some way to ensure that once an engine has passed a particular point in its scanning pattern, no changes are made to the database which cause a new possible input set to be generated, which lies in the part of the search pattern already completed by the inference engine. Allowing the engines to repeat their search patterns just brings us back to the previous problem.

The solution that we seek is one in which each engine could remember the exact input conditions of the sets of data that it had used in its previous computation, and the next time it encountered the same input set, it would ignore it. Note that in this technique, the inference engine may encounter the same input set each time it is invoked, so we must rely on a rapid way in which the determination of whether it had previously seen the input, could be made.

Our solution to this problem involves the use of associative tables. Associative tables, which associate with a key, a prespecified value, allow the convenience of storing arbitrary keys, and rapidly accessing the stored information indexed by the keys. Our implementation of such tables, makes use of hash functions which can take any data structures such as lists, character strings, and indeed even other tables, as the key, and allow accesses with near linear expected time performance. Once an engine has examined a particular input set, it stores in the table the fact that "ENGINE x" has worked on "input data i, input data j...". Whenever the same data is encountered again, it can determine the fact instantly.

The same tables are also used to store information about the current status of the table, namely, whether it has been signalled as a possible candidate to be tried by some other

inference engine, or if it is in a quiescent state. This strategy allows the network model to work successfully, because all the time consuming bookkeeping tasks get reduced to rapid accesses into the associative memory. The supervisory task does not even have to check if an engine is in the "on" or "off" states, because now that task can be relegated to the inference engine itself. The first task of the engine is to determine its state. If it is "off", it exits and allows the next engine in line to start off. If it is on, it searches the blackboard for candidate input sets, and examines each one till it finds one it has not seen. It then performs the relevant actions on the input.

3.4.1 Conclusion

In this chapter we have demonstrated a technique for organizing and controlling inference engines so that the overhead consumed for performing their work, is minimized. We showed that the network organization permits the engines to run in a weakly coupled mode, where the major part of the computational effort required for determining the applicability of engines is left to each engine itself. It obtains a little help from the other modules which determine if it "may" be applicable. This allows easy updating of the knowledge contained in the system, and yet retains the efficiency of a coupled system.

We also showed how the use of associative tables controls the structure of the inference engines and simplifies the task of determining an input set of data for the engines to use.

In the next chapter, we discuss the knowledge that is encoded in the engines for the purpose of analyzing perspective line drawings.

Chapter IV

KNOWLEDGE SOURCES FOR ANALYZING PERSPECTIVE IMAGES

The problem of computer vision is to analyze images of three-dimensional scenes. The world itself can be described in terms of primitive entities and the attributed spatial relationships that hold between them. We do not know what entities are present in a scene being imaged, but we do know the types of entities that could be there. Similarly we know the types of relationships that could be used to describe the spatial arrangement of the entities, but the actual relationships that hold are unknown. The aim of computer vision is to determine these unknowns given the images and measurements made from them.

The problem is constrained by the fact that the process which gives rise to the image is known. It is a deterministic and well defined process whose physics and geometry is well understood. Under these constraints, not all possible choices for world arrangements, will be consistent with the given images. Thus, we can define interpretations to be those spatial arrangements which can be shown to be consistent with any information that can be extracted from the image.

The problem is complicated by the fact that the spatial relationships are not discrete, but have real valued attributes attached to them. For example, if we claim that two lines are parallel in the world, we still have to define the plane in which the lines lie, the distance between the lines and many other such attributes. It is easy to see that if one were to search for the solution over this entire space of real valued attributes, the problem would be intractable.

However, the fact that the projection process is a deterministic process, gives us a large body of coherence conditions which can actually be used not only for determining consistency of interpretations, but also to invert the projected image and compute values for the unknown attributes. This means that the search process does not have to perform its search over the large domain of all possible, real values. It is often sufficient to restrict the search to the finite set of un-attributed relationships and from them compute the real attributes that go along with them.

We have shown in the previous two chapters, that if such a physical system were to be found, the process of generating the interpretations can be viewed as a hypothesis generation and testing problem. The search consists of hypothesizing in a systematic fashion, various predictions that we feel could hold in the world. Then using the knowledge about

the projection process to compute the unknown attributes and to propagate the effect of the known or previously computed values, we can extend the hypothesis further. The process would terminate when we had a valid maximal interpretation or if the current hypothesis proved inconsistent.

We showed that if such knowledge existed, it could be organized as modular computational units, with which we could define consistency and inconsistency of postulated hypotheses. We showed that we do not have to look at all possible ways in which the inference modules can be applied to the generated hypotheses. The application process is essentially linear in nature. We also showed how the modular inference engines can be controlled in a sequential environment.

In both the previous chapters, we indicated that the problem of analyzing perspective line drawings was amenable to such a treatment. In this chapter we look at the equations of perspective geometry and show that indeed it is possible to use the inverse of the perspective transformation provided we are given hypotheses about the spatial configuration of lines in the three-dimensional world. The material in this chapter is a collection of equations which deal with the projection of lines and circles. The basic equations are well known. This chapter serves to collect a number of these equations and techniques in a unified notation.

4.1 GEOMETRY OF PERSPECTIVE PROJECTIONS OF LINES

In the computer vision task, the analysis of the kind described in this chapter would lie somewhere in the middle level processing phase. The low level processing involves essentially image level processing which can be used to remove noise from images, to the point of extracting image features and their properties. High level computer vision is involved with the task of recognizing and identifying the structures in the scene. The middle levels of processing are the least understood aspect of computer vision. It seems clear that these levels need to have progressively more and more information about the world being viewed. How much information and what kind of information is not yet known. We feel that one of the key pieces of information is the knowledge about the image generation process and how it interacts with whatever structures are in the scene. This information does not have any specific object models but rather has general knowledge about the relationships possible between entities in the world. Thus its reasoning power lies in determining a possible arrangement of world entities with respect to the reference frame of the camera itself. To determine the location of the camera with respect to the world requires more information about the world than would be typically available at this stage and therefore it seems

appropriate to refer that task to a higher level reasoning process.

We will consider the camera coordinate system to be a right handed system with the origin being concurrent with the center of the lens. The focal plane is located at a distance "f" down the Y axis and is parallel to the XZ plane. The scene is presented to the camera in front of it. The projection of a point P in the world corresponds to the point at which the ray from the origin of the lens to the point P intersects the screen. The distance "f" is called the focal length of the camera. This arrangement is shown graphically in Figure 5.

Consider a point P at coordinates (x,y,z) in the camera coordinates. From simple projection geometry, we can determine that the projection of P corresponds to the point P' with coordinates (xf/y,zf/y) in the screen coordinates. It is well known that given the location of the screen point P', the point in the world can lie anywhere on the line from the origin through P'. That is the inverse projection is not unique. It has been shown (Har80) that the inverse projection is a line. It is the locus of all points

$$(x,y,z) = L(x_s,f,z_s) \qquad (1)$$

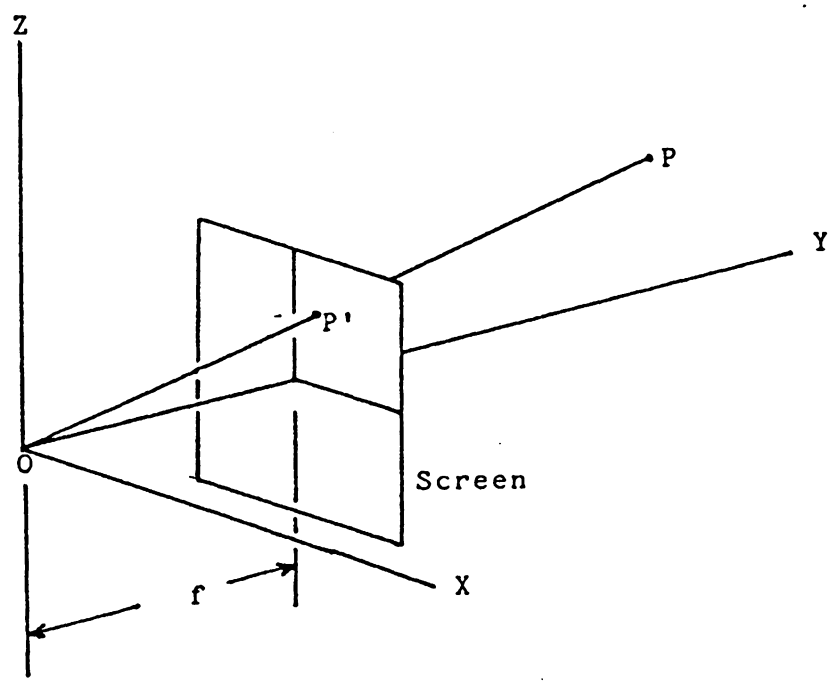


Figure 5: The camera coordinate system
The relation between the camera coordinate system and the world.

for real values of $L > 0.0$. where (x_s, z_s) is the location of the projection in screen coordinates.

It would seem that if the inverse projection of a point is not unique, there would be no hope for obtaining useful information about the world from a single image. However, knowledge about the spatial arrangement in the world provides suitable constraints to make the inverse computation feasible. Haralick (Har80) gives several examples of computations involving line drawings, Barnard (Bar82) provides several conditions under which backprojection of angles can yield useful information. In this chapter, we present several more equations which can be combined to yield interesting computable properties both for circles and for lines.

4.1.1 Vanishing Points

Under perspective projection, the image of a line is a straight line, or if the line is located so that it passes through the origin of the coordinate system, the projection is a point. The easiest way to see this is to consider the plane defined by the origin and the line which is being projected. The projection of the line is the intersection of the plane with the focal plane. If two planes intersect, they intersect in a line. Therefore the projection is a line.

The vanishing point of two parallel lines can be computed, and depends only on the direction cosines (a,b,c) of the lines. From (Har80), we know that the vanishing point of all lines with direction cosines (a,b,c) is:

$$x_v = f\ (\ a\cos\Phi - + b\sin\Theta)/\ D \qquad (2)$$

$$z_v = f\ (a\sin\Phi\sin\Theta - b\sin\Phi\cos\Theta + c\cos\Phi)/\ D \qquad (3)$$

where D is given by

$$D = -a\cos\Phi\sin\Theta + b\cos\Phi\cos\Theta + c\sin\Phi \qquad (4)$$

and theta and phi are the tilt and pan angles of the camera with respect to the coordinate system of the line. In our case, the two coordinate systems are the same, and consequently, both theta and phi are zero. This reduces the equations to:

$$x_v = fa/b \text{ and} \qquad (5)$$

$$z_v = fc/b. \qquad (6)$$

Note that if the line lies entirely in a plane parallel to the XZ plane, the y component of the line direction is 0.0. Consequently the vanishing point disappears to infinity. In subsequent equations, unless we specifically mention it, we will assume that all the vanishing points are not at infinity.

From these equations, we can see that if we know the f and the a,b,c for the line, we can compute the vanishing point. Since the direction cosines represent the unit vector in the line direction, we also know that

$$a^2 + b^2 + c^2 = 1. \qquad (7)$$

If we are given the focal length of the camera in addition to the vanishing point of an unknown line, we can compute the direction cosines from equations (5) and (6) as follows.

From (5) and (6) we have

$$a = x_v\, b \,/\, f \text{ and} \qquad (5')$$

$$c = z_v\, b \,/\, f. \qquad (6')$$

Since f is non-zero, we can perform the division without any trouble. Substitute this expression into (7) to get:

$$(x_v b/f)^2 + b^2 + (z_v b/f)^2 = 1 \qquad (8)$$

This expression can be solved for b, yielding two values.

$$b = \pm\, f*\mathrm{sqrt}(\, 1.0 \,/\, (x_v^2 + f^2 + z_v^2) \qquad (9)$$

The denominator in the subexpression above is a sum of squares and therefore non negative. Further, it is zero only when x_v, z_v and f are all zero, which is not possible in the real world.

Having obtained the value of b, we then substitute back in (5') and (6') to compute the remaining unknowns, a and c.

The physical interpretation of these expression is as follows: All parallel lines (those having the same direction cosines a, b, and c) have the same vanishing point. In particular, the line passing through the origin with direction cosines a,b,c also has the same vanishing point. Since the vanishing point lies on the line, the line through the origin with direction cosines a,b,c must pass through x_v, z_v. Given two points which lie on the line defined by the two points (0,0,0) and (x_v,f,z_v), the direction cosines are given by x_v/L, f/L, z_v/L where L is $\mathrm{sqrt}(x_v^2 + f^2 z_v^2)$. This is the same set of equations obtained above.

If the vanishing point of a line is known to lie at infinity, the line must lie in a plane parallel to the XZ plane, and consequently we know that the value of b must be 0. The line in three dimensions is parallel to its projection, and the a and c values are the same as those for the projection of the line.

4.1.2 Vanishing Trace of a Plane

The locus of the vanishing points of all lines which lie in a plane forms a line on the screen. This line is called the vanishing trace of the plane. Consider a plane with

normal (n_a, n_b, n_c). All lines in the plane have direction cosines (a,b,c) such that

$$an_a + bn_b + cn_c = 0 \qquad (10)$$

Let the plane be such that at least one of n_a or n_c is non zero. That is, the plane is not parallel to the XZ plane. Let the non zero term be n_a. Thus we have an expression for a which is

$$a = - (bn_b + cn_c) / n_a \qquad (11)$$

Substitute equation (11) into the vanishing point of the line with direction cosines (a,b,c) given by (5) and (6) to get:

$$x_v = fa/b = -f (bn_b + cn_c) / b\, n_a \qquad (12)$$

$$z_v = fc/b \qquad (13)$$

Substituting the value of fc/b, from (13) into (12) we get

$$x_v n_a = -z_v n_c - f n_b \qquad (14)$$

This equation is linear in x and z and therefore represents a line on the screen. Another way to see this fact is to recognize that all parallel planes have the same vanishing trace. In particular, the plane that passes through the origin parallel to this family of parallel planes, also has the same vanishing trace. The equation of this plane is

$$n_a x + n_b y + n_c z = 0 \qquad (15)$$

Its intersection with the screen is the required vanishing trace and therefore the equation of the trace is

$$n_a x + n_b f + n_c z = 0 \qquad (16)$$

which is identical to equation (14).

Given the vanishing trace of a plane with unknown normal, we can compute the values for (n_a, n_b, n_c) if we also know the value for f. Let the equation of the vanishing trace be given by $ax + bz + c = 0$. One of the planes in the family of parallel planes which shares this vanishing trace is the one that passes through the origin and contains the given trace on the screen. The normal to the plane may be obtained by taking the cross product of any pair of non-parallel vectors that lie in the plane. In particular, the vector from the origin to the point at which the trace crosses the x axis and the vector from the origin to the point at which the trace crosses the z axis will suffice. These two vectors are $(-c/a, f, 0)$ and $(0, f, -c/b)$ respectively. Thus, with simple algebraic manipulation, the normal vector in un-normalized for is:

$$(n_a, n_b, n_c) = (-fz_i, z_i x_i, -fx_i) \qquad (17)$$

$$\text{with } x_i = -c/a \text{ and } z_i = -c/b \qquad (18)$$

If a is zero, that is, the vanishing trace is parallel to the z axis, n_c is 0, and similarly if the trace is parallel to the x axis, the x component, n_a is 0.

The vanishing trace of a plane can be computed provided we know the non-singular vanishing points of two lines which lie in the plane. The vanishing trace is the straight line joining the two vanishing points.

Planes parallel to the XZ plane have no vanishing trace. All lines which lie in the plane (see previous section), have vanishing points at infinity. Thus if we are given two non-parallel lines in a plane, and both lines are found to have vanishing points at infinity, the plane must be parallel to the focal plane.

If we are given two lines in the plane which are not parallel to each other, and if one of them has a vanishing point at infinity, we can still compute the vanishing trace of the plane. To see this, note that the vanishing point of a line in the plane must lie on the vanishing trace of the plane. Further, the vanishing point must also lie on the extension of the projection of the line itself. Thus, if the vanishing point of the given line is at infinity, it must mean that on the focal plane, the vanishing trace is parallel to the projection of the line with vanishing point at infinity. Through a point on the plane, there is only one

line parallel to a given line, and its equation can be computed.

Let (a_i, c_i) be the vector parallel to the projection of the line whose vanishing point lies at infinity. Let (x_v, z_v) be the vanishing point of the second line. The equation of the vanishing trace in parametric form is then:

$$x = x_v + L\ a \tag{19}$$

$$z = z_v + L\ c \tag{20}$$

where L is a non-negative real number.

Given the equation of the vanishing trace of a plane, we can use that information to compute the vanishing points of all lines that lie in the plane, based on their projections. The vanishing points are the intersection of the vanishing trace with the extension of the projection of the lines.

4.1.3 Computing Focal Length

Both the inverse computations of the last two sections depend on knowing the focal length of the camera. It is possible to compute the focal length if we know the vanishing points of two lines which are known (or hypothesized) perpendicular. The only requirement is that the vanishing points be non singular. We first illustrate this by giving the geometric reason why the technique works, and then provide the algebraic equivalent of it.

Let a and b be two lines which are perpendicular, and let us be given (xa',za') and (xb',zb'), their vanishing points on the screen. Without loss of generality, we can assume that these lines pass through the center of the camera because the vanishing point of all parallel lines are the same, and if two lines a and b are perpendicular, and two lines a' and b' are given with a parallel to a' and b parallel to b', then a' and b' are also perpendicular. This situation is illustrated in Figure 6 in which A' is the given vanishing point of line a (and a'), and B' is the vanishing point of line b (and b').

Consider the three-dimensional triangle OA'B' shown in the figure. The lines OA' and OB' correspond to the vectors a and b respectively. Therefore the angle A'OB' is a right angle. Therefore the point O lies on the semicircle, with A'B' as diameter, which lies in the plane OA'B'. Further the point O on the semicircle is located such that the perpendicular dropped from O on the focal plane, passes through the origin of the screen coordinate system. This information is sufficient for calculating the focal length of the camera.

This fact may be shown algebraically as well. Let the first line have direction cosines (a_x, a_y, a_z) and the second line have direction cosines (b_x, b_y, b_z). Let their vanishing points be given by

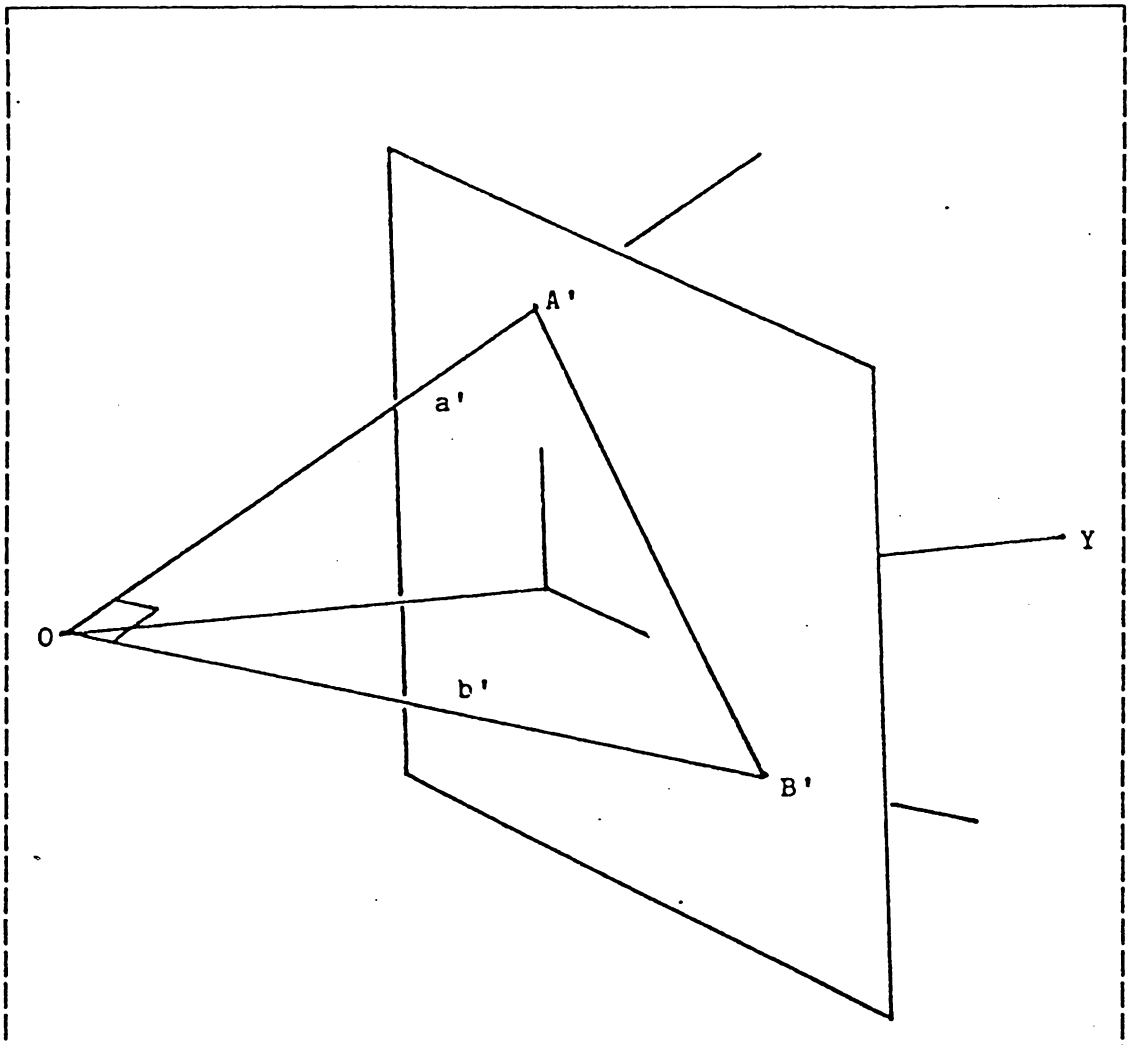


Figure 6: Computing the focal length
Illustrating the configuration for computing focal length given a pair of conjugate vanishing points. Line a' and b' are orthogonal. A' and B' are the vanishing points of the two lines respectively.

$A' = (fa_x/a_y, fa_z/a_y)$ and (21)

$B' = (fb_x/b_y, fb_z/b_y)$ (22)

respectively. If the vanishing points are non singular, we can write the expression for the components of the direction cosines in terms of the vanishing point coordinates and f. Then using the fact that the two lines are normal to each other, we can put:

$$a_x b_x + a_y b_y + a_z b_z = 0 \quad (23)$$

to obtain an expression for f in terms of the vanishing point coordinates. This expression is:

$$f = \text{sqrt} (- x_1 x_2 - z_1 z_2) \quad (24)$$

where A' = (x1,z1) and B' = (x2,z2). Since the focal length is positive, there is no ambiguity in which sign to choose. Vanishing points of orthogonal lines are called <u>conjugate</u> vanishing points.

This same calculation can also be performed with vanishing points of line known to belong to lines which make an angle of Θ degrees with respect to each other. In this case, equation (13) becomes

$$a_x b_x + a_y b_y + a_z b_z = C \quad (25)$$

where C = cos(Θ). If the angle Θ is known, f can be computed.

4.1.4 Center of the Screen Coordinate System

All the equations presented up to this point, rely on knowing the projections of lines and their vanishing points in screen coordinate system where the origin corresponds to the point where the camera axis pierces the focal plane. In most image processing applications, it is often the case that generated images are subimaged. The subimages taken may not be symmetric around the center of the original image. In this case, the equations presented above have to be modified.

In addition to restricting the amount of image available for use, subimaging may also cause the origin of the image coordinates to change. Without loss of generality, let the new screen coordinates (x',z') be given by

$$x' = x + dx \tag{26}$$

$$z' = z + dz \tag{27}$$

where dx and dz are unknown translations. Supposing we are given the vanishing points of three lines which are pairwise orthogonal. These values are given in the translated coordinate system. We have 12 unknowns, namely, the three direc-

tion cosines for each line, f and the unknown dx and dz. However we also have 12 equations in those unknowns: two each from the three vanishing points, three from the pairwise dot products of the three direction cosines and three each dealing with the length of the three unit vectors. Thus all unknowns can be solved for. This fact can be shown geometrically as well.

4.1.5 Projection of Points

The inverse of the perspective projection of a point is not unique. Indeed, actual three-dimensional coordinates of points in space cannot be computed based only on non-attributed spatial relationships between points, lines and planes. If all the linear dimensions of an object are multiplied by a non-zero constant, keeping the focal length f constant, the image will stay exactly the same. However, if the exact coordinates of any one point are known, or even if we know partial information about the coordinates of some points, we can propagate that information in closed form to related points, lines and planes. Haralick (Har80), provides a list of such useful transformations. We list them here in camera centered coordinate system for completeness. In all the following equations, we assume that the projection of the point does not coincide with the origin of the coordinate system.

The coordinates (x,y,z) of a point with projection (x',z') are given by the equations:

$$x = L\ x' \tag{28}$$

$$y = L\ f \tag{29}$$

$$z = L\ z' \tag{30}$$

for some positive value L. If we know one of the three unknown values x,y or z, we can solve for the other two. Supposing we know x, then we have:

$$y = x\ f\ /\ x' \tag{31}$$

$$z = x\ z'/\ x' \tag{32}$$

Similarly, if y is known for the point, we have

$$x = x'\ y'\ /\ f \tag{32}$$

$$z = z'\ y'\ /\ f \tag{33}$$

or if z is know, we can compute

$$x = x'\ z\ /\ z' \tag{33}$$

$$y = f\ z\ /\ z' \tag{34}$$

If two points in the world are known to have a common value in one dimension, we can compute the remaining values, provided we know the difference in one of the other two values. That is if we know that a line lies in one of the planes and we know the non-zero projection of the line on

one of the axes of the plane, we can compute the unknown coordinates of its end points. If for example, we know that the two points (x_1,y_1,z_1) and (x_2,y_2,z_2) are such that $z_1 = z_2 = z$, and we are given $x_1-x_2 = dx$ we can solve for z as:

$$z = \frac{dx\ z'_1\ z'_2}{(x'_1z'_2-x'_2z'_1)} \qquad (35)$$

Similarly if the difference dy in the y coordinates is given, we have

$$z = \frac{dy\ z'_1\ z'_2}{-(z'_1 - z'_2)\ f} \qquad (36)$$

Similar equations can be derived for the cases when the line is known to lie on planes of constant y or constant z. These equations are the counterparts of equations 32 to 37 in (Har80) with the values of Θ and Φ set to zero.

In (Har80), several other interesting numeric relationships are presented for special cases of lines known to be parallel to the axis. Parallel lines in the coordinate planes with known separation can be used to compute point coordinate information from the perspective image. However, to use these equations, we require more and more absolute information about the world. Such information takes us out of the realm of reasoning based purely on the information contained in the image and knowledge of the transformation process, and consequently should be left for higher levels in the computer vision hierarchy.

4.1.6 Projection of a Rectangle

In (Har81) Haralick shows that given a perspective projection of a rectangle of unknown dimensions, we can compute the location of the camera. In addition, given the absolute length and width of the rectangle in the world, we can also determine where it is located in space. In this section we show how hypothesis based reasoning using the relationships presented in previous sections, yields the same result. We show the chain of hypotheses and the knowledge used at each step, and derive the final result.

Let the projection on the screen consist of four lines a,b,c and d as shown in Figure 7, and let these lines correspond to the projection of lines A,B,C and D in space. Consider a hypothesis based reasoning scheme, which can hypothesize about lines and planes. Suppose it knows that lines in space can be parallel, perpendicular or at an arbitrary angle with respect to each other, and that lines can lie in the same plane. In the absence of any other contextual information, one of the possible hypotheses that could be generated would be the fact that lines A and C are parallel, lines B and D are parallel, with lines A,B; B,C; C,D and D,A mutually perpendicular. Further let us hypothesize that all the lines lie in a common plane. Assume that the projected quadrilateral is such that neither of the pairs a,c or b,d are parallel in screen coordinates.

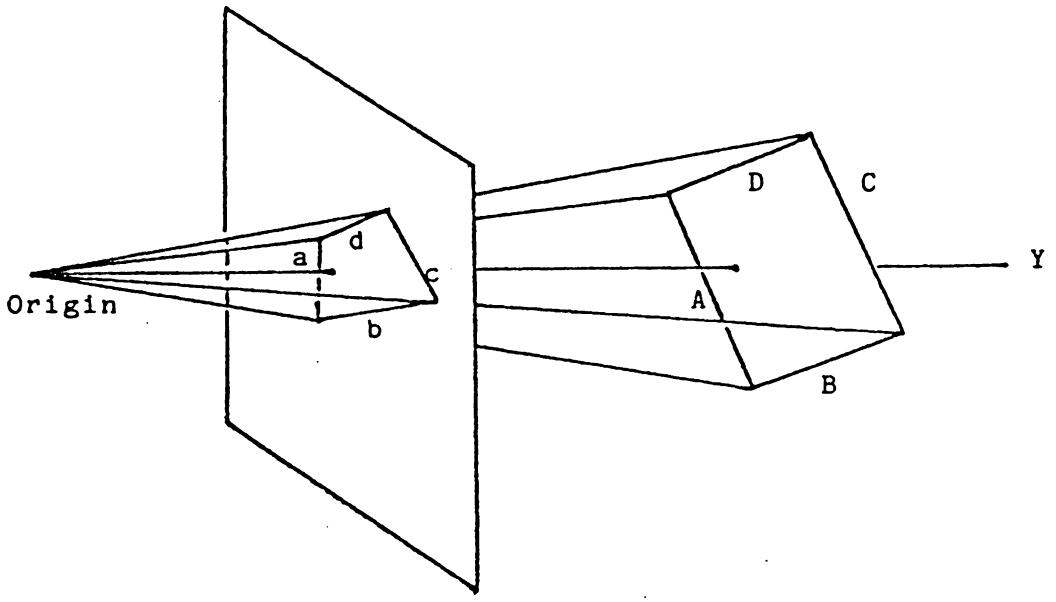


Figure 7: The perspective projection of a rectangle
A quadrilateral, which could be the
perspective projection of a rectangle in space.

Under the hypothesis that A is parallel to C, we can compute the vanishing point of the line A. It is the intersection of the lines a and c. Similarly the vanishing point of line B lies at the intersection of lines b and d. If lines A and B are perpendicular, their vanishing points are conjugate. Consequently we can compute the focal length f and the vanishing trace of the plane ABCD. Given these two pieces of information, we can compute the normal to the plane ABCD. Thus just based on the fact that we assume the quadrilateral to be the perspective projection of a rectangle in space, we can compute the normal to the plane in the camera coordinate system.

These computations give only one solution for the normal. In (Har81), the orientation of the plane is specified in terms of the angles Θ, Φ and Ψ corresponding to the rotation of the camera coordinate axes with respect to the coordinate frame of the plane of the rectangle. When computed in terms of the rotation angles, the solution of Ψ is found to be the arc tangent of some quantity. There is an inherent ambiguity of 180° in that solution. The solutions for Θ and Φ are obtained as trigonometric functions of the angle Ψ. Therefore at first glance it may seem that there is an inconsistency between the result in (Har81) and the result just obtained here. However we show that the two possible solutions in (Har81) are actually identical.

In the equations that follow, the equation numbers in square brackets refer to equation numbers in (Har81).

$$x'_i = x_i\cos\Psi - z_i\sin\Psi \qquad [2]$$

$$z'_i = x_i\sin\Psi + z_i\cos\Psi \qquad [2]$$

Substituting $\Psi+180^\circ$ for Ψ, and using the identities $\sin(\Psi) = -\sin(\Psi+180)$ and $\cos(\Psi) = -\cos(\Psi+180)$, and writing Ψ' for $\Psi+180$, we have

$$\begin{aligned} x''_i &= x_i\cos\Psi' - z_i\sin\Psi' \\ &= -x_i\sin\Psi + z_i\sin\Psi \\ &= -x'_i \end{aligned}$$

Similarly

$$z''_i = -z'_i$$

Substituting these expressions into the equations for Θ and Φ, we can obtain the values Θ' and Φ' corresponding to the second solution Ψ'. These are:

$$\begin{aligned} \tan\Phi' = \; & \{(x_2''z_4''-x_4''z_2'') \\ & (z_1''-z_3'') \\ & -(x_1''z_3''-x_3''z_1'') \\ & (z_2''-z_4'')\} \; / \\ & f[(x_1''-x_3'')(z_2''-z_4'') - \\ & (x_2''-x_4'')(z_1''-z_3'')] \end{aligned} \qquad [10]$$

Using the relation between the single primed and double primed x and z terms derived before, we can see that all

terms involving differences of x's or z's will change sign, and those involving products of the x's and z's will not. Thus we have

$$\tan \Phi' = -\tan \Phi$$

where Φ is the theta corresponding to the first solution for Ψ. If Φ and Φ' are restricted to lie between $\pm 90^{o}$, we have $\Phi' = -\Phi$.

Similarly inserting the values for x_i'' into [12] and noting that $\cos(\Phi') = \cos(\Phi)$, we have the relation,

$$\Theta' = -\Theta.$$

Thus the two solutions are related by the fact that the signs of two of the angles are reversed, and the third is 180^{o} greater. Now consider how the angles Θ, Φ, Ψ are related to the normal of the plane in camera coordinates. The angles were with respect to the plane Z = constant. Thus to find the normal to the plane in the camera coordinate system, we must rotate the vector (0,0,1) by the appropriate inverse rotations $(-\Theta, -\Phi, -\Psi)$ around the appropriate axes in the appropriate order. Doing this transforms the vector (0,0,1) to the vector $(-\cos\Phi\sin\Psi, \sin\Phi, \cos\Phi\cos\Psi)$. Replacing Φ by $-\Phi$ and Ψ by $\Psi+180$, we see that $\cos\Phi$ does not change sign, but $\sin\Psi$, $\sin\Phi$ and $\cos\Psi$ do. Consequently the vector corresponding to the second solution in the camera coordi-

nate system is simply the negative of the vector corresponding to the first solution. The two solutions thus correspond to the same plane.

The advantage of the reasoning path we used above over the solution in terms of the rotation angles is the fact that we do not introduce what look like extra solutions and then have to track them down and eliminate them. Further, we have shown by the reasoning scheme above, that we do not in fact need to know the value of f before the computation proceeds. Indeed, it is possible to compute both the orientation of the rectangle and the focal length of the camera from the perspective projection. After computing the orientation, we can then proceed along a path similar to that in (Har81) to compute the location, if we know either the lengths of the rectangle or the coordinates of one of the corners.

4.2 PERSPECTIVE PROJECTION OF A CIRCLE

In this section we will take the equation of a circle, and apply the perspective transformation to it. Then based on the equation of the resulting ellipse, we will generate numerical relationships between the unknowns similar to those for lines and points. All equations in this section are non-linear, and therefore, it is not possible to get

closed form solutions as is possible for lines. However, this is not a drawback since one can use numerical techniques for their solution in a computer vision system. The technique adopted here complements the equations presented by Chu (Chu83). We will discuss the relationship with Chu's work later in this chapter.

4.2.1 Perspective Projection of a Circle

Let us consider the implicit equation of a circle lying on an arbitrary plane in three-dimensional space. The coordinate system as before is the camera centered coordinate system with the Y axis pointing in the camera look direction.

Let the circle lie on some plane in space with the equation of the plane being given by

$$Ax + By + Cz + D = 0 \qquad (35)$$

Also let the equation be normalized so that the terms (A,B,C) correspond to the unit normal to the plane, with D being the distance of the plane from the origin. Thus we have:

$$A^2 + B^2 + C^2 = 1 \qquad (36)$$

Let the circle be centered at the point with coordinates (x_o, y_o, z_o). Since the circle lies on the plane, the center must also satisfy the equation of the plane. Thus we have:

$$Ax_o + By_o + Cz_o + D = 0 \qquad (37)$$

The equation of the circle is then the locus of all points (x,y,z) which simultaneously satisfy the conditions:

$$(x-x_o)^2 + (y-y_o)^2 + (z-z_o)^2 = r^2 \qquad (38)$$

$$\text{and } Ax + By + Cz + D = 0. \qquad (39)$$

which defines the intersection of a sphere of radius r centered at x_o, y_o, z_o with the plane Ax+By+Cz+D=0 passing through its center.

The equation of the perspective projection of this circle in screen coordinates can be obtained by a transformation of coordinates. We know from the previous section, that the projection of a point (x,y,z) is given by

$$x' = xf/y \qquad (40)$$

$$z' = zf/y \qquad (41)$$

Substituting the expressions for x and z from (40) and (41) into the equations for the circle (38) and (39), we obtain:

$$(x'y/f)^2 + xo^2 - 2x'yx_o/f + y_2 + y_o^2 + (z'y/f)^2 + z_o^2 - 2z'yz_o/f - r^2 = 0 \qquad (42)$$

Writing R^2 for the common subexpression $x_o^2+y_o^2+z_o^2-r^2$ we get

$$x'^2y^2 + R^2 - 2x_ox'yf + y^2f^2 - 2yy_of^2 + z'^2y^2 - 2z_oz'yf = 0 \qquad (43)$$

Similarly substituting (40,41) in (39),the equation of the plane, we get:

$$Ax'y/f + By + Cz'y/f + D = 0 \qquad (44)$$

From (44) we can obtain y in terms of the other variables as:

$$y = -Df / (Ax' + Cz' + Bf) \qquad (45)$$

provided the denominator is non-zero. However, it is easy to see that if the denominator were zero, the value of y would be infinite, and therefore would not satisfy the equation of the circle, if the circle were of finite radius located a finite distance from the camera.

Substituting equation (45) in (43), and removing the primes for convenience, we get

$$x_2D_2f_2/\delta^2 + R^2f^2 + 2xx_oDf^2 + 2Dy_of^3/\delta + z^2D^2f^2/\delta^2 + 2Df^2z_oz/\delta = 0 \qquad (46)$$

where δ is the denominator in equation (45). Since f is non-zero, we may divide by f and cross multiply by δ^2 to get

$$x^2D^2 + R^2[A^2x^2 + C^2z^2 + 2ACxz + B^2f^2 + 2ABxf + 2BCzf] + 2x_oxD[Ax + Cz + Bf] + D^2f^2 + 2y_ofD[Ax + Cz + Bf] + D^2z^2$$

$$+ 2z_o zD[Ax + Cz + Bf] = 0 \qquad (47)$$

Expanding equation 47 and rewriting it in the form $ax^2 + bxz + cz^2 + dx + ez + k = 0$ we get the following result:

$$a = D^2 + A^2R^2 + 2x_oAD$$
$$b = 2(x_oCD + z_oAD + ACR^2)$$
$$c = D^2 + C^2R^2 + 2z_oCD$$
$$d = 2(ABfR^2 + BDx_of + ADy_of)$$
$$e = 2(BCfR^2 + CDy_of + BDz_of)$$
$$k = B^2R^2f^2 + D^2f^2 + 2BDy_of^2 \qquad (48)$$

It is easy to see that the implicit form of the equation of the projected circle is highly non-linear. However, we will show that if the circle lies in planes parallel to the coordinate planes, the equation simplifies considerably. We will also show that if some of the unknown terms are known, we have enough independent equations to solve for the remaining terms. We will then show how these equations can be used in a hypothesis based reasoning process similar to the approach for lines.

4.2.2 Plane of the Circle Parallel to the Screen

First let us consider what happens to the equation of the projection when the circle lies in a Y=constant plane. The equation of the plane $Ax + By + Cz + D = 0$ then becomes $y + D = 0$, with A=C=0 and B=1. Note that as before we retain the condition that $D \neq 0$.

From (48), the equation of the projection becomes

$$x^2D^2 + z^2D^2 + 2xDx_of + 2zDz_of - r^2f^2 = 0 \qquad (49)$$

which corresponds to a circle with radius rf/D centered at $(x_of/D, z_of/D)$ in screen coordinates. Thus the center of the projected circle is the projection of the center of the original circle.

4.2.2.1 Inverse of the projection

Assume that we are given the projection of a circle known to lie in the y=D plane with D > 0, where the value of D is unknown. Let us see some of the inferences which may be drawn about the unknown parameters of the three-dimensional circle. The unknowns are x_o, z_o, D, f, and r.

Let (x_c, z_c) correspond to the center of the projected circle. This can be determined from the given equation of the projection. Let the radius of the projected circle be r'. We have the relation that $x_o = 0$ iff $x_c = 0$ and similarly $z_o = 0$ iff $z_c = 0$. This follows from the fact that both f and D are constrained to be non zero.

If any one of x_o and z_o are $\neq 0$, then

$$x_of = Dx_c \qquad (50)$$

$$z_of = Dz_c \qquad (51)$$

$$r\ f = Dr' \qquad (52)$$

Thus given any one of r, z_o or x_o, the other two values may be calculated. If both x_o, and z_o are zero, we must know any two of f, D or r in order to calculate the remaining unknowns.

4.2.3 Circle on a Z = Constant Plane

Consider now the appearance of a circle on a plane perpendicular to the screen. Since the perspective projection (and hence the locus) of a point P^* obtained from a point P by a rotation about the Y axis through an angle Θ is the same as the rotation of the projection of P about the origin through the same angle, we can consider only z=constant planes without any loss of generality. The appearance of circles on other planes perpendicular to the screen are simply rotations of the corresponding circles on the Z=constant planes.

Let the plane be given by $z + D = 0$ with A=B=0, C=1. Let the circle be centered at (x_o, y_o-D), with a radius r. Note that for the circle to lie entirely in front of the camera in the positive y half space, y_o must be greater than r.

From (48), the equation of the projection becomes:

$$x^2D^2 + 2xzDx_o + z^2(x_o^2+y_o^2-r^2) + z(2fDy_o) + f^2D^2 = 0 \qquad (53)$$

In this equation, note that the coefficient of the x term is identically zero. The determinant of the quadratic terms is

$$\det = D^2(y_o^2 - r^2) \tag{54}$$

Since D^2 is greater than zero, and $y_o > r$, the determinant is greater than zero. Thus the projected conic section is an ellipse. The center of the projected ellipse lies at

$$(fx_o/D^2, -fy_o/D) \tag{55}$$

whereas the projection of the center lies at

$$(fx_o/y_o, -fD/y_o). \tag{56}$$

4.2.3.1 Inverse of the Projection

Note the following interesting properties of (53). If the coefficient of the xz term is 0, the only possible value for x_o is zero. Further if the equation is normalized so that the coefficient of the x^2 term is 1, the constant term is exactly f^2. Equation (53) in normalized terms is

$$x^2 + 2xzx_o/D + z^2([x_o/D]^2+[y_o/D]^2-[r/D]^2) + 2zfy_o/D + f^2 = 0 \tag{57}$$

Note that given the equation of the projection in the form of equation (57), we can first compute f equal to the positive square root of the constant term. We can then determine

x_o/D, y_o/D and r/D. We have to be given one of the terms x_o, y_o or r to be able to compute the remaining two terms. This follows intuitively from the fact that all three are distance terms, and we know that if all linear terms are multiplied by a non-zero constant, the perspective projection remains unchanged. Thus we must know at least one of the distance terms involved to be able to relate the other terms.

What remains to be shown that if we have a circle in any arbitrary plane perpendicular to the screen, we can determine the rotation required to transform it to the z=constant plane.

4.2.4 Circle on any Plane Perpendicular to the Screen

Let the arbitrary plane be given by $Ax + Cz + D = 0$, and consider a rotation of axes through and angle of theta degrees clockwise around the y axis. The situation is shown diagrammatically in Figure 8.

In the new coordinate system, the coordinate of a point (x,y,z) is (x',y,z') with $x' = x\cos\theta - z\sin\theta$ and $z' = x\sin\theta + z\cos\theta$.

Let the rotation angle be given by θ = arctangent2(-D/C, -D/A), with $\sin\theta = -A$ and $\cos\theta = -C$. The equation of the plane in the rotated coordinate system becomes

$$A(x'\cos\theta+z'\sin\theta) + C(-x'\sin\theta+z'\cos\theta) + D = 0 \qquad (58)$$

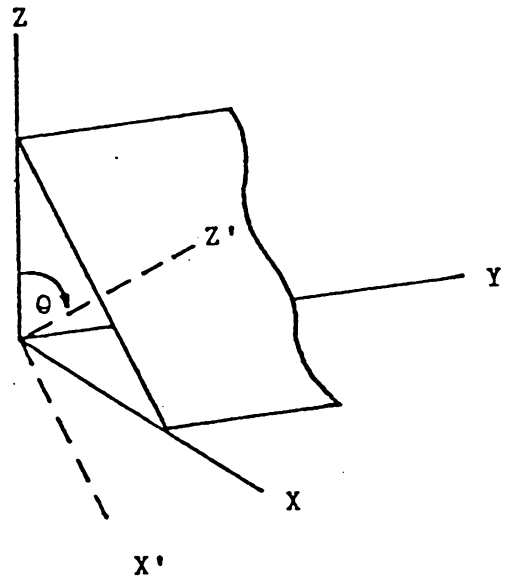


Figure 8: Rotation of the axes
Showing the rotation of axis by an angle of O around the Y axis. The plane with equation Ax + Cz + D = 0 is also shown.

which simplifies to

$z' - D/(A^2+C^2) = 0$ as required.

In the unrotated coordinate system, the equation of the projection of the circle is given by equation (). In that equation, we can see that the ratio of the x term to the z term is precisely A/C which is the tangent of the angle θ that we must rotate the coordinates by. Note that there is no ambiguity in the sign of A/C because y_o and f are constrained to be positive. The sign of AD is the same as the sign of D/A. Similarly the sign of CD is the same as the sign of C/D. The signs can be kept straight by noting that the sign of the constant term must be positive because it is a perfect square. Therefore, the angle through which the coordinates must be rotated is arctangent(A/C). If this curve is rotated in screen coordinates through the appropriate angle θ as before, with the rotation equations as specified, we have:

$$x = x'\cos\theta+z'\sin\theta = -(x'C + z'A) \qquad (60)$$

$$z = x'\sin\theta+z'\sin\theta = x'A - z'C \qquad (61).$$

Substituting (60) and (61) in (59), and on simplifying the result, we get

$$x'^2D^2 - 2x'z'Dx_o' + z^2(x_o'^2+y_o'^2-r^2)$$

$$- 2zfDy_o' + f^2D^2 = 0 \qquad (62)$$

which is precisely the equation of the projection of the rotated circle.

So given an ellipse known or hypothesized to be the perspective projection of a circle on an arbitrary plane perpendicular to the screen, we proceed as follows:

1. Normalize the equation so that the coefficient of the constant term is positive
2. Determine the rotation angle, (hence A and C) given by arctangent(Cx,Cz) where Cx and Cz are the coefficients of the x and z terms in the equation.
3. In the rotated equation compute f^2 which is the constant term divided by the coefficient of the x^2 term.
4. Substitute the value of f so obtained into the coefficients of the other terms to obtain x_o'/D', y_o'/D' and r/D'.

In real life, for circles which lie on the ground plane, we have a reasonably good estimate for the value of D. D corresponds to the height of the viewer. Therefore it is possible to obtain a reasonably good estimate to the actual size and location of the circle.

Equation (53) may also be used to illustrate a surprisingly non-intuitive result. Intuitively one would guess that if the projection of a circle looks like a circle, then the

circle must lie on a plane parallel to the screen. Indeed we have shown in the previous section, that if the circle does lie on a plane parallel to the screen, its projection is circular.

However, in (53) consider the case of a circle in the z = -D plane centered at $(0,y_o,-D)$. Also let D be such that $D^2 = y_o^2-r^2$. This does not violate any of the conditions laid down earlier. Under these circumstances, equation (53) reduces to:

$$x^2D^2 + z^2D^2 + 2zfy_oD + f^2D^2 = 0 \qquad (63)$$

which is the equation of a circle centered at $(0,-fy_o/D)$. The projection of the center of the circle however is at $(o,-fD/y_o)$, which is distinct from the center of the projected ellipse. Thus in this case, although the projection looks circular, the center of the ellipse does not coincide with the projection of the center of the circle as it does in the case when the circle lies on a plane parallel to the screen (See equation (49)).

We illustrated the fact that the projections of circles on two different planes could look similar, with an example. In the next section we will show that the two possible interpretations for a circle whose projection is given are related, and given one interpretation, it is possible to com-

pute the other. This relationship is useful when the equations for the inverse perspective transformation are used in computer vision.

4.3 TWO SOLUTIONS FOR THE PLANE OF THE CIRCLE

In this section we will consider the following question: If we know the focal length of the camera, and if we know how far away the circle is located, given its projection, can we determine the normal to the plane of the circle. If we can determine the normal to the plane, is the normal unique, or are there several solutions possible. Note that in the case of the rectangle discussed in the previous section, we had only one possible solution (barring a sign reversal). In this section, we show that for a circle, there are two solutions, and that these two solutions do not represent the same vector.

It is possible to follow the arguments in this section, with algebraic computations using the equations presented earlier for the general projection of the circle. However, the complexity of the resulting expressions increases so fast that it obscures the relationships that we wish to see. So we will restrict our analysis to a geometric argument which is equally convincing.

We are given an ellipse on the screen which corresponds to the projection of a circle on some arbitrary plane in front of the camera. We are also given the focal length. Let us see what the possible normal to the plane of the circle can be. From the origin, consider the cone formed by projecting rays through each point on the projection. This cone is in general, an elliptical cone. Consider any other plane intersecting this cone. In general (barring degenerate cases), the intersection will be a conic section. If the ellipse itself was formed as the projection of a circle in space, there must be some plane which cuts this cone of rays from the origin, with the intersection conic section being a circle. This situation is shown in Figure 9.

Consider the plane which intersects the cone at right angles to the center line. The resulting right elliptical cone is shown in Figure 10.

Consider the plane P which cuts the cone so that the intersection is a circle. This plane must be inclined with respect to the base plane B of the right elliptical cone in such a way that the line of intersection of P and B lies on the major axis of the ellipse. The reason for this is as follows. Consider the ellipse that results when the plane P cuts the cone. If the intersection line of P and B were aligned with the major axis of the cone, and if the tilt of

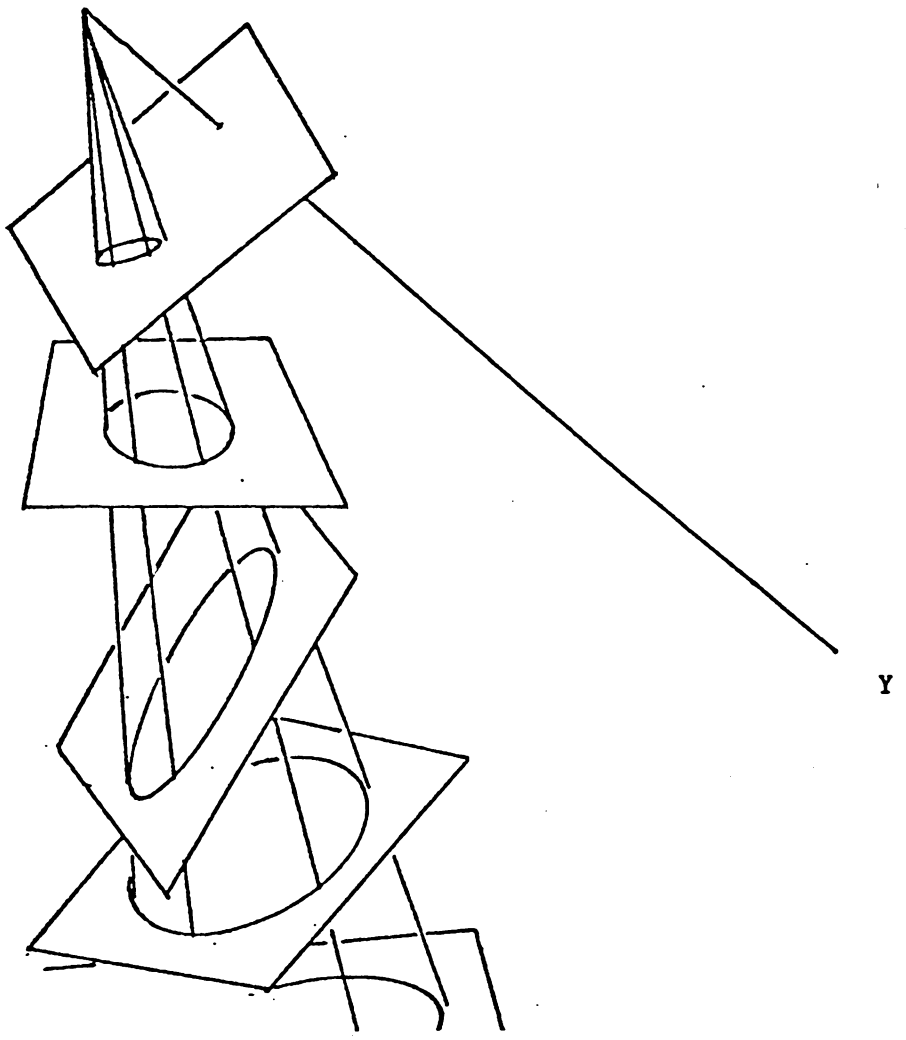


Figure 9: The cone of rays from the origin
This figure shows the cone of rays from the origin passing through the ellipse which corresponds to the projection of the circle. The conic sections which result when the cone is intersected by various planes are also shown.

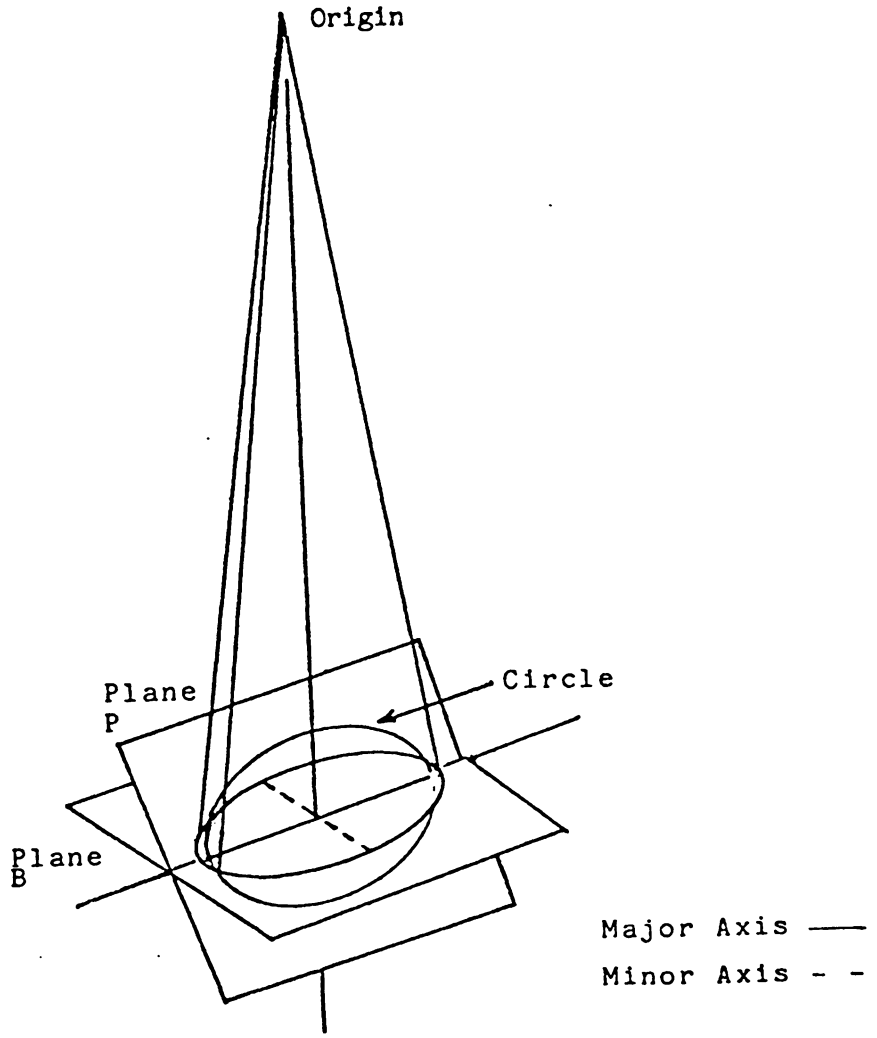


Figure 10: The right elliptical cone
The right elliptical cone corresponding to the cone shown in Figure 5.

the plane P with respect to B was increased from 0 to 90 degrees, the eccentricity of the ellipse in plane P would always be higher than the eccentricity of the ellipse in plane B. However, when the planes are inclined such that the the intersection line of P and B is parallel to the major axis, the eccentricity decreases. At one particular inclination, the eccentricity of the ellipse on plane P becomes equal to that of a circle.

Let the inclination of such a plane be θ^{o} degrees. Now because the elliptical cone is symmetric around the center line of the cone, a plane which is inclined at an angle of $-\theta^{o}$ will also intersect the cone with the same eccentricity for the intersection ellipse. These then are the two solutions for the plane which intersects the visual cone, such that the intersection is a circle.

Unlike the rectangle case, where the two solutions that result correspond to the same physical plane, the two solutions in this case are not identical. Of course if the right elliptical cone itself is actually a right circular cone, the two solutions will be identical. The two solution planes are shown in Figure 11.

To be able to use the analysis in a mathematical form, we need to be able to compute the equation of the axis of the right elliptical cone. Although this can be done given only

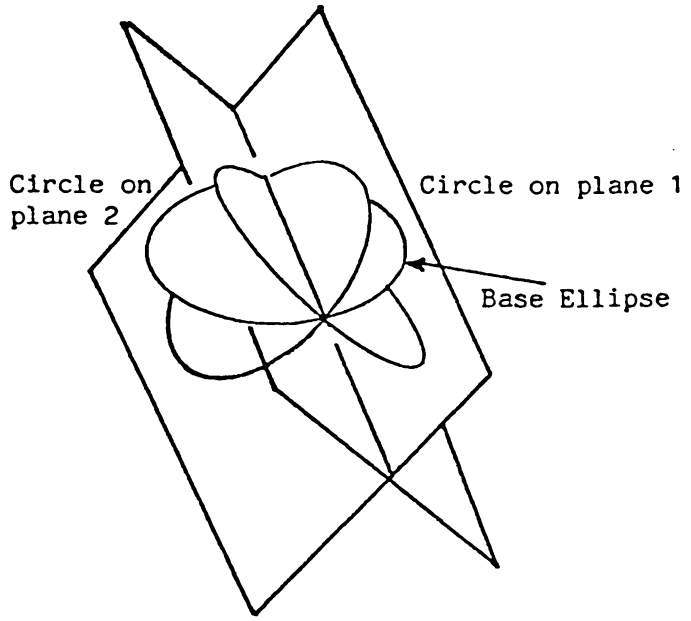


Figure 11: The two solutions for the circle
Showing the two solution planes
and their relationship when the cone is
right elliptical.

the projection of the circle on the screen, the equations involved are not particularly illuminating. The point on the focal plane at which the axis of the cone pierces the projected ellipse does not correspond either with either foci, or the center of the projected ellipse or the projection of the center of the original circle. However, we can, in closed form, compute the angle between the two solution planes provided we are given one of the two solutions. In the next section, we will discuss how this information, and other knowledge about the projection processes may be used to compute unknown parameters of the original circle given only the equation of the projected ellipse and some of the other variables.

4.4 INVERSE PROJECTION PROPERTIES FOR GENERAL CIRCLES

Although, just given the ellipse which forms the projection of a circle in any arbitrary plane, we cannot recover all the unknown parameters of the circle, we can recover some of the unknowns if we known some of the others. In computer vision tasks, it is rarely the case that the circle would be visible in isolation. Taken in context, there is always some extra information available about the image which helps to constrain some of the unknown parameters. For example, we may know the camera focal length either as

a-priori information or based on computations with lines as shown in the previous section. Or we may know that the circle lies on some particular plane. In this section we show how this information can be used to constrain the inverse perspective transformation process.

4.4.1 Computing the Focal Length

We show in this section, that given the normal to the plane of the original circle, and its projection, we can recover the unknown focal length and the ratio of the distance terms.

All the equations involved are non-linear and it is not possible to generate neat closed form equations for the variables involved. The approach that we take will consist of showing that we have the proper number of equations in terms of the knowns and unknowns, to enable a solution to be found. When this knowledge is used on a computer, the equations may be simultaneously solved using numerical techniques for the solution of non-linear equations or using some non-linear least squares minimization techniques. We have successfully used the LMDIF algorithm (Minpack) in the MINPACK subroutine library for solving these equations with very good convergence rates. On the average, only in 7 out of 100 computer runs using random initial guesses, did the algorithm fail to converge.

Let us consider the unknowns that we have to deal with: they are:

1. A, B, C, D, the coefficients of the normal to the plane.
2. x_o, y_o, z_o, the location of the center of the circle in space.
3. r, the radius of the circle.
4. f, the focal length of the camera.

We are given the equation of the projection of the circle in implicit form $ax^2 + bxz + cz^2 + dx + ez + k = 0$. Note that if all the coefficients in the equations of the projected ellipse were divided by a non-zero constant, the equation would still remain the same. That is we have one linear dependency in the coefficients. Let us assume that the equation is normalized so that the coefficient of the x^2 term is 1.0 to eliminate the linear dependency.

If we are given the value of A, B, and C (the normal of the plane of the circle), we have 6 unknowns (D, x_o, y_o, z_o, r, f). To solve for these, we have five equations from the five coefficients of the projected ellipse (note that we put the sixth coefficient to 1.0). In addition we know that x_o, y_o, z_o satisfies the equation of the plane. That gives us the sixth equation. We also know that since perspective projection is scale independent, we can over

constrain the solution process by specifying that the circle we are interested in is at a unit distance from the origin. In other words

$$x_o^2 + y_o^2 + z_o^2 = 1$$

The solution process returns a value for f, the center of the circle and r. Note that if we know the actual radius or one of the terms x_o, y_o or z_o, we can remove the last restriction and reduce the number of unknowns by 1. If we do not know one of these distance terms, the solution for these will still be in the correct ratio. If r, xo, yo, zo are the true values for these variables, the computed values r*, xo*, yo*, zo* will be such that they satisfy r*:xo*:yo*:zo*::r:xo:yo:zo.

The solution is over constrained, and can be solved using a least squared minimization technique.

4.4.2 Normal to the Plane of the Circle Given the Focal Length

If we are given the focal length, we have 8 unknowns, namely A,B,C,D,r,x_o,y_o,z_o. The number of equations is still 8. However, we know from the analysis of the previous section, that there will be two, non-identical solutions for A,B,C and D in general. There is no way to predict which of the two solutions, the optimization process will converge

to. The way we handle this situation, is to let it converge to any of the two solutions. Then we know that the second solution's normal vector is related to the normal of the first solution. By the computations discussed in the previous section, we can compute in closed form the angle between the two normals. We then repeat the optimization with the additional constraint that $A.A' + B.B' + C.C' = \cos(\Phi)$ where Φ is the angle between the normals. This guarantees that the minimization will not converge to the same solution a second time. In the equation above, A',B' and C' are the direction cosines of the normal computed as the first solution.

4.4.3 Normal to the Circle Given the Projection of the Center

The two solutions that can be obtained given only the value of f, can be distinguished from each other by the fact that the projection of the centers of the two circles are different, even though the ellipse that the circles project to are the same. Often it is possible to figure out where the projection of the center of the circle lies in screen coordinates. For example, if we have a rounded right angle, the center of the circle can be figured out knowing the vanishing points of the two tangent lines. The computation involved in doing this is shown graphically in Figure 12.

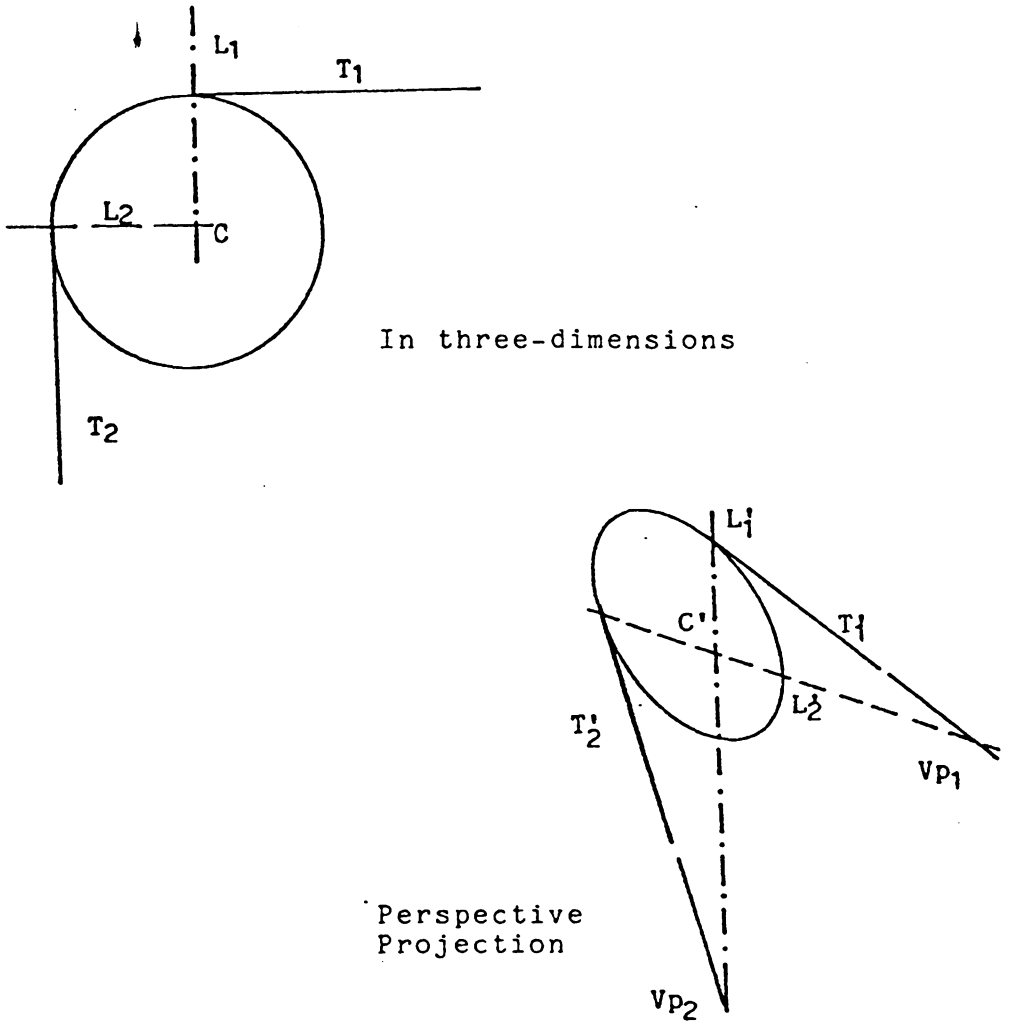


Figure 12: Constructing the projection of the center Graphical construction for figuring out the projection of the center of a circle given two perpendicular tangents.

Given the projection of the center of the circle, and the focal length we are left with 8 unknowns (A,B,C,D, r, x_o, y_o, z_o). To solve for them we have the 8 equations that we had for solving for the normal given the focal length. In addition we have two equations

$$x_c = x_o f / y_o \tag{64}$$

$$z_c = z_o f / y_o \tag{65}$$

where x_c, z_c are the coordinates of the projection of the center of the circle. If the minimization fais to converge, we can use the fact that since we know the f, we can compute the two solutions for A,B, and C. For each of these two solutions, we can compute where the projection of the center of the circle would lie. Then we select the solution for which the distance from the estimated x_c', z_c' to the given x_c, z_c is the least. This method works fine in practice, yielding solutions which are very close to the correct solution.

4.5 COMMENTS ON THE APPROACH OF CHU

The solution technique adopted in this chapter is different from the previous solution technique of Chu (Chu83). We first briefly describe Chu's analysis and then point out several differences between the two.

Chu follows the work done by Haralick in (Har81). Haralick showed that for the perspective projection of a rectangle, the computation of the Θ,Φ,Ψ parameters relating the camera look direction to the plane containing the rectangle, could be separated from the problem of computing the position of the camera in world coordinates. Following the same reasoning, Chu presents a technique for separating the camera angles from the position parameters in the case of the perspective projection for a general ellipse. Given a large set of points in the image which lie on the projected ellipse, it is possible to manipulate the equations of projection so that the translation terms are factored out. The remaining equations forming a very overconstrained system of equations can be solved using non-linear least squares minimization as we do here. The advantage of Chu's technique is that in the minimization, the number of equations depends linearly on the number of points sampled on the projected ellipse. Thus slight noise in the projection and sampling process does not result in large errors in the final result. However, if the equation of the projection that we use would also typically be computed from the given set of data points using overconstrained minimization. Thus we feel that the final errors in both techniques should be similar.

The form of the general ellipse considered in (Chu83) is parametrized by a and b, the major and minor axes of the ellipse; and the rotation angle of the ellipse with respect to the coordinate axes of the plane containing the ellipse. Setting a = b and ignoring the rotation angle (since circles are rotationally symmetric), we arrive at the case discussed for the circles in this chapter.

The form of the equations in (Chu83) however, hides some of the results which can be seen more intuitively in this chapter. Chu apparently does not realize that two solutions would be possible for the camera look angles and that these two would be non parallel solutions. In all experiments reported in (Chu83) there is no mention of the minimization converging to the second solution. Secondly Chu states that the camera look angles can be solved even if the eccèntricity of the original ellipse is unknown or if the the inclination of the major axis of the ellipse is unknown. This assumption is incorrect. The eccentricity of the original ellipse must be known in order to calculate the plane of the ellipse. Consider a very simple example. Let the projection of the ellipse on the screen also be an ellipse. If we assume that the original ellipse had the same eccentricity as that of the projection, one possible solution is that the plane is normal to the screen. If the original ellipse is

assumed to have any other eccentricity, the plane on which it lies cannot be normal to the screen. Thus there are a multiplicity of solutions possible if we try to solve for the eccentricity and the ellipse plane at the same time. Similarly, the angle at which the ellipse is inclined in the world coordinate frame can be shown to directly affect the camera look angles. If both of these are unknown, they cannot be solved for explicitly.

We feel that in the implicit form in which we have expanded the algebra, it gives us additional insight into the geometric nature of the equations. One of the research lines we would like to pursue later, is to expand equations for the projection of general conic curves, in implicit form similar to the development of the equations of the circle. Since the complexity of the equations is greater, such research would need the use of a suitable algebraic manipulation system, which we do not have access to at the present time.

4.6 LOGICAL CONSISTENCY RULES

In the previous sections, we discussed the knowledge about perspective gometry that can be used in a scene analysis system for inferring the attributes of three-dimensional spatial relationships using computations based on the mea-

surements made from the image. In a hypothesis based reasoning system, this knowledge would is encoded as modular inference engines, and consistency of hypotheses is defined in terms of the applications of the engines to the set of predictions in the hypothesis. A hypothesis is inconsistent if there is more than one reasoning path which can be used to compute the value for any attribute, and if any two of the paths yield contradictory values for the attribute.

In addition to the constraints imposed by the geometry of the projection, hypotheses must satisfy additional constraints. All the predictions in the hypothesis are relational tuples, drawn from some spatial relation on the entities. The relations themselves have semantics which define how the various relations interact with each other. These semantic constraints do not depend on the attributes of the relations, but rather, deal with the unattributed part of the relation themselves. For example, consider the spatial relation "parallel" which defines the three-dimensional line segments which are parallel. This relation may have attributes such as the distance between the parallel lines. Consider a hypothesis which states that lines a and b in the scene are parallel and lines b and c in the scene are also parallel. Regardless of the distance between lines a and b and the distance between b and c, we must have another rela-

tional tuple which states that lines a and c are parallel. This follows from the known transitivity of the parallel relation and does not depend on either of the two distances involved. In fact, it is independent of any attributes that may be attached to the parallel relation. Any hypothesis which consists only of the first two tuples, is incomplete. In a hypothesis based reasoning strategy, disallowing incomplete sets of predictions, drastically reduces the space that must be searched. As defined in the previous chapter, the search space consists of all subsets of possible predictions. With the restriction of logical completeness, the space now consists of all subsets of the predictions which are logically complete.

Groups of logically related predictions are termed chunks. Hypotheses can then be defined as chunks which are extracted from the set of all possible relational tuples which can be formed using the entities in the world and the known spatial relationships that may hold between them.

The applications of inference engines used for computing attributes runs in two modes. The first mode of operation consists of computing values for unknown attributes. The second mode consists of re-computing the values using alternate reasoning chains, and declaring hypotheses inconsistent if the values do not match. In a similar fashion, the logi-

cal consistency rules can be used in two modes. First, if we are given a hypotheses, we can use the rules to add the tuples which are logically implied by the tuples already in the hypothesis, and extend the set of predictions to form a "chunk". The second mode of operation is defined in a similar fashion. If the logical consistency rules deem that a subset of the hypothesized predictions imply some other relational tuple, and if the hypothesis also contains the logical negation of the implication, then the hypothesis is inconsistent. In addition, during the search, if the implied tuple has already been shown to be inconsistent with the tuples in the hypotheses (based on criteria such as attribute values, or prior world knowledge), then too, the hypothesis is inconsistent.

In this section, we list a set of rules which are used for determining logical consistency of a set of predictions. Since these rules convert sets of predictions into chunks, we call this knowledge chunking rules.

1. Perpendicular line interactions Given two pairs of lines a,b and b,c such that a is perpendicular to b and b is perpendicular to c, a is also perpendicular to c provided all three lines lie in a single plane.
2. Parallel line interactions Given two pais of lines a,b and b,c such that a is parallel to b and b is

parallel to c, a will always be parallel to c, regardless of the planes that they lie in.

3. Parallel and Perpendicular lines Given two pairs of lines a,b and a,c such that a is parallel to b and a and c are perpendicular, then b is also perpendicular to c.

These three rules, applied recursively to all subsets, reduce the search space drastically. For example, consider the analysis of the rectangle image discussed in a previous section. The image consists of 4 lines. Lines can be pairwise parallel or perpendicular. Thus there are $2\binom{4}{2}$ or 12 possible relational tuples which can be used to describe the world. With these tuples, we can form 2^{12} or 4096 different subsets. However, these subsets include several which are logically inconsistent. For example, the set {(parallel a b),(perpendicular a b)} is one possible subset. Ruling out such subsets, and applying the logical consistency rules defined above, the number of possible logically complete subsets (chunks) turns out to be only 289 - an order of magnitude lower. Further, using consistency criteria to extend partial hypotheses, reduces the effective depth of the tree search. Thus, for the rectangle image, when line a is hypothesized parallel to c, line b is hypothesized parallel to d and a is hypothesized perpendicular to b, all the other

relational tuples follow without a search. Thus the depth of the tree search reduces in this instance from 12 to 3 (again an order of magnitude reduction). After making the three predictions, and applying the consistency criteria, the final hypothesis consists of 6 tuples.

In the next chapter, we will discuss the actual implementation of a reasoning system based on the knowledge presented in this chapter.

Chapter V

IMPLEMENTATION AND EXPERIMENTAL RESULTS

5.1 INTRODUCTION

In this chapter we will describe our implementation of the techniques and theories presented in the previous part of this dissertation. We will describe how the system looks, its input and output data requirements and then describe the results of experimental runs performed on the system.

The actual source code for the system is given in appendix A. The code generated was a mix of RATFOR subroutines for performing specific procedural sections of the inference engine tasks, coupled with PROLOG code for the outer layers of the system. The PROLOG interpreter, written in RATFOR, with several language extensions necessary to support associative tables, arrays, procedural constructs and ease of interface to user supplied subroutines, was used as the base language for the reasoning system. The overall structure will become clear as our discussion progresses.

5.2 INPUT DATA STRUCTURE

The system constructed had knowledge about the relationship of perspective geometry to three-dimensional lines, and circles in arbitrary three dimensional planes. The input to the reasoning phase was presented as a set of PROLOG axioms with the rule name corresponding to the type of entity being described. The entities recognized by the system are:

1. LINE: Line entities are two-dimensional lines extracted from the image. The attributes of the line are the beginning and ending point of the line, and the equation, in screen coordinates of the line. Information about lines was encoded as an axiom:

   (line # (f t) (a b c))

   where # is the identification number of the line, f and t correspond to point identifiers which are known to the system, and a,b,c represents the equation of the projection of the form ax + bz + c = 0.

2. POINT: Point entities are junctions or line end points on the image. The properties of points are their absolute screen coordinates. Thus point information is encoded as:

   (point # (xs zs))

   where # is the unique point identifier, and xs, zs is the x,z coordinates of the projection on the screen.

The point number is referenced by the line descriptor and the arc descriptor.

3. ARCS: Arcs describe curves extracted from the image. Arcs also have unique arc identifiers, and are assumed to be conic sections. The information provided for arcs is:

   (arc # (f t) (a b c d e f k))

   where f and t are the identifiers of the beginning and ending point on the image, and a,b,c,d,e,f,k represents the implicit form of the equation of the arc in the form $ax^2 + bxz + cz^2 + dx + ez + k = 0$. In keeping with the normalization described in the previous chapter the value of a was always expected to be 1.0.

4. PLANE: Plane information included the list of lines and arcs whose three-dimensional counterparts occupied the same plane. Although this information can be obtained by a process of hypothesis, we felt that that would add to the complexity of the search and since partial plane information would be obtainable from the image as segmentation regions, this information could be made part of the "given". Thus the planes were encoded as:

   (plane # linelist arclist)

where linelist and arclist were lists of line and arc identifiers respectively. If a plane contained no lines or arcs, the corresponding lists were left empty.

These entities are extracted from the image and form part of the measurements to be used in determining consistency of the hypotheses generated by the search algorithm. In addition to this information, line intersection information is also precomputed from the image and supplied to the reasoning process as PROLOG rules. This information is supplied as rules called LINE_INTERSECTION. It is encoded as:

(line_intersection (#1 #2) (xi zi))

where #1 and #2 are the identification numbers of lines. By convention we keep the identification number 1 to be smaller than identification number 2. Thus the intersection of line 2 and line 1 will be encoded as (line_intersection (1 2) ...)

The lower half of this intersection matrix, consisting of rules of the form (line_intersection (#2 #1) token) encodes whether the lines actually meet on the image. That is, if the intersection point lies on either of the lines 1 or 2, token is set to "yes". Otherwise it is set to "no". It is this sort of flexibility that PROLOG allows us to use. Alphanumeric tokens can be intermixed with numeric values. In-

formation about lines that physically intersect on the screen is used to prune hypotheses. In addition, we adopt the convention that if the intersection point is at infinity, it is encoded with (xi,zi) set to (infinity,infinity). The equations of the lines are set so that the larger of the two quantities a and b are always 1.0. This avoids very large numbers from being produced, which may lead to overflows during computation.

The system has the knowledge that there are entities in the three-dimensional world which correspond to the entities observed on the image. These entities are:

1. LINE3D: These are the three dimensional lines corresponding to the observed two dimensional lines. We assume that lines in the three dimensional world do not correspond to single points in the image, because such a condition can only arise because of a very special alignment of the camera with the world. The attributes of the three dimensional line entities are:

   (line3d # (xv zv) (a b c))

   where xv and zv correspond to the vanishing point on the image. This information, although it is in screen coordinates, results because of hypotheses made regarding the orientation of the line in space and,

consequently, is part of the three dimensional data structure for the entity. A,b, and c represent the direction cosines of the line in the three-dimensional camera coordinate frame.

2. POINT3D: These are the three dimensional coordinates of the points which correspond to the points and junctions observed in the image. The information is encoded as:

   (point3d # x y z)

3. PLANE3D: Information relevant to planes in the world is the equation of the vanishing trace of the plane, the normal to the plane and the distance of the plane from the origin of the coordinate system. This is encoded as:

   (plane3d # (na nb nc) (ea eb ec) d)

   where the n's describe the normal to the plane, the e's describe the equation of the vanishing trace in the form eax + ebz+ec = 0. The d represents the distance of the plane from the origin. As in the case of the three dimensional lines, note that although the equation of the vanishing trace is in screen coordinates, it results because of hypotheses regarding the three-dimensional orientation of the plane and lines in the plane. It is not computable from the

image alone. Consequently it is stored as part of the three-dimensional data structure.

In addition to these world entities, axioms are asserted for the global variable "focal length" as a rule of the form (focallength f)

At the beginning of the reasoning process, none of the numerical three-dimensional attributes are know. That is represented in the rules by the token "unknown". Ofcourse, if any of this information is known from other analysis before the reasoning phase, it can be supplied to the system in the appropriate slot in the appropriate axiom. For example, if the normal to a plane is known from shape-from- type of analysis, we may encode it as numerical values in the appropriate plane3d rule. User supplied values are never overwritten during the initialization phase.

As the reasoning process proceeds and values are computed for the three-dimensional attributes, the appropriate slots in the axiom are filled in with the computed numerical values. By convention, whenever a piece of three-dimensional information consists of a set of values, either they are all unknown, or they all have numerical values. For example, the direction cosines of a line will be (unknown unknown unknown) or have three real numbers. The only exception to this rule is the x, y and z coordinates of three dimensional

points because using inference engines described for points in the previous chapter, it is possible to have partial information about point coordinates, and then use the partial information for propagation.

The spatial relationships that are used for the propagation of attribute values and for the consistency checking, are the following:

1. PARALLEL: Parallelism between three dimensional lines.
2. PERPENDICULAR: Lines being perpendicular
3. CIRCLE: Arcs being circular in three-dimensional space.

5.3 GLOBAL TABLES AND CONTROL OF ENGINES

The main vehicle for storing control information are the associative tables available in PROLOG. Associative tables can take any of PROLOGs data structures as the key and can store any of the data types as the value associated with the key. Initially, all the possible relational tuples that can be constructed, are generated and stored in global associative tables. There are two tables, called %table and %onoff. %table maps the integer index of the relational tuple to the tuple itself. For example, the value corresponding to integer 4 in %table could be (parallel 1 2). %onoff contains a

token "on" or "off" corresponding to each tuple. If a tuple can still be added to the hypotheses for extension, the token is "on". If however, forward checking or other means have detected that the tuple is inconsistent and cannot be added to the current hypothesis, the tag is set to "off". All changes that are made to the data base including attribute values computed, tuples set to on or off, rules added or deleted, are recorded as axioms of a special rule called "saver". Whenever the tree search recurses a level deeper, the status saving routine is informed of the new level. Whenever the global information is changed, a record is kept of the changed information and the level at which the change was made. This way, recursion does not entail recording the complete state at each level. On backtracking to the previous level, the changes are undone.

Whenever an inference engine examines a particular tuple or a set of tuples, it also records that information in an associative table. The level at which the check was made is also recorded. This information is also undone when control backs up.

5.3.1 Control of Inference Engines

The control of the inference engines is done by a supervisory rule which executes all the engines in a loop. The basic structure of the engines is as follows: The rules corresponding to the engines all have the same name "checker". The supervisor invokes "checker" and if it returns "success", it loops, invoking checker again. If a check fails, the appropriate inference engine returns "fail".

The rules corresponding to the engines are arranged in sequence. The sequential scanning action of the supervisor results from the sequential search performed by PROLOG when looking for a rule to unify with. The first action performed by each rule is to turn off the previous inference engine by appropriate change to a global associative table. It then checks if its own flag is set to "on". If not, it fails. PROLOG's control then passes to the next rule for "checker", which is the next inference engine in the sequence.

The last rule in the sequence of rules for "checker", returns a special token which indicates that all the rules above it were tried, and none of them applied. This is the indication that the hypotheses is successful, because it corresponds to a sequence of applications which terminates when no more engines are applicable (See Chapter II).

5.4 EXPERIMENTAL RESULTS

The system described here was systematically tested with a large variety of test data sets. Some of the data was generated by a graphics program for producing perspective projections of three-dimensional models stored on the computer. Other data was obtained by hand-digitizing perspective drawings on a data tablet. DeVries (DeVries), is an excellent source of perspective drawings, and it was used as a source of data. Figures 13, 14, and 15 show three drawings from the book. Lines were digitized and preprocessed to form the data structures described previously. Appendix B lists the input data structure corresponding to the bold lines shown in these figures.

The results for the computation are shown in Appendix C for the same data set in terms of the parallel and perpendicular lines and the computed parameters for the planes.

5.5 ANALYSIS OF TIMING AND SOLUTIONS GENERATED

In this section we will describe some of the measurements made on the search process. The experiments were carried out using a set of six engines (Engines C1 to C6 in Appendix A), and optionally the entire group of "chunking" rules from the program. Forward checking was turned off and on to measure the relative savings obtained from the extra computation.

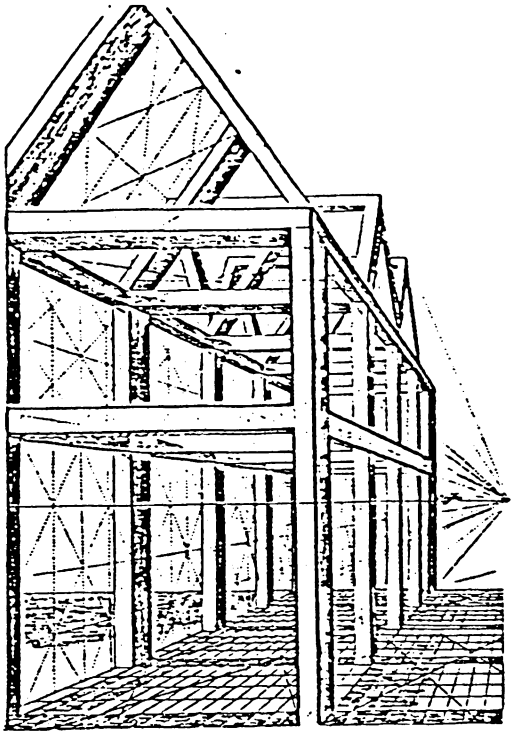


Figure 13: An architectural scene
An architectural scene from DeVries.
The lines digitized are shown bold. The input data structure corresponding to this image is exhibited in Appendix B, and the results in Appendix C.

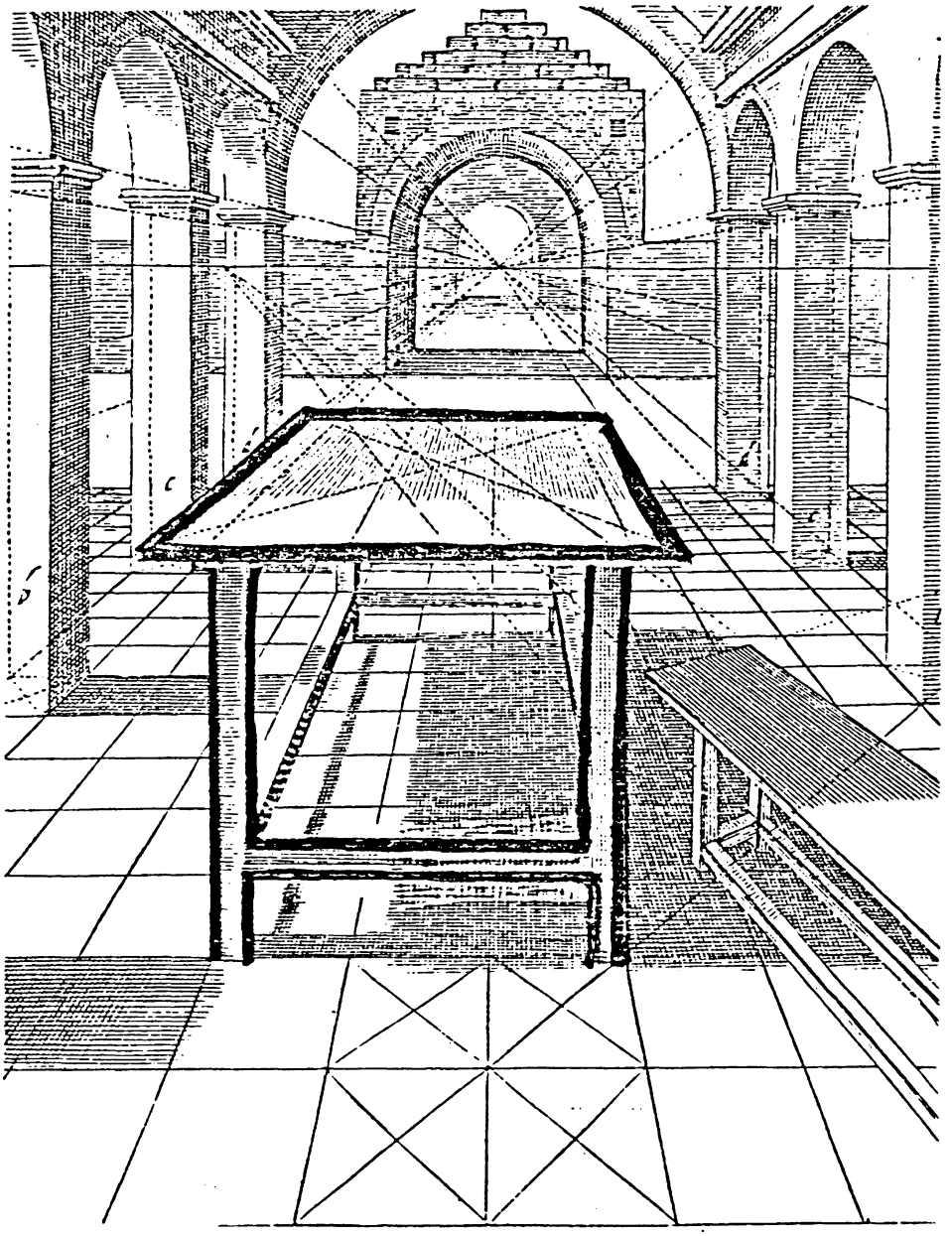


Figure 14: The drawing of a table scene
Another perspective drawing from DeVries.

The input data set that was used for these tests consisted of the projection of a square in the X=constant plane. The test data is shown in Appendix D.

For this test data set, there were 4 lines and one plane. The number of possible relational tuples was 12. This implied that the search tree was 12 levels deep with 4096 (2^{12}) leaf nodes or possible solutions. The number of true solutions (maximal consistent hypotheses) was 10. The number of consistent hypotheses was 289. When logical consistency was also checked along with numerical consistency, the number of true solutions dropped to 5.

Figures 16 and 19 show the cumulative number of times that the inference engines were invoked as a function of the true solutions output, both with forward checking and without. Figures 18 and 21 show that with forward checking in effect, the number of subset solutions generated for each true solution found, decreased. Figures 17 and 20 show the relative cpu time used for each true solution generated. The graph shows that initially forward checking does more work, and that this work pays off only when we want to generate more than one solution. If we only desire to find the first solution, forward checking is an unnecessary burden.

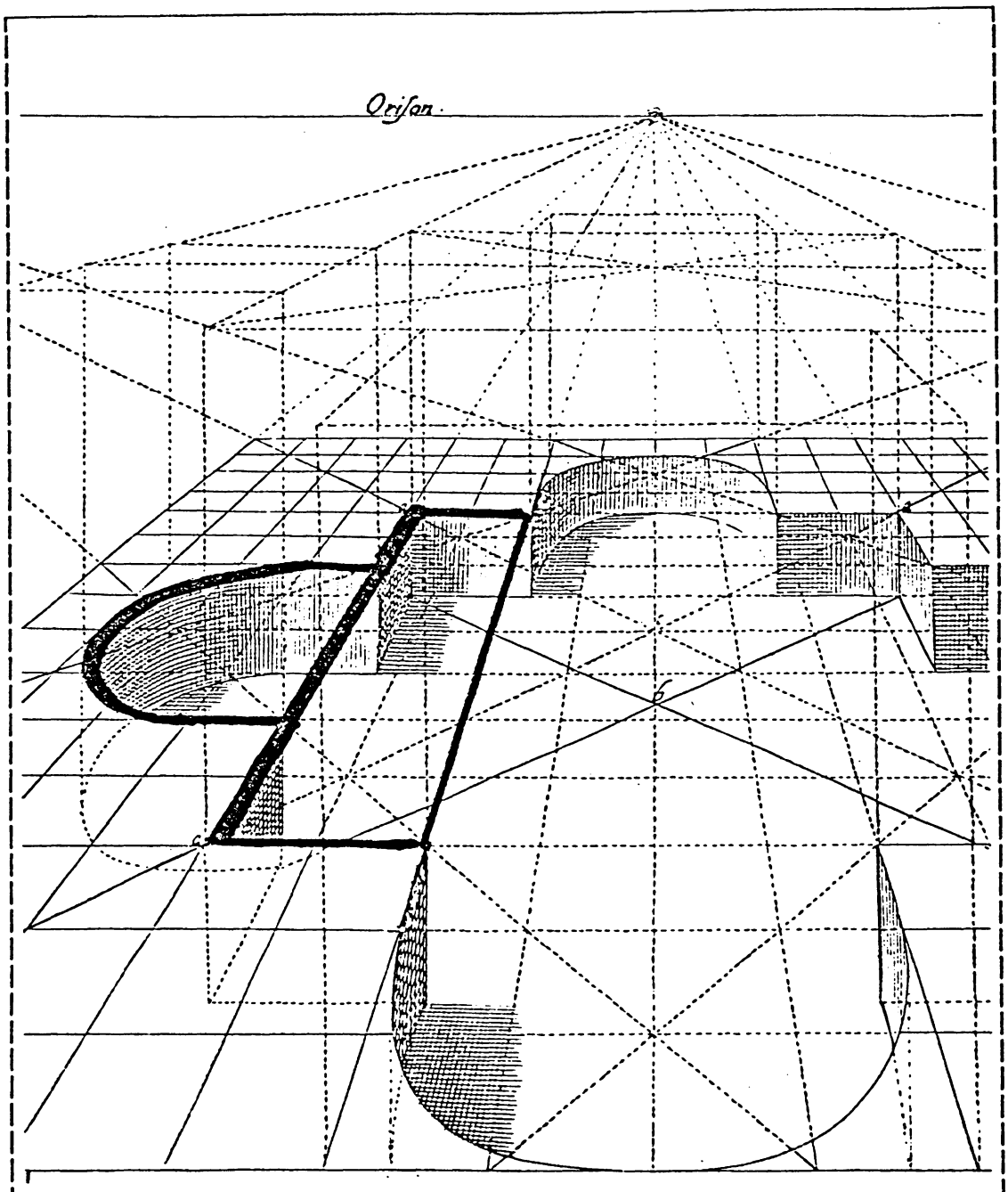


Figure 15: The base of an ornaté fountain
Yet another successfully analyzed image.

Figure 22 shows that without forward checking, the size of hypotheses grow large. The size of the hypotheses whose consistency must be checked increases. The extra work that forward checking performs means that the average size of the hypotheses does not grow. Figure 24 shows that the average cpu time taken to examine a hypothesis of a given size is larger with forward checking that without. The reason for this is that when forward checking is in effect, the hypotheses that the system is called on to analyze, tend to be consistent more often than they are incorrect. This is because, inconsistent tuples get ruled out early. This fact is shown in Figures 28 and 30. Figure 28 shows the number of times that the consistency procedure is invoked as a function of the size of the hypotheses for both consistent and inconsistent hypotheses. The other graphs show similar results for different combinations of cases with and without chunking, with and without forward checking in all combinations.

The conclusions that we may draw from these graphs are as follows: The efficiency of forward checking lies in the fact that it rules out possible subtrees early. This fact pays off only when the ruled out subtree is going to be en-

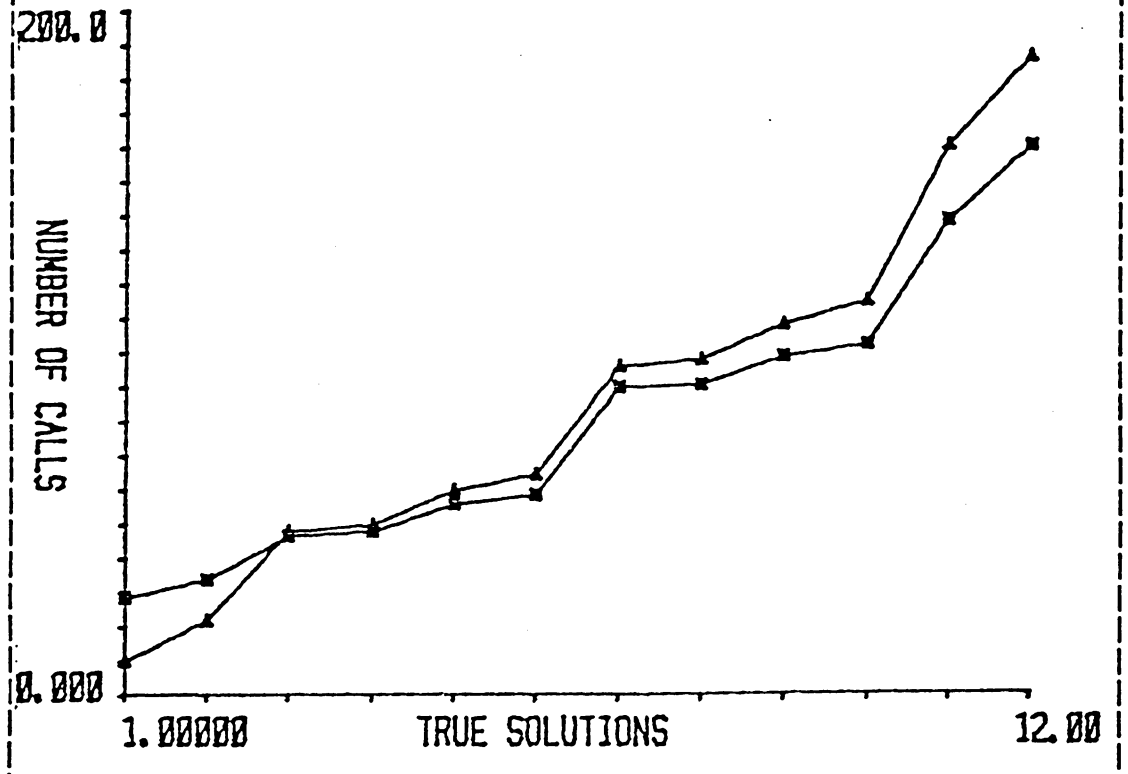


Figure 16: Number of calls without chunking
Number of calls made to the consistency checking module as a function of the number of leaf nodes visited. With forward checking (□) and without (△).

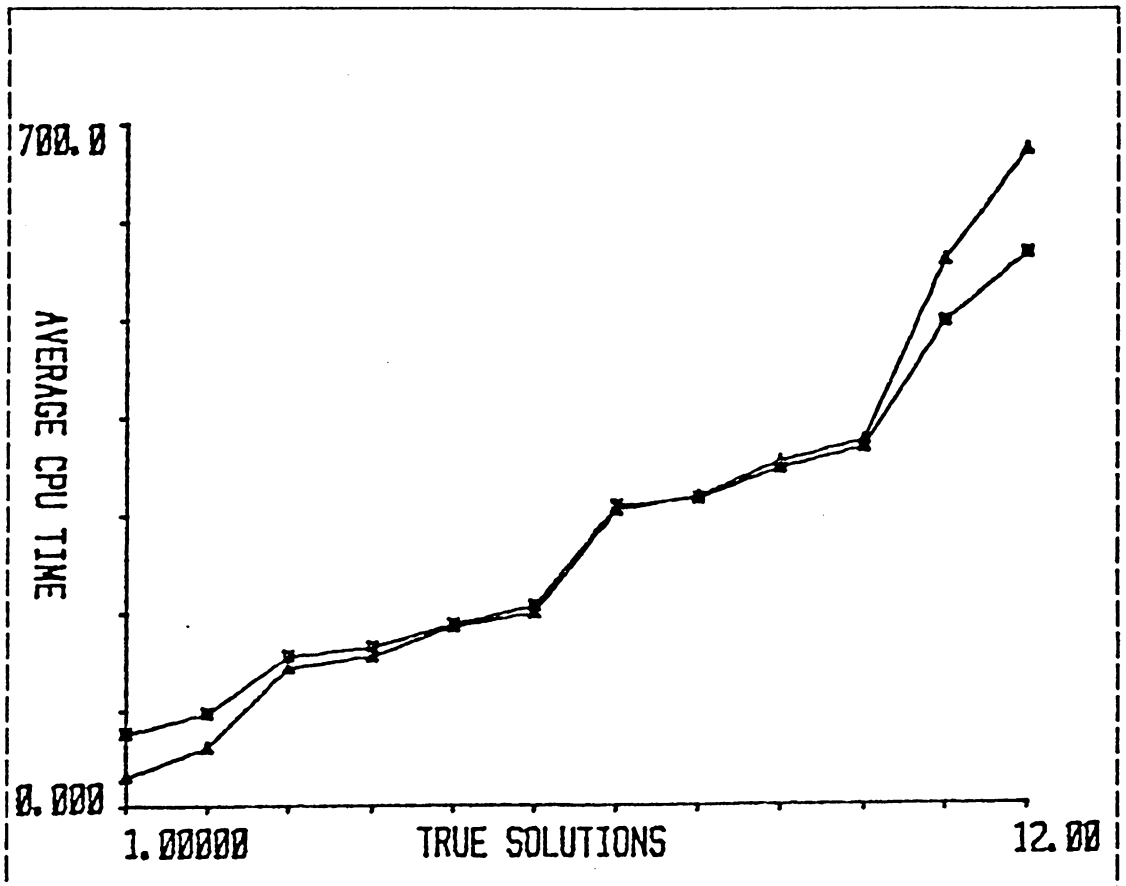


Figure 17: Average CPU time in seconds
Comparison of the CPU time in seconds per solution generated. With forward checking (□) and without (△). Chunking not in effect.

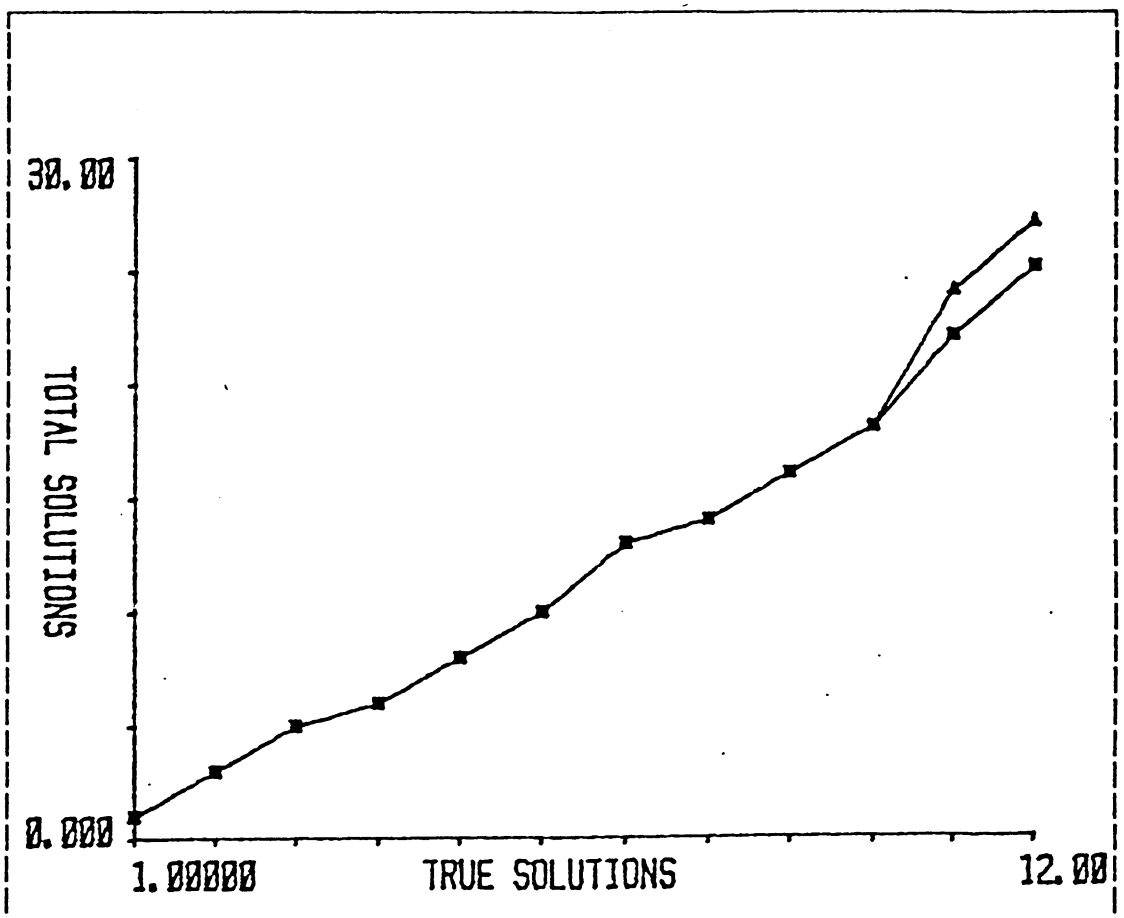


Figure 18: Efficiency of the process
Comparison of the number of true solutions
with the number of total leaf nodes visited.
With forward checking (□) and without (Δ).
Chunking not in effect.

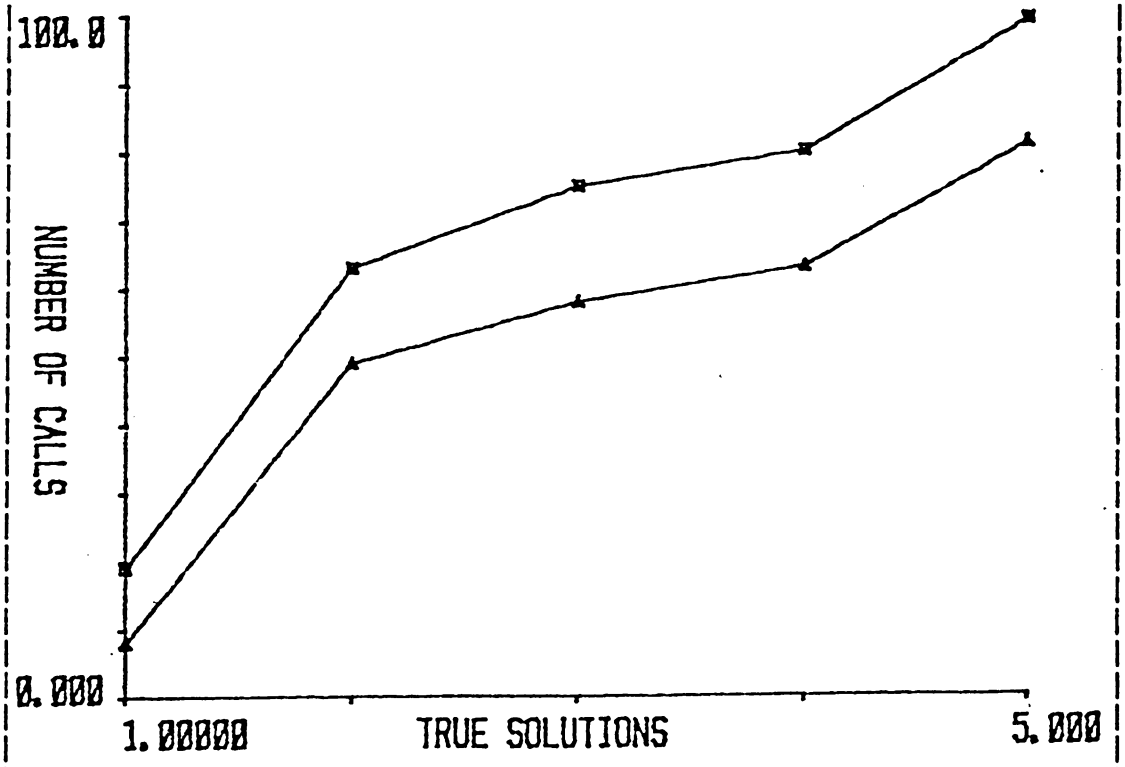


Figure 19: Number of consistency checks
Number of calls to the consistency checking module with chunking in effect. With forward checking (□) and without (△).

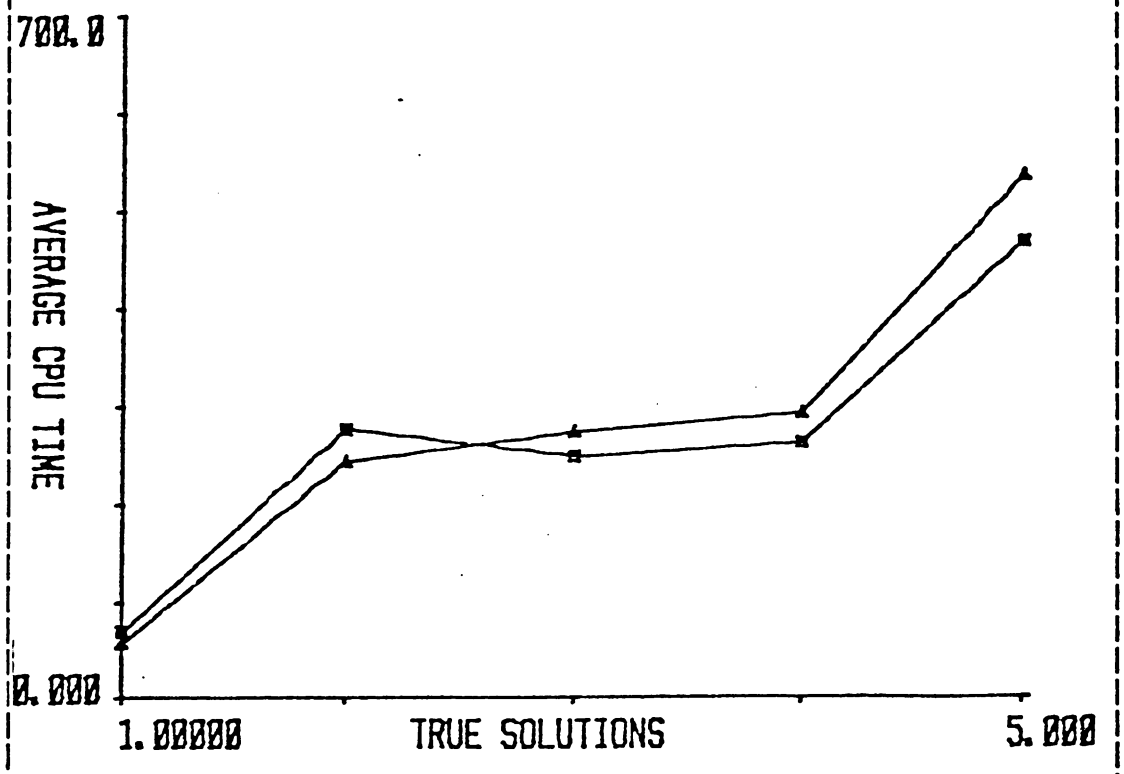


Figure 20: Average CPU time in seconds per solution
The average cpu time as a function of the leaf node visited, with chunking in effect. Note fewer leaf nodes get visited than in figure 17.
With forward checking (□) and without (△).

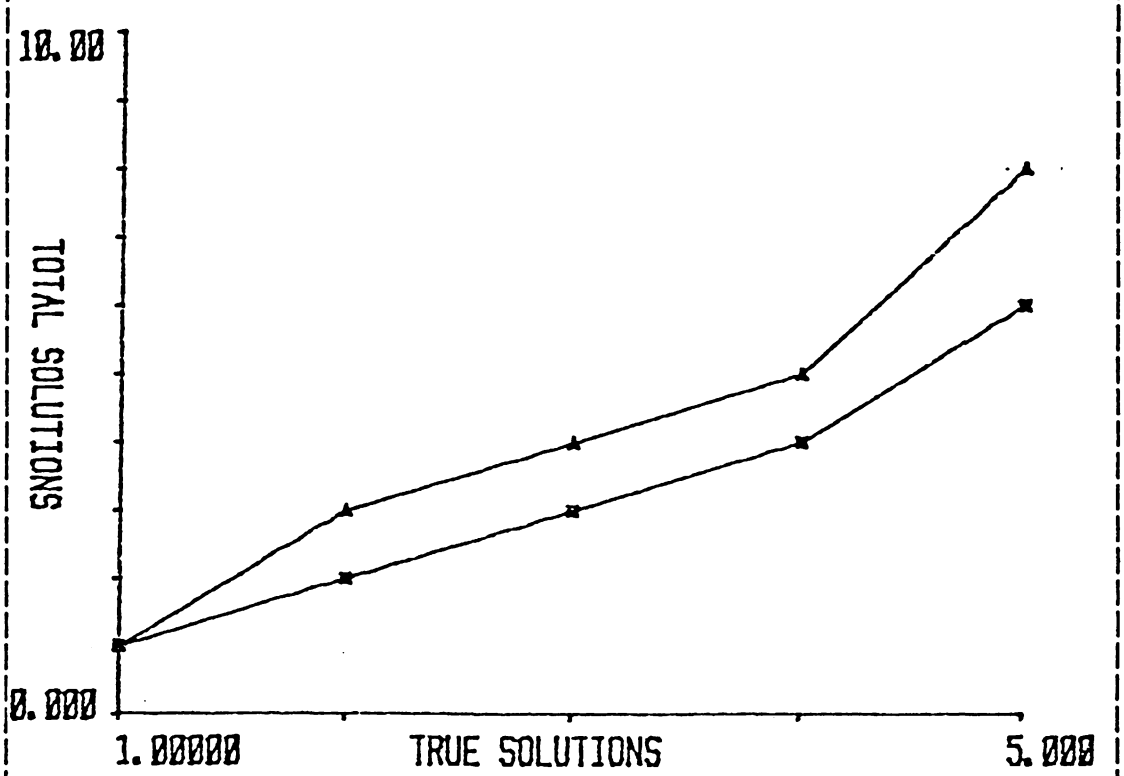


Figure 21: Efficiency of subset checking
Comparison of the number of true solutions generated with the number of leaf nodes visited with forward checking (□) and without (△), with chunking in effect.

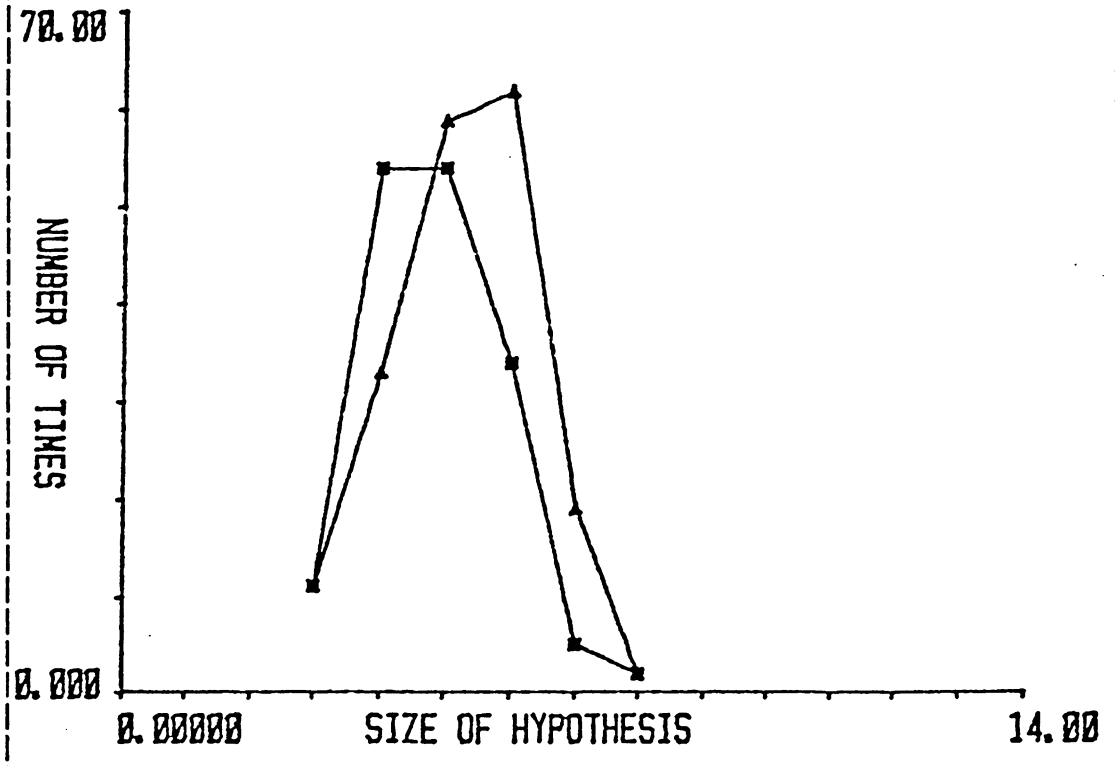


Figure 22: Number of hypotheses vs. size
Number of times the consistency of hypotheses was checked, as a function of hypothesis size. With forward checking (□) and without (△), and chunking disabled.

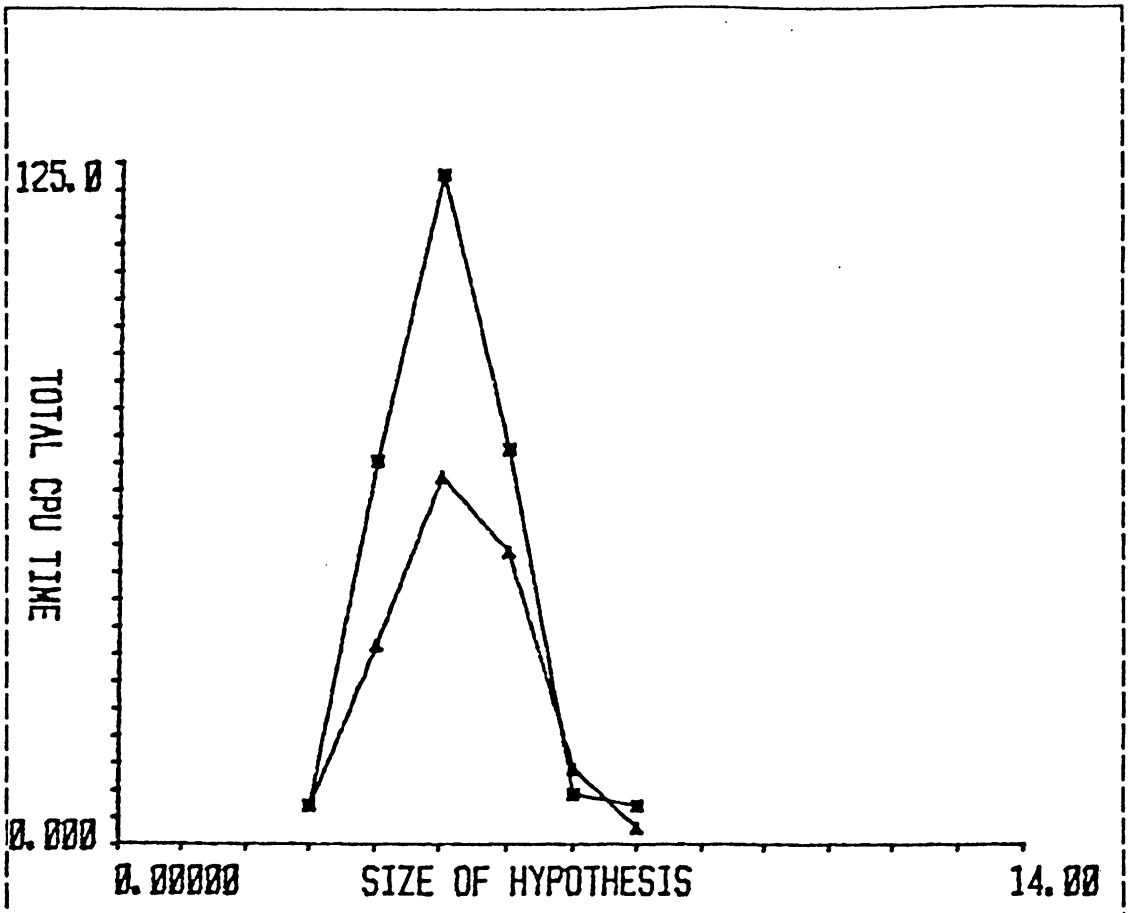


Figure 23: Total CPU time vs. size of hypothesis
The total cpu time in seconds spent in verifying the consistency or inconsistency of a hypothesis for the case with forward checking (□) and without (△). Chunking disabled.

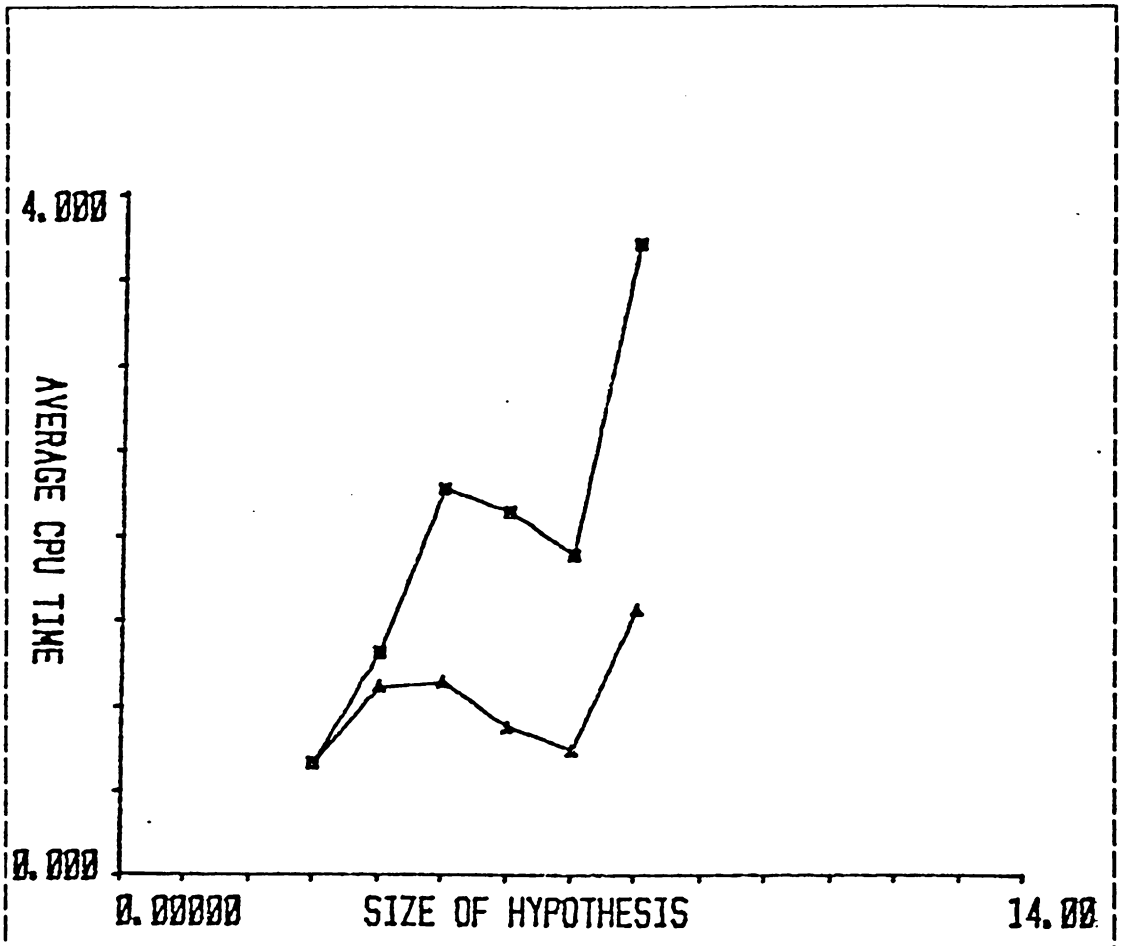


Figure 24: Average CPU time vs. size of hypothesis
Average cpu time (seconds) required for verifying the consistency or inconsistency of a hypothesis with forward checking (□) and without (△). Chunking disabled.

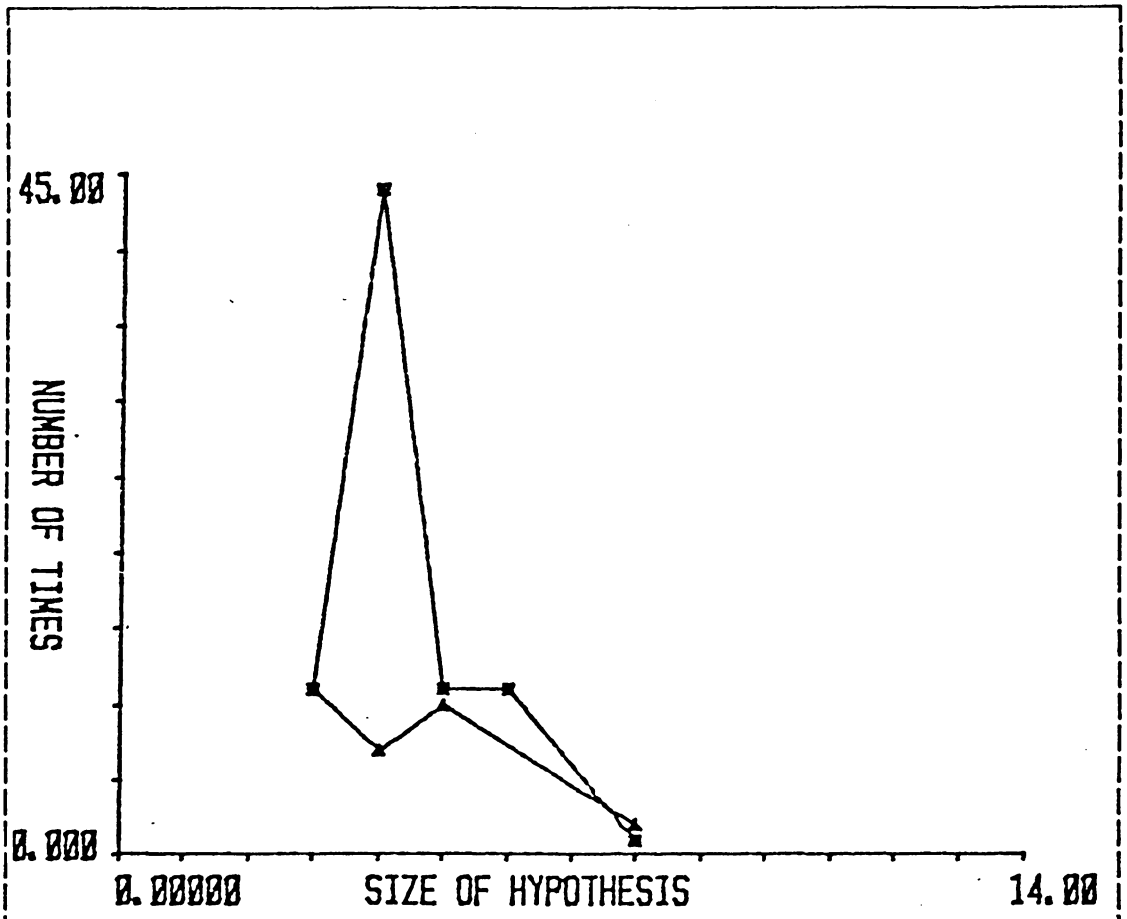


Figure 25: Number of calls vs. size of hypothesis
Number of hypotheses analyzed
as a function of hypothesis size
with (□) and without (△) forward checking.
Chunking enabled.

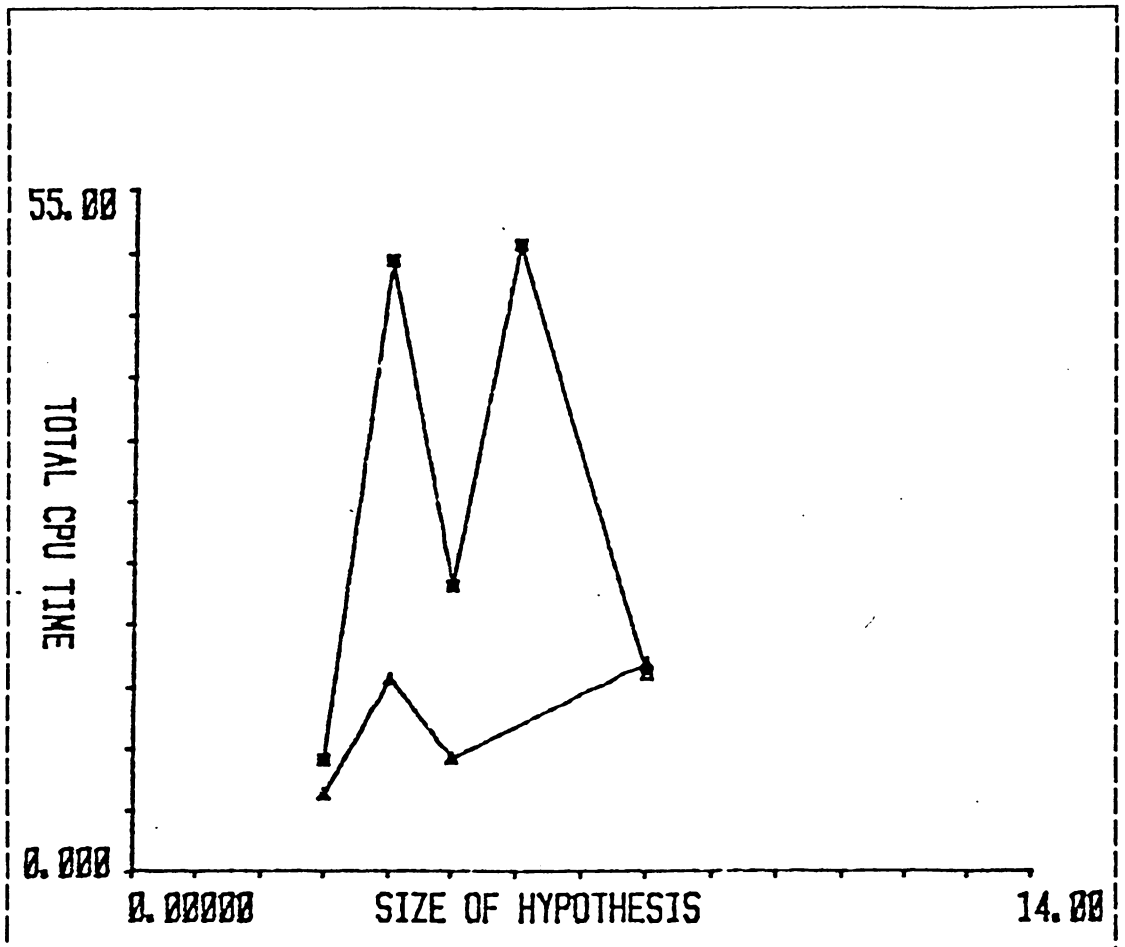


Figure 26: Total CPU time vs. size of hypothesis
Total cpu time in seconds, spent in verifying
the consistency of hypotheses with (□)
and without (△) forward checking
with chunking enabled.

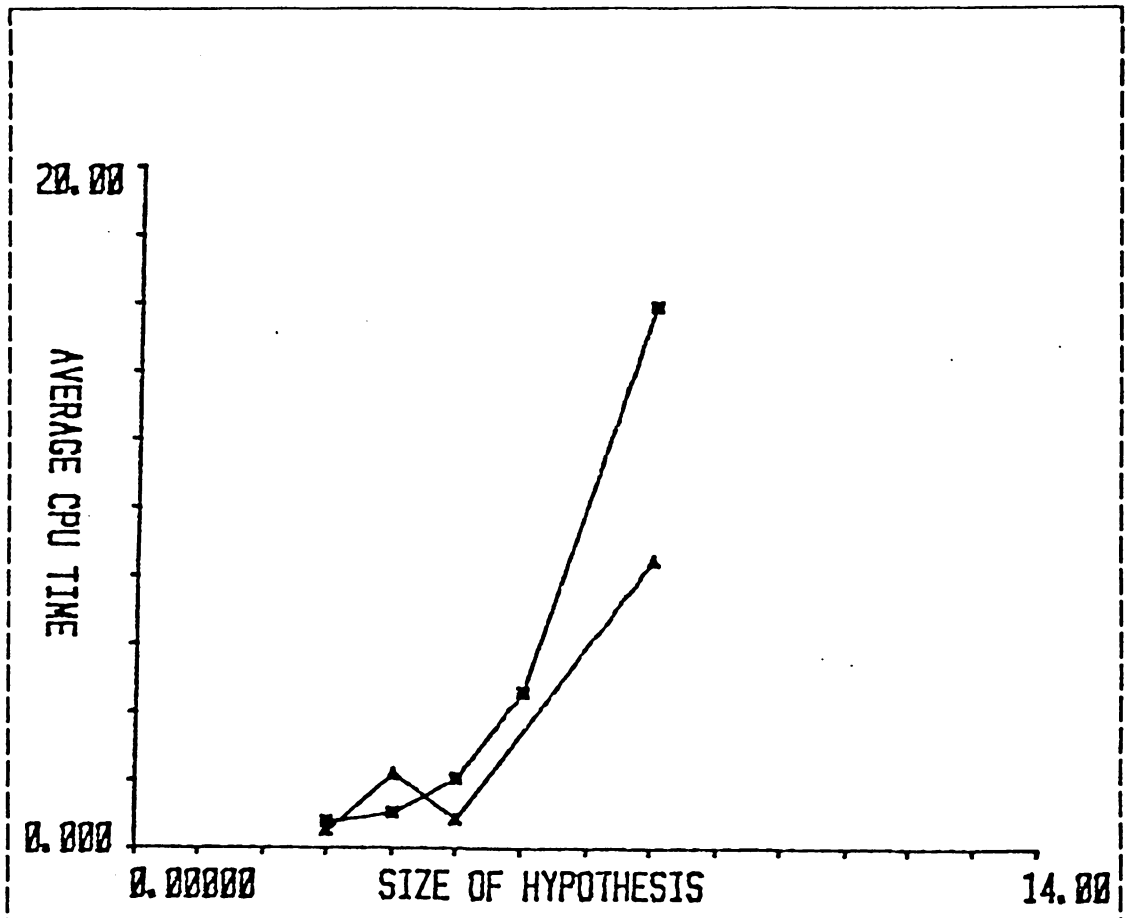


Figure 27: Average CPU time vs. size of hypothesis
Average cpu time (seconds) needed to analyze a given hypothesis as a function of hypothesis size with chunking enabled. With forward checking (□) and without (△).

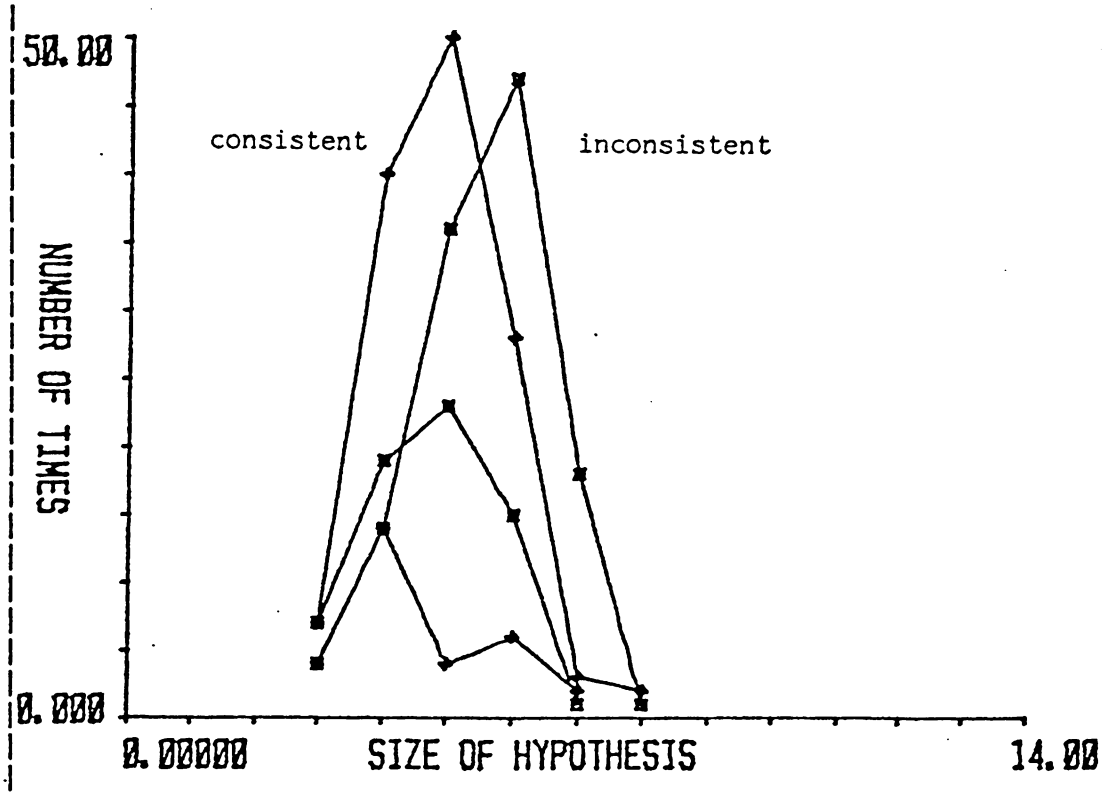


Figure 28: Number of hypotheses by type
Number of consistent and inconsistent hypotheses examined as a function of hypothesis size, with chunking disabled.
With forward checking (◇), and without (□).

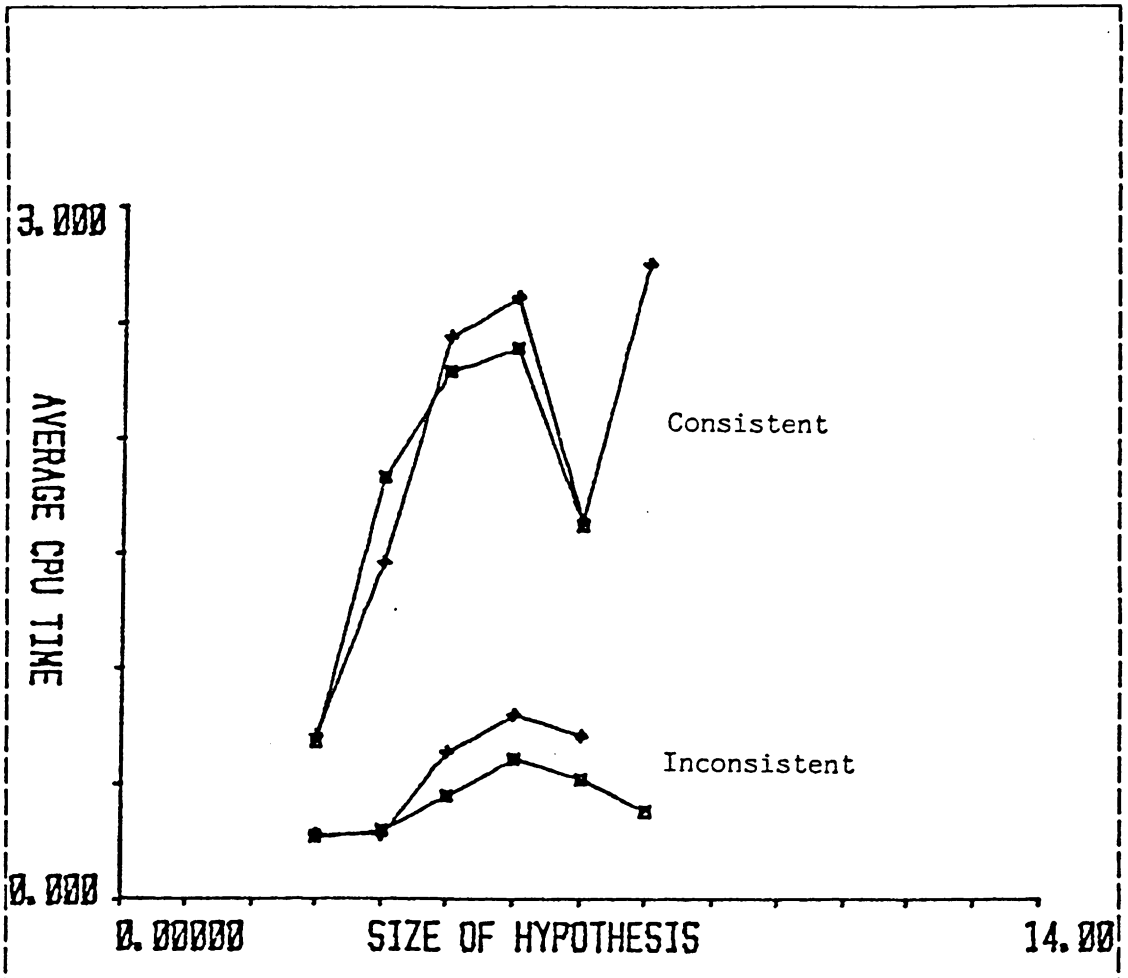


Figure 29: Average CPU time by type
Average cpu time in seconds required for consistent and inconsistent hypotheses vs. the size of the hypothesis.
With forward checking (◇) and without (□), with chunking disabled.

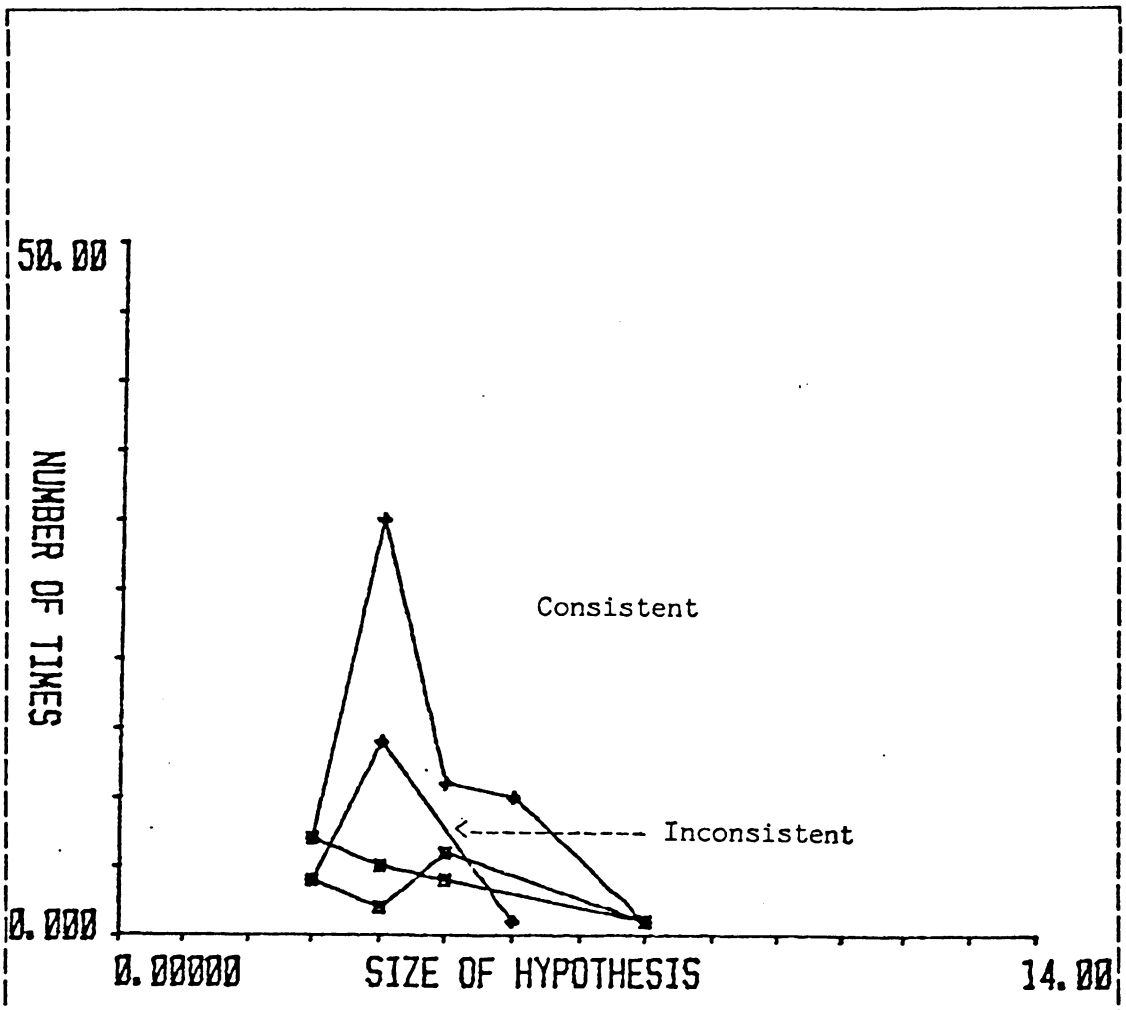


Figure 30: Number of hypotheses by type
Number of consistent and inconsistent hypotheses examined vs. the size of the hypothesis. With chunking in effect, and with (◇) and without (□) forward checking.

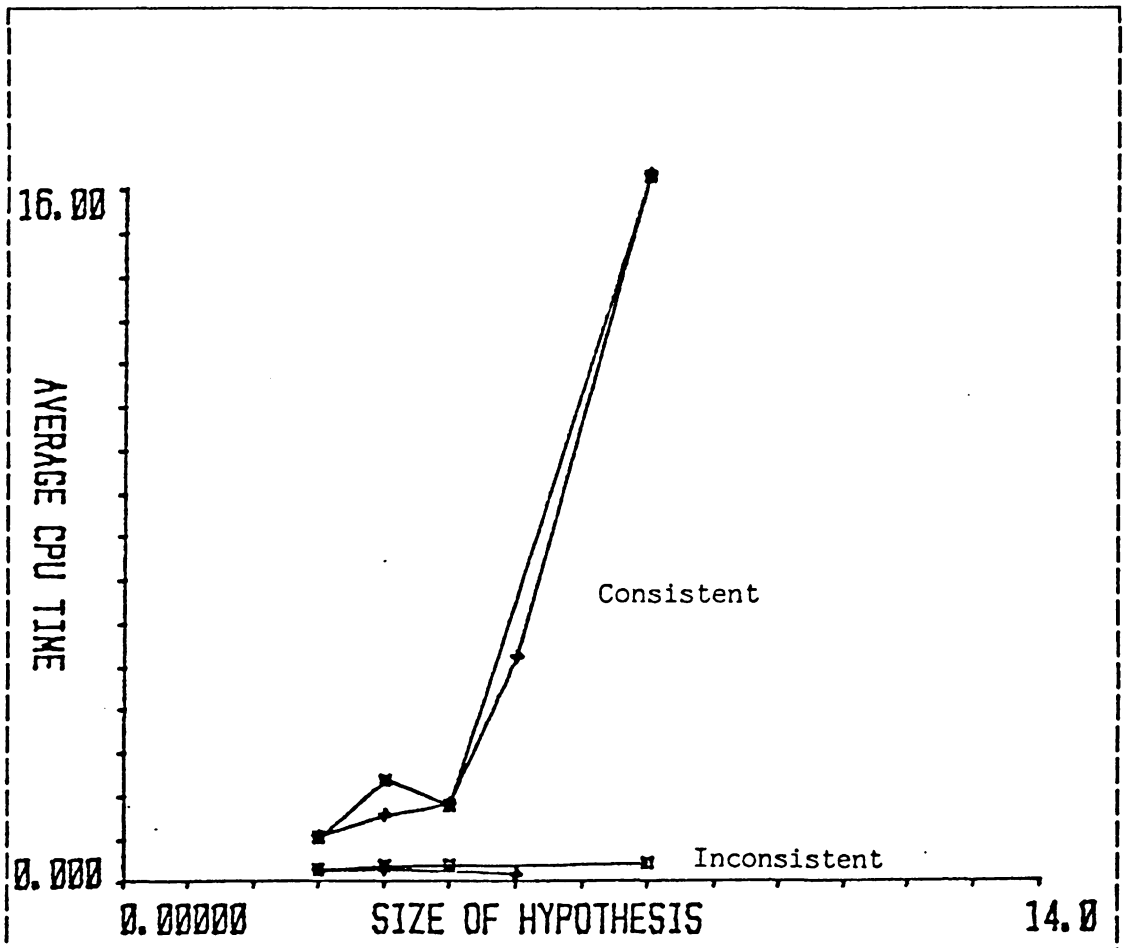


Figure 31: Average CPU time per hypothesis by type
Average cpu time in seconds, required to test the consistency and the inconsistency of hypotheses as a function of size. With chunking in effect and with (◇) and without (□) forward checking.

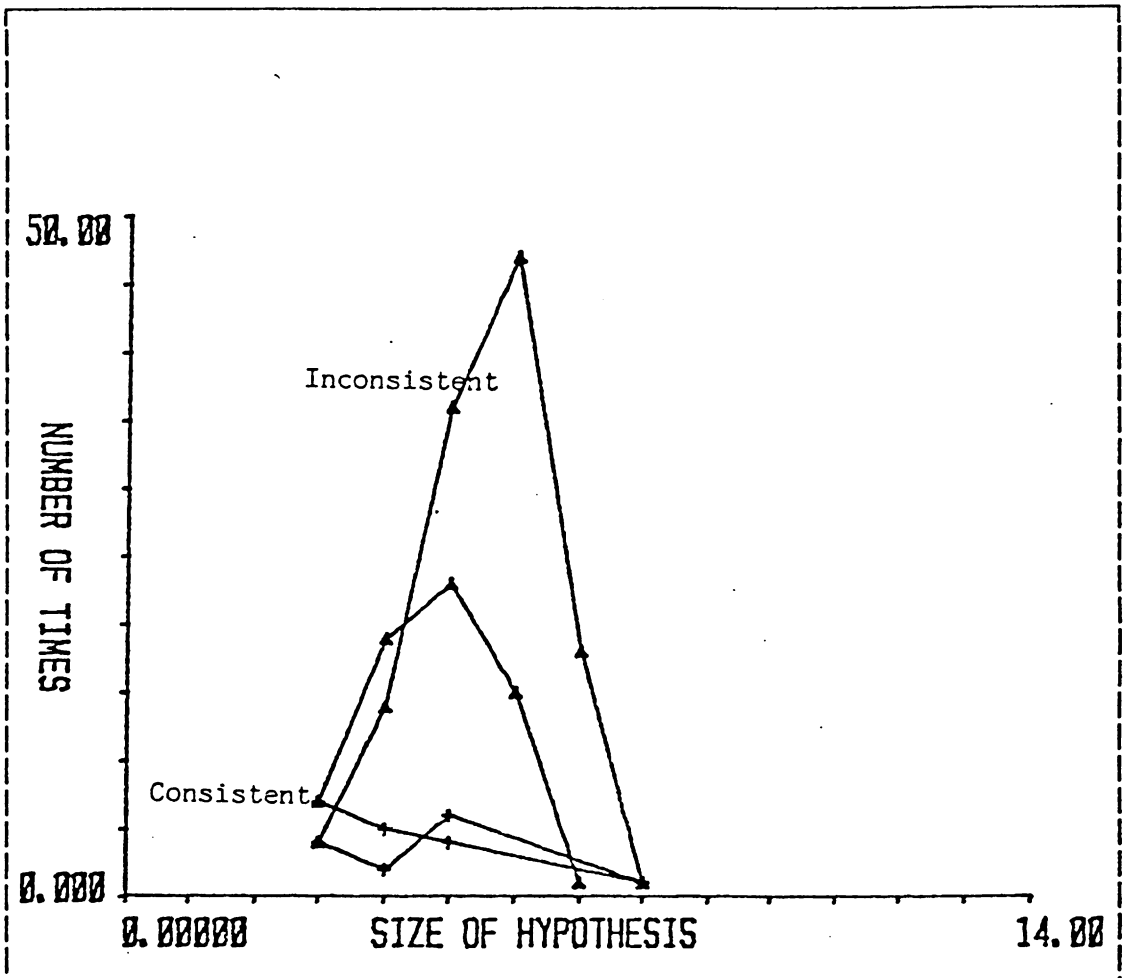


Figure 32: Effect of chunking
Number of consistent hypotheses examined without forward
checking, with (△) and without (+) chunking.
Forward checking turned off.

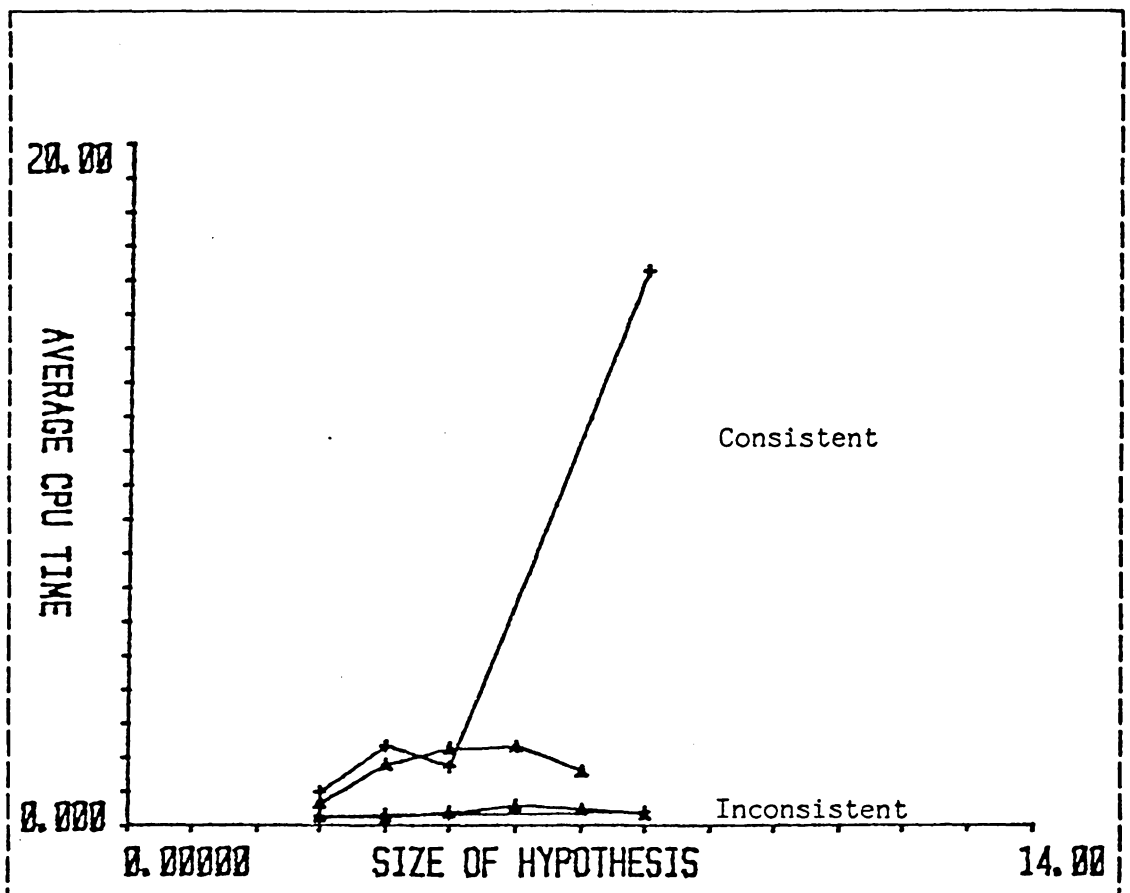


Figure 33: Effect of chunking on average CPU time
Average CPU time in seconds per hypothesis, by size of the hypothesis with (Δ) and without (+) chunking.

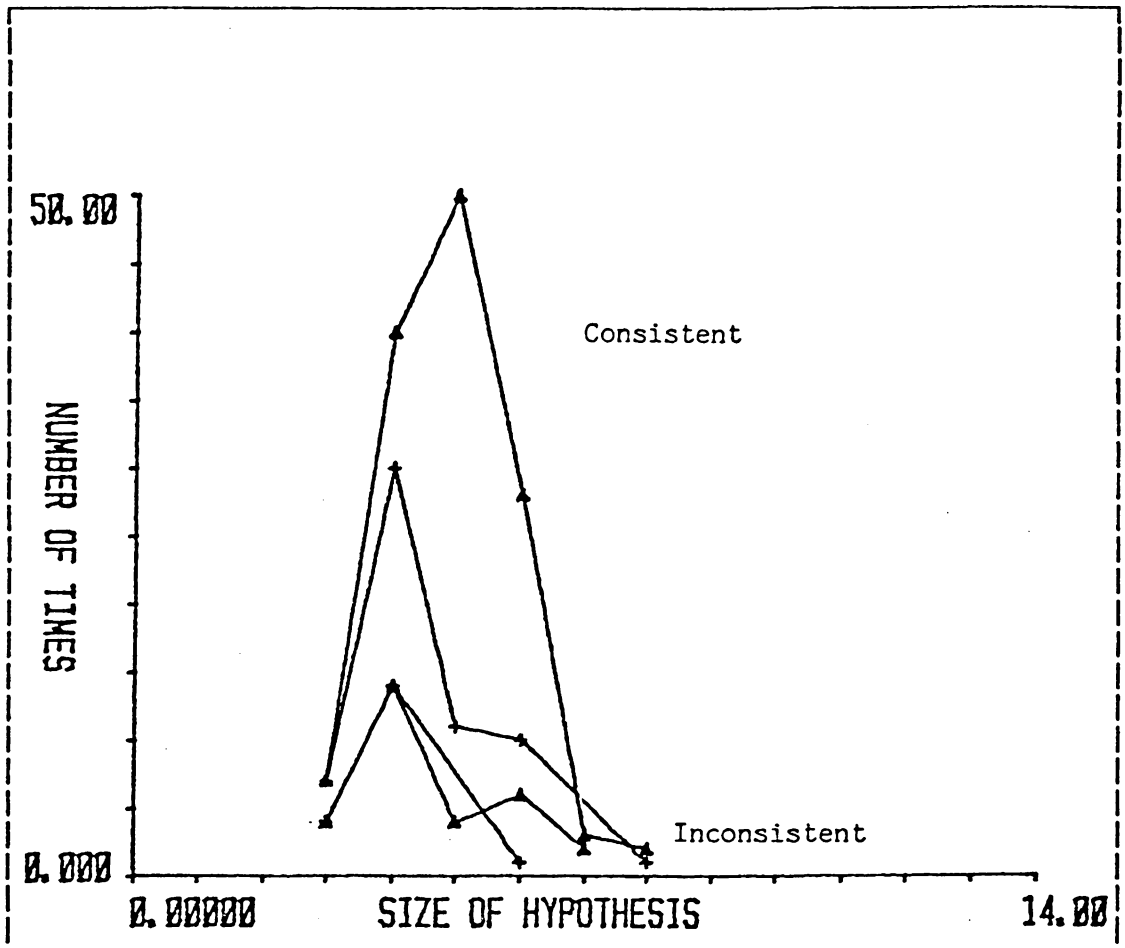


Figure 34: Effect of forward checking
Number of hypotheses examined with (Δ) and without (+) chunking
with forward checking in effect.

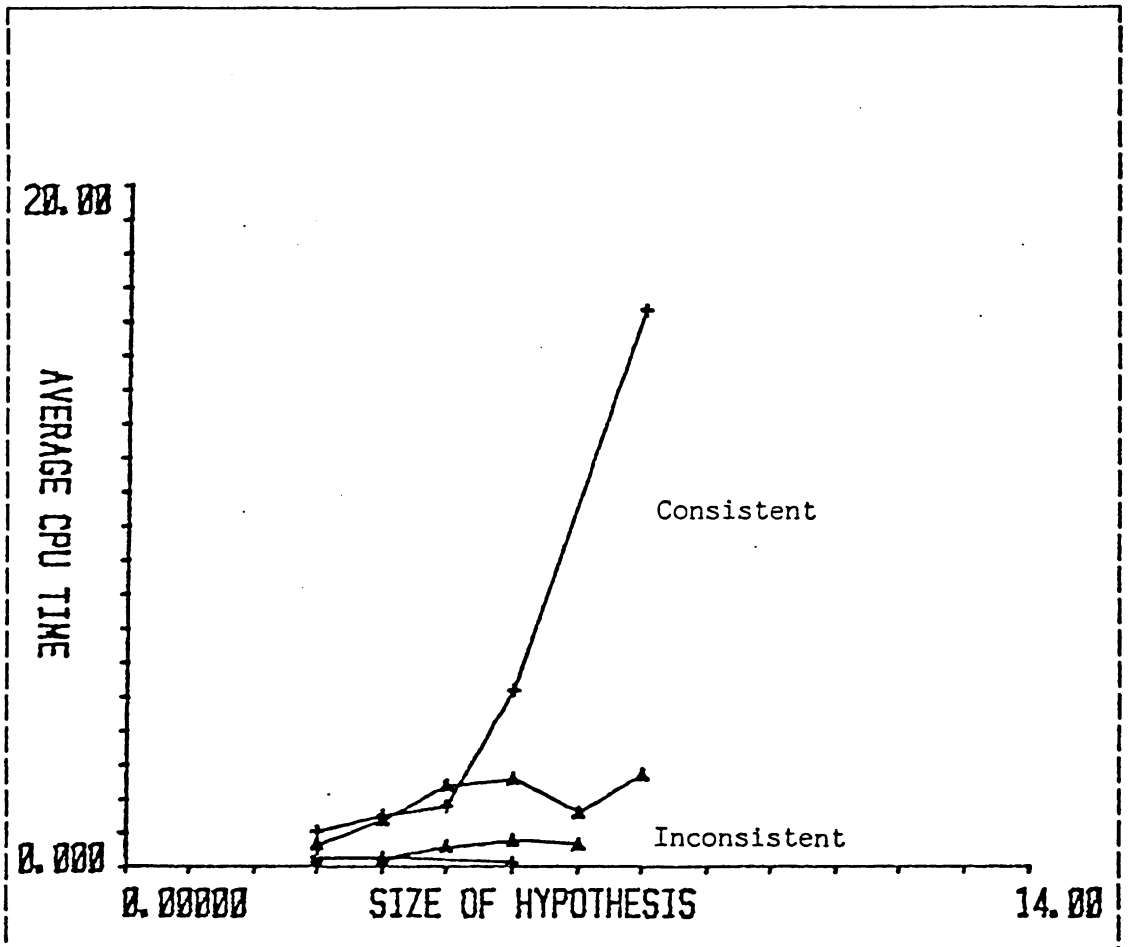


Figure 35: Average CPU time with forward checking
Average cpu time in seconds required as a function of hypothesis
size with forward checking in effect. With (Δ) and without (+) chunking.

countered more than once. If we are only interested in the first solution, this effort is wasted. The other use of forward checking is to restrict the leftmost solution that can be reached in a subtree. If forward checking is not used, then the subset checking algorithm does not have much to go by. The other conclusion is that it is a worthwhile goal to try and keep the size of the hypotheses which must be examined, as low as possible. By incrementally building up the solution, we save on computation times. Further, since the consistent hypotheses take longer to verify (inference engines must be run until no further applications are possible), it takes longer to prove hypotheses consistent rather than finding inconsistencies.

From the analysis of the behavior of the system, we conclude that the process of reasoning about the contents of an image, can be efficiently handled by a hypothesis based reasoning system. Reasoning in this form, allows us to control the possible hypotheses that are generated, and to efficiently prune the tree which has to be searched for solutions. By framing the knowledge of perspective geometry as closed form solutions, we avoid the complexity of the theorem proving approaches, and yet encode knowledge which can be used to effectively generate the inverse of the projection process. The network organization of the inference en-

gines proved to be effective in controlling the supervisory overhead, and yet allowed flexibility in adding new engines. The system itself was constructed by incrementally adding more knowledge, a few engines at a time. We found that this was a fairly easy task, considering the interrelationships between the various modules. Overall, we conclude that the system works well, and that we have gathered a lot of insight into the reasoning process involving the generation and testing of hypotheses. The limiting factor seems to be the time spent internally by PROLOG in searching for rules to apply. If PROLOG is regarded as a medium for testing theories, and when the theories are translated into practical systems, PROLOG is replaced by procedural languages, the experiments were a success.

Chapter VI

LITERATURE REVIEW

In this chapter we will review some of the recent literature in the fields of computer vision, knowledge based systems and other topics relevant to the proposed research. The aim of the chapter is to provide a framework in which our proposal may be examined and results with which our findings can be compared.

The reviews are divided into three sections. The first section deals with vision systems, the second with knowledge systems and the third examines other miscellaneous related topics.

6.1 VISION SYSTEMS

6.1.1 Brooks ACRONYM system

The ACRONYM system is a fairly comprehensive system for the recognition of three dimensional objects from single perspective views using a model based paradigm. In this section we present a brief overview of its salient features in order that we may compare and contrast our strategy with the one embodied in it.

ACRONYM is based on a central thesis that three dimensional objects have properties which stay invariant under

projection for a class of projection parameters. Based on the invariants which can be observed in the image, it is possible to propagate symbolic constraints on free parameters of the imaging process to arrive (if possible) at a consistent interpretation for the entire image. The modelling scheme used was the generalized cylinder approach. The cross sections of the cylinders were chosen from a set of prespecified polygons and the axis functions were restricted to being straight lines or circular arcs. The sweeping rule permitted only bilinear changes to the cross section.

The object parameters and the constraints on their interconnection were specified using symbolic expressions which could involve inequalities. The camera position and the imaging geometry was also specified in similar terms. The constraint satisfaction algorithm was essentially a theorem prover which could handle inequalities and produce estimates of maxima and minima of symbolic inequalities. Although the major part of the system involved trigonometric expressions, the constraint satisfaction system could not handle the nonlinearities introduced by the SIN and COS terms and estimated rougher bounds based on the fact that these functions are restricted to lie between +1 to -1.

The recognition scheme proceeded as follows. From the object description, a graph called the prediction graph was

generated which identified the invariants likely to be observed in the image. The image level routines then used the prediction graph to find corresponding features in the scene. . Based on the ones that it could find, it generated a restriction graph which restricted the free variables in the prediction graph. These new variables were then used to form another prediction graph and the previous step iterated. The process terminated with a restriction graph which contained variables quantified to the most restricted values possible and yet satisfying the constraints imposed by the models. From this it was possible to infer the camera location and geometry.

In contrast with ACRONYM, our system does not use specific object models. We feel that the introduction of the knowledge about perspective projection and the knowledge about specific object models should be at separate levels of the computer vision hierarchy. In addition, ACRONYM's mode of reasoning was essentially theorem proving over the domain of inequalities. Our technique captures the knowledge of the domain in inference engines which have the required mathematical expertise encoded in closed form expressions and reasons over a space equivalent to that of the predicate calculus and is therefore much more efficient.

6.1.2 Generalized Blob technique

Around the same time, we developed the generalized blob technique for the recognition of three dimensional objects from single views. The central idea in this work was the fact that the camera location places a global constraint on the appearance of the entire object. Since the entire object is imaged from a single camera position, any identification of a part in the model with a region on the image constrains the possible appearance of other parts of the object. This meant that the object recognition phase can proceed on a part by part basis keeping the camera location implied by the previously instantiated matches consistent with the camera position suggested by the current match.

As in the ACRONYM work, full generality of the perspective geometry was not used. Perspective normal projection was utilized to make the analytic portion of the matching more tractable. The description scheme was symbolic, decomposing the objects into three categories - sticks, plates, and blobs. The question being answered by this technique was "How little information about the object is necessary to perform object recognition?" As databases of three dimensional objects grow in size and as the environment that a vision system becomes more and more complex, this question will tend to gain importance. Because the more the number of

objects that compete as possible interpretations, the longer it will take to identify the right one. Thus schemes which use the bare minimum of information and yet make correct decisions will gain importance.

Matching proceeded on a part by part basis. A relational error measure was defined between a model and an image. The algorithm looked for the best match - one which minimized the error. The entire scheme was viewed as a consistent labeling problem. Parts in the model were the units and the labels were the regions in the image. Consistency of the implied camera location was required for all matches.

Occlusion was allowed for by allowing units to map to a special label <NULL>. However the net relational error increased because the missing part did not participate in any of the required relations. The overall scheme also produced an estimate for the camera location based on the consistency requirement and identified the object in the database which the given view most closely resembled (based on the relational error).

This was our first effort at utilizing the concept of decomposition of the world in terms of primitive parts and utilizing the coherence properties of the spatial relationships. As in this work, the attribute values (the O, O) of the camera position were computed in closed form. In the

current work, the knowledge of perspective geometry is considerably enhanced. We use full perspective instead of the approximation used in the previous work.

6.1.3 The Hanson and Riseman VISIONS system

The VISIONS system at the University of Massachusetts, (Han83) is a comprehensive system built for the recognition of outdoor scenes. It is based on the integration of high level reasoning into the decision making process, to reason about the interpretation of full color imagery. It is patterned after the HEARSAY-II speech understanding system and is based on the concept of schemas which control the analysis of hypotheses stored on a globally accessible area called the blackboard.

In the VISIONS terminology, the system is provided with a predefined description of the expectations about the world. This definition consists of a hierarchic description which is stored in the Long Term Memory (LTM) of the system. The matching process attempts to construct a description of the scene which is called a "model". The model is constructed in the part of the blackboard called the Short Term Memory (STM). The low level image input to the matching is a preprocessed segmentation of the image in terms of regions, segments and vertices called the RSV representation. This is

produced by a hierarchic processing cone structure which is set up to simulate a parallel processing architecture. The model builder is controlled by a set of independent procedures called Knowledge Sources (KS) which guide the processing. Most of the recognition phase deals with volumes and surfaces and is top down. Appropriate schemas are selected and the knowledge sources produce two-dimensional projections of the expectations which are then compared with the input from the image.

The inexact reasoning scheme used in the VISIONS high level analysis is the Dependency Graph Model of Evidential Reasoning (DGMES) (Low82, Wes82). It is based on expressing the possible environmental events and features as hypotheses in an inference network and combining the evidence using Dempster's rule of combination (Dem67, Dem68). It relies on the fact that it is possible to construct a suitable inference network in which nodes are connected by nodes which denote logical implication, exclusive ors and and conditions over observable events. The starting evidence measures for hypotheses are derived from a feature space analysis of the entities observable in the image. The confidence assigned to an interpretation is directly related to the distance in feature space, of the measurements of an image entity and the predicted value of the measurements if the hypothesis were to be true.

In contrast with our research, VISIONS does not incorporate a large body of perspective knowledge. VISIONS' knowledge is restricted to the concept of spatial arrangements in the image space, and there is no systematic taxonomy of world level entities. The similarity is in the control of the global data sharing over the blackboard data structures, along with the use of independent knowledge sources. Although they too use the concept of a network, the networks are reasoning paths and not control structures. We feel that the ACRONYM system introduces the knowledge of perspective at too high a level, and the VISIONS system attempts to introduce the knowledge at too low a level in the hierarchy of processes.

6.1.4 The Levine image segmentations system

The Levine and Nazif image segmentation system (Lev83) is a very recent effort for the use of a rule based system in the image analysis domain. The technique attempts to integrate two separate information sources about image segmentation into a complete low level segmentation system. The input to the process is an image segmented using a region grower and an edge or line image produced by an edge detector. The rules in the production system try to produce an output segmentation which integrates the information con-

tained in these two representations using heuristic knowledge. The main thrust of the system is to examine ways of ordering the applications of the rules so as to optimize system performance.

The input image is divided into several areas each of which consists of several regions from the initial segmentation and lines from the edge image. For each area, a performance vector is computed which quantifies the disparity between the two representations. Based on the performance vector, rules are invoked which tend to diminish the disparity. This is possibly the first image processing application which works on integrating the knowledge from contradictory sources of information about an image. The system is totally data-driven and does not have any high level knowledge driving it.

The similarity between the Levine segmentation scheme and our system is that both systems use knowledge and rule based control strategies, the segmentation scheme uses fuzzy reasoning for generating the solutions. Our system is inclined more towards the problem of understanding the search space and effectively controlling the search patterns while looking for solutions rather than the problem of developing effective fuzzy evaluation functions and methods for combining inferences.

6.2 KNOWLEDGE BASED SYSTEMS

Starting with the mid 1960's, knowledge based systems have received a great deal of attention as techniques dealing with inexact reasoning and heuristic methods were developed. In this section we review a few of the well known expert systems to examine their techniques for working with inexact information.

Expert systems can be broken up into two major categories: non-consultation systems and consultation systems. Predominant in the first group are the scientific systems such as MACSYMA, HEARSAY, and DENDRAL. The second group consists mainly of the medical diagnosis systems such as MYCIN, PIP and PUFF. Each system shows a different technique for encoding heuristic knowledge and the way in which this knowledge is manipulated.

6.2.1 MACSYMA

MACSYMA is one of the largest symbolic manipulation packages currently in use. It started in the late 1960s with the work of Engleman, Martin and Moses and has undergone about 45 man years of development since then. The domain of MACSYMA's expertise is that of symbolic manipulation of mathematical expressions. One of the major design criteria behind the system was the encoding of large amounts of procedural

knowledge on various mathematical problems into the available database. The control strategy is where heuristics are used. Determination of the next simplification procedure to apply and controlling the way in which results of intermediate simplifications are combined is done heuristically according to a ranking technique. One of the main reasons for the success of MACSYMA is the availability of powerful techniques such as the Risch algorithm for integration (Ris69) which is nearly impossible for human users to apply without the aid of the computer.

In brief then, the main contribution of MACSYMA was to show that it was possible to produce a system in which large bodies of procedural knowledge could be encoded and controlled in a partially heuristic manner to reduce the space over which the system had to search for a solution.

6.2.2 DENDRAL

The DENDRAL system (Led64) was developed to allow exhaustive approach to structure elucidation. Given a set of atoms in a molecule, it could elucidate all possible acyclic molecular combinations that could result. This was augmented by heuristic knowledge from the field of mass spectography to determine the exact structure of chemical compounds.

The basic technique used in the DENDRAL system was a plan, generate and test mechanism. The plan phase was essentially a filtering operation where some inconsistent submolecules were ruled out. The generate mechanism exhaustively generated all possible remaining structures which the test process compared with the obtained mass spectograph results. The main contribution of the DENDRAL project was to demonstrate that heuristic control over a known procedure for total enumeration would work as efficiently as most human experts.

DENDRAL was the first major system which showed used a generate and test paradigm similar to the hypothesis based reasoning scheme developed in our work. The main difference between CONGEN's constraint propagation coupled with exhaustive generation and testing was that there was no effort made to understand the search space and look for domain independent techniques for controlling the search. This, we feel, is the major difference between our technique and the research embodied in DENDRAL.

6.2.3 MYCIN

MYCIN and the other medical consultants built up on the knowledge systems research of the sixties. The problem domain of MYCIN was the field of diagnosis and therapy recom-

mendations for infectious diseases (Sho76). The main contribution of this system was the first in depth analysis and application of heuristic reasoning. MYCIN used confidence factors (numerical quantities in the range -1 to 1) to rate the available clinical evidence and techniques for combining this inexact knowledge for hypothesis formulation. The organization of the knowledge was in the form of production rules and heuristic techniques for combining the confidence factors. Conclusions were reached by a simple depth first evaluation of the and/or goal tree of diseases. The main question raised by the MYCIN system was whether the technique of heuristically determined certainty factors is appropriate for all knowledge areas.

The backward chaining control structure and the confidence factor method for ranking the hypotheses was extracted from the MYCIN system and became EMYCIN, a general control structure for the encoding of expertise in any domain. The control strategy, augmented with a frame based technique for grouping rules together was used in the implementation of CENTAUR (Aki80).

6.2.4 PIP

PIP was one of the first expert systems to use the concept of frames (Pau76). The knowledge was organized into a set of frames. Each frame dealt with a particular type of illness an was related to other frames through values in different frame slots. Frames could trigger other frames, be declared complementary to certain frames or be declared direct consequents of other frames. Each frame had a set of slots which could be filled in with a subset of possible symptoms. These served as local decision criteria which would directly confirm or deny the applicability of a frame. If there was insufficient evidence to confirm a frame based on local criteria, heuristic rules were provided to measure a strength value of a frame and combine it with causally related frames. This provided the system a local as well as a global decision strategy. One of the slots in the frame was a strength slot which encoded the applicability of the frame and its findings.

6.2.5 HEARSAY-II

HEARSAY-I (Red76) and HEARSAY-II (Erm80) were the first systems to incorporate the use of a global database of hypotheses which were accessible to all cooperating processes in the system. This control structure, called a blackboard,

has since been used in several different knowledge representation systems such as CRYSALIS (Eng77), for X-ray crystallography, SAFE (Bal76) and the VISIONS system described earlier. Several processes would examine the contents of the blackboard and depending on the contents, would fire. Some of these processes could add hypotheses for a top-down analysis and yet others would be data driven based on the observations posted on the blackboard. The expertise area for this system was that of speech recognition. The main contribution of this system was the development of a robust control strategy more than the actual application's success.

Further information on these and other expert systems can be found in the Handbook of Artificial Intelligence Volume II (Bar82).

6.3 OTHER RELEVANT RESEARCH

Other recent works have addressed the problem of utilizing knowledge about the perspective process and introducing non-model based knowledge of the world into the reasoning process. The first and most important work we wish to cite is the paper by Haralick (Har80) which laid down the groundwork for the work in this dissertation. Haralick collected in one place the equations of perspective geometry and showed how some of them could be used for computing unknown

parameters of the world. The second paper on the analysis of the perspective projection of rectangles (Har81), built up on this start. Chu (Chu83) extended the rectangle results to conic sections and regular polygons. The use of vanishing points in perspective images by Barnard (Bar82), dealt with the problem of detecting vanishing points in greytone images, and presented several results that could be obtained using a Hough space analysis for back projecting angles. Recently, Barnard (Bar84) showed how right angled corners in images can be used to extract a "perceptual basis" for the visual scene. This work is similar in philosophy to ours in as much as that it too deals with a hypothetical situation with explicit computation of the attributes of the basis. However, Barnard does not extend the ideas to the problem of analyzing the images using similar techniques.

In the 3D MOSAIC system (Her84), Herman and Kanade report the use of world information in a system which hypothesizes the plausible three-dimensional structure of the world based on multiple views. However, their world information is syntactic, and consists of knowledge of how edge junction labels change with corresponding changes in viewpoint. Some knowledge of projection geometry is also encoded in the system to extract directions of movement between successive images and deals with the way lines relate to the the vertical

vanishing points of the images. Their analysis however, does not form hypotheses based on the geometry of the projection alone, and assume that both the vertical vanishing point and the focal length is known a-priori.

Another technique for using model independent knowledge about the projection process is the technique of skewed symmetry analysis for determining surface orientation, when the object being viewed is known to have an axis of symmetry. Several researchers have worked on this problem (Kan81, Ken80, Fri84) and a large body of results exists to substantiate the usefulness of the technique. The main difference between the skewed symmetry research and the research presented in this dissertation is that skewed symmetry analysis does not work for perspective projection. Under perspective projection, orthogonal symmetry of original figures does not become skewed symmetry. For example, the perspective projection of a rectangle is a general quadrilateral. However, under orthographic projection, the image is a parallelogram from which the axes of skewed symmetry may be extracted.

The use of circular features which project to ellipses,has been used in the past (Mul81,Wit81) to extract surface normal information. In (Mul81), the projection was assumed to be perspective-normal, and in (Wit81) it was assumed to be pure orthographic projection. Contrast this

with the use of the full perspective projection in this work.

Chapter VII

CONCLUSION

We have presented a new way of looking at the problem of analyzing perspective line drawings. We showed that we can consider the world to consist of entities related by attributed spatial relations, and the process of interpretation can be considered as a hypothesis generation and testing. We showed that by not considering the process as theorem proving over the space of real numbers, we achieve considerable savings in computational efficiency.

We showed that the process of perspective projection is such that based on hypothesized spatial relationships in the world, we can compute values for several attributes of the world in closed form. The same process that computes values for the attributes can be used to define consistency of the hypotheses.

We constructed a system for the reasoning task based on interpreted PROLOG augmented with procedural sections and associative memories.

In summary, we can say that we have contributed a technique that should prove useful in the middle levels of computer vision systems. By progressively adding world knowledge in the successive stages of image analysis, we may ultimate-

ly reach the goal of being able understand unconstrained scenes with the same aptitude as that displayed by the human visual system.

BIBLIOGRAPHY

(Aik80) Aikins J., "Prototypes and Production Rules: A Knowledge Representation for Computer Consultations", PhD. Dissertation, Department of Computer Science, Stanford 1980.

(Bak57) Baker S.F., "Induction and Hypothesis- A Study of the Logic of Confirmation", Cornell University Press, Ithaca, N.Y. 1957.

(Bar82) Barnard S. T., "Interpreting Perspective Images", Technical Note 271, AI Center, Computer Science and Technology Division, SRI International.

(BaF82) Barr A. and Figenbaum E.A., "The Handbook of Artificial Intelligence Volume I and II", published by William Kaufmann, Inc. 1982.

(Bal76) Balzer R.M., Goldman N., and Wile D. "On the Transformational Implementation Approach to Programming", Second ICSE, 1976.

(Ber82) Berman S, Parag P., Lee C.S.G., "Computer Recognition of Overlapping Parts Using a Single Camera" IEEE Proc PRIP, 1982.

(Bin71) Binford T.O., "Visual Perception by Computer", IEEE Conf. on SMC, 1971.

(Bol81) Bolles, R. and Cain; SRI symmetry analysis.

(Bro81) Brooks R.A., "Symbolic Reasoning among Three Dimensional Models and Two Dimensional Images", AI Special Volume on Computer Vision, AI 17, 1981.

(Cha79) Chang and Lee ***** GET COMPLETE REFRENCE>

(Chu82) Chu Y. H., "Recognition of Wire Frame Objects from Single Perspective Views", PhD. Dissertation, VPI&SU, 1982.

(Dat81) Date C.J., "An Introduction to Database Systems", Third Edition, Addison-Wesley Publishing Company, 1981.

(Dem67) Dempster A.P., "Upper and Lower Probabilities Induced by a Multivalued Mapping", Annals of Mathematical Statistics, Vol 38, 1967.

(Dem68) Dempster A.P., "A Generalization of Baysean Inference", J.R.Stat. Soc. Series B, No. 30. 1968.

(Duh54) Duhem P.M.M., "La Theorie Physique: Son Objet, Sa Structure", Translated to "The Aim and Structure of Physical Theory", by Wiener P.P., Princeton University Press 1954.

(Eng77) Engelmore R.S., and Nii H.P., "A Knowledge-Based System for the Interpretation of Protein X-ray crystallographic Data", Heuristic Programming Project Rep. No. HPP-77-2, Computer Science Dept., Stanford University.

(Erm80) Erman L.D., Hayes-Roth F., Lesser V.R., and Reddy D.R., "The Hearsay-II Speech Understanding System: Integrating Knowledge to resolve uncertainty", Computing Surveys, 12(2), 1980.

(Feu82) Feustel C.D. and Shapiro L.G., "The Nearest Neighbor Problem in an Abstract Space", Pattern Recognition Letters, V1, December 1982.

(Fri84) Friedberg, S.A., Brown C.M., "Finding Axes of Skewed Symmetry", 7th. ICPR, Vol I, 1984.

(Han83) Hanson A.R., and Riseman E.M., "A summary of Image Understanding Research at the University of Massachusetts", COINS technical report 83-35, 1983.

(Har79) Haralick R.M., Elliott G., "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", Proc 6th IJCAI, 1979.

(Har80) Haralick R.M., " Using Perspective Transformations In Scene Analysis", CGIP 13, pp191-221, 1980.

(Har81) Haralick R.M., "Determining Camera Parameters from the Perspective Projection of a Rectangle"

(Hea21) Heath T., "A History of Greek Mathematics", Volume II, Oxford University Press", 1921.

(Her84) Herman M., Kanade T., "The 3D MOSAIC Scene Understanding System: Incremental Reconstruction of 3D

Scenes from Complex Images", Department of Computer Science, Carnegie Mellon University, Technical Report Number CMU-CS-84-102, February 1984.

(Kan81) Kanade T, "Recovery of Three-Dimensional Shape of an Object from a Single View", Artificial Intelligence 17, 1981.

(Ken80) Kender J.R., "Shape from Texture", PhD. Thesis, Computer Science Department, Carnegie Mellon University, 1980.

(Led64) Lederberg J., "DENDRAL-64 A System for Computer Construction, Enumeration and Notation of Organic Molecules as Tree Structures and Cyclic Graphs. Part I, Notaional Algorithm For Tree Structures", Rep. No. CR-57029, NASA.

(Len82) Lenat D.B., Davis R., "Knowledge Based Systems in Artificial Intelligence", McGrawhill International, Advanced Computer Science Series, 1982.

(Lev83) Levine M. and Nazif, "Rule-Based Image Segmentation: A Dynamic Control Strategy Approach", TR-83-8, Computer Vision and Robotics Laboratory, Department of Electrical Engg., McGill University, 1983.

(Low80) Lowrance J.D., "Dependancy Graph Models of Evidential Support", PhD. Dissertation, COINS Dept., University of Massachusetts, Amherst, December 1978.

(Mul81) Mulgaonkar P.G., "Recognizing Three Dimensional Objects from single Perspective Views", M.S. Thesis, VPI&SU, 1981.

(Mul84) Mulgaonkar P.G., Shapiro L.G., Haralick R.M., "Matching Sticks Plates and Blobs Objects using Geometric and Relational Constraints", Image and Vision Computing, March 1984.

(Mul82) Mulgaonkar P.G., "Feasability of Solutions Generated by Consistant Labeling for One Dimensional Occlusion", unpublished.

(Par81) Parma C.C., Hanson A.R., and Riseman E.M., "Experiments in Schema-Driven Interpretation of a Natural Scene", in "Digital Image Processing", ed. J.C.Simon and R.M.Haralick, D.Reidel Publishing Co., Dorducht, Holland, 1981.

(Pau76) Pauker S., Gorry G.A., Kassirer J. and Schwartz W., "Towards the Simulation of Clinical Cognition-- Taking a Present Illness by Computer", Amer. Jou. of Medicine, 60 1976.

(Per82) Perkins W.A., "Model-Based Component Board Inspection", IEEE Proc. PRIP, 1982.

(Red76) Reddy R., Erman L., Fennell R., and Neely R., "The Hearsay Speech Understanding System: An Example of the Recognition Process", IEEE Trans. Computers. C-25, 1976.

(Res64) Rescher N., "Hypothetical Reasoning", Amsterdam North Holland Publishing Company, 1964.

(Rum82) Rummell P. and Beutel W., "A Model-Based Image Analysis System for Workpiece Recognition", IEEE-PRIP 1982.

(Ris69) Risch R., "The Problem of Integration in Finite Terms", Trans. of the AMS, Vol 139, 1969.

(Ros83) Rosenfeld A.R., "Survey, Picture Processing: 1982", CGIP 22, 1983.

(Sha80a) Shapiro L.G. "A Structural Model of Shape", IEEE PAMI, Vol 2, 1980.

(Sha80b) Shapiro L.G., Mulgaonkar P.G., Haralick R.M., Moriarty J.D., "A Generalized Blob Model for Three Dimensional Object Description"., Second IEEE Workshop on Picture Description and Management, 1980.

(Sha82) Shapiro L.G., and Haralick R.M., "Organization of Relational Models for Scene Analysis", IEEE PAMI, 1982.

(Sha83a) Shapiro L.G., "Computer Vision Systems: Past Present and Future", in NATO ASI Series, Vol F4, ed. R.M.Haralick, Springer Verlag, 1983.

(Sha83b) Shapiro L.G., and Haralick R.M., "A Hierarchical Relational Model of the Bulkhead", Preliminary report to Martin Marietta, Department of Computer Science, VPI&SU, 1983.

(Sho76) Shortliffe E.H., "Computer Based Medical Consultations: MYCIN", American Elsevier, 1976.

(Ten76) Tenenbaum, J.M., and Barrow, H.G., "MSYS: A System for Reasoning about Scenes", SRI Technical Note 121, March 1976.

(Tho83) Thorpe C.E., Shafer S., "Topological Correspondance in Line Drawings of Multiple Views of Objects", Techincal Report, CMU, 1983.

(Ull81) Ullmann J., Haralick R.M., Shapiro L.G., "Computer Architecture for Solving Consistent Labelling Problems", Technical Report, Department of Computer Science, VPI&SU.

(Ull82) Ullmann J. "An Investigation into Occlusion in One Dimension", Technical Report, Department of Computer Science, University of Sheffield, 1982.

(Und75) Underwood S.A., "Visual Learning from Multiple Views", IEEE Trans. on Computers, C-24#6, 1975.

(Voe77) Voelcker H.B., Requicha A.A.G., "Geometric Modelling for Mechanical Parts and Processes", COMPUTER, No. 12, 1977.

(Vri68) deVries J.V., "Perspective", Dover Publications, 1968.

(Wes82) Wesley L.P. and Hanson A.R., "The Use of an Evidential-Based Model for Representing Knowledge and Reasoning about Images in the VISIONS System", IEEE-PRIP 1982.

(Wit81) Witkin, A.P., "Recovering Surface Shape and Orientation from Texture", Artificial Intelligence, 17, 1981.

Appendix A

DESCRIPTION OF THE PROLOG AND RATFOR CODE

In this section, we describe the implementation of the reasoning system. The actual source code is available separately and is not included as part of this dissertation. The description is at a fairly high level. We will not go into the actual implementation except in excerpted form to illustrate essential aspects of the code. However, the description that follows is sufficient for understanding the execution trace that is presented in Appendix C.

A.1 THE TOP LEVEL

The very top level of the program performs initializtion of all the global associative tables and global control variables involved. The program has several global variables which are used to control various aspects of the execution. These variables are:

1. %bttsflag. This can be set to "new" or "old". If it is set to "new", the program uses a non-recursive implementation of the back-tracking tree search to generate and test hypotheses. If "old" is specified, a recursive algorithm is employed. The non-recursive algorithm is faster and uses far less memory than the

recursive version. However the recursive version is easier to debug.

2. %mode. If %mode is set to "all", the search looks for all maximal solutions. If it is set to "best", the search terminates when the best solution is found.
3. %forchkflag. If this flag is set to "on", then forward checking is used at all nodes in the tree. It is faster to set this flag to "off" if only the first solution is desired.
4. %extendflag. If %extendflag is "on", then heuristics are applied to generate new tuples to be added to extend the hypothesis.
5. %traceflag. Setting %traceflag "on", produces a printout of the trace of the program execution. One such trace is shown in Appendix C. The default value is "off".

Once all the global variables are initialized, the program then generates all the possible tuples. Note that although the number of tuples may be large, it is still polynomial in the number of lines in the data set. The actual number generated is $2\binom{n}{2}+m$, where n is the number of lines in the image and m is the number of arcs. These tuples are stored in two tables, the first of which maps an integer in-

dex number to the tuple itself and the second maps the tuple to the token "on" or "off". Initially all tuples are "on", that is, they are candidates for addition to the hypothesis. Once a tuple is added to a hypothesis, or if it is determined that a tuple cannot be added to a hypothesis, it is turned "off".

Once these tuples have been generated and stored, the appropriate backtracking tree search is invoked. The selection of the backtracking algorithm is determined by the user selected choice of the flag %bttsflag.

A.2 THE BACKTRACKING TREE SEARCH

Regardless of which algorithm is actually employed, both algorithms perform identical actions when seen in a high-level perspective. If the search is at the bottom of the tree, that is, all the tuples in the tables have either been put into the hypothesis, or have been discarded as invalid, the search prints out the generated interpretation. The interpretation is printed in two forms. First, a readable output is provided of all the points, lines, arcs, and planes in the input data set. This output shows the identifier for the entity, and its two dimensional measurements, its associated three dimensional attributes such as coordinates, direction cosines, normals and vanishing points. Second, a

list is made of all the relational tuples that were used in arriving at the computations which generated the three-dimensional attribute values. Also listed is a summary which describes the computations that were performed upto this stage in the search. We will describe later in this section, the performance criterion that is printed out with each result.

If the search is at an internal node in the tree, several actions are possible. Note that at level l in the tree, the decision that has to be made is whether the tuple l should be added to the current partial hypothesis or not. If tuple l is "off", it means that either it has already been added to the hypothesis by the "chunking" process, or it has been ruled out on the basis of previously accepted tuples. In either case, the current tuple is not a candidate for addition to the hypothesis, and the search process takes the right branch in the tree and decends a level.

If the tuple is "on", then it is a potential candidate for extending the hypothesis. In this case, two checks are performed. First, the "leftmost-solution" check (See Chapter II) is applied. If the leftmost possible solution is a subset of any previously generated solution, there is no use going lower in the tree. In this case the control backs up. Note that for efficiency purposes, the lefmost-solution

checking algorithm is coded so that it does not perform any checking if there are no previously generated solutions or if the rightmost node is not a subset of any previously generated solution.

If the tuple l is still admissible at this stage, the %mode flag is checked. If it is "best", then we desire a gobally maximal solution (the optimal solution). Otherwise all maximal solutions are to be found. To do this, we apply the function "g" (Chapter II), to the leftmost possible node in the subtree. Note that "g" measures the size of the hypothesis. If this size is smaller than any previously generated solution, any solution in the subtree below the current node cannot be the optimal one. Thus if this check fails and the control mode requires us to find the "best" solution, we back up.

Once we have determined if it is worthwhile decending the tree, the program now has to determine, if the current tuple can indeed be added to the hypothesis. The progam maintains a queue of tuples to be hypothesized. This queue is called %fifo. The current tuple is added to the tail of this queue. Note that this queue is emptied at each level, and therefore, the tuple l is at the head of the queue. There are two ways in which a tuple can be added to the queue. It can be added in a "with-extend" mode or in a "no-extend" mode.

If a tuple is added in a "with-extend" mode, a heuristic is employed to "extend" the hypothesis further. The heuristic used is as follows: If the tuple being added is (parallel a b), and lines a and b are known to lie in a common plane, all other lines in the same plane which intersect either a or b at the same point, are also also added to the queue as being parallel to a or b. For example if line c is such that the intersection point of a and c coincides with the intersection point of a and b, and if a, b, and c are co-planar, then the tuple (parallel a c) is added to the queue, provided that (parallel a c) has not been turned off by any previous computation. This is purely a speed-up and does not in any way alter the theory developed in the main body of the dissertation. The tuple (parallel a c) is added in a "no-extend" mode to the queue, because if we were to try to extend (parallel a c), we would simply get (parallel a b) back, and we already have that on the queue.

Once all extensions have been added to the queue, the queue is passed to the chunking algorithm. This part of the code examines each of the tuples on the queue, and applies the chunking rules defined in Chapter II. Let the tuple being examined be l, and let there be a tuple k in the partial hypothesis, such that k and l => m, where m is some other tuple. If m is already in the hypothesis, no action is tak-

en. If m is still "on", i.e. a candidate for addition to the hypothesis, it is added at the end of the queue. If however, m is "off", i.e. has been ruled off because of previous forward checking, the chunking fails. Note that the chunking rules are such that if k and l => m, then l and m => k and k and m => l. Therefore, if the tuple m is added to the queue, we do not apply the chunker to it. Thus the chunking rules are applied only to the tuples that were present in the queue when the chunker was invoked, and is not applied to the new tuples generated as a result of the applications.

If the chunker succeeds, that is if all the logical implications of tuples in the queue and tuples already in the hypothesis are either already part of the hypothesis, or have been added to the queue, the consistency checker is invoked.

The consistency checker first hypothesizes all the tuples in the queue. It then invokes the procedure "lp", to propagate the attributes to see if an inconsistency is present. If an inconsistency is present, "lp" fails.

If either the chunker or lp fails, control of the tree search moves to the right at level l. All the tuples that have been hypothesized at this level are removed, and the system state is restored to what it was when this level was entered. The tuple l and all its implications, (the tuples

generated by the chunking) are marked "off", that is, they are not admissible into the current partial hypothesis. After doing this, control descends one level lower in the tree.

If however, the chunking and consistency checking succeed, the tuple and all its implications are retained. Now if the forward checking has been turned on, all the remaining tuples which are still "on" are examined. Let j be the next tuple which is still "on". Forward checking adds the tuple j to the queue in a "no-extend" mode and invokes the chunking algorithm and the consistency checker on the queue so generated. If it fails, then the tuple j and all its implications are marked "off". The advantages and disadvantages of using the forward checking have been discussed in Chapter V.

This process is repeated until the bottom of the tree is reached. Once the bottom of the tree is reached, control then starts backing up the tree. On backing upto some level l where the search had taken a left branch on the way down, the tuple corresponding to l is turned off, and control then tries the right branch. All the previous steps are repeated.

The process terminates when the search backs up past the top of the tree.

A.3 STATE RECORDING

To enable the tree-search to backtrack, it must keep track of the state of the hypothesis, and the state of the inference engines at each level of the tree. It is un-economical (although conceptually very simple), to record at each level, the exact state of the system. The state consists of knowing which tuples are currently "on" and which ones are "off", the tuples which are in the partial hypothesis, the values for the attributes computed on levels above the current level, indications as to which engines have seen which input tuple sets, and the engines which are currently enabled. The program follows an alternative route. Whenever a change is made to any of these entities, a record is kept of the change. For example, if an engine records the fact that it has seen a particular set of tuples, it also records the fact that this action was taken at level 1. When control backtracks to 1, we can then recover the previous state by examining the changes that were made at this level. Although it is simpler to store the entire state at each level, it uses up a lot of memory because all information generated at level i needs to be stored for all levels below i. The technique we use, requires some work to restore the previous state, because we have to "undo" the actions. However the storage savings justify this extra work.

A.4 APPLICATION OF INFERENCE ENGINES

Inference engines are all coded as different clauses of the same PROLOG rule. Thus the sequencing job of the supervisor (Chapter III) is performed by the PROLOG interpreter itself without having to separately code the function. To see how it works, consider the actions of the interpreter. When a rule is invoked, it scans through the clauses in the sequential order in which they occur, until the first successful clause is applied, or all the clauses have been tried. Each clause is encoded following a standard from. This form is shown below using the notation of the PROLOG interpreter in which the system is implemented.

To interpret the form, we briefly describe the syntax of the interpreter. All clauses are of the form (<precondition> if <postcondition>) where precondition is a single list. <postcondition> is a list of s-expressions. Whenever the precondition is invoked, the s-expressions in the postcondition list are executed. There is an implicit conjunction between the elements of the postcondition list. If any of the postconditions fail, the clause fails and the next clause corresponding to the precondition is tried. Postcondition elements are tried using a backtracking search. For example, if the post condition list consists of two s-expressions (a) and (b), if the invokation of b fails, then

control backtracks to the (a) and tries the next possible clause for a. The builtin rule (cut) acts as a "weir" through which control can move forward but not backwards. Thus if a (cut) is encountered during backtracking, the current clause fails. In addition, even if more clauses remain for the precondition, they are not tried and the postcondition itself is declared unsatisfiable. Variables in PROLOG are indicated as *name. The builtin function <- assigns the second argument to the first. The conditional selection is performed by the (cond ...) statement. The other rules and functions shown in the example below are (notexamined ...) which succeeds if the current inference engine has not previously examined the specified input set; (examined ...) which marks the input set as having been visited by the current engine; (turnengine ...) which turns the specified engine "on" or "off"; and (isengine ...) which queries if the specified engine is "on" or "off". Engines are named in sequence begining which C1. They are all encoded as clauses of the same rule "checker". The last engine in the set of engines is named C* and it sets the value of the argument *s to a special flag which indicates that all previous engines were tried and failed.

```
((checker *s) if

      (turnengine <n-1> off) ; turn of previous engine
                             ; in the sequence
      (isengine   <n>   on)  ; Proceed only if this engine
                             ; has been "activated"
```

```
     (....)                    ; precondition test
     (....)                    ; to find the input set

     (notexamined ....)        ; Proceed only if the
                               ; precondition set determined
                               ; above has not been previously
                               ; operated on.
     (cut)                     ; The next section of code is
                               ; the procedural application of
                               ; the equations of perspective/
                               ; If  there is a failure, then
                               ; the engine terminates.
     (examined ...)            ; Mark the input set as having
                               ; been visited. The next time
                               ; this set is found, the rule
                               ; "notexamined" will fail.
   (.....)                     ; Compute new values for
                               ; attributes.


   (cond (== <old-value> unknown)

                               ; if there was no old value for
                               ; that attribute:

         (<- <value> <new-value>)
         (<- *s succeed)

   elsecond

         (¬  <old-value> <new-value>)
         (<- *s succeed)

                                ; if old value was not
                                ; unknown, the
                                ; value must match the
                                ; new value.
   else

        (<- *s fail)            ; Otherwise the application
                                ; fails.

)
```

Note what happens in the example shown. The first action taken by the engine is to turn off the previous engine in

the sequence. For example C2 turns C1 off. By the definition of the semantics of PROLOG, if the clause corresponding to the engine C2 is invoked, that indicates that the clause corresponding to C1 failed. The reason why engines fail will be clarified later. At this stage it suffices to say that engines fail because they are either disbled or if they are enabled but cannot find a input set to which they can apply. In either case, the engine should be removed from the search sequence. This is precisely what happens when an engine is turned off.

The second action is to check if the current engine is on. If it is not on, no further action is taken. The rule (isengine <name> on) fails, and the PROLOG control strategy declares the current clause failed. It then tries the next clause in sequence. For example if C2 is disabled, (isengine C2 on) fails, and PROLOG tries the next clause corresponding to engine C3. Thus no time is spent is searching for the preconditions of engine C2. This is exactly the requirement set down by the the network organization model (Chapter III).

Suppose an engine, say C2, is active. It then tries to find in the global database, the required input tuples. If no tuples are found, the clause corresponding to C2 fails and the next engine in sequence is tried. If however, a pos-

sible input set is located, the rule (notexamined ...) is invoked. This rule fails if the located input set has previously been examined. PROLOG's backtracking search now comes into effect, and the search for an input set resumes. This process of searching for an input set and checking its history can terminate in one of two ways. First, if all located input sets have been previously visited, the process fails and the clause C2 is declared unsatisfiable. In this case control passes to the next clause C3. Alternately, the search may locate an input set for which (notexamined ...) succeeds. In this case control passes on past the (cut). The input set so located is marked as visited.

The cut plays an important role in this framework. Once the (cut) is passed, the actions taken by the clause consist of applying the closed form inverse projection equation to the located input set. If this application fails, there is something wrong: the hypothesis is inconsistent. In this case, no further engines need to be applied. Thus when control backs into the (cut), the entire engine application is declared a failure.

Once the new values for the appropriate attribute have been computed, they are compared with any previously known values. If the comparison fails, the argument *s is set to "fail". Otherwise the argument is set to "success".

The invokation of the rule (checker *s) is done repeatedly. From the view point of the invoking rule, one of four things happen.

1. (checker *s) fails. In this case the hypothesis is inconsistent.
2. *s is returned as "fail". In this case too, the hypothesis is inconsistent.
3. *s returns "success". When this happens, the particular engine application succeeds. A hypothesis is consistent only when all possible applications are completed, that is, when a successfully terminating sequence is completed. Thus, the loop continues and (checker *s) is invoked again. Note that before any engine applies the equations that it embodies, it marks off the particular input set as having been visited. Therefore when the rule (checker *s) is invoked again in the loop, it will not repeat any previous calculations.
4. If all the clauses are tried and found not applicable, *s is returned set to the special termination flag by the last clause in the sequence. When this flag is detected, a successful terminating sequence has been executed, and the partial hypothesis is consistent. The invoking procedure then continues down the search tree.

A.5 TRACE INFORMATION

When the trace flag %traceflag is set to "on", the search process prints out trace information. This information consists of the following:

1. When all the tuples in the set P are constructed and placed in the appopriate table, they are printed out. The order in which they are printed correspond to the order in which they will be tried for possible inclusion into the partial hypothesis being constructed.
2. Whenever the left branch is taken in the tree, the tuple corresponding to the level in the tree is placed in the queue of predictions. This fact is echoed in the output as "Add to FIFO <tuple>".
3. If the tuple is extended by the extension heuristic described earlier, the fact is echoed to the output.
4. The chunking rules which are invoked are also printed. They are printed in the following format: <Chunker name> <input tuple> <tuple already in the hypothesis> => <tuple being proposed>.
5. If the proposed tuple cannot be added to the queue because it is already been found inconsistent, the fact is noted by the trace.
6. When tuples are picked off the list and actually hypothesized, the trace is marked to read "HYPOTHESIZE <tuple>".

7. Inference engine applications are also listed in the trace. The name of the engine and the corresponding input set are listed.
8. If the engine computation results in an inconsistent value for some attribute, the old and new value for the attribute are listed.

A.6 EFFICIENCY OF THE SEARCH

To be able to determine how much work is performed in searching for the optimal hypothesis, we count the number of times the consistency checking procedure is invoked. This figure is reported for each solution generated in the search. To see how this number represents the efficiency of the procedure, look at the structure of the tree itself. The depth of the tree corresponds to the number of tuples generated, which has been previously been shown to be $2n(n-1)+m$ where n is the number of lines in the image and m is the number of arcs. If the tree search was a simple brute force technique, it would have to examine the consistency of the partial hypotheses at each level of the hypothesis. The efficiency of the search is reflected by the fact that the number of times consistency is checked is considerably smaller than the depth of the tree.

This means that the search is very efficient in ruling out possibilities ahead of time and in grouping the tuples efficiently. The smaller this number, the faster the search.

Appendix B

INPUT DATA FOR TEST EXAMPLES

In this Appendix, we show the test data digitized and preprocessed for the images shown in Chapter V. The data was digitized on a Bit-Pad plotter, and truncated to reduce the resolution. Digitized values were scaled to a 50x50 unit size, keeping one place of decimals in the result. Thus this resolution corresponds to what would be obtained in a 500x500 image where the x and z coordinates were restricted to be integer pixel values.

Appendix C contains the results of the search process on the data shown in this chapter. Both the data and the results refer to entities by integer identifiers. These identifiers are marked on the drawings of the data shown in this appendix. Line numbers are written as integers, point identifiers are shown in a circle and curve identifiers are shown in a box.

B.1 ARCHITECTURAL SCENE

The data in this section, corresponds to the drawing shown in Figure 13 in Chapter V. The digitized lines are shown in Figure 36, and correspond to the bold lines shown in Figure 13 The figure contains 31 points, and 18 lines on two planes.

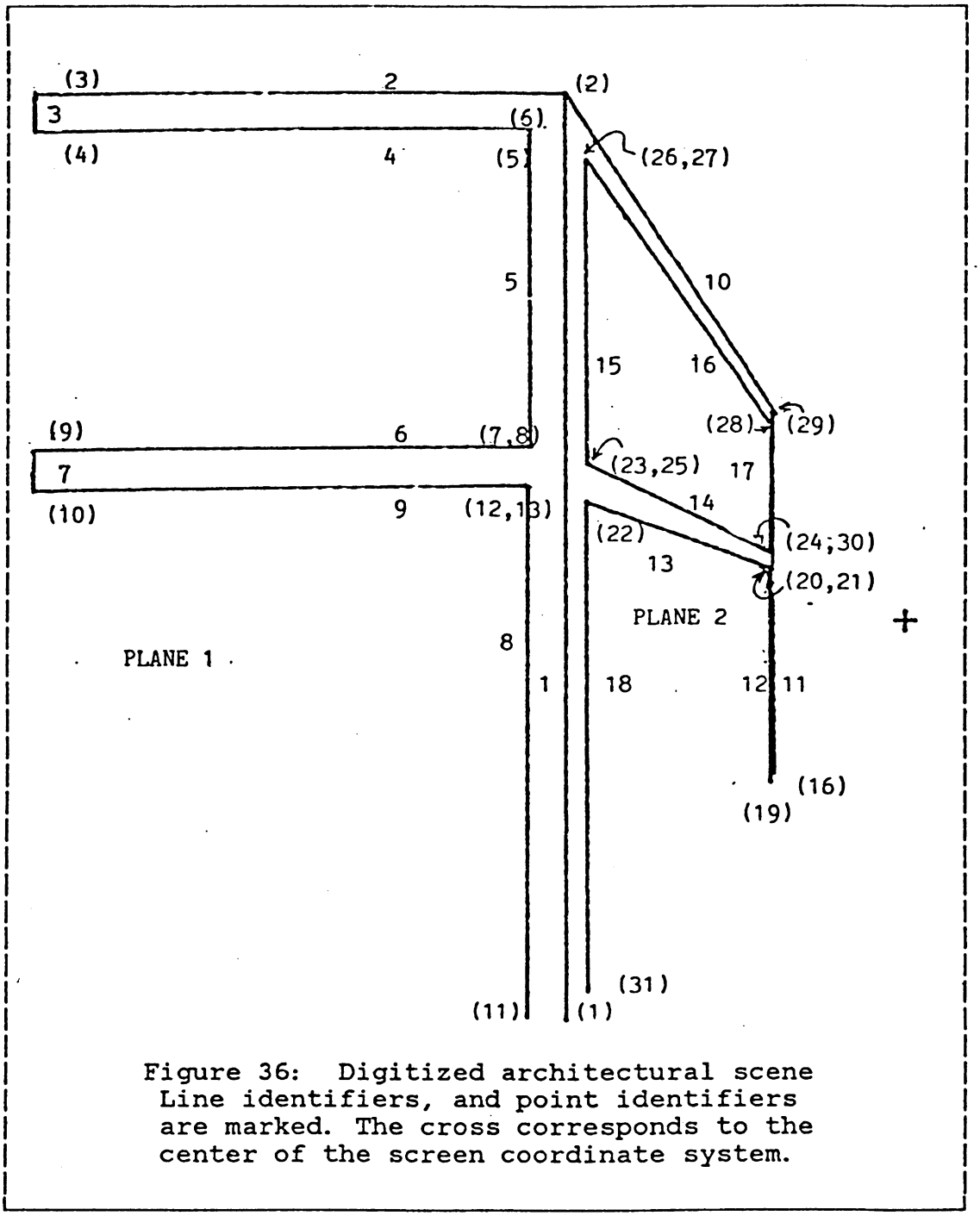


Figure 36: Digitized architectural scene Line identifiers, and point identifiers are marked. The cross corresponds to the center of the screen coordinate system.

PREPROCESSED DATA

```
( assert

( ( line 1 ( 1 2 )
     (    1.00000    0.00000    2.44720 ) ) )
( ( line3d 1 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 2 ( 2 3 )
     (    0.00000    1.00000   -3.72000 ) ) )
( ( line3d 2 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 3 ( 3 4 )
     (    1.00000    0.00000    6.38200 ) ) )
( ( line3d 3 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 4 ( 4 5 )
     (    0.00000    1.00000   -3.41000 ) ) )
( ( line3d 4 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 5 ( 6 7 )
     (    1.00000    0.00000    2.79200 ) ) )
( ( line3d 5 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 6 ( 8 9 )
     (    0.00000    1.00000   -1.22000 ) ) )
( ( line3d 6 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 7 ( 9 10 )
     (    1.00000    0.00000    6.35200 ) ) )
( ( line3d 7 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 8 ( 11 12 )
     (    1.00000    0.00000    2.75200 ) ) )
( ( line3d 8 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 9 ( 13 10 )
     (    0.00000    1.00000   -0.94000 ) ) )
( ( line3d 9 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 10 ( 2 14 )
```

```
     (     1.00000     0.69163     0.00000 ) ) )
( ( line3d 10 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 11 ( 15 16 )
     (     1.00000     0.00000     0.98200 ) ) )
( ( line3d 11 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 12 ( 19 20 )
     (     1.00000     0.00000     0.96200 ) ) )
( ( line3d 12 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 13 ( 21 22 )
     (     0.35606     1.00000     0.00000 ) ) )
( ( line3d 13 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 14 ( 23 24 )
     (     0.47727     1.00000     0.00000 ) ) )
( ( line3d 14 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 15 ( 25 26 )
     (     1.00000     0.00000     2.32200 ) ) )
( ( line3d 15 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 16 ( 27 28 )
     (     1.00000     0.73224     0.00000 ) ) )
( ( line3d 16 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 17 ( 29 30 )
     (     1.00000     0.00000     0.99200 ) ) )
( ( line3d 17 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( line 18 ( 22 31 )
     (     1.00000     0.00000     2.31200 ) ) )
( ( line3d 18 ( unknown unknown )
     ( unknown unknown unknown ) ) )

( ( point 1 (    -2.38200    -2.86000 ) ) )
( ( point3d 1 unknown unknown unknown ) )

( ( point 2 (    -2.53200     3.72000 ) ) )
( ( point3d 2 unknown unknown unknown ) )
```

```
( ( point 3 (    -6.38200     3.67000 ) ) )
( ( point3d 3 unknown unknown unknown ) )

( ( point 4 (    -6.38200     3.41000 ) ) )
( ( point3d 4 unknown unknown unknown ) )

( ( point 5 (    -2.78200     3.46000 ) ) )
( ( point3d 5 unknown unknown unknown ) )

( ( point 6 (    -2.79200     3.45000 ) ) )
( ( point3d 6 unknown unknown unknown ) )

( ( point 7 (    -2.74200     1.22000 ) ) )
( ( point3d 7 unknown unknown unknown ) )

( ( point 8 (    -2.73200     1.22000 ) ) )
( ( point3d 8 unknown unknown unknown ) )

( ( point 9 (    -6.35200     1.17000 ) ) )
( ( point3d 9 unknown unknown unknown ) )

( ( point 10 (    -6.35200     0.87000 ) ) )
( ( point3d 10 unknown unknown unknown ) )

( ( point 11 (    -2.75200     0.94000 ) ) )
( ( point3d 11 unknown unknown unknown ) )

( ( point 12 (    -2.67200    -2.85000 ) ) )
( ( point3d 12 unknown unknown unknown ) )

( ( point 13 (    -2.74200     0.94000 ) ) )
( ( point3d 13 unknown unknown unknown ) )

( ( point 14 (    -0.96200     1.45000 ) ) )
( ( point3d 14 unknown unknown unknown ) )

( ( point 15 (    -0.98200     1.46000 ) ) )
( ( point3d 15 unknown unknown unknown ) )

( ( point 16 (    -0.93200    -1.10000 ) ) )
( ( point3d 16 unknown unknown unknown ) )

( ( point 17 (    -0.93200    -1.11000 ) ) )
( ( point3d 17 unknown unknown unknown ) )

( ( point 18 (    -0.97200    -1.15000 ) ) )
( ( point3d 18 unknown unknown unknown ) )

( ( point 19 (    -0.96200    -1.15000 ) ) )
( ( point3d 19 unknown unknown unknown ) )
```

```
( ( point 20 (    -0.99200     0.36000 ) ) )
( ( point3d 20 unknown unknown unknown ) )

( ( point 21 (    -0.99200     0.37000 ) ) )
( ( point3d 21 unknown unknown unknown ) )

( ( point 22 (    -2.31200     0.84000 ) ) )
( ( point3d 22 unknown unknown unknown ) )

( ( point 23 (    -0.99200     0.48000 ) ) )
( ( point3d 23 unknown unknown unknown ) )

( ( point 24 (    -2.31200     1.11000 ) ) )
( ( point3d 24 unknown unknown unknown ) )

( ( point 25 (    -2.32200     1.11000 ) ) )
( ( point3d 25 unknown unknown unknown ) )

( ( point 26 (    -2.37200     3.23000 ) ) )
( ( point3d 26 unknown unknown unknown ) )

( ( point 27 (    -2.36200     3.23000 ) ) )
( ( point3d 27 unknown unknown unknown ) )

( ( point 28 (    -1.02200     1.40000 ) ) )
( ( point3d 28 unknown unknown unknown ) )

( ( point 29 (    -0.99200     1.40000 ) ) )
( ( point3d 29 unknown unknown unknown ) )

( ( point 30 (    -0.98200     0.49000 ) ) )
( ( point3d 30 unknown unknown unknown ) )

( ( point 31 (    -2.24200    -2.66000 ) ) )
( ( point3d 31 unknown unknown unknown ) )

( ( line_intersection ( 1 2 ) (    -2.53200     3.72000 ) ) )
( ( line_intersection ( 2 1 ) yes ) )

( ( line_intersection ( 1 3 ) ( infinity infinity ) ) )
( ( line_intersection ( 3 1 ) no ) )

( ( line_intersection ( 1 4 ) (    -2.44720     3.41000 ) ) )
( ( line_intersection ( 4 1 ) no ) )

( ( line_intersection ( 1 5 ) ( infinity infinity ) ) )
( ( line_intersection ( 5 1 ) no ) )

( ( line_intersection ( 1 6 ) (    -2.44720     1.22000 ) ) )
```

```
( ( line_intersection ( 6 1 ) no ) )

( ( line_intersection ( 1 7 ) ( infinity infinity ) ) )
( ( line_intersection ( 7 1 ) no ) )

( ( line_intersection ( 1 8 ) ( infinity infinity ) ) )
( ( line_intersection ( 8 1 ) no ) )

( ( line_intersection ( 1 9 ) (   -2.44720    0.94000 ) ) )
( ( line_intersection ( 9 1 ) no ) )

( ( line_intersection ( 1 10 ) (   -2.53200    3.72000 ) ) )
( ( line_intersection ( 10 1 ) yes ) )

( ( line_intersection ( 1 11 ) ( infinity infinity ) ) )
( ( line_intersection ( 11 1 ) no ) )

( ( line_intersection ( 1 12 ) ( infinity infinity ) ) )
( ( line_intersection ( 12 1 ) no ) )

( ( line_intersection ( 1 13 ) (   -2.44720    0.87135 ) ) )
( ( line_intersection ( 13 1 ) no ) )

( ( line_intersection ( 1 14 ) (   -2.44720    1.16798 ) ) )
( ( line_intersection ( 14 1 ) no ) )

( ( line_intersection ( 1 15 ) ( infinity infinity ) ) )
( ( line_intersection ( 15 1 ) no ) )

( ( line_intersection ( 1 16 ) (   -2.44720    3.34207 ) ) )
( ( line_intersection ( 16 1 ) no ) )

( ( line_intersection ( 1 17 ) ( infinity infinity ) ) )
( ( line_intersection ( 17 1 ) no ) )

( ( line_intersection ( 1 18 ) ( infinity infinity ) ) )
( ( line_intersection ( 18 1 ) no ) )

( ( line_intersection ( 2 3 ) (   -6.38200    3.67000 ) ) )
( ( line_intersection ( 3 2 ) yes ) )

( ( line_intersection ( 2 4 ) ( infinity infinity ) ) )
( ( line_intersection ( 4 2 ) no ) )

( ( line_intersection ( 2 5 ) (  -2.79200    3.72000 ) ) )
( ( line_intersection ( 5 2 ) no ) )

( ( line_intersection ( 2 6 ) ( infinity infinity ) ) )
( ( line_intersection ( 6 2 ) no ) )
```

```
( ( line_intersection ( 2 7 ) (   -6.35200    3.72000 ) ) )
( ( line_intersection ( 7 2 ) no ) )

( ( line_intersection ( 2 8 ) (   -2.75200    3.72000 ) ) )
( ( line_intersection ( 8 2 ) no ) )

( ( line_intersection ( 2 9 ) ( infinity infinity ) ) )
( ( line_intersection ( 9 2 ) no ) )

( ( line_intersection ( 2 10 ) (   -2.53200    3.72000 ) ) )
( ( line_intersection ( 10 2 ) yes ) )

( ( line_intersection ( 2 11 ) (   -0.98200    3.72000 ) ) )
( ( line_intersection ( 11 2 ) no ) )

( ( line_intersection ( 2 12 ) (   -0.96200    3.72000 ) ) )
( ( line_intersection ( 12 2 ) no ) )

( ( line_intersection ( 2 13 ) (  -10.44765    3.72000 ) ) )
( ( line_intersection ( 13 2 ) no ) )

( ( line_intersection ( 2 14 ) (   -7.79428    3.72000 ) ) )
( ( line_intersection ( 14 2 ) no ) )

( ( line_intersection ( 2 15 ) (   -2.32200    3.72000 ) ) )
( ( line_intersection ( 15 2 ) no ) )

( ( line_intersection ( 2 16 ) (   -2.72393    3.72000 ) ) )
( ( line_intersection ( 16 2 ) no ) )

( ( line_intersection ( 2 17 ) (   -0.99200    3.72000 ) ) )
( ( line_intersection ( 17 2 ) no ) )

( ( line_intersection ( 2 18 ) (   -2.31200    3.72000 ) ) )
( ( line_intersection ( 18 2 ) no ) )

( ( line_intersection ( 3 4 ) (   -6.38200    3.41000 ) ) )
( ( line_intersection ( 4 3 ) yes ) )

( ( line_intersection ( 3 5 ) ( infinity infinity ) ) )
( ( line_intersection ( 5 3 ) no ) )

( ( line_intersection ( 3 6 ) (   -6.38200    1.22000 ) ) )
( ( line_intersection ( 6 3 ) no ) )

( ( line_intersection ( 3 7 ) ( infinity infinity ) ) )
( ( line_intersection ( 7 3 ) no ) )

( ( line_intersection ( 3 8 ) ( infinity infinity ) ) )
( ( line_intersection ( 8 3 ) no ) )
```

```
( ( line_intersection ( 3 9 ) (   -6.38200    0.94000 ) ) )
( ( line_intersection ( 9 3 ) no ) )

( ( line_intersection ( 3 10 ) (   -6.38200    9.22748 ) ) )
( ( line_intersection ( 10 3 ) no ) )

( ( line_intersection ( 3 11 ) ( infinity infinity ) ) )
( ( line_intersection ( 11 3 ) no ) )

( ( line_intersection ( 3 12 ) ( infinity infinity ) ) )
( ( line_intersection ( 12 3 ) no ) )

( ( line_intersection ( 3 13 ) (   -6.38200    2.27238 ) ) )
( ( line_intersection ( 13 3 ) no ) )

( ( line_intersection ( 3 14 ) (   -6.38200    3.04596 ) ) )
( ( line_intersection ( 14 3 ) no ) )

( ( line_intersection ( 3 15 ) ( infinity infinity ) ) )
( ( line_intersection ( 15 3 ) no ) )

( ( line_intersection ( 3 16 ) (   -6.38200    8.71572 ) ) )
( ( line_intersection ( 16 3 ) no ) )

( ( line_intersection ( 3 17 ) ( infinity infinity ) ) )
( ( line_intersection ( 17 3 ) no ) )

( ( line_intersection ( 3 18 ) ( infinity infinity ) ) )
( ( line_intersection ( 18 3 ) no ) )

( ( line_intersection ( 4 5 ) (   -2.79200    3.41000 ) ) )
( ( line_intersection ( 5 4 ) no ) )

( ( line_intersection ( 4 6 ) ( infinity infinity ) ) )
( ( line_intersection ( 6 4 ) no ) )

( ( line_intersection ( 4 7 ) (   -6.35200    3.41000 ) ) )
( ( line_intersection ( 7 4 ) no ) )

( ( line_intersection ( 4 8 ) (   -2.75200    3.41000 ) ) )
( ( line_intersection ( 8 4 ) no ) )

( ( line_intersection ( 4 9 ) ( infinity infinity ) ) )
( ( line_intersection ( 9 4 ) no ) )

( ( line_intersection ( 4 10 ) (   -2.35846    3.41000 ) ) )
( ( line_intersection ( 10 4 ) no ) )

( ( line_intersection ( 4 11 ) (   -0.98200    3.41000 ) ) )
```

```
( ( line_intersection ( 11 4 ) no ) )

( ( line_intersection ( 4 12 ) (   -0.96200    3.41000 ) ) )
( ( line_intersection ( 12 4 ) no ) )

( ( line_intersection ( 4 13 ) (   -9.57701    3.41000 ) ) )
( ( line_intersection ( 13 4 ) no ) )

( ( line_intersection ( 4 14 ) (   -7.14476    3.41000 ) ) )
( ( line_intersection ( 14 4 ) no ) )

( ( line_intersection ( 4 15 ) (   -2.32200    3.41000 ) ) )
( ( line_intersection ( 15 4 ) no ) )

( ( line_intersection ( 4 16 ) (   -2.49694    3.41000 ) ) )
( ( line_intersection ( 16 4 ) no ) )

( ( line_intersection ( 4 17 ) (   -0.99200    3.41000 ) ) )
( ( line_intersection ( 17 4 ) no ) )

( ( line_intersection ( 4 18 ) (   -2.31200    3.41000 ) ) )
( ( line_intersection ( 18 4 ) no ) )

( ( line_intersection ( 5 6 ) (   -2.79200    1.22000 ) ) )
( ( line_intersection ( 6 5 ) no ) )

( ( line_intersection ( 5 7 ) ( infinity infinity ) ) )
( ( line_intersection ( 7 5 ) no ) )

( ( line_intersection ( 5 8 ) ( infinity infinity ) ) )
( ( line_intersection ( 8 5 ) no ) )

( ( line_intersection ( 5 9 ) (   -2.79200    0.94000 ) ) )
( ( line_intersection ( 9 5 ) no ) )

( ( line_intersection ( 5 10 ) (   -2.79200    4.03684 ) ) )
( ( line_intersection ( 10 5 ) no ) )

( ( line_intersection ( 5 11 ) ( infinity infinity ) ) )
( ( line_intersection ( 11 5 ) no ) )

( ( line_intersection ( 5 12 ) ( infinity infinity ) ) )
( ( line_intersection ( 12 5 ) no ) )

( ( line_intersection ( 5 13 ) (   -2.79200    0.99412 ) ) )
( ( line_intersection ( 13 5 ) no ) )

( ( line_intersection ( 5 14 ) (   -2.79200    1.33255 ) ) )
( ( line_intersection ( 14 5 ) no ) )
```

```
( ( line_intersection ( 5 15 ) ( infinity infinity ) ) )
( ( line_intersection ( 15 5 ) no ) )

( ( line_intersection ( 5 16 ) (   -2.79200    3.81295 ) ) )
( ( line_intersection ( 16 5 ) no ) )

( ( line_intersection ( 5 17 ) ( infinity infinity ) ) )
( ( line_intersection ( 17 5 ) no ) )

( ( line_intersection ( 5 18 ) ( infinity infinity ) ) )
( ( line_intersection ( 18 5 ) no ) )

( ( line_intersection ( 6 7 ) (   -6.35200    1.17000 ) ) )
( ( line_intersection ( 7 6 ) yes ) )

( ( line_intersection ( 6 8 ) (   -2.75200    1.22000 ) ) )
( ( line_intersection ( 8 6 ) no ) )

( ( line_intersection ( 6 9 ) ( infinity infinity ) ) )
( ( line_intersection ( 9 6 ) no ) )

( ( line_intersection ( 6 10 ) (   -0.84379    1.22000 ) ) )
( ( line_intersection ( 10 6 ) no ) )

( ( line_intersection ( 6 11 ) (   -0.98200    1.22000 ) ) )
( ( line_intersection ( 11 6 ) no ) )

( ( line_intersection ( 6 12 ) (   -0.96200    1.22000 ) ) )
( ( line_intersection ( 12 6 ) no ) )

( ( line_intersection ( 6 13 ) (   -3.42638    1.22000 ) ) )
( ( line_intersection ( 13 6 ) no ) )

( ( line_intersection ( 6 14 ) (   -2.55619    1.22000 ) ) )
( ( line_intersection ( 14 6 ) no ) )

( ( line_intersection ( 6 15 ) (   -2.32200    1.22000 ) ) )
( ( line_intersection ( 15 6 ) no ) )

( ( line_intersection ( 6 16 ) (   -0.89333    1.22000 ) ) )
( ( line_intersection ( 16 6 ) no ) )

( ( line_intersection ( 6 17 ) (   -0.99200    1.22000 ) ) )
( ( line_intersection ( 17 6 ) no ) )

( ( line_intersection ( 6 18 ) (   -2.31200    1.22000 ) ) )
( ( line_intersection ( 18 6 ) no ) )

( ( line_intersection ( 7 8 ) ( infinity infinity ) ) )
( ( line_intersection ( 8 7 ) no ) )
```

```
( ( line_intersection ( 7 9 ) (   -6.35200    0.87000 ) ) )
( ( line_intersection ( 9 7 ) yes ) )

( ( line_intersection ( 7 10 ) (   -6.35200    9.18410 ) ) )
( ( line_intersection ( 10 7 ) no ) )

( ( line_intersection ( 7 11 ) ( infinity infinity ) ) )
( ( line_intersection ( 11 7 ) no ) )

( ( line_intersection ( 7 12 ) ( infinity infinity ) ) )
( ( line_intersection ( 12 7 ) no ) )

( ( line_intersection ( 7 13 ) (   -6.35200    2.26170 ) ) )
( ( line_intersection ( 13 7 ) no ) )

( ( line_intersection ( 7 14 ) (   -6.35200    3.03164 ) ) )
( ( line_intersection ( 14 7 ) no ) )

( ( line_intersection ( 7 15 ) ( infinity infinity ) ) )
( ( line_intersection ( 15 7 ) no ) )

( ( line_intersection ( 7 16 ) (   -6.35200    8.67474 ) ) )
( ( line_intersection ( 16 7 ) no ) )

( ( line_intersection ( 7 17 ) ( infinity infinity ) ) )
( ( line_intersection ( 17 7 ) no ) )

( ( line_intersection ( 7 18 ) ( infinity infinity ) ) )
( ( line_intersection ( 18 7 ) no ) )

( ( line_intersection ( 8 9 ) (   -2.75200    0.94000 ) ) )
( ( line_intersection ( 9 8 ) no ) )

( ( line_intersection ( 8 10 ) (   -2.75200    3.97901 ) ) )
( ( line_intersection ( 10 8 ) no ) )

( ( line_intersection ( 8 11 ) ( infinity infinity ) ) )
( ( line_intersection ( 11 8 ) no ) )

( ( line_intersection ( 8 12 ) ( infinity infinity ) ) )
( ( line_intersection ( 12 8 ) no ) )

( ( line_intersection ( 8 13 ) (   -2.75200    0.97988 ) ) )
( ( line_intersection ( 13 8 ) no ) )

( ( line_intersection ( 8 14 ) (   -2.75200    1.31346 ) ) )
( ( line_intersection ( 14 8 ) no ) )

( ( line_intersection ( 8 15 ) ( infinity infinity ) ) )
```

```
( ( line_intersection ( 15 8 ) no ) )

( ( line_intersection ( 8 16 ) (   -2.75200    3.75833 ) ) )
( ( line_intersection ( 16 8 ) no ) )

( ( line_intersection ( 8 17 ) ( infinity infinity ) ) )
( ( line_intersection ( 17 8 ) no ) )

( ( line_intersection ( 8 18 ) ( infinity infinity ) ) )
( ( line_intersection ( 18 8 ) no ) )

( ( line_intersection ( 9 10 ) (   -0.65013    0.94000 ) ) )
( ( line_intersection ( 10 9 ) no ) )

( ( line_intersection ( 9 11 ) (   -0.98200    0.94000 ) ) )
( ( line_intersection ( 11 9 ) no ) )

( ( line_intersection ( 9 12 ) (   -0.96200    0.94000 ) ) )
( ( line_intersection ( 12 9 ) no ) )

( ( line_intersection ( 9 13 ) (   -2.64000    0.94000 ) ) )
( ( line_intersection ( 13 9 ) no ) )

( ( line_intersection ( 9 14 ) (   -1.96952    0.94000 ) ) )
( ( line_intersection ( 14 9 ) no ) )

( ( line_intersection ( 9 15 ) (   -2.32200    0.94000 ) ) )
( ( line_intersection ( 15 9 ) no ) )

( ( line_intersection ( 9 16 ) (   -0.68831    0.94000 ) ) )
( ( line_intersection ( 16 9 ) no ) )

( ( line_intersection ( 9 17 ) (   -0.99200    0.94000 ) ) )
( ( line_intersection ( 17 9 ) no ) )

( ( line_intersection ( 9 18 ) (   -2.31200    0.94000 ) ) )
( ( line_intersection ( 18 9 ) no ) )

( ( line_intersection ( 10 11 ) (   -0.98200    1.41983 ) ) )
( ( line_intersection ( 11 10 ) no ) )

( ( line_intersection ( 10 12 ) (   -0.96200    1.39092 ) ) )
( ( line_intersection ( 12 10 ) no ) )

( ( line_intersection ( 10 13 ) (    0.00000    0.00000 ) ) )
( ( line_intersection ( 13 10 ) no ) )

( ( line_intersection ( 10 14 ) (    0.00000    0.00000 ) ) )
( ( line_intersection ( 14 10 ) no ) )
```

```
( ( line_intersection ( 10 15 ) (   -2.32200    3.35729 ) ) )
( ( line_intersection ( 15 10 ) no ) )

( ( line_intersection ( 10 16 ) ( 0.00000 0.00000 ) ) )
( ( line_intersection ( 16 10 ) no ) )

( ( line_intersection ( 10 17 ) (   -0.99200    1.43429 ) ) )
( ( line_intersection ( 17 10 ) no ) )

( ( line_intersection ( 10 18 ) (   -2.31200    3.34283 ) ) )
( ( line_intersection ( 18 10 ) no ) )

( ( line_intersection ( 11 12 ) ( infinity infinity ) ) )
( ( line_intersection ( 12 11 ) no ) )

( ( line_intersection ( 11 13 ) (   -0.98200    0.34965 ) ) )
( ( line_intersection ( 13 11 ) no ) )

( ( line_intersection ( 11 14 ) (   -0.98200    0.46868 ) ) )
( ( line_intersection ( 14 11 ) no ) )

( ( line_intersection ( 11 15 ) ( infinity infinity ) ) )
( ( line_intersection ( 15 11 ) no ) )

( ( line_intersection ( 11 16 ) (   -0.98200    1.34109 ) ) )
( ( line_intersection ( 16 11 ) no ) )

( ( line_intersection ( 11 17 ) ( infinity infinity ) ) )
( ( line_intersection ( 17 11 ) no ) )

( ( line_intersection ( 11 18 ) ( infinity infinity ) ) )
( ( line_intersection ( 18 11 ) no ) )

( ( line_intersection ( 12 13 ) (   -0.93200    0.33185 ) ) )
( ( line_intersection ( 13 12 ) no ) )

( ( line_intersection ( 12 14 ) (   -0.93200    0.44482 ) ) )
( ( line_intersection ( 14 12 ) no ) )

( ( line_intersection ( 12 15 ) ( infinity infinify ) ) )
( ( line_intersection ( 15 12 ) no ) )

( ( line_intersection ( 12 16 ) (   -0.93200    1.27281 ) ) )
( ( line_intersection ( 16 12 ) no ) )

( ( line_intersection ( 12 17 ) ( infinity infinity ) ) )
( ( line_intersection ( 17 12 ) no ) )

( ( line_intersection ( 12 18 ) ( -0.932 0.34253 ) ) )
( ( line_intersection ( 18 12 ) no ) )
```

```
( ( line_intersection ( 12 13 ) (   -0.96200    0.34253 ) ) )
( ( line_intersection ( 13 12 ) no ) )

( ( line_intersection ( 12 14 ) (   -0.96200    0.45914 ) ) )
( ( line_intersection ( 14 12 ) no ) )

( ( line_intersection ( 12 15 ) ( infinity infinity ) ) )
( ( line_intersection ( 15 12 ) no ) )

( ( line_intersection ( 12 16 ) (   -0.96200    1.31378 ) ) )
( ( line_intersection ( 16 12 ) no ) )

( ( line_intersection ( 12 17 ) ( infinity infinity ) ) )
( ( line_intersection ( 17 12 ) no ) )

( ( line_intersection ( 12 18 ) ( infinity infinity ) ) )
( ( line_intersection ( 18 12 ) no ) )

( ( line_intersection ( 13 14 ) (    0.00000    0.00000 ) ) )
( ( line_intersection ( 14 13 ) no ) )

( ( line_intersection ( 13 15 ) (   -2.32200    0.82677 ) ) )
( ( line_intersection ( 15 13 ) no ) )

( ( line_intersection ( 13 16 ) (    0.00000    0.00000 ) ) )
( ( line_intersection ( 16 13 ) no ) )

( ( line_intersection ( 13 17 ) (   -0.99200    0.35321 ) ) )
( ( line_intersection ( 17 13 ) no ) )

( ( line_intersection ( 13 18 ) (   -2.31200    0.84000 ) ) )
( ( line_intersection ( 18 13 ) yes ) )

( ( line_intersection ( 14 15 ) (   -2.32200    1.10823 ) ) )
( ( line_intersection ( 15 14 ) no ) )

( ( line_intersection ( 14 16 ) (    0.00000    0.00000 ) ) )
( ( line_intersection ( 16 14 ) no ) )

( ( line_intersection ( 14 17 ) (   -0.99200    0.47345 ) ) )
( ( line_intersection ( 17 14 ) no ) )

( ( line_intersection ( 14 18 ) (   -2.31200    1.10345 ) ) )
( ( line_intersection ( 18 14 ) no ) )

( ( line_intersection ( 15 16 ) (   -2.32200    3.17109 ) ) )
( ( line_intersection ( 16 15 ) no ) )

( ( line_intersection ( 15 17 ) ( infinity infinity ) ) )
```

```
( ( line_intersection ( 17 15 ) no ) )

( ( line_intersection ( 15 18 ) ( infinity infinity ) ) )
( ( line_intersection ( 18 15 ) no ) )

( ( line_intersection ( 16 17 ) (   -0.99200    1.35475 ) ) )
( ( line_intersection ( 17 16 ) no ) )

( ( line_intersection ( 16 18 ) (   -2.31200    3.15743 ) ) )
( ( line_intersection ( 18 16 ) no ) )

( ( line_intersection ( 17 18 ) ( infinity infinity ) ) )
( ( line_intersection ( 18 17 ) no ) )


 )

( <-  %numpts 31 )
( <-  %numlins 18 )

( assert

((plane 1 (1 2 3 4 5 6 7 8 9) () ))
((plane 2 (1 10 11 12 13 14 15 16 17 18) () ))
((plane3d 1 (unknown unknown unknown)
            (unknown unknown unknown)
             unknown))
((plane3d 2 (unknown unknown unknown)
            (unknown unknown unknown)
             unknown))

)

(<- %numplane 2)
```

B.2 THE TABLE SCENE

The data in this section corresponds to the table scene in Figure 14, Chapter V. The digitized image is shown in Figure 37.

THE PREPROCESSED DATA

```
( assert

( ( line 1 ( 1 2 ) (   -0.81395    1.00000  -0.034884 ) ) )
```

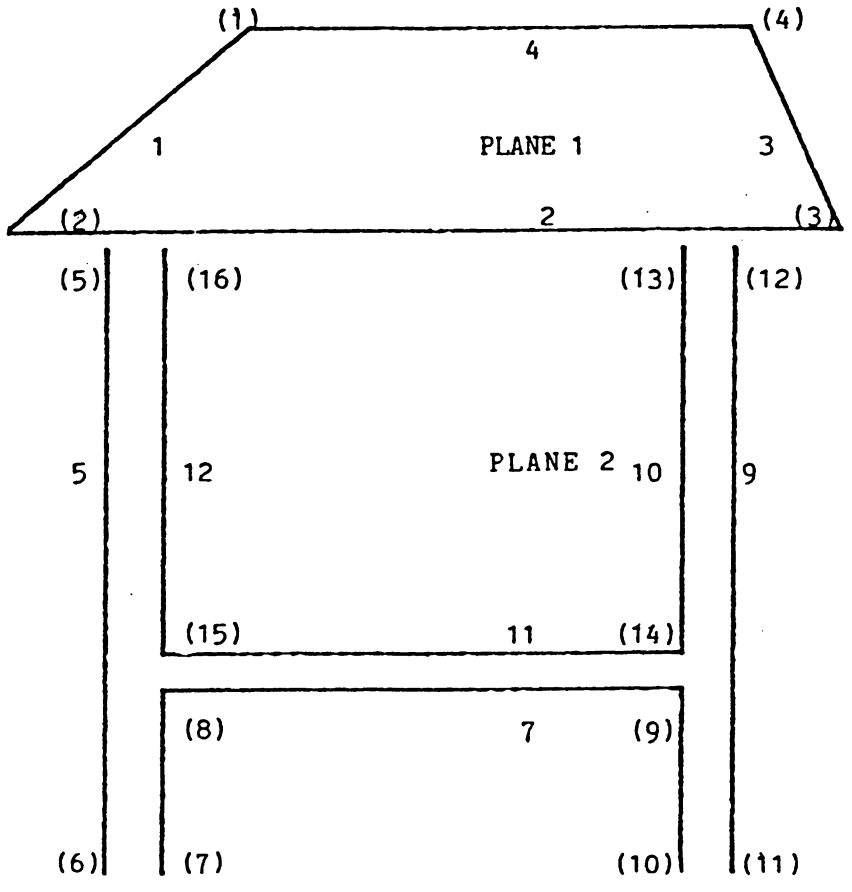


Figure 37: The Digitized Table Image

```
( ( line3d 1 ( unknown unknown )
             ( unknown unknown unknown ) ) )

( ( line 2 ( 2 3 ) (     0.00000     1.00000     3.75000 ) ) )
( ( line3d 2 ( unknown unknown )
             ( unknown unknown unknown ) ) )

( ( line 3 ( 3 4 ) (     1.00000     0.45714    -0.98571 ) ) )
( ( line3d 3 ( unknown unknown )
             ( unknown unknown unknown ) ) )

( ( line 4 ( 4 1 ) (     0.00000     1.00000     2.00000 ) ) )
( ( line3d 4 ( unknown unknown )
             ( unknown unknown unknown ) ) )

( ( line 5 ( 5 6 ) (     1.00000     0.00000     3.75000 ) ) )
( ( line3d 5 ( unknown unknown )
             ( unknown unknown unknown ) ) )

( ( line 6 ( 7 8 ) (     1.00000     0.00000     3.25000 ) ) )
( ( line3d 6 ( unknown unknown )
             ( unknown unknown unknown ) ) )

( ( line 7 ( 8 9 ) (     0.00000     1.00000     7.70000 ) ) )
( ( line3d 7 ( unknown unknown )
             ( unknown unknown unknown ) ) )

( ( line 8 ( 9 10 ) (     1.00000     0.00000    -1.35000 ) ) )
( ( line3d 8 ( unknown unknown )
             ( unknown unknown unknown ) ) )

( ( line 9 ( 11 12 ) (     1.00000     0.00000    -1.80000 ) ) )
( ( line3d 9 ( unknown unknown )
             ( unknown unknown unknown ) ) )

( ( line 10 ( 13 14 ) (     1.00000     0.00000    -1.35000 ) ) )
( ( line3d 10 ( unknown unknown )
              ( unknown unknown unknown ) ) )

( ( line 11 ( 14 15 ) (     0.00000     1.00000     7.40000 ) ) )
( ( line3d 11 ( unknown unknown )
              ( unknown unknown unknown ) ) )

( ( line 12 ( 15 16 ) (     1.00000     0.00000     3.25000 ) ) )
( ( line3d 12 ( unknown unknown )
              ( unknown unknown unknown ) ) )

( ( point 1 (    -2.50000    -2.00000 ) ) )
( ( point3d 1 unknown unknown unknown ) )
```

```
( ( point 2 (    -4.65000    -3.75000 ) ) )
( ( point3d 2 unknown unknown unknown ) )

( ( point 3 (     2.70000    -3.75000 ) ) )
( ( point3d 3 unknown unknown unknown ) )

( ( point 4 (     1.90000    -2.00000 ) ) )
( ( point3d 4 unknown unknown unknown ) )

( ( point 5 (    -3.75000    -3.90000 ) ) )
( ( point3d 5 unknown unknown unknown ) )

( ( point 6 (    -3.75000    -9.30000 ) ) )
( ( point3d 6 unknown unknown unknown ) )

( ( point 7 (    -3.25000    -9.30000 ) ) )
( ( point3d 7 unknown unknown unknown ) )

( ( point 8 (    -3.25000    -7.70000 ) ) )
( ( point3d 8 unknown unknown unknown ) )

( ( point 9 (     1.35000    -7.70000 ) ) )
( ( point3d 9 unknown unknown unknown ) )

( ( point 10 (     1.35000    -9.30000 ) ) )
( ( point3d 10 unknown unknown unknown ) )

( ( point 11 (     1.80000    -9.30000 ) ) )
( ( point3d 11 unknown unknown unknown ) )

( ( point 12 (     1.80000    -3.90000 ) ) )
( ( point3d 12 unknown unknown unknown ) )

( ( point 13 (     1.35000    -3.90000 ) ) )
( ( point3d 13 unknown unknown unknown ) )

( ( point 14 (     1.35000    -7.40000 ) ) )
( ( point3d 14 unknown unknown unknown ) )

( ( point 15 (    -3.25000    -7.40000 ) ) )
( ( point3d 15 unknown unknown unknown ) )

( ( point 16 (    -3.25000    -3.90000 ) ) )
( ( point3d 16 unknown unknown unknown ) )

( ( line_intersection ( 1 2 ) (    -4.65000    -3.75000 ) ) )
( ( line_intersection ( 2 1 ) yes ) )

( ( line_intersection ( 1 3 ) (     0.70678     0.61017 ) ) )
( ( line_intersection ( 3 1 ) no ) )
```

```
( ( line_intersection ( 1 4 ) (   -2.50000   -2.00000 ) ) )
( ( line_intersection ( 4 1 ) yes ) )

( ( line_intersection ( 1 5 ) (   -3.75000   -3.01744 ) ) )
( ( line_intersection ( 5 1 ) no ) )

( ( line_intersection ( 1 6 ) (   -3.25000   -2.61047 ) ) )
( ( line_intersection ( 6 1 ) no ) )

( ( line_intersection ( 1 7 ) (   -9.50286   -7.70000 ) ) )
( ( line_intersection ( 7 1 ) no ) )

( ( line_intersection ( 1 8 ) (    1.35000    1.13372 ) ) )
( ( line_intersection ( 8 1 ) no ) )

( ( line_intersection ( 1 9 ) (    1.80000    1.50000 ) ) )
( ( line_intersection ( 9 1 ) no ) )

( ( line_intersection ( 1 10 ) (    1.35000    1.13372 ) ) )
( ( line_intersection ( 10 1 ) no ) )

( ( line_intersection ( 1 11 ) (   -9.13429   -7.40000 ) ) )
( ( line_intersection ( 11 1 ) no ) )

( ( line_intersection ( 1 12 ) (   -3.25000   -2.61047 ) ) )
( ( line_intersection ( 12 1 ) no ) )

( ( line_intersection ( 2 3 ) (    2.70000   -3.75000 ) ) )
( ( line_intersection ( 3 2 ) yes ) )

( ( line_intersection ( 2 4 ) ( infinity infinity ) ) )
( ( line_intersection ( 4 2 ) no ) )

( ( line_intersection ( 2 5 ) (   -3.75000   -3.75000 ) ) )
( ( line_intersection ( 5 2 ) no ) )

( ( line_intersection ( 2 6 ) (   -3.25000   -3.75000 ) ) )
( ( line_intersection ( 6 2 ) no ) )

( ( line_intersection ( 2 7 ) ( infinity infinity ) ) )
( ( line_intersection ( 7 2 ) no ) )

( ( line_intersection ( 2 8 ) (    1.35000   -3.75000 ) ) )
( ( line_intersection ( 8 2 ) no ) )

( ( line_intersection ( 2 9 ) (    1.80000   -3.75000 ) ) )
( ( line_intersection ( 9 2 ) no ) )

( ( line_intersection ( 2 10 ) (    1.35000   -3.75000 ) ) )
```

```
( ( line_intersection ( 10 2 ) no ) )

( ( line_intersection ( 2 11 ) ( infinity infinity ) ) )
( ( line_intersection ( 11 2 ) no ) )

( ( line_intersection ( 2 12 ) (   -3.25000   -3.75000 ) ) )
( ( line_intersection ( 12 2 ) no ) )

( ( line_intersection ( 3 4 ) (    1.90000   -2.00000 ) ) )
( ( line_intersection ( 4 3 ) yes ) )

( ( line_intersection ( 3 5 ) (   -3.75000   10.35937 ) ) )
( ( line_intersection ( 5 3 ) no ) )

( ( line_intersection ( 3 6 ) (   -3.25000    9.26562 ) ) )
( ( line_intersection ( 6 3 ) no ) )

( ( line_intersection ( 3 7 ) (    4.50572   -7.70000 ) ) )
( ( line_intersection ( 7 3 ) no ) )

( ( line_intersection ( 3 8 ) (    1.35000   -0.79688 ) ) )
( ( line_intersection ( 8 3 ) no ) )

( ( line_intersection ( 3 9 ) (    1.80000   -1.78125 ) ) )
( ( line_intersection ( 9 3 ) no ) )

( ( line_intersection ( 3 10 ) (    1.35000   -0.79688 ) ) )
( ( line_intersection ( 10 3 ) no ) )

( ( line_intersection ( 3 11 ) (    4.36857   -7.40000 ) ) )
( ( line_intersection ( 11 3 ) no ) )

( ( line_intersection ( 3 12 ) (   -3.25000    9.26562 ) ) )
( ( line_intersection ( 12 3 ) no ) )

( ( line_intersection ( 4 5 ) (   -3.75000   -2.00000 ) ) )
( ( line_intersection ( 5 4 ) no ) )

( ( line_intersection ( 4 6 ) (   -3.25000   -2.00000 ) ) )
( ( line_intersection ( 6 4 ) no ) )

( ( line_intersection ( 4 7 ) ( infinity infinity ) ) )
( ( line_intersection ( 7 4 ) no ) )

( ( line_intersection ( 4 8 ) (    1.35000   -2.00000 ) ) )
( ( line_intersection ( 8 4 ) no ) )

( ( line_intersection ( 4 9 ) (    1.80000   -2.00000 ) ) )
( ( line_intersection ( 9 4 ) no ) )
```

```
( ( line_intersection ( 4 10 ) (     1.35000   -2.00000 ) ) )
( ( line_intersection ( 10 4 ) no ) )

( ( line_intersection ( 4 11 ) ( infinity infinity ) ) )
( ( line_intersection ( 11 4 ) no ) )

( ( line_intersection ( 4 12 ) (   -3.25000   -2.00000 ) ) )
( ( line_intersection ( 12 4 ) no ) )

( ( line_intersection ( 5 6 ) ( infinity infinity ) ) )
( ( line_intersection ( 6 5 ) no ) )

( ( line_intersection ( 5 7 ) (   -3.75000   -7.70000 ) ) )
( ( line_intersection ( 7 5 ) no ) )

( ( line_intersection ( 5 8 ) ( infinity infinity ) ) )
( ( line_intersection ( 8 5 ) no ) )

( ( line_intersection ( 5 9 ) ( infinity infinity ) ) )
( ( line_intersection ( 9 5 ) no ) )

( ( line_intersection ( 5 10 ) ( infinity infinity ) ) )
( ( line_intersection ( 10 5 ) no ) )

( ( line_intersection ( 5 11 ) (   -3.75000   -7.40000 ) ) )
( ( line_intersection ( 11 5 ) no ) )

( ( line_intersection ( 5 12 ) ( infinity infinity ) ) )
( ( line_intersection ( 12 5 ) no ) )

( ( line_intersection ( 6 7 ) (   -3.25000   -7.70000 ) ) )
( ( line_intersection ( 7 6 ) yes ) )

( ( line_intersection ( 6 8 ) ( infinity infinity ) ) )
( ( line_intersection ( 8 6 ) no ) )

( ( line_intersection ( 6 9 ) ( infinity infinity ) ) )
( ( line_intersection ( 9 6 ) no ) )

( ( line_intersection ( 6 10 ) ( infinity infinity ) ) )
( ( line_intersection ( 10 6 ) no ) )

( ( line_intersection ( 6 11 ) (   -3.25000   -7.40000 ) ) )
( ( line_intersection ( 11 6 ) no ) )

( ( line_intersection ( 6 12 ) ( infinity infinity ) ) )
( ( line_intersection ( 12 6 ) no ) )

( ( line_intersection ( 7 8 ) (     1.35000   -7.70000 ) ) )
( ( line_intersection ( 8 7 ) yes ) )
```

```
( ( line_intersection ( 7 9 ) (     1.80000   -7.70000 ) ) )
( ( line_intersection ( 9 7 ) no ) )

( ( line_intersection ( 7 10 ) (    1.35000   -7.70000 ) ) )
( ( line_intersection ( 10 7 ) no ) )

( ( line_intersection ( 7 11 ) ( infinity infinity ) ) )
( ( line_intersection ( 11 7 ) no ) )

( ( line_intersection ( 7 12 ) (   -3.25000   -7.70000 ) ) )
( ( line_intersection ( 12 7 ) no ) )

( ( line_intersection ( 8 9 ) ( infinity infinity ) ) )
( ( line_intersection ( 9 8 ) no ) )

( ( line_intersection ( 8 10 ) ( infinity infinity ) ) )
( ( line_intersection ( 10 8 ) no ) )

( ( line_intersection ( 8 11 ) (    1.35000   -7.40000 ) ) )
( ( line_intersection ( 11 8 ) no ) )

( ( line_intersection ( 8 12 ) ( infinity infinity ) ) )
( ( line_intersection ( 12 8 ) no ) )

( ( line_intersection ( 9 10 ) ( infinity infinity ) ) )
( ( line_intersection ( 10 9 ) no ) )

( ( line_intersection ( 9 11 ) (    1.80000   -7.40000 ) ) )
( ( line_intersection ( 11 9 ) no ) )

( ( line_intersection ( 9 12 ) ( infinity infinity ) ) )
( ( line_intersection ( 12 9 ) no ) )

( ( line_intersection ( 10 11 ) (    1.35000   -7.40000 ) ) )
( ( line_intersection ( 11 10 ) yes ) )

( ( line_intersection ( 10 12 ) ( infinity infinity ) ) )
( ( line_intersection ( 12 10 ) no ) )

( ( line_intersection ( 11 12 ) (   -3.25000   -7.40000 ) ) )
( ( line_intersection ( 12 11 ) yes ) )


 )
; Number of points and lines
( <-  %numpts 16 )
( <-  %numlins 12 )
; Information about planes
(assert
```

```
( ( plane 1 ( 1 2 3 4 ) ( ) ) )
( ( plane 2 ( 5 6 7 8 9 10 11 12 ) ( ) ) )
( ( plane3d 1 ( unknown unknown unknown )
              (unknown unknown unknown) unknown))
( ( plane3d 2 ( unknown unknown unknown )
              (unknown unknown unknown) unknown))
)
; Number of planes
(<- %numplane 2)
```

B.3 IMAGE OF A RECTANGLE

In this section, we work with a synthetic image of a rectangle which lies in the z=constant plane, and show how knowledge about one of the coordinates of a point propagate to other points. The image of the rectangle is shown in Figure 38.

PREPROCESSED DATA

```
( assert
 ( ( line 1 ( 1 2 ) ( 1.0 1.0 0.0) ) )
 ( ( line 2 ( 2 3 ) (  0.0 1.0 2.0) ) )
 ( ( line 3 ( 3 4 ) ( 1.0 -1.0 0.0 ) ) )
 ( ( line 4 ( 1 4 ) (  0.0 1.0 1.0) ) )
)
(assert
    ((line_intersection (1 2) (-2.0 -2.0))        )
    ((line_intersection (1 3) (0.0 0.0))          )
    ((line_intersection (1 4) (-1.0 -1.0))        )
    ((line_intersection (2 3) (2.0 -2.0))         )
    ((line_intersection (2 4) (infinity infinity) ))
    ((line_intersection (3 4) (1.0 -1.0))         )
    ((line_intersection (2 1) yes))
    ((line_intersection (3 1) no))
    ((line_intersection (4 1) yes))
    ((line_intersection (3 2) yes))
    ((line_intersection (4 2) no))
    ((line_intersection (4 3) yes))
)
(assert
 ( ( line3d 1 ( unknown unknown )
              ( unknown unknown unknown ) ) )
 ( ( line3d 2 ( unknown unknown )
```

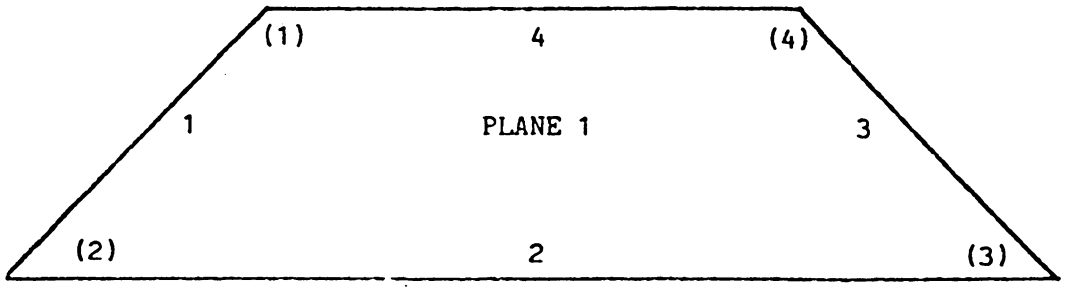


Figure 38: Synthetic image of a rectangle
The rectangle lies in the Z = -5 plane.
We know f and the z coordinate of the point marked 1.

```
                ( unknown unknown unknown ) ) )
 ( ( line3d 3 ( unknown unknown )
                ( unknown unknown unknown ) ) )
 ( ( line3d 4 ( unknown unknown )
                ( unknown unknown unknown ) ) )
)
(assert
  ( ( plane 1 ( 1 3 2 4) () ) )
  ( ( plane3d 1 ( unknown unknown unknown )
      ( unknown unknown unknown )
      unknown) )
)
(assert

    ((point 1 (-1.0 -1.0) ))
    ((point 2 (-2.0 -2.0) ))
    ((point 3 (2.0 -2.0) ))
    ((point 4 (1.0 -1.0) ))
    ((point3d 1 unknown unknown -5.0   ))
    ((point3d 2 unknown unknown unknown))
    ((point3d 3 unknown unknown unknown))
    ((point3d 4 unknown unknown unknown))
)
(assert ((focallength 1.0)) )
(<- %numplane     1 )
(<- %numlins      4 )
(<- %numpts       4 )
```

B.4 FOUNTAIN IMAGE

The data in this section corresponds to the digitized drawing of the scene shown in Figure 15 in Chapter V. It is part of the perspective projection of the base of a fountain.

PREPROCESSED DATA

```
( assert

( ( line 1 ( 1 2 ) (    0.00000    1.00000   20.13181 ) ) )
( ( line3d 1 ( unknown unknown ) ( unknown unknown unknown ) ) )

( ( line 2 ( 3 4 ) (    1.00000   -0.29765    0.22584 ) ) )
( ( line3d 2 ( unknown unknown ) ( unknown unknown unknown ) ) )
```

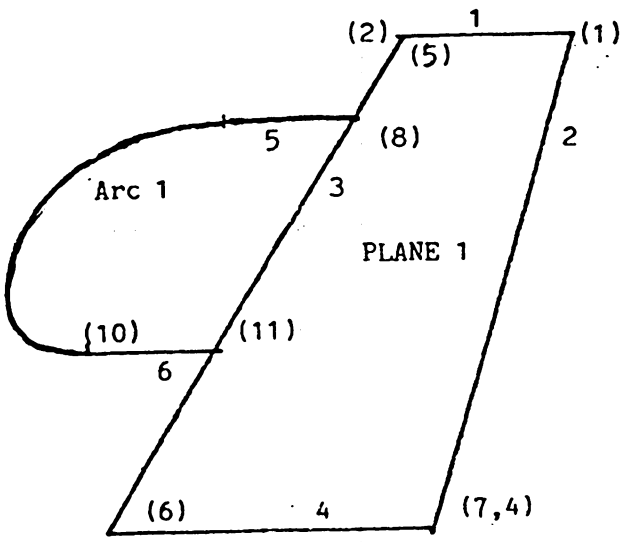


Figure 39: Digitized image of a fountain

```
( ( line 3 ( 5 6 ) (     1.00000   -0.61478   -0.10906 ) ) )
( ( line3d 3 ( unknown unknown ) ( unknown unknown unknown ) ) )

( ( line 4 ( 6 7 ) (     0.00000    1.00000   37.40454 ) ) )
( ( line3d 4 ( unknown unknown ) ( unknown unknown unknown ) ) )

( ( line 5 ( 8 9 ) (     0.00000    1.00000   22.90454 ) ) )
( ( line3d 5 ( unknown unknown ) ( unknown unknown unknown ) ) )

( ( line 6 ( 10 11 ) (     0.00000    1.00000   30.95000 ) ) )
( ( line3d 6 ( unknown unknown ) ( unknown unknown unknown ) ) )
( ( arc ( 9 10 ) ( 1.00000 -0.992731 4.686244
        14.31676 232.8405  3220.830 ) ) )

( ( point 1 (   -6.25000  -20.13181 ) ) )
( ( point3d 1 unknown unknown unknown ) )

( ( point 2 (  -12.47727  -20.13181 ) ) )
( ( point3d 2 unknown unknown unknown ) )

( ( point 3 (  -11.38637  -37.49545 ) ) )
( ( point3d 3 unknown unknown unknown ) )

( ( point 4 (   -6.20455  -20.08636 ) ) )
( ( point3d 4 unknown unknown unknown ) )

( ( point 5 (  -12.29545  -20.17727 ) ) )
( ( point3d 5 unknown unknown unknown ) )

( ( point 6 (  -22.88637  -37.40454 ) ) )
( ( point3d 6 unknown unknown unknown ) )

( ( point 7 (  -11.38637  -37.35909 ) ) )
( ( point3d 7 unknown unknown unknown ) )

( ( point 8 (  -13.97727  -22.90454 ) ) )
( ( point3d 8 unknown unknown unknown ) )

( ( point 9 (  -17.56818  -22.90454 ) ) )
( ( point3d 9 unknown unknown unknown ) )

( ( point 10 (  -23.52273  -30.95000 ) ) )
( ( point3d 10 unknown unknown unknown ) )

( ( point 11 (  -18.75000  -30.99545 ) ) )
( ( point3d 11 unknown unknown unknown ) )

( ( line_intersection ( 1 2 ) (   -6.21808  -20.13181 ) ) )
```

```
( ( line_intersection ( 2 1 ) no ) )

( ( line_intersection ( 1 3 ) (  -12.26751  -20.13181 ) ) )
( ( line_intersection ( 3 1 ) no ) )

( ( line_intersection ( 1 4 ) ( infinity infinity ) ) )
( ( line_intersection ( 4 1 ) no ) )

( ( line_intersection ( 1 5 ) ( infinity infinity ) ) )
( ( line_intersection ( 5 1 ) no ) )

( ( line_intersection ( 1 6 ) ( infinity infinity ) ) )
( ( line_intersection ( 6 1 ) no ) )

( ( line_intersection ( 2 3 ) (   -0.00504017   -0.005604 ) ) )
( ( line_intersection ( 3 2 ) no ) )

( ( line_intersection ( 2 4 ) (  -11.35931  -37.40454 ) ) )
( ( line_intersection ( 4 2 ) no ) )

( ( line_intersection ( 2 5 ) (   -7.04338  -22.90454 ) ) )
( ( line_intersection ( 5 2 ) no ) )

( ( line_intersection ( 2 6 ) (   -9.43812  -30.95000 ) ) )
( ( line_intersection ( 6 2 ) no ) )

( ( line_intersection ( 3 4 ) (  -22.88637  -37.40454 ) ) )
( ( line_intersection ( 4 3 ) yes ) )

( ( line_intersection ( 3 5 ) (  -13.97212  -22.90454 ) ) )
( ( line_intersection ( 5 3 ) no ) )

( ( line_intersection ( 3 6 ) (  -18.91827  -30.95000 ) ) )
( ( line_intersection ( 6 3 ) no ) )

( ( line_intersection ( 4 5 ) ( infinity infinity ) ) )
( ( line_intersection ( 5 4 ) no ) )

( ( line_intersection ( 4 6 ) ( infinity infinity ) ) )
( ( line_intersection ( 6 4 ) no ) )

( ( line_intersection ( 5 6 ) ( infinity infinity ) ) )
( ( line_intersection ( 6 5 ) no ) )


 )
( <-  %numpts 11 )
( <-  %numlins 6 )
( <-  %numarc 1 )
( <-  %numplane 1 )
```

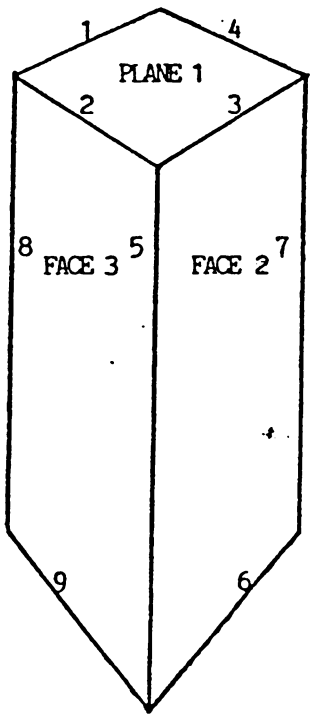
```
(assert
( ( plane 1 ( 1 2 3 4 5 6 ) ( 1 ) ) )
( ( plane3d 1 ( unknown unknown unknown )
              ( unknown unknown unknown )
                unknown ) )
)
```

B.5 IMAGE OF A RECTANGULAR PARALLELIPIPED

This data set consists of the digitized image of a rectangular parallelipiped. The image of the object is shown in Figure 40. It consists of a parallelipiped with its top face parallel to the XY plane located below the origin. This image was used to perform an interesting experiment. As the image shows, three faces of the solid are visible. These faces are numbered 1 through 3. The first experiment that was run on this image used the data set shown below. In the second experiment, the reasoning system was given erroneous information on purpose to see what its response would be. The system was told that the faces 2 and 3 were actually in the same plane. The reader is urged to match his wits against that of the system to try and figure out a three-dimensional configuration of entities which would produce the given image and yet satisfy the specified constraint. The result produced by the system for these two cases is shown in Appendix C. In the case with the erroneous input, the result is indeed unexpected (although it is correct).

PREPROCESSED DATA

```
( assert
```



Note: In one experiment, FACE 2 and FACE 3 were declared to lie in the same plane. In another experiment they were on independent planes. Both experiments are reported in the text.

Figure 40: Image of a Rectangular Parallelipiped

In one test, the faces marked 2 and 3 were forced to lie in the same plane.

```
( ( line3d 1 ( unknown unknown )
    ( unknown unknown unknown ) ) )

( ( line 2 ( 2 3 )
    (    0.42857    1.00000    1.28571 ) ) )
( ( line3d 2 ( unknown unknown )
    ( unknown unknown unknown ) ) )

( ( line 3 ( 3 4 )
    (   -0.60000    1.00000    1.80000 ) ) )
( ( line3d 3 ( unknown unknown )
    ( unknown unknown unknown ) ) )

( ( line 4 ( 4 1 )
    (    0.60000    1.00000    1.80000 ) ) )
( ( line3d 4 ( unknown unknown )
    ( unknown unknown unknown ) ) )

( ( line 5 ( 4 5 )
    (    1.00000    0.00000    0.00000 ) ) )
( ( line3d 5 ( unknown unknown )
    ( unknown unknown unknown ) ) )

( ( line 6 ( 5 6 )
    (    1.00000   -0.83333   -3.00000 ) ) )
( ( line3d 6 ( unknown unknown )
    ( unknown unknown unknown ) ) )

( ( line 7 ( 6 3 )
    (    1.00000    0.00000   -0.50000 ) ) )
( ( line3d 7 ( unknown unknown )
    ( unknown unknown unknown ) ) )

( ( line 8 ( 1 7 )
    (    1.00000    0.00000    0.50000 ) ) )
( ( line3d 8 ( unknown unknown )
    ( unknown unknown unknown ) ) )

( ( line 9 ( 7 5 )
    (    1.00000    0.83333    3.00000 ) ) )
( ( line3d 9 ( unknown unknown )
    ( unknown unknown unknown ) ) )

( ( point 1 (   -0.50000   -1.50000 ) ) )
( ( point3d 1 unknown unknown unknown ) )

( ( point 2 (    0.00000   -1.28571 ) ) )
( ( point3d 2 unknown unknown unknown ) )

( ( point 3 (    0.50000   -1.50000 ) ) )
```

```
( ( point3d 3 unknown unknown unknown ) )

( ( point 4 (     0.00000   -1.80000 ) ) )
( ( point3d 4 unknown unknown unknown ) )

( ( point 5 (     0.00000   -3.60000 ) ) )
( ( point3d 5 unknown unknown unknown ) )

( ( point 6 (     0.50000   -3.00000 ) ) )
( ( point3d 6 unknown unknown unknown ) )

( ( point 7 (    -0.50000   -3.00000 ) ) )
( ( point3d 7 unknown unknown unknown ) )

( ( line_intersection ( 1 2 )
    (     0.00000   -1.28571 ) ) )
( ( line_intersection ( 2 1 ) yes ) )

( ( line_intersection ( 1 3 )
    (     3.00001 0.0000069539 ) ) )
( ( line_intersection ( 3 1 ) no ) )

( ( line_intersection ( 1 4 )
    (    -0.50000   -1.50000 ) ) )
( ( line_intersection ( 4 1 ) yes ) )

( ( line_intersection ( 1 5 )
    (     0.00000   -1.28571 ) ) )
( ( line_intersection ( 5 1 ) no ) )

( ( line_intersection ( 1 6 )
    (     3.00000 0.0000031524 ) ) )
( ( line_intersection ( 6 1 ) no ) )

( ( line_intersection ( 1 7 )
    (     0.50000   -1.07143 ) ) )
( ( line_intersection ( 7 1 ) no ) )

( ( line_intersection ( 1 8 )
    (    -0.50000   -1.50000 ) ) )
( ( line_intersection ( 8 1 ) yes ) )

( ( line_intersection ( 1 9 )
    (    -1.42105   -1.89474 ) ) )
( ( line_intersection ( 9 1 ) no ) )

( ( line_intersection ( 2 3 )
    (     0.50000   -1.50000 ) ) )
( ( line_intersection ( 3 2 ) yes ) )
```

```
( ( line_intersection ( 2 4 )
    (    -3.00001 0.0000069539 ) ) )
( ( line_intersection ( 4 2 ) no ) )

( ( line_intersection ( 2 5 )
    (     0.00000   -1.28571 ) ) )
( ( line_intersection ( 5 2 ) no ) )

( ( line_intersection ( 2 6 )
    (     1.42105   -1.89474 ) ) )
( ( line_intersection ( 6 2 ) no ) )

( ( line_intersection ( 2 7 )
    (     0.50000   -1.50000 ) ) )
( ( line_intersection ( 7 2 ) yes ) )

( ( line_intersection ( 2 8 )
    (    -0.50000   -1.07143 ) ) )
( ( line_intersection ( 8 2 ) no ) )

( ( line_intersection ( 2 9 )
    (    -3.00000 0.0000031524 ) ) )
( ( line_intersection ( 9 2 ) no ) )

( ( line_intersection ( 3 4 )
    (     0.00000   -1.80000 ) ) )
( ( line_intersection ( 4 3 ) yes ) )

( ( line_intersection ( 3 5 )
    (     0.00000   -1.80000 ) ) )
( ( line_intersection ( 5 3 ) yes ) )

( ( line_intersection ( 3 6 )
    (     3.00000    0.00000 ) ) )
( ( line_intersection ( 6 3 ) no ) )

( ( line_intersection ( 3 7 )
    (     0.50000   -1.50000 ) ) )
( ( line_intersection ( 7 3 ) yes ) )

( ( line_intersection ( 3 8 )
    (    -0.50000   -2.10000 ) ) )
( ( line_intersection ( 8 3 ) no ) )

( ( line_intersection ( 3 9 )
    (    -1.00000   -2.40000 ) ) )
( ( line_intersection ( 9 3 ) no ) )

( ( line_intersection ( 4 5 )
    (     0.00000   -1.80000 ) ) )
```

```
( ( line_intersection ( 5 4 ) yes ) )

( ( line_intersection ( 4 6 )
    (     1.00000    -2.40000 ) ) )
( ( line_intersection ( 6 4 ) no ) )

( ( line_intersection ( 4 7 )
    (     0.50000    -2.10000 ) ) )
( ( line_intersection ( 7 4 ) no ) )

( ( line_intersection ( 4 8 )
    (    -0.50000    -1.50000 ) ) )
( ( line_intersection ( 8 4 ) yes ) )

( ( line_intersection ( 4 9 )
    (    -3.00000     0.00000 ) ) )
( ( line_intersection ( 9 4 ) no ) )

( ( line_intersection ( 5 6 )
    (     0.00000    -3.60000 ) ) )
( ( line_intersection ( 6 5 ) yes ) )

( ( line_intersection ( 5 7 )
    ( infinity infinity ) ) )
( ( line_intersection ( 7 5 ) no ) )

( ( line_intersection ( 5 8 )
    ( infinity infinity ) ) )
( ( line_intersection ( 8 5 ) no ) )

( ( line_intersection ( 5 9 )
    (     0.00000    -3.60000 ) ) )
( ( line_intersection ( 9 5 ) yes ) )

( ( line_intersection ( 6 7 )
    (     0.50000    -3.00000 ) ) )
( ( line_intersection ( 7 6 ) yes ) )

( ( line_intersection ( 6 8 )
    (    -0.50000    -4.20000 ) ) )
( ( line_intersection ( 8 6 ) no ) )

( ( line_intersection ( 6 9 )
    (     0.00000    -3.60000 ) ) )
( ( line_intersection ( 9 6 ) yes ) )

( ( line_intersection ( 7 8 )
    ( infinity infinity ) ) )
( ( line_intersection ( 8 7 ) no ) )
```

```
( ( line_intersection ( 7 9 )
    (    0.50000   -4.20000 ) ) )
( ( line_intersection ( 9 7 ) no ) )

( ( line_intersection ( 8 9 )
    (   -0.50000   -3.00000 ) ) )
( ( line_intersection ( 9 8 ) yes ) )


 )
( <-  %numpts 7 )
( <-  %numlins 9 )
(assert
((plane 1 (1 2 3 4) () ))
((plane 2 (3 5 6 7) () ))
((plane 3 (2 5 8 9) () ))
((plane3d 1 (unknown unknown unknown)
    (unknown unknown unknown)
    unknown))
((plane3d 2 (unknown unknown unknown)
    (unknown unknown unknown)
    unknown))
((plane3d 3 (unknown unknown unknown)
    (unknown unknown unknown)
    unknown))
)
(<- %numplane 3)
```

B.6 COMPLEX IMAGE OF THE TABLE SCENE

In addition to the image of the table shown earlier, another data set was created with more lines than before. The digitized line drawing is shown in Figure 41. The actual preprocessed data set is not included in this dissertation for space. However, the dataset can be recreated directly from the result shown in the Appendix C.

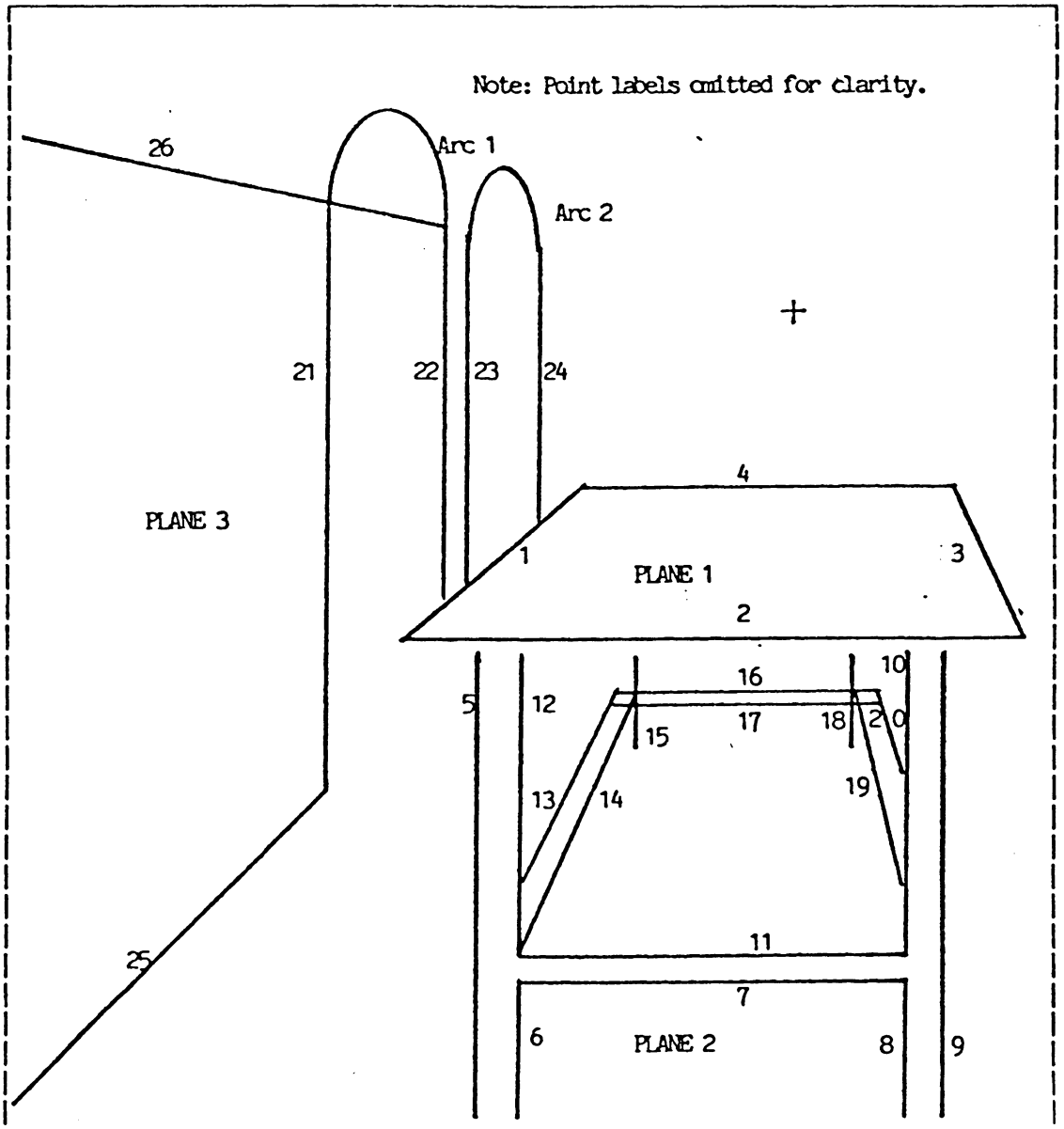
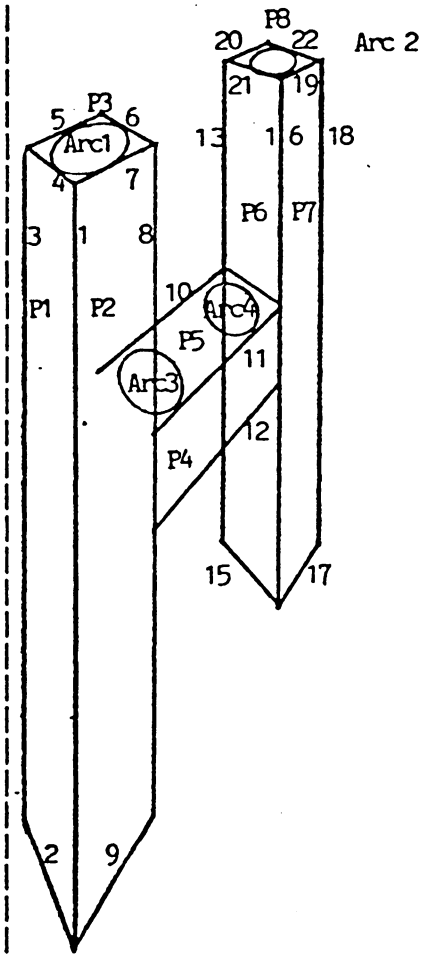


Figure 41: Complicated line drawing of the table scene

B.7 TOWERS IMAGE

An artificial image was constructed to show the system working with arcs. The image is shown in Figure 42. Again, to conserve space, the actual preprocessed data is not shown. It too can be determined from the results shown in Appendix C.



Point labels are not indicated for legibility. They can be determined from the printout of the dataset. Plane identifications shown as P#.

Figure 42: Artificial image of two towers

Appendix C

RESULTS FOR THE SAMPLE DATA

In this appendix, we show the results obtained by the search algorithm for the test data sets shown in Appendix B.

C.1 RESULTS FOR THE ARCHITECTURAL SCENE

In this section, we list the results of the search procedure for finding the best solution to the architectural scene shown in Figure 1, Chapter V. The listing is annotated, with all comments indicated by a semicolon (;) in column 1. To interpret the listing, note the following:

1. Whenever predictions are generated, they are added to a queue of tuples to be added to the hypothesis at that level. They are then checked to make sure that they will not violate logical correctness of the hypothesis. The action of adding tuples to the queue is denoted as "ADD TO FIFO (...)" or as "ADD TO FIFO WITHOUT EXTEND (...)"
2. Tuples added "WITHOUT EXTEND" are not used to extend new hypotheses using heuristic techniques.
3. "PERFORMING EXTEND" marks the spot where the vanishing point heuristic is used to extend the hypothesis further.

4. When tuples are selected from the queue and actually made part of the hypothesis, the listing indicates it as "HYPOTHESIZE (...)".
5. Whenever an inference engine is applied to a hypothesis, it records the fact as "APPLY <engine name> <input tuple set>".
6. Whenever logical consistency of a hypothesis is checked, it is recorded as "E<n> <input conditions>".
7. If any consistency check or logical check fails, the fact is recorded giving the reason why the check fails.
8. Each level of the tree is marked as "... save <level>" when the level is encountered on the way down the tree, or as "... restore <level>" if it is encountered on the way up.
9. To save space, if no new information is added for several levels in a row, those levels are not shown in the trace.

```
        WELCOME to the NEW PROLOG
          4-AUG-1984 16:36:43.41

loading file HYP.PRO
Finished loading HYP.PRO
loading file CHUNKER.PRO
Finished loading CHUNKER.PRO
Loading file COMPUTE.PRO
finished loading COMPUTE.PRO
loading test2p and test2pl
;
;                    Turn on the trace flag
;
(<- %traceflag on)
:- ( ( <- on on ) )
;                    Print best solution found
(<- %mode best)
:- ( ( <- best best ) )
;                    Use non recursive search
(<- %bttsflag new)
:- ( ( <- new new ) )
;                   Start the search
(search)
;                   The list of possible spatial
;                   relational tuple (P'). All
;                   hypotheses are subsets of this list
Possible relational tuples
[ 1 ] ( ( parallel 1 2 ) )
[ 2 ] ( ( perpendicular 1 2 ) )
[ 3 ] ( ( parallel 1 3 ) )
[ 4 ] ( ( perpendicular 1 3 ) )
[ 5 ] ( ( parallel 1 4 ) )
[ 6 ] ( ( perpendicular 1 4 ) )
[ 7 ] ( ( parallel 1 5 ) )
[ 8 ] ( ( perpendicular 1 5 ) )
[ 9 ] ( ( parallel 1 6 ) )
[ 10 ] ( ( perpendicular 1 6 ) )
[ 11 ] ( ( parallel 1 7 ) )
[ 12 ] ( ( perpendicular 1 7 ) )
[ 13 ] ( ( parallel 1 8 ) )
[ 14 ] ( ( perpendicular 1 8 ) )
[ 15 ] ( ( parallel 1 9 ) )
[ 16 ] ( ( perpendicular 1 9 ) )
[ 17 ] ( ( parallel 1 10 ) )
[ 18 ] ( ( perpendicular 1 10 ) )
[ 19 ] ( ( parallel 1 11 ) )
[ 20 ] ( ( perpendicular 1 11 ) )
[ 21 ] ( ( parallel 1 12 ) )
[ 22 ] ( ( perpendicular 1 12 ) )
[ 23 ] ( ( parallel 1 13 ) )
```

```
[ 24 ] ( ( perpendicular 1 13 ) )
[ 25 ] ( ( parallel 1 14 ) )
[ 26 ] ( ( perpendicular 1 14 ) )
[ 27 ] ( ( parallel 1 15 ) )
[ 28 ] ( ( perpendicular 1 15 ) )
[ 29 ] ( ( parallel 1 16 ) )
[ 30 ] ( ( perpendicular 1 16 ) )
[ 31 ] ( ( parallel 1 17 ) )
[ 32 ] ( ( perpendicular 1 17 ) )
[ 33 ] ( ( parallel 1 18 ) )
[ 34 ] ( ( perpendicular 1 18 ) )
[ 35 ] ( ( parallel 2 3 ) )
[ 36 ] ( ( perpendicular 2 3 ) )
[ 37 ] ( ( parallel 2 4 ) )
[ 38 ] ( ( perpendicular 2 4 ) )
[ 39 ] ( ( parallel 2 5 ) )
[ 40 ] ( ( perpendicular 2 5 ) )
[ 41 ] ( ( parallel 2 6 ) )
[ 42 ] ( ( perpendicular 2 6 ) )
[ 43 ] ( ( parallel 2 7 ) )
[ 44 ] ( ( perpendicular 2 7 ) )
[ 45 ] ( ( parallel 2 8 ) )
[ 46 ] ( ( perpendicular 2 8 ) )
[ 47 ] ( ( parallel 2 9 ) )
[ 48 ] ( ( perpendicular 2 9 ) )
[ 49 ] ( ( parallel 2 10 ) )
[ 50 ] ( ( perpendicular 2 10 ) )
[ 51 ] ( ( parallel 2 11 ) )
[ 52 ] ( ( perpendicular 2 11 ) )
[ 53 ] ( ( parallel 2 12 ) )
[ 54 ] ( ( perpendicular 2 12 ) )
[ 55 ] ( ( parallel 2 13 ) )
[ 56 ] ( ( perpendicular 2 13 ) )
[ 57 ] ( ( parallel 2 14 ) )
[ 58 ] ( ( perpendicular 2 14 ) )
[ 59 ] ( ( parallel 2 15 ) )
[ 60 ] ( ( perpendicular 2 15 ) )
[ 61 ] ( ( parallel 2 16 ) )
[ 62 ] ( ( perpendicular 2 16 ) )
[ 63 ] ( ( parallel 2 17 ) )
[ 64 ] ( ( perpendicular 2 17 ) )
[ 65 ] ( ( parallel 2 18 ) )
[ 66 ] ( ( perpendicular 2 18 ) )
[ 67 ] ( ( parallel 3 4 ) )
[ 68 ] ( ( perpendicular 3 4 ) )
[ 69 ] ( ( parallel 3 5 ) )
[ 70 ] ( ( perpendicular 3 5 ) )
[ 71 ] ( ( parallel 3 6 ) )
[ 72 ] ( ( perpendicular 3 6 ) )
[ 73 ] ( ( parallel 3 7 ) )
```

```
[  74 ] (  ( perpendicular 3 7 ) )
[  75 ] (  ( parallel 3 8 ) )
[  76 ] (  ( perpendicular 3 8 ) )
[  77 ] (  ( parallel 3 9 ) )
[  78 ] (  ( perpendicular 3 9 ) )
[  79 ] (  ( parallel 3 10 ) )
[  80 ] (  ( perpendicular 3 10 ) )
[  81 ] (  ( parallel 3 11 ) )
[  82 ] (  ( perpendicular 3 11 ) )
[  83 ] (  ( parallel 3 12 ) )
[  84 ] (  ( perpendicular 3 12 ) )
[  85 ] (  ( parallel 3 13 ) )
[  86 ] (  ( perpendicular 3 13 ) )
[  87 ] (  ( parallel 3 14 ) )
[  88 ] (  ( perpendicular 3 14 ) )
[  89 ] (  ( parallel 3 15 ) )
[  90 ] (  ( perpendicular 3 15 ) )
[  91 ] (  ( parallel 3 16 ) )
[  92 ] (  ( perpendicular 3 16 ) )
[  93 ] (  ( parallel 3 17 ) )
[  94 ] (  ( perpendicular 3 17 ) )
[  95 ] (  ( parallel 3 18 ) )
[  96 ] (  ( perpendicular 3 18 ) )
[  97 ] (  ( parallel 4 5 ) )
[  98 ] (  ( perpendicular 4 5 ) )
[  99 ] (  ( parallel 4 6 ) )
[  100 ] (  ( perpendicular 4 6 ) )
[  101 ] (  ( parallel 4 7 ) )
[  102 ] (  ( perpendicular 4 7 ) )
[  103 ] (  ( parallel 4 8 ) )
[  104 ] (  ( perpendicular 4 8 ) )
[  105 ] (  ( parallel 4 9 ) )
[  106 ] (  ( perpendicular 4 9 ) )
[  107 ] (  ( parallel 4 10 ) )
[  108 ] (  ( perpendicular 4 10 ) )
[  109 ] (  ( parallel 4 11 ) )
[  110 ] (  ( perpendicular 4 11 ) )
[  111 ] (  ( parallel 4 12 ) )
[  112 ] (  ( perpendicular 4 12 ) )
[  113 ] (  ( parallel 4 13 ) )
[  114 ] (  ( perpendicular 4 13 ) )
[  115 ] (  ( parallel 4 14 ) )
[  116 ] (  ( perpendicular 4 14 ) )
[  117 ] (  ( parallel 4 15 ) )
[  118 ] (  ( perpendicular 4 15 ) )
[  119 ] (  ( parallel 4 16 ) )
[  120 ] (  ( perpendicular 4 16 ) )
[  121 ] (  ( parallel 4 17 ) )
[  122 ] (  ( perpendicular 4 17 ) )
[  123 ] (  ( parallel 4 18 ) )
```

```
[ 124 ] ( ( perpendicular 4 18 ) )
[ 125 ] ( ( parallel 5 6 ) )
[ 126 ] ( ( perpendicular 5 6 ) )
[ 127 ] ( ( parallel 5 7 ) )
[ 128 ] ( ( perpendicular 5 7 ) )
[ 129 ] ( ( parallel 5 8 ) )
[ 130 ] ( ( perpendicular 5 8 ) )
[ 131 ] ( ( parallel 5 9 ) )
[ 132 ] ( ( perpendicular 5 9 ) )
[ 133 ] ( ( parallel 5 10 ) )
[ 134 ] ( ( perpendicular 5 10 ) )
[ 135 ] ( ( parallel 5 11 ) )
[ 136 ] ( ( perpendicular 5 11 ) )
[ 137 ] ( ( parallel 5 12 ) )
[ 138 ] ( ( perpendicular 5 12 ) )
[ 139 ] ( ( parallel 5 13 ) )
[ 140 ] ( ( perpendicular 5 13 ) )
[ 141 ] ( ( parallel 5 14 ) )
[ 142 ] ( ( perpendicular 5 14 ) )
[ 143 ] ( ( parallel 5 15 ) )
[ 144 ] ( ( perpendicular 5 15 ) )
[ 145 ] ( ( parallel 5 16 ) )
[ 146 ] ( ( perpendicular 5 16 ) )
[ 147 ] ( ( parallel 5 17 ) )
[ 148 ] ( ( perpendicular 5 17 ) )
[ 149 ] ( ( parallel 5 18 ) )
[ 150 ] ( ( perpendicular 5 18 ) )
[ 151 ] ( ( parallel 6 7 ) )
[ 152 ] ( ( perpendicular 6 7 ) )
[ 153 ] ( ( parallel 6 8 ) )
[ 154 ] ( ( perpendicular 6 8 ) )
[ 155 ] ( ( parallel 6 9 ) )
[ 156 ] ( ( perpendicular 6 9 ) )
[ 157 ] ( ( parallel 6 10 ) )
[ 158 ] ( ( perpendicular 6 10 ) )
[ 159 ] ( ( parallel 6 11 ) )
[ 160 ] ( ( perpendicular 6 11 ) )
[ 161 ] ( ( parallel 6 12 ) )
[ 162 ] ( ( perpendicular 6 12 ) )
[ 163 ] ( ( parallel 6 13 ) )
[ 164 ] ( ( perpendicular 6 13 ) )
[ 165 ] ( ( parallel 6 14 ) )
[ 166 ] ( ( perpendicular 6 14 ) )
[ 167 ] ( ( parallel 6 15 ) )
[ 168 ] ( ( perpendicular 6 15 ) )
[ 169 ] ( ( parallel 6 16 ) )
[ 170 ] ( ( perpendicular 6 16 ) )
[ 171 ] ( ( parallel 6 17 ) )
[ 172 ] ( ( perpendicular 6 17 ) )
[ 173 ] ( ( parallel 6 18 ) )
```

[174] ((perpendicular 6 18))
[175] ((parallel 7 8))
[176] ((perpendicular 7 8))
[177] ((parallel 7 9))
[178] ((perpendicular 7 9))
[179] ((parallel 7 10))
[180] ((perpendicular 7 10))
[181] ((parallel 7 11))
[182] ((perpendicular 7 11))
[183] ((parallel 7 12))
[184] ((perpendicular 7 12))
[185] ((parallel 7 13))
[186] ((perpendicular 7 13))
[187] ((parallel 7 14))
[188] ((perpendicular 7 14))
[189] ((parallel 7 15))
[190] ((perpendicular 7 15))
[191] ((parallel 7 16))
[192] ((perpendicular 7 16))
[193] ((parallel 7 17))
[194] ((perpendicular 7 17))
[195] ((parallel 7 18))
[196] ((perpendicular 7 18))
[197] ((parallel 8 9))
[198] ((perpendicular 8 9))
[199] ((parallel 8 10))
[200] ((perpendicular 8 10))
[201] ((parallel 8 11))
[202] ((perpendicular 8 11))
[203] ((parallel 8 12))
[204] ((perpendicular 8 12))
[205] ((parallel 8 13))
[206] ((perpendicular 8 13))
[207] ((parallel 8 14))
[208] ((perpendicular 8 14))
[209] ((parallel 8 15))
[210] ((perpendicular 8 15))
[211] ((parallel 8 16))
[212] ((perpendicular 8 16))
[213] ((parallel 8 17))
[214] ((perpendicular 8 17))
[215] ((parallel 8 18))
[216] ((perpendicular 8 18))
[217] ((parallel 9 10))
[218] ((perpendicular 9 10))
[219] ((parallel 9 11))
[220] ((perpendicular 9 11))
[221] ((parallel 9 12))
[222] ((perpendicular 9 12))
[223] ((parallel 9 13))

```
[ 224 ] ( ( perpendicular 9 13 ) )
[ 225 ] ( ( parallel 9 14 ) )
[ 226 ] ( ( perpendicular 9 14 ) )
[ 227 ] ( ( parallel 9 15 ) )
[ 228 ] ( ( perpendicular 9 15 ) )
[ 229 ] ( ( parallel 9 16 ) )
[ 230 ] ( ( perpendicular 9 16 ) )
[ 231 ] ( ( parallel 9 17 ) )
[ 232 ] ( ( perpendicular 9 17 ) )
[ 233 ] ( ( parallel 9 18 ) )
[ 234 ] ( ( perpendicular 9 18 ) )
[ 235 ] ( ( parallel 10 11 ) )
[ 236 ] ( ( perpendicular 10 11 ) )
[ 237 ] ( ( parallel 10 12 ) )
[ 238 ] ( ( perpendicular 10 12 ) )
[ 239 ] ( ( parallel 10 13 ) )
[ 240 ] ( ( perpendicular 10 13 ) )
[ 241 ] ( ( parallel 10 14 ) )
[ 242 ] ( ( perpendicular 10 14 ) )
[ 243 ] ( ( parallel 10 15 ) )
[ 244 ] ( ( perpendicular 10 15 ) )
[ 245 ] ( ( parallel 10 16 ) )
[ 246 ] ( ( perpendicular 10 16 ) )
[ 247 ] ( ( parallel 10 17 ) )
[ 248 ] ( ( perpendicular 10 17 ) )
[ 249 ] ( ( parallel 10 18 ) )
[ 250 ] ( ( perpendicular 10 18 ) )
[ 251 ] ( ( parallel 11 12 ) )
[ 252 ] ( ( perpendicular 11 12 ) )
[ 253 ] ( ( parallel 11 13 ) )
[ 254 ] ( ( perpendicular 11 13 ) )
[ 255 ] ( ( parallel 11 14 ) )
[ 256 ] ( ( perpendicular 11 14 ) )
[ 257 ] ( ( parallel 11 15 ) )
[ 258 ] ( ( perpendicular 11 15 ) )
[ 259 ] ( ( parallel 11 16 ) )
[ 260 ] ( ( perpendicular 11 16 ) )
[ 261 ] ( ( parallel 11 17 ) )
[ 262 ] ( ( perpendicular 11 17 ) )
[ 263 ] ( ( parallel 11 18 ) )
[ 264 ] ( ( perpendicular 11 18 ) )
[ 265 ] ( ( parallel 12 13 ) )
[ 266 ] ( ( perpendicular 12 13 ) )
[ 267 ] ( ( parallel 12 14 ) )
[ 268 ] ( ( perpendicular 12 14 ) )
[ 269 ] ( ( parallel 12 15 ) )
[ 270 ] ( ( perpendicular 12 15 ) )
[ 271 ] ( ( parallel 12 16 ) )
[ 272 ] ( ( perpendicular 12 16 ) )
[ 273 ] ( ( parallel 12 17 ) )
```

```
[ 274 ] ( ( perpendicular 12 17 ) )
[ 275 ] ( ( parallel 12 18 ) )
[ 276 ] ( ( perpendicular 12 18 ) )
[ 277 ] ( ( parallel 13 14 ) )
[ 278 ] ( ( perpendicular 13 14 ) )
[ 279 ] ( ( parallel 13 15 ) )
[ 280 ] ( ( perpendicular 13 15 ) )
[ 281 ] ( ( parallel 13 16 ) )
[ 282 ] ( ( perpendicular 13 16 ) )
[ 283 ] ( ( parallel 13 17 ) )
[ 284 ] ( ( perpendicular 13 17 ) )
[ 285 ] ( ( parallel 13 18 ) )
[ 286 ] ( ( perpendicular 13 18 ) )
[ 287 ] ( ( parallel 14 15 ) )
[ 288 ] ( ( perpendicular 14 15 ) )
[ 289 ] ( ( parallel 14 16 ) )
[ 290 ] ( ( perpendicular 14 16 ) )
[ 291 ] ( ( parallel 14 17 ) )
[ 292 ] ( ( perpendicular 14 17 ) )
[ 293 ] ( ( parallel 14 18 ) )
[ 294 ] ( ( perpendicular 14 18 ) )
[ 295 ] ( ( parallel 15 16 ) )
[ 296 ] ( ( perpendicular 15 16 ) )
[ 297 ] ( ( parallel 15 17 ) )
[ 298 ] ( ( perpendicular 15 17 ) )
[ 299 ] ( ( parallel 15 18 ) )
[ 300 ] ( ( perpendicular 15 18 ) )
[ 301 ] ( ( parallel 16 17 ) )
[ 302 ] ( ( perpendicular 16 17 ) )
[ 303 ] ( ( parallel 16 18 ) )
[ 304 ] ( ( perpendicular 16 18 ) )
[ 305 ] ( ( parallel 17 18 ) )
[ 306 ] ( ( perpendicular 17 18 ) )
;                          Total of 306 possible tuples.
;                          The search space is 2**306
```

```
forward 1 1
 ..............  save  1
Add to FIFO ( ( parallel 1 2 ) )
HYPOTHESIS ( ( parallel 1 2 ) )
Intersecting parallel lines 1 2
 ..................  restorestate 1
 ..............  save  1
forward 2 2
 ..............  save  2
Add to FIFO ( ( perpendicular 1 2 ) )
HYPOTHESIS ( ( perpendicular 1 2 ) )
forward 3 3
 ..............  save  3
Add to FIFO ( ( parallel 1 3 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 1 5 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 3 5 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 1 7 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 3 7 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 1 8 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 3 8 ) )
E10 parallel 1 3 and perp . 1 . 2 => perpendicular 2 3
Add to FIFO ( ( perpendicular 2 3 ) )
E10 parallel 1 5 and perp . 1 . 2 => perpendicular 2 5
Add to FIFO ( ( perpendicular 2 5 ) )
E10 parallel 1 7 and perp . 1 . 2 => perpendicular 2 7
Add to FIFO ( ( perpendicular 2 7 ) )
E10 parallel 1 8 and perp . 1 . 2 => perpendicular 2 8
Add to FIFO ( ( perpendicular 2 8 ) )
HYPOTHESIS ( ( parallel 1 3 ) )
HYPOTHESIS ( ( parallel 1 5 ) )
HYPOTHESIS ( ( parallel 3 5 ) )
HYPOTHESIS ( ( parallel 1 7 ) )
HYPOTHESIS ( ( parallel 3 7 ) )
HYPOTHESIS ( ( parallel 1 8 ) )
HYPOTHESIS ( ( parallel 3 8 ) )
HYPOTHESIS ( ( perpendicular 2 3 ) )
HYPOTHESIS ( ( perpendicular 2 5 ) )
HYPOTHESIS ( ( perpendicular 2 7 ) )
HYPOTHESIS ( ( perpendicular 2 8 ) )
Apply c1 ( parallel 1 3 )
Apply c1 ( parallel 1 5 )
Apply c1 ( parallel 3 5 )
Apply c1 ( parallel 1 7 )
Apply c1 ( parallel 3 7 )
```

```
Apply c1 ( parallel 1 8 )
Apply c1 ( parallel 3 8 )
APPLY c17 ( line3d 1 )
APPLY c17 ( line3d 3 )
APPLY c17 ( line3d 5 )
APPLY c17 ( line3d 7 )
APPLY c17 ( line3d 8 )
forward 4 4
 ..............  save  4
forward 5 5
 ..............  save  5
forward 6 6
 ..............  save  6
Add to FIFO ( ( perpendicular 1 4 ) )
E2 perpendicular 1 4 and 1 2 => parallel 2 4
Add to FIFO ( ( parallel 2 4 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 2 6 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 4 6 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 2 9 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 4 9 ) )
E14 parallel 1 3 and perp 1 4 => perp 3 4
Add to FIFO ( ( perpendicular 3 4 ) )
E14 parallel 1 5 and perp 1 4 => perp 4 5
Add to FIFO ( ( perpendicular 4 5 ) )
E14 parallel 1 7 and perp 1 4 => perp 4 7
Add to FIFO ( ( perpendicular 4 7 ) )
E14 parallel 1 8 and perp 1 4 => perp 4 8
Add to FIFO ( ( perpendicular 4 8 ) )
HYPOTHESIS ( ( perpendicular 1 4 ) )
HYPOTHESIS ( ( parallel 2 4 ) )
HYPOTHESIS ( ( parallel 2 6 ) )
HYPOTHESIS ( ( parallel 4 6 ) )
HYPOTHESIS ( ( parallel 2 9 ) )
HYPOTHESIS ( ( parallel 4 9 ) )
HYPOTHESIS ( ( perpendicular 3 4 ) )
HYPOTHESIS ( ( perpendicular 4 5 ) )
HYPOTHESIS ( ( perpendicular 4 7 ) )
HYPOTHESIS ( ( perpendicular 4 8 ) )
Apply c1 ( parallel 2 4 )
Apply c1 ( parallel 2 6 )
Apply c1 ( parallel 4 6 )
Apply c1 ( parallel 2 9 )
Apply c1 ( parallel 4 9 )
APPLY c17 ( line3d 2 )
APPLY c17 ( line3d 4 )
APPLY c17 ( line3d 6 )
```

```
APPLY c17 ( line3d 9 )
Apply c30 ( plane 1 ) ( line3d 1 ) ( line3d 2 )
forward 7 7
 ..............  save  7
forward 8 8
 ..............  save  8
forward 9 9
 ..............  save  9
forward 10 10
 ..............  save  10
Add to FIFO ( ( perpendicular 1 6 ) )
E2 perpendicular 1 6 and 1 2 => parallel 2 6
E2 perpendicular 1 6 and 1 4 => parallel 4 6
E14 parallel 1 3 and perp 1 6 => perp 3 6
Add to FIFO ( ( perpendicular 3 6 ) )
E14 parallel 1 5 and perp 1 6 => perp 5 6
Add to FIFO ( ( perpendicular 5 6 ) )
E14 parallel 1 7 and perp 1 6 => perp 6 7
Add to FIFO ( ( perpendicular 6 7 ) )
E14 parallel 1 8 and perp 1 6 => perp 6 8
Add to FIFO ( ( perpendicular 6 8 ) )
E16 parallel 2 6 and perp 1 6 => perp 1 2
E16 parallel 4 6 and perp 1 6 => perp 1 4
HYPOTHESIS ( ( perpendicular 1 6 ) )
HYPOTHESIS ( ( perpendicular 3 6 ) )
HYPOTHESIS ( ( perpendicular 5 6 ) )
HYPOTHESIS ( ( perpendicular 6 7 ) )
HYPOTHESIS ( ( perpendicular 6 8 ) )
forward 11 11
 ..............  save  11
forward 12 12
 ..............  save  12
forward 13 13
 ..............  save  13
forward 14 14
 ..............  save  14
forward 15 15
 ..............  save  15
forward 16 16
 ..............  save  16
Add to FIFO ( ( perpendicular 1 9 ) )
E2 perpendicular 1 9 and 1 2 => parallel 2 9
E2 perpendicular 1 9 and 1 4 => parallel 4 9
E2 perpendicular 1 9 and 1 6 => parallel 6 9
Add to FIFO ( ( parallel 6 9 ) )
E14 parallel 1 3 and perp 1 9 => perp 3 9
Add to FIFO ( ( perpendicular 3 9 ) )
E14 parallel 1 5 and perp 1 9 => perp 5 9
Add to FIFO ( ( perpendicular 5 9 ) )
E14 parallel 1 7 and perp 1 9 => perp 7 9
```

```
Add to FIFO ( ( perpendicular 7 9 ) )
E14 parallel 1 8 and perp 1 9 => perp 8 9
Add to FIFO ( ( perpendicular 8 9 ) )
E16 parallel 2 9 and perp 1 9 => perp 1 2
E16 parallel 4 9 and perp 1 9 => perp 1 4
HYPOTHESIS ( ( perpendicular 1 9 ) )
HYPOTHESIS ( ( parallel 6 9 ) )
HYPOTHESIS ( ( perpendicular 3 9 ) )
HYPOTHESIS ( ( perpendicular 5 9 ) )
HYPOTHESIS ( ( perpendicular 7 9 ) )
HYPOTHESIS ( ( perpendicular 8 9 ) )
Apply c1 ( parallel 6 9 )
forward 17 17
 ............. save  17
Add to FIFO ( ( parallel 1 10 ) )
E6 parallel 1 10 and 1 3 => parallel 3 10
Add to FIFO ( ( parallel 3 10 ) )
E6 parallel 1 10 and 1 5 => parallel 5 10
Add to FIFO ( ( parallel 5 10 ) )
E6 parallel 1 10 and 1 7 => parallel 7 10
Add to FIFO ( ( parallel 7 10 ) )
E6 parallel 1 10 and 1 8 => parallel 8 10
Add to FIFO ( ( parallel 8 10 ) )
E10 parallel 1 10 and perp . 1 . 2 => perpendicular 2 10
Add to FIFO ( ( perpendicular 2 10 ) )
E10 parallel 1 10 and perp . 1 . 4 => perpendicular 4 10
Add to FIFO ( ( perpendicular 4 10 ) )
E10 parallel 1 10 and perp . 1 . 6 => perpendicular 6 10
Add to FIFO ( ( perpendicular 6 10 ) )
E10 parallel 1 10 and perp . 1 . 9 => perpendicular 9 10
Add to FIFO ( ( perpendicular 9 10 ) )
HYPOTHESIS ( ( parallel 1 10 ) )
HYPOTHESIS ( ( parallel 3 10 ) )
HYPOTHESIS ( ( parallel 5 10 ) )
HYPOTHESIS ( ( parallel 7 10 ) )
HYPOTHESIS ( ( parallel 8 10 ) )
HYPOTHESIS ( ( perpendicular 2 10 ) )
HYPOTHESIS ( ( perpendicular 4 10 ) )
HYPOTHESIS ( ( perpendicular 6 10 ) )
HYPOTHESIS ( ( perpendicular 9 10 ) )
Intersecting parallel lines 1 10
 ................. restorestate 17
 ............. save  17
forward 18 18
 ............. save  18
Add to FIFO ( ( perpendicular 1 10 ) )
E14 parallel 1 3 and perp 1 10 => perp 3 10
Add to FIFO ( ( perpendicular 3 10 ) )
E14 parallel 1 5 and perp 1 10 => perp 5 10
Add to FIFO ( ( perpendicular 5 10 ) )
```

```
E14 parallel 1 7 and perp 1 10 => perp 7 10
Add to FIFO ( ( perpendicular 7 10 ) )
E14 parallel 1 8 and perp 1 10 => perp 8 10
Add to FIFO ( ( perpendicular 8 10 ) )
HYPOTHESIS ( ( perpendicular 1 10 ) )
HYPOTHESIS ( ( perpendicular 3 10 ) )
HYPOTHESIS ( ( perpendicular 5 10 ) )
HYPOTHESIS ( ( perpendicular 7 10 ) )
HYPOTHESIS ( ( perpendicular 8 10 ) )
forward 19 19
 ..............  save  19
Add to FIFO ( ( parallel 1 11 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 1 12 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 11 12 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 1 15 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 11 15 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 1 17 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 11 17 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 1 18 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 11 18 ) )
E6 parallel 1 11 and 1 3 => parallel 3 11
Add to FIFO ( ( parallel 3 11 ) )
E6 parallel 1 11 and 1 5 => parallel 5 11
Add to FIFO ( ( parallel 5 11 ) )
E6 parallel 1 11 and 1 7 => parallel 7 11
Add to FIFO ( ( parallel 7 11 ) )
E6 parallel 1 11 and 1 8 => parallel 8 11
Add to FIFO ( ( parallel 8 11 ) )
E10 parallel 1 11 and perp . 1 . 2 => perpendicular 2 11
Add to FIFO ( ( perpendicular 2 11 ) )
E10 parallel 1 11 and perp . 1 . 4 => perpendicular 4 11
Add to FIFO ( ( perpendicular 4 11 ) )
E10 parallel 1 11 and perp . 1 . 6 => perpendicular 6 11
Add to FIFO ( ( perpendicular 6 11 ) )
E10 parallel 1 11 and perp . 1 . 9 => perpendicular 9 11
Add to FIFO ( ( perpendicular 9 11 ) )
E10 parallel 1 11 and perp . 1 . 10 => perpendicular 10 11
Add to FIFO ( ( perpendicular 10 11 ) )
E6 parallel 1 12 and 1 3 => parallel 3 12
Add to FIFO ( ( parallel 3 12 ) )
E6 parallel 1 12 and 1 5 => parallel 5 12
Add to FIFO ( ( parallel 5 12 ) )
```

```
E6 parallel 1 12 and 1 7 => parallel 7 12
Add to FIFO ( ( parallel 7 12 ) )
E6 parallel 1 12 and 1 8 => parallel 8 12
Add to FIFO ( ( parallel 8 12 ) )
E10 parallel 1 12 and perp . 1 . 2 => perpendicular 2 12
Add to FIFO ( ( perpendicular 2 12 ) )
E10 parallel 1 12 and perp . 1 . 4 => perpendicular 4 12
Add to FIFO ( ( perpendicular 4 12 ) )
E10 parallel 1 12 and perp . 1 . 6 => perpendicular 6 12
Add to FIFO ( ( perpendicular 6 12 ) )
E10 parallel 1 12 and perp . 1 . 9 => perpendicular 9 12
Add to FIFO ( ( perpendicular 9 12 ) )
E10 parallel 1 12 and perp . 1 . 10 => perpendicular 10 12
Add to FIFO ( ( perpendicular 10 12 ) )
E6 parallel 1 15 and 1 3 => parallel 3 15
Add to FIFO ( ( parallel 3 15 ) )
E6 parallel 1 15 and 1 5 => parallel 5 15
Add to FIFO ( ( parallel 5 15 ) )
E6 parallel 1 15 and 1 7 => parallel 7 15
Add to FIFO ( ( parallel 7 15 ) )
E6 parallel 1 15 and 1 8 => parallel 8 15
Add to FIFO ( ( parallel 8 15 ) )
E10 parallel 1 15 and perp . 1 . 2 => perpendicular 2 15
Add to FIFO ( ( perpendicular 2 15 ) )
E10 parallel 1 15 and perp . 1 . 4 => perpendicular 4 15
Add to FIFO ( ( perpendicular 4 15 ) )
E10 parallel 1 15 and perp . 1 . 6 => perpendicular 6 15
Add to FIFO ( ( perpendicular 6 15 ) )
E10 parallel 1 15 and perp . 1 . 9 => perpendicular 9 15
Add to FIFO ( ( perpendicular 9 15 ) )
E10 parallel 1 15 and perp . 1 . 10 => perpendicular 10 15
Add to FIFO ( ( perpendicular 10 15 ) )
E6 parallel 1 17 and 1 3 => parallel 3 17
Add to FIFO ( ( parallel 3 17 ) )
E6 parallel 1 17 and 1 5 => parallel 5 17
Add to FIFO ( ( parallel 5 17 ) )
E6 parallel 1 17 and 1 7 => parallel 7 17
Add to FIFO ( ( parallel 7 17 ) )
E6 parallel 1 17 and 1 8 => parallel 8 17
Add to FIFO ( ( parallel 8 17 ) )
E10 parallel 1 17 and perp . 1 . 2 => perpendicular 2 17
Add to FIFO ( ( perpendicular 2 17 ) )
E10 parallel 1 17 and perp . 1 . 4 => perpendicular 4 17
Add to FIFO ( ( perpendicular 4 17 ) )
E10 parallel 1 17 and perp . 1 . 6 => perpendicular 6 17
Add to FIFO ( ( perpendicular 6 17 ) )
E10 parallel 1 17 and perp . 1 . 9 => perpendicular 9 17
Add to FIFO ( ( perpendicular 9 17 ) )
E10 parallel 1 17 and perp . 1 . 10 => perpendicular 10 17
Add to FIFO ( ( perpendicular 10 17 ) )
```

```
E6 parallel 1 18 and 1 3 => parallel 3 18
Add to FIFO ( ( parallel 3 18 ) )
E6 parallel 1 18 and 1 5 => parallel 5 18
Add to FIFO ( ( parallel 5 18 ) )
E6 parallel 1 18 and 1 7 => parallel 7 18
Add to FIFO ( ( parallel 7 18 ) )
E6 parallel 1 18 and 1 8 => parallel 8 18
Add to FIFO ( ( parallel 8 18 ) )
E10 parallel 1 18 and perp . 1 . 2 => perpendicular 2 18
Add to FIFO ( ( perpendicular 2 18 ) )
E10 parallel 1 18 and perp . 1 . 4 => perpendicular 4 18
Add to FIFO ( ( perpendicular 4 18 ) )
E10 parallel 1 18 and perp . 1 . 6 => perpendicular 6 18
Add to FIFO ( ( perpendicular 6 18 ) )
E10 parallel 1 18 and perp . 1 . 9 => perpendicular 9 18
Add to FIFO ( ( perpendicular 9 18 ) )
E10 parallel 1 18 and perp . 1 . 10 => perpendicular 10 18
Add to FIFO ( ( perpendicular 10 18 ) )
HYPOTHESIS ( ( parallel 1 11 ) )
HYPOTHESIS ( ( parallel 1 12 ) )
HYPOTHESIS ( ( parallel 11 12 ) )
HYPOTHESIS ( ( parallel 1 15 ) )
HYPOTHESIS ( ( parallel 11 15 ) )
HYPOTHESIS ( ( parallel 1 17 ) )
HYPOTHESIS ( ( parallel 11 17 ) )
HYPOTHESIS ( ( parallel 1 18 ) )
HYPOTHESIS ( ( parallel 11 18 ) )
HYPOTHESIS ( ( parallel 3 11 ) )
HYPOTHESIS ( ( parallel 5 11 ) )
HYPOTHESIS ( ( parallel 7 11 ) )
HYPOTHESIS ( ( parallel 8 11 ) )
HYPOTHESIS ( ( perpendicular 2 11 ) )
HYPOTHESIS ( ( perpendicular 4 11 ) )
HYPOTHESIS ( ( perpendicular 6 11 ) )
HYPOTHESIS ( ( perpendicular 9 11 ) )
HYPOTHESIS ( ( perpendicular 10 11 ) )
HYPOTHESIS ( ( parallel 3 12 ) )
HYPOTHESIS ( ( parallel 5 12 ) )
HYPOTHESIS ( ( parallel 7 12 ) )
HYPOTHESIS ( ( parallel 8 12 ) )
HYPOTHESIS ( ( perpendicular 2 12 ) )
HYPOTHESIS ( ( perpendicular 4 12 ) )
HYPOTHESIS ( ( perpendicular 6 12 ) )
HYPOTHESIS ( ( perpendicular 9 12 ) )
HYPOTHESIS ( ( perpendicular 10 12 ) )
HYPOTHESIS ( ( parallel 3 15 ) )
HYPOTHESIS ( ( parallel 5 15 ) )
HYPOTHESIS ( ( parallel 7 15 ) )
HYPOTHESIS ( ( parallel 8 15 ) )
HYPOTHESIS ( ( perpendicular 2 15 ) )
```

```
HYPOTHESIS ( ( perpendicular 4 15 ) )
HYPOTHESIS ( ( perpendicular 6 15 ) )
HYPOTHESIS ( ( perpendicular 9 15 ) )
HYPOTHESIS ( ( perpendicular 10 15 ) )
HYPOTHESIS ( ( parallel 3 17 ) )
HYPOTHESIS ( ( parallel 5 17 ) )
HYPOTHESIS ( ( parallel 7 17 ) )
HYPOTHESIS ( ( parallel 8 17 ) )
HYPOTHESIS ( ( perpendicular 2 17 ) )
HYPOTHESIS ( ( perpendicular 4 17 ) )
HYPOTHESIS ( ( perpendicular 6 17 ) )
HYPOTHESIS ( ( perpendicular 9 17 ) )
HYPOTHESIS ( ( perpendicular 10 17 ) )
HYPOTHESIS ( ( parallel 3 18 ) )
HYPOTHESIS ( ( parallel 5 18 ) )
HYPOTHESIS ( ( parallel 7 18 ) )
HYPOTHESIS ( ( parallel 8 18 ) )
HYPOTHESIS ( ( perpendicular 2 18 ) )
HYPOTHESIS ( ( perpendicular 4 18 ) )
HYPOTHESIS ( ( perpendicular 6 18 ) )
HYPOTHESIS ( ( perpendicular 9 18 ) )
HYPOTHESIS ( ( perpendicular 10 18 ) )
Apply c1 ( parallel 1 11 )
Apply c1 ( parallel 1 12 )
Apply c1 ( parallel 11 12 )
Apply c1 ( parallel 1 15 )
Apply c1 ( parallel 11 15 )
Apply c1 ( parallel 1 17 )
Apply c1 ( parallel 11 17 )
Apply c1 ( parallel 1 18 )
Apply c1 ( parallel 11 18 )
Apply c1 ( parallel 3 11 )
Apply c1 ( parallel 5 11 )
Apply c1 ( parallel 7 11 )
Apply c1 ( parallel 8 11 )
Apply c1 ( parallel 3 12 )
Apply c1 ( parallel 5 12 )
Apply c1 ( parallel 7 12 )
Apply c1 ( parallel 8 12 )
Apply c1 ( parallel 3 15 )
Apply c1 ( parallel 5 15 )
Apply c1 ( parallel 7 15 )
Apply c1 ( parallel 8 15 )
Apply c1 ( parallel 3 17 )
Apply c1 ( parallel 5 17 )
Apply c1 ( parallel 7 17 )
Apply c1 ( parallel 8 17 )
Apply c1 ( parallel 3 18 )
Apply c1 ( parallel 5 18 )
Apply c1 ( parallel 7 18 )
```

```
Apply c1 ( parallel 8 18 )
APPLY c17 ( line3d 11 )
APPLY c17 ( line3d 12 )
APPLY c17 ( line3d 15 )
APPLY c17 ( line3d 17 )
APPLY c17 ( line3d 18 )
forward 20 20
 ..............  save  20
forward 21 21
 ..............  save  21
forward 22 22
 ..............  save  22
forward 23 23
 ..............  save  23
forward 24 24
 ..............  save  24
Add to FIFO ( ( perpendicular 1 13 ) )
E2 perpendicular 1 13 and 1 10 => parallel 10 13
Add to FIFO ( ( parallel 10 13 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 10 14 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 13 14 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 10 16 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 13 16 ) )
E14 parallel 1 3 and perp 1 13 => perp 3 13
Add to FIFO ( ( perpendicular 3 13 ) )
E14 parallel 1 5 and perp 1 13 => perp 5 13
Add to FIFO ( ( perpendicular 5 13 ) )
E14 parallel 1 7 and perp 1 13 => perp 7 13
Add to FIFO ( ( perpendicular 7 13 ) )
E14 parallel 1 8 and perp 1 13 => perp 8 13
Add to FIFO ( ( perpendicular 8 13 ) )
E14 parallel 1 11 and perp 1 13 => perp 11 13
Add to FIFO ( ( perpendicular 11 13 ) )
E14 parallel 1 12 and perp 1 13 => perp 12 13
Add to FIFO ( ( perpendicular 12 13 ) )
E14 parallel 1 15 and perp 1 13 => perp 13 15
Add to FIFO ( ( perpendicular 13 15 ) )
E14 parallel 1 17 and perp 1 13 => perp 13 17
Add to FIFO ( ( perpendicular 13 17 ) )
E14 parallel 1 18 and perp 1 13 => perp 13 18
Add to FIFO ( ( perpendicular 13 18 ) )
HYPOTHESIS ( ( perpendicular 1 13 ) )
HYPOTHESIS ( ( parallel 10 13 ) )
HYPOTHESIS ( ( parallel 10 14 ) )
HYPOTHESIS ( ( parallel 13 14 ) )
HYPOTHESIS ( ( parallel 10 16 ) )
```

```
HYPOTHESIS ( ( parallel 13 16 ) )
HYPOTHESIS ( ( perpendicular 3 13 ) )
HYPOTHESIS ( ( perpendicular 5 13 ) )
HYPOTHESIS ( ( perpendicular 7 13 ) )
HYPOTHESIS ( ( perpendicular 8 13 ) )
HYPOTHESIS ( ( perpendicular 11 13 ) )
HYPOTHESIS ( ( perpendicular 12 13 ) )
HYPOTHESIS ( ( perpendicular 13 15 ) )
HYPOTHESIS ( ( perpendicular 13 17 ) )
HYPOTHESIS ( ( perpendicular 13 18 ) )
Apply c1 ( parallel 10 13 )
Apply c1 ( parallel 10 14 )
Apply c1 ( parallel 13 14 )
Apply c1 ( parallel 10 16 )
Apply c1 ( parallel 13 16 )
APPLY c4 ( line3d 1 ) ( line3d 10 ) ( plane 2 )
APPLY c4 ( line3d 1 ) ( line3d 13 ) ( plane 2 )
APPLY c4 ( line3d 1 ) ( line3d 14 ) ( plane 2 )
APPLY c4 ( line3d 1 ) ( line3d 16 ) ( plane 2 )
APPLY c4 ( line3d 11 ) ( line3d 10 ) ( plane 2 )
APPLY c4 ( line3d 11 ) ( line3d 13 ) ( plane 2 )
APPLY c4 ( line3d 11 ) ( line3d 14 ) ( plane 2 )
APPLY c4 ( line3d 11 ) ( line3d 16 ) ( plane 2 )
APPLY c4 ( line3d 12 ) ( line3d 10 ) ( plane 2 )
APPLY c4 ( line3d 12 ) ( line3d 13 ) ( plane 2 )
APPLY c4 ( line3d 12 ) ( line3d 14 ) ( plane 2 )
APPLY c4 ( line3d 12 ) ( line3d 16 ) ( plane 2 )
APPLY c4 ( line3d 15 ) ( line3d 10 ) ( plane 2 )
APPLY c4 ( line3d 15 ) ( line3d 13 ) ( plane 2 )
APPLY c4 ( line3d 15 ) ( line3d 14 ) ( plane 2 )
APPLY c4 ( line3d 15 ) ( line3d 16 ) ( plane 2 )
APPLY c4 ( line3d 17 ) ( line3d 10 ) ( plane 2 )
APPLY c4 ( line3d 17 ) ( line3d 13 ) ( plane 2 )
APPLY c4 ( line3d 17 ) ( line3d 14 ) ( plane 2 )
APPLY c4 ( line3d 17 ) ( line3d 16 ) ( plane 2 )
APPLY c4 ( line3d 18 ) ( line3d 10 ) ( plane 2 )
APPLY c4 ( line3d 18 ) ( line3d 13 ) ( plane 2 )
APPLY c4 ( line3d 18 ) ( line3d 14 ) ( plane 2 )
APPLY c4 ( line3d 18 ) ( line3d 16 ) ( plane 2 )
APPLY c5 ( line3d 1 ) ( plane 2 )
APPLY c5 ( line3d 11 ) ( plane 2 )
APPLY c5 ( line3d 12 ) ( plane 2 )
APPLY c5 ( line3d 15 ) ( plane 2 )
APPLY c5 ( line3d 17 ) ( plane 2 )
APPLY c5 ( line3d 18 ) ( plane 2 )
APPLY c5 ( line3d 10 ) ( plane 2 )
APPLY c5 ( line3d 13 ) ( plane 2 )
APPLY c5 ( line3d 14 ) ( plane 2 )
APPLY c5 ( line3d 16 ) ( plane 2 )
APPLY c16 ( plane3d 2 )
```

```
APPLY c18 ( line3d 10 )
APPLY c18 ( line3d 13 )
APPLY c18 ( line3d 14 )
APPLY c18 ( line3d 16 )
forward 25 25
 ..............  save  25
forward 26 26
 ..............  save  26
Add to FIFO ( ( perpendicular 1 14 ) )
E2 perpendicular 1 14 and 1 10 => parallel 10 14
E2 perpendicular 1 14 and 1 13 => parallel 13 14
E14 parallel 1 3 and perp 1 14 => perp 3 14
Add to FIFO ( ( perpendicular 3 14 ) )
E14 parallel 1 5 and perp 1 14 => perp 5 14
Add to FIFO ( ( perpendicular 5 14 ) )
E14 parallel 1 7 and perp 1 14 => perp 7 14
Add to FIFO ( ( perpendicular 7 14 ) )
E14 parallel 1 8 and perp 1 14 => perp 8 14
Add to FIFO ( ( perpendicular 8 14 ) )
E14 parallel 1 11 and perp 1 14 => perp 11 14
Add to FIFO ( ( perpendicular 11 14 ) )
E14 parallel 1 12 and perp 1 14 => perp 12 14
Add to FIFO ( ( perpendicular 12 14 ) )
E14 parallel 1 15 and perp 1 14 => perp 14 15
Add to FIFO ( ( perpendicular 14 15 ) )
E14 parallel 1 17 and perp 1 14 => perp 14 17
Add to FIFO ( ( perpendicular 14 17 ) )
E14 parallel 1 18 and perp 1 14 => perp 14 18
Add to FIFO ( ( perpendicular 14 18 ) )
E16 parallel 10 14 and perp 1 14 => perp 1 10
E16 parallel 13 14 and perp 1 14 => perp 1 13
HYPOTHESIS ( ( perpendicular 1 14 ) )
HYPOTHESIS ( ( perpendicular 3 14 ) )
HYPOTHESIS ( ( perpendicular 5 14 ) )
HYPOTHESIS ( ( perpendicular 7 14 ) )
HYPOTHESIS ( ( perpendicular 8 14 ) )
HYPOTHESIS ( ( perpendicular 11 14 ) )
HYPOTHESIS ( ( perpendicular 12 14 ) )
HYPOTHESIS ( ( perpendicular 14 15 ) )
HYPOTHESIS ( ( perpendicular 14 17 ) )
HYPOTHESIS ( ( perpendicular 14 18 ) )
forward 27 27
 ..............  save  27
forward 28 28
 ..............  save  28
forward 29 29
 ..............  save  29
forward 30 30
 ..............  save  30
Add to FIFO ( ( perpendicular 1 16 ) )
```

```
E2 perpendicular 1 16 and 1 10 => parallel 10 16
E2 perpendicular 1 16 and 1 13 => parallel 13 16
E2 perpendicular 1 16 and 1 14 => parallel 14 16
Add to FIFO ( ( parallel 14 16 ) )
E14 parallel 1 3 and perp 1 16 => perp 3 16
Add to FIFO ( ( perpendicular 3 16 ) )
E14 parallel 1 5 and perp 1 16 => perp 5 16
Add to FIFO ( ( perpendicular 5 16 ) )
E14 parallel 1 7 and perp 1 16 => perp 7 16
Add to FIFO ( ( perpendicular 7 16 ) )
E14 parallel 1 8 and perp 1 16 => perp 8 16
Add to FIFO ( ( perpendicular 8 16 ) )
E14 parallel 1 11 and perp 1 16 => perp 11 16
Add to FIFO ( ( perpendicular 11 16 ) )
E14 parallel 1 12 and perp 1 16 => perp 12 16
Add to FIFO ( ( perpendicular 12 16 ) )
E14 parallel 1 15 and perp 1 16 => perp 15 16
Add to FIFO ( ( perpendicular 15 16 ) )
E14 parallel 1 17 and perp 1 16 => perp 16 17
Add to FIFO ( ( perpendicular 16 17 ) )
E14 parallel 1 18 and perp 1 16 => perp 16 18
Add to FIFO ( ( perpendicular 16 18 ) )
E16 parallel 10 16 and perp 1 16 => perp 1 10
E16 parallel 13 16 and perp 1 16 => perp 1 13
HYPOTHESIS ( ( perpendicular 1 16 ) )
HYPOTHESIS ( ( parallel 14 16 ) )
HYPOTHESIS ( ( perpendicular 3 16 ) )
HYPOTHESIS ( ( perpendicular 5 16 ) )
HYPOTHESIS ( ( perpendicular 7 16 ) )
HYPOTHESIS ( ( perpendicular 8 16 ) )
HYPOTHESIS ( ( perpendicular 11 16 ) )
HYPOTHESIS ( ( perpendicular 12 16 ) )
HYPOTHESIS ( ( perpendicular 15 16 ) )
HYPOTHESIS ( ( perpendicular 16 17 ) )
HYPOTHESIS ( ( perpendicular 16 18 ) )
Apply c1 ( parallel 14 16 )
forward 31 31
 ..............  save  31
forward 32 32
 ..............  save  32
forward 33 33
 ..............  save  33
forward 34 34
 ..............  save  34
forward 35 35
 ..............  save  35
forward 36 36
 ..............  save  36
forward 37 37
 ..............  save  37
```

```
forward 38 38
 ..............  save  38
forward 39 39
 ..............  save  39
forward 40 40
 ..............  save  40
forward 41 41
 ..............  save  41
forward 42 42
 ..............  save  42
forward 43 43
 ..............  save  43
forward 44 44
 ..............  save  44
forward 45 45
 ..............  save  45
forward 46 46
 ..............  save  46
forward 47 47
 ..............  save  47
forward 48 48
 ..............  save  48
forward 49 49
 ..............  save  49
Add to FIFO ( ( parallel 2 10 ) )
E6 parallel 2 10 and 2 4 => parallel 4 10
Add to FIFO ( ( parallel 4 10 ) )
E6 parallel 2 10 and 2 6 => parallel 6 10
Add to FIFO ( ( parallel 6 10 ) )
E6 parallel 2 10 and 2 9 => parallel 9 10
Add to FIFO ( ( parallel 9 10 ) )
E8 parallel 2 10 and 10 13 => parallel 2 13
Add to FIFO ( ( parallel 2 13 ) )
E8 parallel 2 10 and 10 14 => parallel 2 14
Add to FIFO ( ( parallel 2 14 ) )
E8 parallel 2 10 and 10 16 => parallel 2 16
Add to FIFO ( ( parallel 2 16 ) )
E10 parallel 2 10 and perp . 2 . 3 => perpendicular 3 10
E10 parallel 2 10 and perp . 2 . 5 => perpendicular 5 10
E10 parallel 2 10 and perp . 2 . 7 => perpendicular 7 10
E10 parallel 2 10 and perp . 2 . 8 => perpendicular 8 10
E10 parallel 2 10 and perp . 2 . 11 => perpendicular 10 11
E10 parallel 2 10 and perp . 2 . 12 => perpendicular 10 12
E10 parallel 2 10 and perp . 2 . 15 => perpendicular 10 15
E10 parallel 2 10 and perp . 2 . 17 => perpendicular 10 17
E10 parallel 2 10 and perp . 2 . 18 => perpendicular 10 18
E11 parallel 2 10 and perpendicular 1 2 => perp 1 10
E12 parallel 2 10 and perp 1 10 => perp 1 2
E12 parallel 2 10 and perp 3 10 => perp 2 3
E12 parallel 2 10 and perp 5 10 => perp 2 5
```

```
E12 parallel 2 10 and perp 7 10 => perp 2 7
E12 parallel 2 10 and perp 8 10 => perp 2 8
E13 parallel 2 10 and perp 10 11 => perp 2 11
E13 parallel 2 10 and perp 10 12 => perp 2 12
E13 parallel 2 10 and perp 10 15 => perp 2 15
E13 parallel 2 10 and perp 10 17 => perp 2 17
E13 parallel 2 10 and perp 10 18 => perp 2 18
HYPOTHESIS ( ( parallel 2 10 ) )
HYPOTHESIS ( ( parallel 4 10 ) )
HYPOTHESIS ( ( parallel 6 10 ) )
HYPOTHESIS ( ( parallel 9 10 ) )
HYPOTHESIS ( ( parallel 2 13 ) )
HYPOTHESIS ( ( parallel 2 14 ) )
HYPOTHESIS ( ( parallel 2 16 ) )
Intersecting parallel lines 2 10
 ..................  restorestate 49
 ..............  save  49
forward 50 50
 ..............  save  50
Add to FIFO ( ( perpendicular 2 10 ) )
E14 parallel 2 4 and perp 2 10 => perp 4 10
Add to FIFO ( ( perpendicular 4 10 ) )
E14 parallel 2 6 and perp 2 10 => perp 6 10
Add to FIFO ( ( perpendicular 6 10 ) )
E14 parallel 2 9 and perp 2 10 => perp 9 10
Add to FIFO ( ( perpendicular 9 10 ) )
E15 parallel 10 13 and perp 2 10 => perp 2 13
Add to FIFO ( ( perpendicular 2 13 ) )
E15 parallel 10 14 and perp 2 10 => perp 2 14
Add to FIFO ( ( perpendicular 2 14 ) )
E15 parallel 10 16 and perp 2 10 => perp 2 16
Add to FIFO ( ( perpendicular 2 16 ) )
HYPOTHESIS ( ( perpendicular 2 10 ) )
HYPOTHESIS ( ( perpendicular 4 10 ) )
HYPOTHESIS ( ( perpendicular 6 10 ) )
HYPOTHESIS ( ( perpendicular 9 10 ) )
HYPOTHESIS ( ( perpendicular 2 13 ) )
HYPOTHESIS ( ( perpendicular 2 14 ) )
HYPOTHESIS ( ( perpendicular 2 16 ) )
forward 51 51
 ..............  save  51
forward 52 52
 ..............  save  52


;          All lines on plane 1 have been correctly related
;          by the reasoning process by this level.

forward 111 111
 ..............  save  111
```

```
forward 112 112
 ..............  save  112
forward 113 113
 ..............  save  113
Add to FIFO ( ( parallel 4 13 ) )
E6 parallel 4 13 and 4 6 => parallel 6 13
Add to FIFO ( ( parallel 6 13 ) )
E6 parallel 4 13 and 4 9 => parallel 9 13
Add to FIFO ( ( parallel 9 13 ) )
E7 parallel 4 13 and 2 4 => parallel 2 13
Chunker e7 fails, cannot instantiate ( parallel 2 13 )
 ................. restorestate 113
 ..............  save  113
forward 114 114
 ..............  save  114
Add to FIFO ( ( perpendicular 4 13 ) )
E14 parallel 4 6 and perp 4 13 => perp 6 13
Add to FIFO ( ( perpendicular 6 13 ) )
E14 parallel 4 9 and perp 4 13 => perp 9 13
Add to FIFO ( ( perpendicular 9 13 ) )
E15 parallel 13 14 and perp 4 13 => perp 4 14
Add to FIFO ( ( perpendicular 4 14 ) )
E15 parallel 13 16 and perp 4 13 => perp 4 16
Add to FIFO ( ( perpendicular 4 16 ) )
E16 parallel 10 13 and perp 4 13 => perp 4 10
E17 parallel 2 4 and perp 4 13 => perp 2 13
HYPOTHESIS ( ( perpendicular 4 13 ) )
HYPOTHESIS ( ( perpendicular 6 13 ) )
HYPOTHESIS ( ( perpendicular 9 13 ) )
HYPOTHESIS ( ( perpendicular 4 14 ) )
HYPOTHESIS ( ( perpendicular 4 16 ) )
forward 115 115
 ..............  save  115
forward 116 116
 ..............  save  116
forward 117 117
 ..............  save  117
forward 118 118
 ..............  save  118
forward 119 119
 ..............  save  119
forward 120 120
 ..............  save  120
forward 121 121
 ..............  save  121
forward 122 122
 ..............  save  122
forward 123 123
 ..............  save  123
forward 124 124
```

```
 ..............  save  124
forward 125 125
 ..............  save  125
forward 126 126
 ..............  save  126
forward 127 127
 ..............  save  127
Add to FIFO ( ( parallel 5 7 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 5 8 ) )
EXTENSION
Add to FIFO without extend ( ( parallel 7 8 ) )
E6 parallel 5 7 and 5 11 => parallel 7 11
E6 parallel 5 7 and 5 12 => parallel 7 12
E6 parallel 5 7 and 5 15 => parallel 7 15
E6 parallel 5 7 and 5 17 => parallel 7 17
E6 parallel 5 7 and 5 18 => parallel 7 18
E7 parallel 5 7 and 1 5 => parallel 1 7
E7 parallel 5 7 and 3 5 => parallel 3 7
E8 parallel 5 7 and 7 11 => parallel 5 11
E8 parallel 5 7 and 7 12 => parallel 5 12
E8 parallel 5 7 and 7 15 => parallel 5 15
E8 parallel 5 7 and 7 17 => parallel 5 17
E8 parallel 5 7 and 7 18 => parallel 5 18
E9 parallel 5 7 and 1 7 => parallel 1 5
E9 parallel 5 7 and 3 7 => parallel 3 5
E10 parallel 5 7 and perp . 5 . 6 => perpendicular 6 7
E10 parallel 5 7 and perp . 5 . 9 => perpendicular 7 9
E10 parallel 5 7 and perp . 5 . 10 => perpendicular 7 10
E10 parallel 5 7 and perp . 5 . 13 => perpendicular 7 13
E10 parallel 5 7 and perp . 5 . 14 => perpendicular 7 14
E10 parallel 5 7 and perp . 5 . 16 => perpendicular 7 16
E11 parallel 5 7 and perpendicular 2 5 => perp 2 7
E11 parallel 5 7 and perpendicular 4 5 => perp 4 7
E12 parallel 5 7 and perp 2 7 => perp 2 5
E12 parallel 5 7 and perp 4 7 => perp 4 5
E12 parallel 5 7 and perp 6 7 => perp 5 6
E13 parallel 5 7 and perp 7 9 => perp 5 9
E13 parallel 5 7 and perp 7 10 => perp 5 10
E13 parallel 5 7 and perp 7 13 => perp 5 13
E13 parallel 5 7 and perp 7 14 => perp 5 14
E13 parallel 5 7 and perp 7 16 => perp 5 16
E6 parallel 5 8 and 5 11 => parallel 8 11
E6 parallel 5 8 and 5 12 => parallel 8 12
E6 parallel 5 8 and 5 15 => parallel 8 15
E6 parallel 5 8 and 5 17 => parallel 8 17
E6 parallel 5 8 and 5 18 => parallel 8 18
E7 parallel 5 8 and 1 5 => parallel 1 8
E7 parallel 5 8 and 3 5 => parallel 3 8
E8 parallel 5 8 and 8 11 => parallel 5 11
```

```
E8 parallel 5 8 and 8 12 => parallel 5 12
E8 parallel 5 8 and 8 15 => parallel 5 15
E8 parallel 5 8 and 8 17 => parallel 5 17
E8 parallel 5 8 and 8 18 => parallel 5 18
E9 parallel 5 8 and 1 8 => parallel 1 5
E9 parallel 5 8 and 3 8 => parallel 3 5
E10 parallel 5 8 and perp . 5 . 6 => perpendicular 6 8
E10 parallel 5 8 and perp . 5 . 9 => perpendicular 8 9
E10 parallel 5 8 and perp . 5 . 10 => perpendicular 8 10
E10 parallel 5 8 and perp . 5 . 13 => perpendicular 8 13
E10 parallel 5 8 and perp . 5 . 14 => perpendicular 8 14
E10 parallel 5 8 and perp . 5 . 16 => perpendicular 8 16
E11 parallel 5 8 and perpendicular 2 5 => perp 2 8
E11 parallel 5 8 and perpendicular 4 5 => perp 4 8
E12 parallel 5 8 and perp 2 8 => perp 2 5
E12 parallel 5 8 and perp 4 8 => perp 4 5
E12 parallel 5 8 and perp 6 8 => perp 5 6
E13 parallel 5 8 and perp 8 9 => perp 5 9
E13 parallel 5 8 and perp 8 10 => perp 5 10
E13 parallel 5 8 and perp 8 13 => perp 5 13
E13 parallel 5 8 and perp 8 14 => perp 5 14
E13 parallel 5 8 and perp 8 16 => perp 5 16
E6 parallel 7 8 and 7 11 => parallel 8 11
E6 parallel 7 8 and 7 12 => parallel 8 12
E6 parallel 7 8 and 7 15 => parallel 8 15
E6 parallel 7 8 and 7 17 => parallel 8 17
E6 parallel 7 8 and 7 18 => parallel 8 18
E7 parallel 7 8 and 1 7 => parallel 1 8
E7 parallel 7 8 and 3 7 => parallel 3 8
E8 parallel 7 8 and 8 11 => parallel 7 11
E8 parallel 7 8 and 8 12 => parallel 7 12
E8 parallel 7 8 and 8 15 => parallel 7 15
E8 parallel 7 8 and 8 17 => parallel 7 17
E8 parallel 7 8 and 8 18 => parallel 7 18
E9 parallel 7 8 and 1 8 => parallel 1 7
E9 parallel 7 8 and 3 8 => parallel 3 7
E10 parallel 7 8 and perp . 7 . 9 => perpendicular 8 9
E10 parallel 7 8 and perp . 7 . 10 => perpendicular 8 10
E10 parallel 7 8 and perp . 7 . 13 => perpendicular 8 13
E10 parallel 7 8 and perp . 7 . 14 => perpendicular 8 14
E10 parallel 7 8 and perp . 7 . 16 => perpendicular 8 16
E11 parallel 7 8 and perpendicular 2 7 => perp 2 8
E11 parallel 7 8 and perpendicular 4 7 => perp 4 8
E11 parallel 7 8 and perpendicular 6 7 => perp 6 8
E12 parallel 7 8 and perp 2 8 => perp 2 7
E12 parallel 7 8 and perp 4 8 => perp 4 7
E12 parallel 7 8 and perp 6 8 => perp 6 7
E13 parallel 7 8 and perp 8 9 => perp 7 9
E13 parallel 7 8 and perp 8 10 => perp 7 10
E13 parallel 7 8 and perp 8 13 => perp 7 13
```

```
E13 parallel 7 8 and perp 8 14 => perp 7 14
E13 parallel 7 8 and perp 8 16 => perp 7 16
HYPOTHESIS ( ( parallel 5 7 ) )
HYPOTHESIS ( ( parallel 5 8 ) )
HYPOTHESIS ( ( parallel 7 8 ) )
Apply c1 ( parallel 5 7 )
Apply c1 ( parallel 5 8 )
Apply c1 ( parallel 7 8 )
forward 128 128
 ..............  save  128
forward 129 129
 ..............  save  129


;                Nothing happens between level 129 and 163
;                Intermediate trace is eliminated for space


forward 163 163
 ..............  save  163
forward 164 164
 ..............  save  164
forward 165 165
 ..............  save  165
Add to FIFO ( ( parallel 6 14 ) )
E6 parallel 6 14 and 6 9 => parallel 9 14
Add to FIFO ( ( parallel 9 14 ) )
E7 parallel 6 14 and 2 6 => parallel 2 14
Chunker e7 fails, cannot instantiate ( parallel 2 14 )
 ................  restorestate 165
 ..............  save  165
forward 166 166
 ..............  save  166
Add to FIFO ( ( perpendicular 6 14 ) )
E14 parallel 6 9 and perp 6 14 => perp 9 14
Add to FIFO ( ( perpendicular 9 14 ) )
E15 parallel 14 16 and perp 6 14 => perp 6 16
Add to FIFO ( ( perpendicular 6 16 ) )
E16 parallel 10 14 and perp 6 14 => perp 6 10
E16 parallel 13 14 and perp 6 14 => perp 6 13
E17 parallel 2 6 and perp 6 14 => perp 2 14
E17 parallel 4 6 and perp 6 14 => perp 4 14
HYPOTHESIS ( ( perpendicular 6 14 ) )
HYPOTHESIS ( ( perpendicular 9 14 ) )
HYPOTHESIS ( ( perpendicular 6 16 ) )
forward 167 167
 ..............  save  167
forward 168 168
 ..............  save  168
```

```
;                    Nothing new happens between level 169 and
;                    227, intermediate trace deleted for space


 .............. save  227
forward 228 228
 .............. save  228
forward 229 229
 .............. save  229
Add to FIFO ( ( parallel 9 16 ) )
E7 parallel 9 16 and 2 9 => parallel 2 16
Chunker e7 fails, cannot instantiate ( parallel 2 16 )
 ................. restorestate 229
 .............. save  229
forward 230 230
 .............. save  230
Add to FIFO ( ( perpendicular 9 16 ) )
E16 parallel 10 16 and perp 9 16 => perp 9 10
E16 parallel 13 16 and perp 9 16 => perp 9 13
E16 parallel 14 16 and perp 9 16 => perp 9 14
E17 parallel 2 9 and perp 9 16 => perp 2 16
E17 parallel 4 9 and perp 9 16 => perp 4 16
E17 parallel 6 9 and perp 9 16 => perp 6 16
HYPOTHESIS ( ( perpendicular 9 16 ) )
forward 231 231
 .............. save  231
forward 232 232

;                 Trace between levels 232 and 267 deleted
;                 for space.


forward 267 267
 .............. save  267
forward 268 268
 .............. save  268
forward 269 269
 .............. save  269
Add to FIFO ( ( parallel 12 15 ) )
E7 parallel 12 15 and 1 12 => parallel 1 15
E7 parallel 12 15 and 11 12 => parallel 11 15
E7 parallel 12 15 and 3 12 => parallel 3 15
E7 parallel 12 15 and 5 12 => parallel 5 15
E7 parallel 12 15 and 7 12 => parallel 7 15
E7 parallel 12 15 and 8 12 => parallel 8 15
E9 parallel 12 15 and 1 15 => parallel 1 12
E9 parallel 12 15 and 11 15 => parallel 11 12
E9 parallel 12 15 and 3 15 => parallel 3 12
E9 parallel 12 15 and 5 15 => parallel 5 12
```

```
E9 parallel 12 15 and 7 15 => parallel 7 12
E9 parallel 12 15 and 8 15 => parallel 8 12
E10 parallel 12 15 and perp . 12 . 13 => perpendicular 13 15
E10 parallel 12 15 and perp . 12 . 14 => perpendicular 14 15
E10 parallel 12 15 and perp . 12 . 16 => perpendicular 15 16
E11 parallel 12 15 and perpendicular 2 12 => perp 2 15
E11 parallel 12 15 and perpendicular 4 12 => perp 4 15
E11 parallel 12 15 and perpendicular 6 12 => perp 6 15
E11 parallel 12 15 and perpendicular 9 12 => perp 9 15
E11 parallel 12 15 and perpendicular 10 12 => perp 10 15
E12 parallel 12 15 and perp 2 15 => perp 2 12
E12 parallel 12 15 and perp 4 15 => perp 4 12
E12 parallel 12 15 and perp 6 15 => perp 6 12
E12 parallel 12 15 and perp 9 15 => perp 9 12
E12 parallel 12 15 and perp 10 15 => perp 10 12
E12 parallel 12 15 and perp 13 15 => perp 12 13
E12 parallel 12 15 and perp 14 15 => perp 12 14
E13 parallel 12 15 and perp 15 16 => perp 12 16
HYPOTHESIS ( ( parallel 12 15 ) )
Apply c1 ( parallel 12 15 )
Old vanishing point line 12 is infinity infinity
Old vanishing point line 15 is infinity infinity
New vaninshing point is infinity infinify
forward 270 270
 ..............  save  270
forward 271 271
 ..............  save  271
forward 272 272
 ..............  save  272
forward 273 273
 ..............  save  273
forward 274 274
 ..............  save  274
Add to FIFO ( ( perpendicular 12 17 ) )
E2 perpendicular 12 17 and 12 13 => parallel 13 17
Chunker e2 fails, cannot instantiate ( parallel 13 17 )
 ................. restorestate 274
 ..............  save  274
forward 275 275
 ..............  save  275
forward 276 276
 ..............  save  276
Add to FIFO ( ( perpendicular 12 18 ) )
E2 perpendicular 12 18 and 12 13 => parallel 13 18
Chunker e2 fails, cannot instantiate ( parallel 13 18 )
 ................. restorestate 276
 ..............  save  276
forward 277 277
 ..............  save  277
```

```
;              Trace between 278 and 295 deleted


 .............. save  295
forward 296 296
 .............. save  296
forward 297 297
 .............. save  297
forward 298 298
 .............. save  298
Add to FIFO ( ( perpendicular 15 17 ) )
E2 perpendicular 15 17 and 15 16 => parallel 16 17
Chunker e2 fails, cannot instantiate ( parallel 16 17 )
 ................. restorestate 298
 .............. save  298
forward 299 299
 .............. save  299
forward 300 300
 .............. save  300
Add to FIFO ( ( perpendicular 15 18 ) )
E2 perpendicular 15 18 and 15 16 => parallel 16 18
Chunker e2 fails, cannot instantiate ( parallel 16 18 )
 ................. restorestate 300
 .............. save  300
forward 301 301
 .............. save  301
forward 302 302
 .............. save  302
forward 303 303
 .............. save  303
forward 304 304
 .............. save  304
forward 305 305
 .............. save  305
Add to FIFO ( ( parallel 17 18 ) )
E7 parallel 17 18 and 1 17 => parallel 1 18
E7 parallel 17 18 and 11 17 => parallel 11 18
E7 parallel 17 18 and 3 17 => parallel 3 18
E7 parallel 17 18 and 5 17 => parallel 5 18
E7 parallel 17 18 and 7 17 => parallel 7 18
E7 parallel 17 18 and 8 17 => parallel 8 18
E9 parallel 17 18 and 1 18 => parallel 1 17
E9 parallel 17 18 and 11 18 => parallel 11 17
E9 parallel 17 18 and 3 18 => parallel 3 17
E9 parallel 17 18 and 5 18 => parallel 5 17
E9 parallel 17 18 and 7 18 => parallel 7 17
E9 parallel 17 18 and 8 18 => parallel 8 17
E11 parallel 17 18 and perpendicular 2 17 => perp 2 18
E11 parallel 17 18 and perpendicular 4 17 => perp 4 18
```

```
E11 parallel 17 18 and perpendicular 6 17 => perp 6 18
E11 parallel 17 18 and perpendicular 9 17 => perp 9 18
E11 parallel 17 18 and perpendicular 10 17 => perp 10 18
E11 parallel 17 18 and perpendicular 13 17 => perp 13 18
E11 parallel 17 18 and perpendicular 14 17 => perp 14 18
E11 parallel 17 18 and perpendicular 16 17 => perp 16 18
E12 parallel 17 18 and perp 2 18 => perp 2 17
E12 parallel 17 18 and perp 4 18 => perp 4 17
E12 parallel 17 18 and perp 6 18 => perp 6 17
E12 parallel 17 18 and perp 9 18 => perp 9 17
E12 parallel 17 18 and perp 10 18 => perp 10 17
E12 parallel 17 18 and perp 13 18 => perp 13 17
E12 parallel 17 18 and perp 14 18 => perp 14 17
E12 parallel 17 18 and perp 16 18 => perp 16 17
HYPOTHESIS ( ( parallel 17 18 ) )
Apply c1 ( parallel 17 18 )
forward 306 306
 ..............  save  306
forward 307 307
 ..............  save  307
```

```
;
;                          The computed solution
;

Computed focal distance unknown


POINT 1
 ....... projection at   -2.38200   -2.86000
 ....... space location unknown unknown unknown

POINT 2
 ....... projection at   -2.53200    3.72000
 ....... space location unknown unknown unknown

POINT 3
 ....... projection at   -6.38200    3.67000
 ....... space location unknown unknown unknown

POINT 4
 ....... projection at   -6.38200    3.41000
 ....... space location unknown unknown unknown

POINT 5
 ....... projection at   -2.78200    3.46000
 ....... space location unknown unknown unknown

POINT 6
 ....... projection at   -2.79200    3.45000
 ....... space location unknown unknown unknown

POINT 7
 ....... projection at   -2.74200    1.22000
 ....... space location unknown unknown unknown

POINT 8
 ....... projection at   -2.73200    1.22000
 ....... space location unknown unknown unknown

POINT 9
 ....... projection at   -6.35200    1.17000
 ....... space location unknown unknown unknown

POINT 10
 ....... projection at   -6.35200    0.87000
 ....... space location unknown unknown unknown

POINT 11
 ....... projection at   -2.75200    0.94000
 ....... space location unknown unknown unknown
```

```
POINT 12
 ....... projection at   -2.67200   -2.85000
 ....... space location unknown unknown unknown

POINT 13
 ....... projection at   -2.74200    0.94000
 ....... space location unknown unknown unknown

POINT 14
 ....... projection at   -0.96200    1.45000
 ....... space location unknown unknown unknown

POINT 15
 ....... projection at   -0.98200    1.46000
 ....... space location unknown unknown unknown

POINT 16
 ....... projection at   -0.93200   -1.10000
 ....... space location unknown unknown unknown

POINT 17
 ....... projection at   -0.93200   -1.11000
 ....... space location unknown unknown unknown

POINT 18
 ....... projection at   -0.97200   -1.15000
 ....... space location unknown unknown unknown

POINT 19
 ....... projection at   -0.96200   -1.15000
 ....... space location unknown unknown unknown

POINT 20
 ....... projection at   -0.99200    0.36000
 ....... space location unknown unknown unknown

POINT 21
 ....... projection at   -0.99200    0.37000
 ....... space location unknown unknown unknown

POINT 22
 ....... projection at   -2.31200    0.84000
 ....... space location unknown unknown unknown

POINT 23
 ....... projection at   -0.99200    0.48000
 ....... space location unknown unknown unknown

POINT 24
```

```
 ....... projection at   -2.31200     1.11000
 ....... space location unknown unknown unknown

POINT 25
 ....... projection at   -2.32200     1.11000
 ....... space location unknown unknown unknown

POINT 26
 ....... projection at   -2.37200     3.23000
 ....... space location unknown unknown unknown

POINT 27
 ....... projection at   -2.36200     3.23000
 ....... space location unknown unknown unknown

POINT 28
 ....... projection at   -1.02200     1.40000
 ....... space location unknown unknown unknown

POINT 29
 ....... projection at   -0.99200     1.40000
 ....... space location unknown unknown unknown

POINT 30
 ....... projection at   -0.98200     0.49000
 ....... space location unknown unknown unknown

POINT 31
 ....... projection at   -2.24200    -2.66000
 ....... space location unknown unknown unknown
```

LINE 1
 ..... from 1 to 2
 ..... equation 1.00000 x + 0.00000 z + 2.44720
 ..... vanishing point infinity infinity
 ..... direction cosines 0.00000 0.00000 1.00000

LINE 2
 ..... from 2 to 3
 ..... equation 0.00000 x + 1.00000 z + -3.72000
 ..... vanishing point infinity infinity
 ..... direction cosines 1.00000 0.00000 0.00000

LINE 3
 ..... from 3 to 4
 ..... equation 1.00000 x + 0.00000 z + 6.38200
 ..... vanishing point infinity infinity
 ..... direction cosines 0.00000 0.00000 1.00000

LINE 4
 ..... from 4 to 5
 ..... equation 0.00000 x + 1.00000 z + -3.41000
 ..... vanishing point infinity infinity
 ..... direction cosines 1.00000 0.00000 0.00000

LINE 5
 ..... from 6 to 7
 ..... equation 1.00000 x + 0.00000 z + 2.79200
 ..... vanishing point infinity infinity
 ..... direction cosines 0.00000 0.00000 1.00000

LINE 6
 ..... from 8 to 9
 ..... equation 0.00000 x + 1.00000 z + -1.22000
 ..... vanishing point infinity infinity
 ..... direction cosines 1.00000 0.00000 0.00000

LINE 7
 ..... from 9 to 10
 ..... equation 1.00000 x + 0.00000 z + 6.35200
 ..... vanishing point infinity infinity
 ..... direction cosines 0.00000 0.00000 1.00000

LINE 8
 ..... from 11 to 12
 ..... equation 1.00000 x + 0.00000 z + 2.75200
 ..... vanishing point infinity infinity
 ..... direction cosines 0.00000 0.00000 1.00000

LINE 9

```
 ....... from 13 to 10
 ....... equation    0.00000 x +    1.00000 z +   -0.94000
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 10
 ....... from 2 to 14
 ....... equation    1.00000 x +    0.69163 z +    0.00000
 ....... vanishing point    0.00000    0.00000
 ....... direction cosines    0.00000    1.00000    0.00000

LINE 11
 ....... from 15 to 16
 ....... equation    1.00000 x +    0.00000 z +    0.98200
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 12
 ....... from 19 to 20
 ....... equation    1.00000 x +    0.00000 z +    0.96200
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 13
 ....... from 21 to 22
 ....... equation    0.35606 x +    1.00000 z +    0.00000
 ....... vanishing point    0.00000    0.00000
 ....... direction cosines    0.00000    1.00000    0.00000

LINE 14
 ....... from 23 to 24
 ....... equation    0.47727 x +    1.00000 z +    0.00000
 ....... vanishing point    0.00000    0.00000
 ....... direction cosines    0.00000    1.00000    0.00000

LINE 15
 ....... from 25 to 26
 ....... equation    1.00000 x +    0.00000 z +    2.32200
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 16
 ....... from 27 to 28
 ....... equation    1.00000 x +    0.73224 z +    0.00000
 ....... vanishing point    0.00000    0.00000
 ....... direction cosines    0.00000    1.00000    0.00000

LINE 17
 ....... from 29 to 30
 ....... equation    1.00000 x +    0.00000 z +    0.99200
```

```
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 18
 ....... from 22 to 31
 ....... equation    1.00000 x +    0.00000 z +    2.31200
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000
```

```
PLANE 1
 ....... lines ( 1 2 3 4 5 6 7 8 9 )
 ....... vanishing trace
 .......      infinity x + infinity z + infinity
 ....... direction cosines of normal
 .......      0.00000    1.00000    0.00000
 ....... distance from origin unknown

PLANE 2
 ....... lines ( 1 10 11 12 13 14 15 16 17 18 )
 ....... vanishing trace
 .......      1.00000 x +     0.00000 z +     0.00000
 ....... direction cosines of normal
 .......      1.00000    0.00000    0.00000
 ....... distance from origin unknown

;                    The spatial relationships in the
;                interpretation
( ( ( parallel ) ) ( ( parallel 1 3 ) ) ( ( parallel 1 5 ) )
( ( parallel 3 5 ) ) ( ( parallel 1 7 ) ) ( ( parallel 3 7 )
) ( ( parallel 1 8 ) ) ( ( parallel 3 8 ) ) ( ( parallel 2 4
) ) ( ( parallel 2 6 ) ) ( ( parallel 4 6 ) ) ( ( parallel 2
9 ) ) ( ( parallel 4 9 ) ) ( ( parallel 6 9 ) ) ( ( parallel
1 11 ) ) ( ( parallel 1 12 ) ) ( ( parallel 11 12 ) ) ( (
parallel 1 15 ) ) ( ( parallel 11 15 ) ) ( ( parallel 1 17 )
) ( ( parallel 11 17 ) ) ( ( parallel 1 18 ) ) ( ( parallel
11 18 ) ) ( ( parallel 3 11 ) ) ( ( parallel 5 11 ) ) ( (
parallel 7 11 ) ) ( ( parallel 8 11 ) ) ( ( parallel 3 12 )
) ( ( parallel 5 12 ) ) ( ( parallel 7 12 ) ) ( ( parallel 8
12 ) ) ( ( parallel 3 15 ) ) ( ( parallel 5 15 ) ) ( ( par-
allel 7 15 ) ) ( ( parallel 8 15 ) ) ( ( parallel 3 17 ) ) (
( parallel 5 17 ) ) ( ( parallel 7 17 ) ) ( ( parallel 8 17
) ) ( ( parallel 3 18 ) ) ( ( parallel 5 18 ) ) ( ( parallel
7 18 ) ) ( ( parallel 8 18 ) ) ( ( parallel 10 13 ) ) ( (
parallel 10 14 ) ) ( ( parallel 13 14 ) ) ( ( parallel 10 16
) ) ( ( parallel 13 16 ) ) ( ( parallel 14 16 ) ) ( ( paral-
lel 5 7 ) ) ( ( parallel 5 8 ) ) ( ( parallel 7 8 ) ) ( (
parallel 12 15 ) ) ( ( parallel 17 18 ) ) ) ( ( ( perpendi-
cular ) ) ( ( perpendicular 1 2 ) ) ( ( perpendicular 2 3 )
) ( ( perpendicular 2 5 ) ) ( ( perpendicular 2 7 ) ) ( (
perpendicular 2 8 ) ) ( ( perpendicular 1 4 ) ) ( ( perpen-
dicular 3 4 ) ) ( ( perpendicular 4 5 ) ) ( ( perpendicular
4 7 ) ) ( ( perpendicular 4 8 ) ) ( ( perpendicular 1 6 ) )
( ( perpendicular 3 6 ) ) ( ( perpendicular 5 6 ) ) ( ( per-
pendicular 6 7 ) ) ( ( perpendicular 6 8 ) ) ( ( perpendicu-
lar 1 9 ) ) ( ( perpendicular 3 9 ) ) ( ( perpendicular 5 9
) ) ( ( perpendicular 7 9 ) ) ( ( perpendicular 8 9 ) ) ( (
perpendicular 1 10 ) ) ( ( perpendicular 3 10 ) ) ( ( per-
```

pendicular 5 10)) ((perpendicular 7 10)) ((perpendicular 8 10)) ((perpendicular 2 11)) ((perpendicular 4 11)) ((perpendicular 6 11)) ((perpendicular 9 11)) ((perpendicular 10 11)) ((perpendicular 2 12)) ((perpendicula 4 12)) ((perpendicular 6 12)) ((perpendicular 9 12)) ((perpendicular 10 12)) ((perpendicular 2 15)) ((perpendicula 4 15)) ((perpendicular 6 15)) ((perpendicular 9 15)) ((perpendicular 10 15)) ((perpendicular 2 17)) ((perpendicula 4 17)) ((perpendicular 6 17)) ((perpendicular 9 17)) ((perpendicular 10 17)) ((perpendicular 2 18)) ((perpendicula 4 18)) ((perpendicular 6 18)) ((perpendicular 9 18)) ((perpendicular 10 18)) ((perpendicular 1 13)) ((perpendicula 3 13)) ((perpendicular 5 13)) ((perpendicular 7 13)) ((perpendicular 8 13)) ((perpendicular 11 13)) ((perpendicula 12 13)) ((perpendicular 13 15)) ((perpendicular 13 17)) ((perpendicular 13 18)) ((perpendicular 1 14)) ((perpendicula 3 14)) ((perpendicular 5 14)) ((perpendicular 7 14)) ((perpendicular 8 14)) ((perpendicular 11 14)) ((perpendicula 12 14)) ((perpendicular 14 15)) ((perpendicular 14 17)) ((perpendicular 14 18)) ((perpendicular 1 16)) ((perpendicula 3 16)) ((perpendicular 5 16)) ((perpendicular 7 16)) ((perpendicular 8 16)) ((perpendicular 11 16)) ((perpendicula 12 16)) ((perpendicular 15 16)) ((perpendicular 16 17)) ((perpendicular 16 18)) ((perpendicular 2 10)) ((perpendicula 4 10)) ((perpendicular 6 10)) ((perpendicular 9 10)) ((perpendicular 2 13)) ((perpendicular 2 14)) ((perpendicular 2 16)) ((perpendicular 4 13)) ((perpendicular 6 13)) ((perpendicular 9 13)) ((perpendicular 4 14)) ((perpendicular 4 16)) ((perpendicular 6 14)) ((perpendicular 9 14)) ((perpendicular 6 16)) ((perpendicular 9 16)))

```
;                    Summary of the results
27 applications of consistency checking engines
number of true solutions 1
number of total solutions 1
cpu time 5838.90967 seconds
```

C.2 RESULTS FOR THE TABLE IMAGE

In this section, we show the results generated by the search process for the image shown in Figure 2, in Chapter IV. The listing is annotated to point out interesting features of the search process and to explain the results. All comments are denoted by a semicolon (;) in column 1 of the listing. The image consists of 16 points, 12 lines and 2 planes.

```
  WELCOME to the NEW PROLOG
    4-AUG-1984 16:36:43.41

loading file HYP.PRO
Finished loading HYP.PRO
loading file CHUNKER.PRO
Finished loading CHUNKER.PRO
Loading file COMPUTE.PRO
finished loading COMPUTE.PRO
;
;
;                          We are looking for the
;                          Best solution. Set the
;                          search mode appropriately
;
;
(<- %mode best)
:- ( ( <- best best ) )
;
;
;                         This run uses the non-recursive
;                         search algorithm BTTS2.(See
;                         program listing in Appendix A
;
(<- %bttsflag new)
:- ( ( <- new new ) )
(search)
;
;
;                        This is the final result printed
;                        by the system.

Computed focal distance unknown


POINT 1
 ....... projection at   -2.50000   -2.00000
 ....... space location unknown unknown unknown

POINT 2
 ....... projection at   -4.65000   -3.75000
 ....... space location unknown unknown unknown

POINT 3
 ....... projection at    2.70000   -3.75000
 ....... space location unknown unknown unknown

POINT 4
 ....... projection at    1.90000   -2.00000
 ....... space location unknown unknown unknown
```

```
POINT 5
 ....... projection at   -3.75000   -3.90000
 ....... space location unknown unknown unknown

POINT 6
 ....... projection at   -3.75000   -9.30000
 ....... space location unknown unknown unknown

POINT 7
 ....... projection at   -3.25000   -9.30000
 ....... space location unknown unknown unknown

POINT 8
 ....... projection at   -3.25000   -7.70000
 ....... space location unknown unknown unknown

POINT 9
 ....... projection at    1.35000   -7.70000
 ....... space location unknown unknown unknown

POINT 10
 ....... projection at    1.35000   -9.30000
 ....... space location unknown unknown unknown

POINT 11
 ....... projection at    1.80000   -9.30000
 ....... space location unknown unknown unknown

POINT 12
 ....... projection at    1.80000   -3.90000
 ....... space location unknown unknown unknown

POINT 13
 ....... projection at    1.35000   -3.90000
 ....... space location unknown unknown unknown

POINT 14
 ....... projection at    1.35000   -7.40000
 ....... space location unknown unknown unknown

POINT 15
 ....... projection at   -3.25000   -7.40000
 ....... space location unknown unknown unknown

POINT 16
 ....... projection at   -3.25000   -3.90000
 ....... space location unknown unknown unknown
```

```
LINE 1
 ....... from 1 to 2
 ....... equation   -0.81395 x +    1.00000 z +  -0.034884
 ....... vanishing point    0.70678    0.61017
 ....... direction cosines unknown unknown unknown

LINE 2
 ....... from 2 to 3
 ....... equation    0.00000 x +    1.00000 z +    3.75000
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 3
 ....... from 3 to 4
 ....... equation    1.00000 x +    0.45714 z +   -0.98571
 ....... vanishing point    0.70678    0.61017
 ....... direction cosines unknown unknown unknown

LINE 4
 ....... from 4 to 1
 ....... equation    0.00000 x +    1.00000 z +    2.00000
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 5
 ....... from 5 to 6
 ....... equation    1.00000 x +    0.00000 z +    3.75000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 6
 ....... from 7 to 8
 ....... equation    1.00000 x +    0.00000 z +    3.25000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 7
 ....... from 8 to 9
 ....... equation    0.00000 x +    1.00000 z +    7.70000
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 8
 ....... from 9 to 10
 ....... equation    1.00000 x +    0.00000 z +   -1.35000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 9
 ....... from 11 to 12
```

```
 ....... equation    1.00000 x +    0.00000 z +   -1.80000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 10
 ....... from 13 to 14
 ....... equation    1.00000 x +    0.00000 z +   -1.35000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 11
 ....... from 14 to 15
 ....... equation    0.00000 x +    1.00000 z +    7.40000
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 12
 ....... from 15 to 16
 ....... equation    1.00000 x +    0.00000 z +    3.25000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000
```

```
PLANE 1
 ....... lines ( 1 2 3 4 )
 ....... vanishing trace
 .......      0.00000 x +     1.00000 z +    -0.61017
 ....... direction cosines of normal
 .......      unknown unknown unknown
 ....... distance from origin unknown

PLANE 2
 ....... lines ( 5 6 7 8 9 10 11 12 )
 ....... vanishing trace
 .......      infinity x + infinity z + infinity
 ....... direction cosines of normal
 .......      0.00000     1.00000     0.00000
 ....... distance from origin unknown
```

```
;
;
;                          The final hypotheses consists
;                          of generated spatial relationships
;                          between the three-dimensional
;                          world entities.
;
( ( ( parallel ) ) ( ( parallel 1 3 ) ) ( ( parallel 2 4 ) )
( ( parallel 2 7 ) ) ( ( parallel 4 7 ) ) ( ( parallel 2 11
) ) ( ( parallel 4 11 ) ) ( ( parallel 7 11 ) ) ( ( parallel
5 6 ) ) ( ( parallel 5 8 ) ) ( ( parallel 6 8 ) ) ( ( paral-
lel 5 9 ) ) ( ( parallel 6 9 ) ) ( ( parallel 5 10 ) ) ( (
parallel 6 10 ) ) ( ( parallel 5 12 ) ) ( ( parallel 6 12 )
) ( ( parallel 8 9 ) ) ( ( parallel 8 10 ) ) ( ( parallel 9
10 ) ) ( ( parallel 8 12 ) ) ( ( parallel 9 12 ) ) ( ( par-
allel 10 12 ) ) ) ( ( ( perpendicular ) ) ( ( perpendicular
1 2 ) ) ( ( perpendicular 2 3 ) ) ( ( perpendicular 1 4 ) )
( ( perpendicular 3 4 ) ) ( ( perpendicular 1 5 ) ) ( ( per-
pendicular 3 5 ) ) ( ( perpendicular 1 6 ) ) ( ( perpendicu-
lar 3 6 ) ) ( ( perpendicular 1 7 ) ) ( ( perpendicular 3 7
) ) ( ( perpendicular 1 8 ) ) ( ( perpendicular 3 8 ) ) ( (
perpendicular 1 9 ) ) ( ( perpendicular 3 9 ) ) ( ( perpen-
dicular 1 10 ) ) ( ( perpendicular 3 10 ) ) ( ( perpendicu-
lar 1 11 ) ) ( ( perpendicular 3 11 ) ) ( ( perpendicular 1
12 ) ) ( ( perpendicular 3 12 ) ) ( ( perpendicular 2 5 ) )
( ( perpendicular 4 5 ) ) ( ( perpendicular 2 6 ) ) ( ( per-
pendicular 4 6 ) ) ( ( perpendicular 5 7 ) ) ( ( perpendicu-
lar 6 7 ) ) ( ( perpendicular 2 8 ) ) ( ( perpendicular 4 8
) ) ( ( perpendicular 7 8 ) ) ( ( perpendicular 2 9 ) ) ( (
perpendicular 4 9 ) ) ( ( perpendicular 7 9 ) ) ( ( perpen-
dicular 2 10 ) ) ( ( perpendicular 4 10 ) ) ( ( perpendicu-
lar 7 10 ) ) ( ( perpendicular 5 11 ) ) ( ( perpendicular 6
11 ) ) ( ( perpendicular 8 11 ) ) ( ( perpendicular 9 11 ) )
( ( perpendicular 10 11 ) ) ( ( perpendicular 2 12 ) ) ( (
perpendicula 4 12 ) ) ( ( perpendicular 7 12 ) ) ( ( perpen-
dicular 11 12 ) ) )
```

```
;                          Statistics about the run
37 applications of lp      ; The consistency checker
                           ; was called 37 times.
number of true solutions 1
```

```
number of total solutions 1
cpu time 2063.50000 seconds  ; Accumulated CPU time.
```

```
Garbage collector invoked
 Memory status

Total memory (words)     800000
Maximum number of cells  129999
Number of atom cells defined  658
Total cell space in use  53248 cells
Currently allocated cells0 cells
Garbage collection after 10000 cells


   System table and stack status

Protection stack: MAX =    8000 Current =    2414
Function  stack1: MAX =    1000 Current =  1
Function  stack2: MAX =    1000 Current =  1
Change stack:MAX =    2000 Current =    1686
Bind   stack:MAX =    4000 Current =     217
Local  stack:MAX =    6000 Current =     221
Atom Hash table:  MAX =    1003 Current =659
Global variables: MAX =501 Current =      30
Successful examinations
;This table shows how many times the consistency
;checker was invoked as a function of hypothesis size
;in the cases when the hypotheses succeeded, and
;the total and average cpu time per invokation
;Size   CountTotal CPU    Average CPU (sec)
4    1       1.41998       1.41998
6    1       3.03998       3.03998
9    1       27.13001      27.13001
11   1       1.00998       1.00998
13   1       1.11002       1.11002
15   1       1.20001       1.20001
17   1       4.79001       4.79001
19   1       1.38000       1.38000
21   1       1.47000       1.47000
23   1       1.58002       1.58002
25   1       5.55005       5.55005
27   1       5.66998       5.66998
29   1       1.87000       1.87000
33   1       18.94000      18.94000
36   1       2.73004       2.73004
39   1       7.32001       7.32001
42   1       3.03000       3.03000
50   1       38.97998      38.97998
54   1       2.52997       2.52997
63   1      194.52002      194.52002
68   1      106.84010      106.84010
69   1      34.69995       34.69995
```

```
Failed examinations
;     This table is similar to the previous one,
;     except that it is for inconsistent hypotheses
;Size     Count     Total CPU     Average CPU (sec)
4        1         0.22998       0.22998
13       1         1.29001       1.29001
16       1         4.84003       4.84003
19       1         1.58002       1.58002
22       1         1.73999       1.73999
25       1         1.91000       1.91000
27       1         1.27997       1.27997
28       1         2.04001       2.04001
30       1         2.38000       2.38000
31       1         2.26001       2.26001
34       1         6.27002       6.27002
40       1         7.55005       7.55005
42       1         7.87000       7.87000
47       1         8.20001       8.20001
SUMMARY OF RUN
Number of true solutions 1
Number of total solutions 1
Number of lp calls 37
Total time 2078.68994
( ( search ) )
```

Note that the average CPU time to handle an inconsistent hypothesis is smaller than the average time for a consistent hypothesis. As explained in Chapter IV, the reason is that to prove some hypothesis consistent, the sequence of application must terminate with success, that is, all possible applicable engines must be examined. In the case of an inconsistent hypothesis, applications terminate as soon as the first inconsistency is found.

Further note that for the lines in the plane corresponding to the top of the table, not much information could be generated. Although we know the vanishing trace for plane

1, we do not know the focal length of the camera. Since the vanishing trace does not pass through the origin of the system, the normal to the plane depends on the focal length, which could not be computed. The same reason (focal length unknown) explains why the direction of lines 1 and 3 could not be determined. In the next section, we repeat this run with a user supplied focal length, and show the difference.

C.3 RESULTS FOR THE TABLE IMAGE GIVEN THE FOCAL LENGTH

In this section, we show the results obtained for the table image if we know the focal length a-priori. Note that because plane 1 did not turn out to be exactly horizontal, the focal length could not be computed automatically. However, if the focal length is known and supplied to the system, the normal to the plane can be computed. Specifically, compare the result obtained in this section for lines 1 and 3 and for plane 1 with the corresponding results in the previous section. Note that with the focal length known, we can compute the directions of lines 1 and 3 and the normal to the plane.

```
Computed focal distance 10 ; NOTE: This was supplied as
                           ; prior knowledge

POINT 1
 ....... projection at   -2.50000   -2.00000
 ....... space location unknown unknown unknown

POINT 2
 ....... projection at   -4.65000   -3.75000
```

```
 ....... space location unknown unknown unknown

POINT 3
 ....... projection at    2.70000   -3.75000
 ....... space location unknown unknown unknown

POINT 4
 ....... projection at    1.90000   -2.00000
 ....... space location unknown unknown unknown

POINT 5
 ....... projection at   -3.75000   -3.90000
 ....... space location unknown unknown unknown

POINT 6
 ....... projection at   -3.75000   -9.30000
 ....... space location unknown unknown unknown

POINT 7
 ....... projection at   -3.25000   -9.30000
 ....... space location unknown unknown unknown

POINT 8
 ....... projection at   -3.25000   -7.70000
 ....... space location unknown unknown unknown

POINT 9
 ....... projection at    1.35000   -7.70000
 ....... space location unknown unknown unknown

POINT 10
 ....... projection at    1.35000   -9.30000
 ....... space location unknown unknown unknown

POINT 11
 ....... projection at    1.80000   -9.30000
 ....... space location unknown unknown unknown

POINT 12
 ....... projection at    1.80000   -3.90000
 ....... space location unknown unknown unknown

POINT 13
 ....... projection at    1.35000   -3.90000
 ....... space location unknown unknown unknown

POINT 14
 ....... projection at    1.35000   -7.40000
 ....... space location unknown unknown unknown
```

```
POINT 15
 ....... projection at   -3.25000   -7.40000
 ....... space location unknown unknown unknown

POINT 16
 ....... projection at   -3.25000   -3.90000
 ....... space location unknown unknown unknown
```

```
LINE 1
 ....... from 1 to 2
 ....... equation   -0.81395 x +     1.00000 z +  -0.034884
 ....... vanishing point     0.70678     0.61017
 ....... direction cosines   0.070372    0.99567   0.060753

LINE 2
 ....... from 2 to 3
 ....... equation    0.00000 x +     1.00000 z +    3.75000
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 3
 ....... from 3 to 4
 ....... equation    1.00000 x +     0.45714 z +   -0.98571
 ....... vanishing point     0.70678     0.61017
 ....... direction cosines   0.070372    0.99567   0.060753

LINE 4
 ....... from 4 to 1
 ....... equation    0.00000 x +     1.00000 z +    2.00000
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 5
 ....... from 5 to 6
 ....... equation    1.00000 x +     0.00000 z +    3.75000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 6
 ....... from 7 to 8
 ....... equation    1.00000 x +     0.00000 z +    3.25000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 7
 ....... from 8 to 9
 ....... equation    0.00000 x +     1.00000 z +    7.70000
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 8
 ....... from 9 to 10
 ....... equation    1.00000 x +     0.00000 z +   -1.35000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 9
```

```
 ....... from 11 to 12
 ....... equation    1.00000 x +    0.00000 z +   -1.80000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 10
 ....... from 13 to 14
 ....... equation    1.00000 x +    0.00000 z +   -1.35000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 11
 ....... from 14 to 15
 ....... equation    0.00000 x +    1.00000 z +    7.40000
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 12
 ....... from 15 to 16
 ....... equation    1.00000 x +    0.00000 z +    3.25000
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000


PLANE 1
 ....... lines ( 1 2 3 4 )
 ....... vanishing trace
 .......      0.00000 x +    1.00000 z +   -0.61017
 ....... direction cosines of normal
 .......      0.00000  -0.060904    0.99814
 ....... distance from origin unknown

PLANE 2
 ....... lines ( 5 6 7 8 9 10 11 12 )
 ....... vanishing trace
 .......      infinity x + infinity z + infinity
 ....... direction cosines of normal
 .......      0.00000    1.00000    0.00000
 ....... distance from origin unknown
```

```
( ( ( parallel ) ) ( ( parallel 1 3 ) ) ( ( parallel 2 4 ) )
( ( parallel 2 7 ) ) ( ( parallel 4 7 ) ) ( ( parallel 2 11
) ) ( ( parallel 4 11 ) ) ( ( parallel 7 11 ) ) ( ( parallel
5 6 ) ) ( ( parallel 5 8 ) ) ( ( parallel 6 8 ) ) ( ( paral-
lel 5 9 ) ) ( ( parallel 6 9 ) ) ( ( parallel 5 10 ) ) ( (
parallel 6 10 ) ) ( ( parallel 5 12 ) ) ( ( parallel 6 12 )
) ( ( parallel 8 9 ) ) ( ( parallel 8 10 ) ) ( ( parallel 9
10 ) ) ( ( parallel 8 12 ) ) ( ( parallel 9 12 ) ) ( ( par-
allel 10 12 ) ) ) ( ( ( perpendicular ) ) ( ( perpendicular
1 2 ) ) ( ( perpendicular 2 3 ) ) ( ( perpendicular 1 4 ) )
( ( perpendicular 3 4 ) ) ( ( perpendicular 1 5 ) ) ( ( per-
pendicular 3 5 ) ) ( ( perpendicular 1 6 ) ) ( ( perpendicu-
lar 3 6 ) ) ( ( perpendicular 1 7 ) ) ( ( perpendicular 3 7
) ) ( ( perpendicular 1 8 ) ) ( ( perpendicular 3 8 ) ) ( (
perpendicular 1 9 ) ) ( ( perpendicular 3 9 ) ) ( ( perpen-
dicular 1 10 ) ) ( ( perpendicular 3 10 ) ) ( ( perpendicu-
lar 1 11 ) ) ( ( perpendicular 3 11 ) ) ( ( perpendicular 1
12 ) ) ( ( perpendicular 3 12 ) ) ( ( perpendicular 2 5 ) )
( ( perpendicular 4 5 ) ) ( ( perpendicular 2 6 ) ) ( ( per-
pendicular 4 6 ) ) ( ( perpendicular 5 7 ) ) ( ( perpendicu-
lar 6 7 ) ) ( ( perpendicular 2 8 ) ) ( ( perpendicular 4 8
) ) ( ( perpendicular 7 8 ) ) ( ( perpendicular 2 9 ) ) ( (
perpendicular 4 9 ) ) ( ( perpendicular 7 9 ) ) ( ( perpen-
dicular 2 10 ) ) ( ( perpendicular 4 10 ) ) ( ( perpendicu-
lar 7 10 ) ) ( ( perpendicular 5 11 ) ) ( ( perpendicular 6
11 ) ) ( ( perpendicular 8 11 ) ) ( ( perpendicular 9 11 ) )
( ( perpendicular 10 11 ) ) ( ( perpendicular 2 12 ) ) ( (
perpendicula 4 12 ) ) ( ( perpendicular 7 12 ) ) ( ( perpen-
dicular 11 12 ) ) )
;                          Result summary
37 applications of lp
number of true solutions 1
number of total solutions 1
cpu time 2234.18018 seconds
```

C.4 RESULTS FOR THE RECTANGLE IN THE Z PLANE

In this section, we show the results obtained for the rectangle data of the previous section. Note that we started out with knowledge about the coordinates of one point and the focal length. From this, the coordinates of all points are computed as are the directions of all the lines in the image.

```
Computed focal distance     1.00000


POINT 1
 ....... projection at    -1.00000    -1.00000
 ....... space location    -5.00000     5.00000    -5.00000

POINT 2
 ....... projection at    -2.00000    -2.00000
 ....... space location    -5.00000     2.50000    -5.00000

POINT 3
 ....... projection at     2.00000    -2.00000
 ....... space location     5.00000     2.50000    -5.00000

POINT 4
 ....... projection at     1.00000    -1.00000
 ....... space location     5.00000     5.00000    -5.00000


LINE 1
 ....... from 1 to 2
 ....... equation     1.00000 x +     1.00000 z +     0.00000
 ....... vanishing point     0.00000     0.00000
 ....... direction cosines     0.00000     1.00000     0.00000

LINE 2
 ....... from 2 to 3
 ....... equation     0.00000 x +     1.00000 z +     2.00000
 ....... vanishing point infinity infinity
 ....... direction cosines     1.00000     0.00000     0.00000

LINE 3
 ....... from 3 to 4
 ....... equation     1.00000 x +    -1.00000 z +     0.00000
 ....... vanishing point     0.00000     0.00000
 ....... direction cosines     0.00000     1.00000     0.00000

LINE 4
 ....... from 1 to 4
 ....... equation     0.00000 x +     1.00000 z +     1.00000
 ....... vanishing point infinity infinity
 ....... direction cosines     1.00000     0.00000     0.00000
```

```
PLANE 1
 ....... lines ( 1 3 2 4 )
 ....... vanishing trace
 .......       0.00000 x +     1.00000 z +     0.00000
 ....... direction cosines of normal
 .......       0.00000     0.00000     1.00000
 ....... distance from origin     5.00000

( ( ( parallel ) ) ( ( parallel 1 3 ) ) ( ( parallel 2 4 ) )
)

( ( ( perpendicular ) ) ( ( perpendicular 1 2 ) ) ( ( per-
pendicular 2 3 ) ) ( ( perpendicular 1 4 ) ) ( ( perpendicu-
lar 3 4 ) ) )
4 applications of lp
number of true solutions 1
number of total solutions 1
cpu time   50.70999 seconds
```

C.4.1 Results for the fountain scene

In this section, we exhibit the results for the fountain scene shown in Figure 3 in Chapter V and Figure 39 shown in Appendix B. Note the following observations about the inference process involved:

1. The lines 1,2,3 and 4 were determined to be a rectangle and used to compute the normal to the plane 1. Since two of the lines (1 and 4) were actually parallel in the image, the focal length could not be calculated.
2. Knowing the direction of the normal to the plane, and using the hypothesis that the observed arc (Arc 1) was a actually a circle in three-dimensions, the focal length could be calculated. The calculated value is shown in the result.

3. This calculated value was used to re-compute the direction cosines of the normal to the plane, and the vanishing trace of the plane, and the two computed values were close enough to be considered the same (less than 0.1% difference). Thus the hypothesis was declared consistent.

```
Computed focal distance   56.75236


POINT 1
 ....... projection at   -6.25000  -20.13181
 ....... space location unknown unknown unknown

POINT 2
 ....... projection at  -12.47727  -20.13181
 ....... space location unknown unknown unknown

POINT 3
 ....... projection at  -11.38637  -37.49545
 ....... space location unknown unknown unknown

POINT 4
 ....... projection at   -6.20455  -20.08636
 ....... space location unknown unknown unknown

POINT 5
 ....... projection at  -12.29545  -20.17727
 ....... space location unknown unknown unknown

POINT 6
 ....... projection at  -22.88637  -37.40454
 ....... space location unknown unknown unknown

POINT 7
 ....... projection at  -11.38637  -37.35909
 ....... space location unknown unknown unknown

POINT 8
 ....... projection at  -13.97727  -22.90454
 ....... space location unknown unknown unknown

POINT 9
 ....... projection at  -17.56818  -22.90454
 ....... space location unknown unknown unknown

POINT 10
 ....... projection at  -23.52273  -30.95000
 ....... space location unknown unknown unknown

POINT 11
 ....... projection at  -18.75000  -30.99545
 ....... space location unknown unknown unknown
```

```
LINE 1
 ....... from 1 to 2
 ....... equation    0.00000 x +    1.00000 z +   20.13181
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 2
 ....... from 3 to 1
 ....... equation    1.00000 x +   -0.29765 z +  0.0022584
 ....... vanishing point -0.0050402 -0.0056040
 ....... direction cosines    0.00000    1.00000    0.00000

LINE 3
 ....... from 2 to 6
 ....... equation    1.00000 x +   -0.61478 z + -0.0010906
 ....... vanishing point -0.0050402 -0.0056040
 ....... direction cosines    0.00000    1.00000    0.00000

LINE 4
 ....... from 6 to 3
 ....... equation    0.00000 x +    1.00000 z +   37.40454
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 5
 ....... from 8 to 9
 ....... equation    0.00000 x +    1.00000 z +   22.90454
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000

LINE 6
 ....... from 10 to 11
 ....... equation    0.00000 x +    1.00000 z +   30.95000
 ....... vanishing point infinity infinity
 ....... direction cosines    1.00000    0.00000    0.00000


ARC 1
 ....... from 9 to 10
 ....... equation    1.00000 x2 +   -0.99273 xz +    4.68624 z2
 .......          +   14.31676 x +  232.84050 z + 3220.83008


PLANE 1
 ....... lines ( 1 2 3 4 5 6 )
 ....... arcs ( 1 )
```

```
....... vanishing trace
.......      0.00000 x +     1.00000 z +   0.0056040
....... direction cosines of normal
.......      0.00000     0.00000     1.00000
....... distance from origin unknown
```

```
;                    Final relations

( ( ( parallel ) ) ( ( parallel 2 3 ) ) ( ( parallel 1 4 ) )
( ( parallel
     1 5 ) ) ( ( parallel 4 5 ) ) ( ( parallel 1 6 )  ) ( (
parallel
     4 6 ) ) ( ( parallel 5 6 ) ) ) ( ( ( perpendicular )  )
( ( perpendicular 1 2 ) ) ( ( perpendicular
     1 3 ) ) ( ( perpendicular 2 4 ) ) ( ( perpendicular 3 4
     ) ) ( ( perpendicular 2 5 ) ) ( ( perpendicular 3 5 ) )
( ( perpendicular
     2 6 ) ) ( ( perpendicular 3 6 ) ) ) ( ( ( circle ) ) (
( circle 1 ) ) )
;                         Summary of run
7 applications of lp
number of true solutions 1
number of total solutions 1
cpu time  170.21002 seconds
```

C.5 RESULTS FOR THE RECTANGULAR PARALLELIPIPID

With the correct input, the rectangular parallelipiped was correctly analyzed. The results are shown below.

```
Computed focal distance    3.00001


POINT 1
 ....... projection at   -0.50000   -1.50000
 ....... space location unknown unknown unknown

POINT 2
 ....... projection at    0.00000   -1.28571
 ....... space location    0.00000 unknown unknown

POINT 3
 ....... projection at    0.50000   -1.50000
 ....... space location unknown unknown unknown

POINT 4
 ....... projection at    0.00000   -1.80000
 ....... space location    0.00000 unknown unknown

POINT 5
 ....... projection at    0.00000   -3.60000
 ....... space location    0.00000 unknown unknown

POINT 6
 ....... projection at    0.50000   -3.00000
 ....... space location unknown unknown unknown

POINT 7
 ....... projection at   -0.50000   -3.00000
 ....... space location unknown unknown unknown
```

```
LINE 1
 ....... from 1 to 2
 ....... equation   -0.42857 x +     1.00000 z +     1.28571
 ....... vanishing point    3.00001 0.0000069539
 ....... direction cosines     0.70711     0.70711 0.0000016390

LINE 2
 ....... from 2 to 3
 ....... equation    0.42857 x +     1.00000 z +     1.28571
 ....... vanishing point   -3.00001 0.0000069539
 ....... direction cosines    -0.70711     0.70711 0.0000016390

LINE 3
 ....... from 3 to 4
 ....... equation   -0.60000 x +     1.00000 z +     1.80000
 ....... vanishing point    3.00001 0.0000069539
 ....... direction cosines     0.70711     0.70711 0.0000016390

LINE 4
 ....... from 4 to 1
 ....... equation    0.60000 x +     1.00000 z +     1.80000
 ....... vanishing point   -3.00001 0.0000069539
 ....... direction cosines    -0.70711     0.70711 0.0000016390

LINE 5
 ....... from 4 to 5
 ....... equation    1.00000 x +     0.00000 z +     0.00000
 ....... vanishing point infinity infinity
 ....... direction cosines     0.00000     0.00000     1.00000

LINE 6
 ....... from 5 to 6
 ....... equation    1.00000 x +    -0.83333 z +    -3.00000
 ....... vanishing point    3.00001 0.000012016
 ....... direction cosines     0.70711     0.70711 0.0000028323

LINE 7
 ....... from 6 to 3
 ....... equation    1.00000 x +     0.00000 z +    -0.50000
 ....... vanishing point infinity infinity
 ....... direction cosines     0.00000     0.00000     1.00000

LINE 8
 ....... from 1 to 7
 ....... equation    1.00000 x +     0.00000 z +     0.50000
 ....... vanishing point infinity infinity
 ....... direction cosines     0.00000     0.00000     1.00000

LINE 9
```

```
....... from 7 to 5
....... equation    1.00000 x +     0.83333 z +     3.00000
....... vanishing point   -3.00001 0.000012016
....... direction cosines   -0.70711     0.70711 0.0000028323
```

```
PLANE 1
 ....... lines ( 1 2 3 4 )
 ....... vanishing trace
 .......      0.00000 x +     1.00000 z + -0.0000069539
 ....... direction cosines of normal
 .......      0.00000 -0.0000023180     1.00000
 ....... distance from origin unknown

PLANE 2
 ....... lines ( 3 5 6 7 )
 ....... vanishing trace
 .......     1.00000 x +     0.00000 z +    -3.00001
 ....... direction cosines of normal
 .......      0.70711    -0.70711     0.00000
 ....... distance from origin unknown

PLANE 3
 ....... lines ( 2 5 8 9 )
 ....... vanishing trace
 .......      1.00000 x +     0.00000 z +     3.00001
 ....... direction cosines of normal
 .......      0.70711     0.70711     0.00000
 ....... distance from origin unknown

( ( ( parallel ) ) ( ( parallel 1 3 ) ) ( ( parallel 2 4 ) )
( ( parallel 5 8 ) ) ( ( parallel 5 7 ) ) ( ( parallel 7 8 )
) ) ( ( ( perpendicular ) ) ( ( perpendicular 1 2 ) ) ( (
perpendicular 2 3 ) ) ( ( perpendicular 1 4 ) ) ( ( perpen-
dicular 3 4 ) ) ( ( perpendicular 1 5 ) ) ( ( perpendicular
3 5 ) ) ( ( perpendicular 1 6 ) ) ( ( perpendicular 3 6 ) )
( ( perpendicular 1 7 ) ) ( ( perpendicular 3 7 ) ) ( ( per-
pendicular 1 8 ) ) ( ( perpendicular 3 8 ) ) ( ( perpendicu-
lar 1 9 ) ) ( ( perpendicular 3 9 ) ) ( ( perpendicular 2 5
) ) ( ( perpendicular 4 5 ) ) ( ( perpendicular 2 6 ) ) ( (
perpendicular 4 6 ) ) ( ( perpendicular 2 7 ) ) ( ( perpen-
dicular 4 7 ) ) ( ( perpendicular 2 8 ) ) ( ( perpendicular
4 8 ) ) ( ( perpendicular 6 9 ) ) ) ( ( ( circle ) ) )


38 applications of lp
number of true solutions 1
number of total solutions 1
cpu time  570.17999 seconds
```

C.6 INCONSISTENT DATA

When the system was asked to interpret the rectangular parallelipiped image with the condition that faces 2 and 3 were forced to lie in the same plane, it produced a very interesting result. Its result forced the object to be planar! Both the resulting planes: the one containing face 1 and the other containing face 2 were said to be horizontal (parallel to the XY plane). The top face was determined to be the projection of a square, and the remaining lines in the plane were so inclined that their projection matched that in the given image. The result is shown below:

```
Computed focal distance    3.00001


POINT 1
 ....... projection at   -0.50000   -1.50000
 ....... space location unknown unknown unknown

POINT 2
 ....... projection at    0.00000   -1.28571
 ....... space location    0.00000 unknown unknown

POINT 3
 ....... projection at    0.50000   -1.50000
 ....... space location unknown unknown unknown

POINT 4
 ....... projection at    0.00000   -1.80000
 ....... space location    0.00000 unknown unknown

POINT 5
 ....... projection at    0.00000   -3.60000
 ....... space location    0.00000 unknown unknown

POINT 6
 ....... projection at    0.50000   -3.00000
 ....... space location unknown unknown unknown

POINT 7
 ....... projection at   -0.50000   -3.00000
 ....... space location unknown unknown unknown
```

```
LINE 1
 ....... from 1 to 2
 ....... equation   -0.42857 x +     1.00000 z +     1.28571
 ....... vanishing point     3.00001 0.0000069539
 ....... direction cosines     0.70711     0.70711 0.0000016390

LINE 2
 ....... from 2 to 3
 ....... equation    0.42857 x +     1.00000 z +     1.28571
 ....... vanishing point    -3.00001 0.0000069539
 ....... direction cosines    -0.70711     0.70711 0.0000016390

LINE 3
 ....... from 3 to 4
 ....... equation   -0.60000 x +     1.00000 z +     1.80000
 ....... vanishing point     3.00001 0.0000069539
 ....... direction cosines     0.70711     0.70711 0.0000016390

LINE 4
 ....... from 4 to 1
 ....... equation    0.60000 x +     1.00000 z +     1.80000
 ....... vanishing point    -3.00001 0.0000069539
 ....... direction cosines    -0.70711     0.70711 0.0000016390

LINE 5
 ....... from 4 to 5
 ....... equation    1.00000 x +     0.00000 z +     0.00000
 ....... vanishing point     0.00000 0.0000069539
 ....... direction cosines     0.00000     1.00000 0.0000023180

LINE 6
 ....... from 5 to 6
 ....... equation    1.00000 x +    -0.83333 z +    -3.00000
 ....... vanishing point     3.00001 0.0000069539
 ....... direction cosines     0.70711     0.70711 0.0000016390

LINE 7
 ....... from 6 to 3
 ....... equation    1.00000 x +     0.00000 z +    -0.50000
 ....... vanishing point     0.50000 0.0000069539
 ....... direction cosines     0.16440     0.98639 0.0000022864

LINE 8
 ....... from 1 to 7
 ....... equation    1.00000 x +     0.00000 z +     0.50000
 ....... vanishing point    -0.50000 0.0000069539
 ....... direction cosines    -0.16440     0.98639 0.0000022864

LINE 9
```

```
....... from 7 to 5
....... equation    1.00000 x +     0.83333 z +     3.00000
....... vanishing point   -3.00001 0.0000069539
....... direction cosines   -0.70711     0.70711 0.0000016390
```

```
PLANE 1
 ....... lines ( 1 2 4 )
 ....... vanishing trace
 .......      0.00000 x +     1.00000 z + -0.0000069539
 ....... direction cosines of normal
 .......      0.00000 -0.0000023180     1.00000
 ....... distance from origin unknown

PLANE 2
 ....... lines ( 2 3 5 6 7 8 9 )
 ....... vanishing trace
 .......      0.00000 x +     1.00000 z + -0.0000069539
 ....... direction cosines of normal
 .......      0.00000 -0.0000023180     1.00000
 ....... distance from origin unknown

( ( ( parallel ) ) ( ( parallel 1 3 ) ) ( ( parallel 2 4 ) )
) ( ( ( perpendicular ) ) ( ( perpendicular 1 2 ) ) ( ( per-
pendicular 2 3 ) ) ( ( perpendicular 1 4 ) ) ( ( perpendicu-
lar 3 4 ) ) ( ( perpendicular 1 6 ) ) ( ( perpendicular 3 6
) ) ( ( perpendicular 1 9 ) ) ( ( perpendicular 3 9 ) ) ( (
perpendicular 4 6 ) ) ( ( perpendicular 2 6 ) ) ( ( perpen-
dicular 4 9 ) ) ( ( perpendicular 2 9 ) ) ) ( ( ( circle ) )
)


61 applications of lp
number of true solutions 1
number of total solutions 1
cpu time 1064.48010 seconds
```

C.7 COMPLEX TABLE IMAGE

The results for the complex table image are shown below. Note that although there were 652 levels in the tree, only 124 applications of the consistency checker were required.

```
Computed focal distance     4.474056


POINT 1
 ....... projection at    -2.50000    -2.00000
 ....... space location unknown unknown unknown

POINT 2
 ....... projection at    -4.65000    -3.75000
 ....... space location unknown unknown unknown

POINT 3
 ....... projection at     2.70000    -3.75000
 ....... space location unknown unknown unknown

POINT 4
 ....... projection at     1.90000    -2.00000
 ....... space location unknown unknown unknown

POINT 5
 ....... projection at    -3.75000    -3.90000
 ....... space location unknown unknown unknown

POINT 6
 ....... projection at    -3.75000    -9.30000
 ....... space location unknown unknown unknown

POINT 7
 ....... projection at    -3.25000    -9.30000
 ....... space location unknown unknown unknown

POINT 8
 ....... projection at    -3.25000    -7.70000
 ....... space location unknown unknown unknown

POINT 9
 ....... projection at     1.35000    -7.70000
 ....... space location unknown unknown unknown

POINT 10
 ....... projection at     1.35000    -9.30000
 ....... space location unknown unknown unknown

POINT 11
 ....... projection at     1.80000    -9.30000
 ....... space location unknown unknown unknown

POINT 12
 ....... projection at     1.80000    -3.90000
 ....... space location unknown unknown unknown
```

```
POINT 13
 ....... projection at    1.35000   -3.90000
 ....... space location unknown unknown unknown

POINT 14
 ....... projection at    1.35000   -7.40000
 ....... space location unknown unknown unknown

POINT 15
 ....... projection at   -3.25000   -7.40000
 ....... space location unknown unknown unknown

POINT 16
 ....... projection at   -3.25000   -3.90000
 ....... space location unknown unknown unknown

POINT 17
 ....... projection at   -3.20000   -6.50000
 ....... space location unknown unknown unknown

POINT 18
 ....... projection at   -2.10000   -4.35000
 ....... space location unknown unknown unknown

POINT 19
 ....... projection at   -1.85000   -4.35000
 ....... space location unknown unknown unknown

POINT 20
 ....... projection at   -1.85000   -3.95000
 ....... space location unknown unknown unknown

POINT 21
 ....... projection at   -1.85000   -5.00000
 ....... space location unknown unknown unknown

POINT 22
 ....... projection at    1.00000   -4.35000
 ....... space location unknown unknown unknown

POINT 23
 ....... projection at   -2.10000   -4.50000
 ....... space location unknown unknown unknown

POINT 24
 ....... projection at    1.00000   -4.50000
 ....... space location unknown unknown unknown

POINT 25
```

```
 ....... projection at    0.70000   -3.95000
 ....... space location unknown unknown unknown

POINT 26
 ....... projection at    0.70000   -5.00000
 ....... space location unknown unknown unknown

POINT 27
 ....... projection at    0.75000   -4.35000
 ....... space location unknown unknown unknown

POINT 28
 ....... projection at    1.30000   -6.60000
 ....... space location unknown unknown unknown

POINT 29
 ....... projection at    1.30000   -5.30000
 ....... space location unknown unknown unknown

POINT 30
 ....... projection at   -5.50000   -5.50000
 ....... space location unknown unknown unknown

POINT 31
 ....... projection at   -5.50000    1.30000
 ....... space location unknown unknown unknown

POINT 32
 ....... projection at   -4.15000    1.00000
 ....... space location unknown unknown unknown

POINT 33
 ....... projection at   -4.15000   -3.30000
 ....... space location unknown unknown unknown

POINT 34
 ....... projection at   -3.90000    0.90000
 ....... space location unknown unknown unknown

POINT 35
 ....... projection at   -3.90000   -3.10000
 ....... space location unknown unknown unknown

POINT 36
 ....... projection at   -3.05000    0.75000
 ....... space location unknown unknown unknown

POINT 37
 ....... projection at   -3.05000   -2.40000
 ....... space location unknown unknown unknown
```

```
POINT 38
 ....... projection at   -9.20000   -9.20000
 ....... space location unknown unknown unknown

POINT 39
 ....... projection at   -9.20000    2.10000
 ....... space location unknown unknown unknown
```

```
LINE 1
 ....... from 1 to 2
 ....... equation   -0.81395 x +    1.00000 z +  -0.034884
 ....... vanishing point    0.70678    0.61017
 ....... direction cosines  0.58427    0.63576    0.50441

LINE 2
 ....... from 2 to 3
 ....... equation    0.00000 x +    1.00000 z +   3.75000
 ....... vanishing point infinity infinity
 ....... direction cosines  1.00000    0.00000    0.00000

LINE 3
 ....... from 3 to 4
 ....... equation    1.00000 x +    0.45714 z +  -0.98571
 ....... vanishing point    0.70678    0.61017
 ....... direction cosines  0.58427    0.63576    0.50441

LINE 4
 ....... from 4 to 1
 ....... equation    0.00000 x +    1.00000 z +   2.00000
 ....... vanishing point infinity infinity
 ....... direction cosines  1.00000    0.00000    0.00000

LINE 5
 ....... from 5 to 6
 ....... equation    1.00000 x +    0.00000 z +   3.75000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000    0.00000    1.00000

LINE 6
 ....... from 7 to 8
 ....... equation    1.00000 x +    0.00000 z +   3.25000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000    0.00000    1.00000

LINE 7
 ....... from 8 to 9
 ....... equation    0.00000 x +    1.00000 z +   7.70000
 ....... vanishing point infinity infinity
 ....... direction cosines  1.00000    0.00000    0.00000

LINE 8
 ....... from 9 to 10
 ....... equation    1.00000 x +    0.00000 z +  -1.35000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000    0.00000    1.00000

LINE 9
```

```
 ....... from 11 to 12
 ....... equation    1.00000 x +    0.00000 z +   -1.80000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000    0.00000    1.00000

LINE 10
 ....... from 13 to 14
 ....... equation    1.00000 x +    0.00000 z +   -1.35000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000    0.00000    1.00000

LINE 11
 ....... from 14 to 15
 ....... equation    0.00000 x +    1.00000 z +    7.40000
 ....... vanishing point infinity infinity
 ....... direction cosines  1.00000    0.00000    0.00000

LINE 12
 ....... from 15 to 16
 ....... equation    1.00000 x +    0.00000 z +    3.25000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000    0.00000    1.00000

LINE 13
 ....... from 17 to 18
 ....... equation    1.00000 x +   -0.51163 z +   -0.12558
 ....... vanishing point    0.33117    0.40183
 ....... direction cosines  0.35657    0.82805    0.43265

LINE 14
 ....... from 15 to 19
 ....... equation    1.00000 x +   -0.45902 z +   -0.14672
 ....... vanishing point    0.33117    0.40183
 ....... direction cosines  0.35657    0.82805    0.43265

LINE 15
 ....... from 20 to 21
 ....... equation    1.00000 x +    0.00000 z +    1.85000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000    0.00000    1.00000

LINE 16
 ....... from 18 to 22
 ....... equation    0.00000 x +    1.00000 z +    4.35000
 ....... vanishing point infinity infinity
 ....... direction cosines  1.00000    0.00000    0.00000

LINE 17
 ....... from 23 to 24
 ....... equation    0.00000 x +    1.00000 z +    4.50000
```

```
 ....... vanishing point infinity infinity
 ....... direction cosines  1.00000     0.00000     0.00000

LINE 18
 ....... from 25 to 26
 ....... equation    1.00000 x +     0.00000 z +    -0.70000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000     0.00000     1.00000

LINE 19
 ....... from 27 to 28
 ....... equation    1.00000 x +     0.24444 z +     0.31333
 ....... vanishing point   -0.10656    -0.84590
 ....... direction cosines-0.092806     0.66980    -0.73672

LINE 20
 ....... from 22 to 29
 ....... equation    1.00000 x +     0.31579 z +     0.37368
 ....... vanishing point   -0.10656    -0.84590
 ....... direction cosines-0.092806     0.66980    -0.73672

LINE 21
 ....... from 30 to 31
 ....... equation    1.00000 x +     0.00000 z +     5.50000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000     0.00000     1.00000

LINE 22
 ....... from 32 to 33
 ....... equation    1.00000 x +     0.00000 z +     4.15000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000     0.00000     1.00000

LINE 23
 ....... from 34 to 35
 ....... equation    1.00000 x +     0.00000 z +     3.90000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000     0.00000     1.00000

LINE 24
 ....... from 36 to 37
 ....... equation    1.00000 x +     0.00000 z +     3.05000
 ....... vanishing point infinity infinity
 ....... direction cosines  0.00000     0.00000     1.00000

LINE 25
 ....... from 30 to 38
 ....... equation    1.00000 x +    -1.00000 z +     0.00000
 ....... vanishing point unknown unknown
 ....... direction cosines unknown unknown unknown
```

LINE 26
 ..... from 32 to 39
 ..... equation 0.21782 x + 1.00000 z + -0.096039
 ..... vanishing point unknown unknown
 ..... direction cosines unknown unknown unknown

```
ARC 1
 ....... from 31 to 32
 ....... equation   1.00000 x2 +  0.01302 xz + 0.08217 z2
 .......          + 9.4580748 x + -0.038023 z +  22.47287

ARC 2
 ....... from 35 to 36
 ....... equation ************************

PLANE 1
 ....... lines ( 1 2 3 4 )
 ....... vanishing trace
 .......     0.00000 x +    1.00000 z +   -0.61017
 ....... direction cosines of normal
 .......     0.00000   -0.62153    0.78339
 ....... distance from origin unknown

PLANE 2
 ....... lines ( 5 6 7 8 9 10 11 12 )
 ....... vanishing trace
 .......      infinity x + infinity z + infinity
 ....... direction cosines of normal
 .......      0.00000    1.00000    0.00000
 ....... distance from origin unknown

PLANE 3
 ....... lines ( 21 22 23 24 25 26 )
 ....... arcs  ( 1 2 )
 ....... vanishing trace
 .......      unknown x + unknown z + unknown
 ....... direction cosines of normal
 .......      1.000000 0.000000 0.00000
 ....... distance from origin unknown
```

(((parallel)) ((parallel 1 3)) ((parallel 2 4)) ((parallel 2 7)) ((parallel 4 7)) ((parallel 2 11)) ((parallel 4 11)) ((parallel 7 11)) ((parallel 2 16)) ((parallel 4 16)) ((parallel 7 16)) ((parallel 11 16)) ((parallel 2 17)) ((parallel 4 17)) ((parallel 7 17)) ((parallel 11 17)) ((parallel 16 17)) ((parallel 5 6)) ((parallel 5 8)) ((parallel 6 8)) ((parallel 5 9)) ((parallel 6 9)) ((parallel 5 10)) ((parallel 6 10)) ((parallel 5 12)) ((parallel 6 12)) ((parallel 5 15)) ((parallel 6 15)) ((parallel 8 15)) ((parallel 9 15)) ((parallel 10 15)) ((parallel 12 15)) ((parallel 5 18)) ((parallel 6 18)) ((parallel 8 18)) ((parallel 9 18)) ((parallel 10 18)) ((parallel 12 18)) ((parallel 15 18)) ((parallel 5 21)) ((parallel 6 21)) ((parallel 8 21)) ((parallel 9 21)) ((parallel 10 21)) ((parallel 12 21)) ((parallel 15 21)) ((parallel 18 21)) ((parallel 5 22)) ((parallel 6 22)) ((parallel 8 22)) ((parallel 9 22)) ((parallel 10 22)) ((parallel 12 22)) ((parallel 15 22)) ((parallel 18 22)) ((parallel 21 22)) ((parallel 21 23)) ((parallel 22 23)) ((parallel 21 24)) ((parallel 22 24)) ((parallel 5 23)) ((parallel 6 23)) ((parallel 8 23)) ((parallel 9 23)) ((parallel 10 23)) ((parallel 12 23)) ((parallel 15 23)) ((parallel 18 23)) ((parallel 5 24)) ((parallel 6 24)) ((parallel 8 24)) ((parallel 9 24)) ((parallel 10 24)) ((parallel 12 24)) ((parallel 15 24)) ((parallel 18 24)) ((parallel 23 24)) ((parallel 8 9)) ((parallel 8 10)) ((parallel 9 10)) ((parallel 8 12)) ((parallel 9 12)) ((parallel 10 12)) ((parallel 13 14)) ((parallel 19 20)) ((parallel 25 26))) (((perpendicular)) ((perpendicular 1 2)) ((perpendicular 2 3)) ((perpendicular 1 4)) ((perpendicular 3 4)) ((perpendicular 1 5)) ((perpendicular 3 5)) ((perpendicular 1 6)) ((perpendicular 3 6)) ((perpendicular 1 7)) ((perpendicular 3 7)) ((perpendicular 1 8)) ((perpendicular 3 8)) ((perpendicular 1 9)) ((perpendicular 3 9)) ((perpendicular 1 10)) ((perpendicular 3 10)) ((perpendicular 1 11)) ((perpendicular 3 11)) ((perpendicular 1 12)) ((perpendicular 3 12)) ((perpendicular 1 13)) ((perpendicular 3 13)) ((perpendicular 1 14)) ((perpendicular 3 14)) ((perpendicular 1 15)) ((perpendicular 3 15)) ((perpendicular 1 16)) ((perpendicular 3 16)) ((perpendicular 1 17)) ((perpendicular 3 17)) ((perpendicular 1 18)) ((perpendicular 3 18)) ((perpendicular 1 19)) ((perpendicular 3 19)) ((perpendicular 1 20)) ((perpendicular 3 20)) ((perpendicular 1 21)

) ((perpendicular 3 21)) ((perpendicular 1 22)) ((perpendicular 3 22)) ((perpendicular 1 23)) ((perpendicular 3 23)) ((perpendicular 1 24)) ((perpendicular 3 24)) ((perpendicular 1 25)) ((perpendicular 3 25)) ((perpendicular 1 26)) ((perpendicular 3 26)) ((perpendicular 2 5)) ((perpendicular 4 5)) ((perpendicular 2 6)) ((perpendicular 4 6)) ((perpendicular 5 7)) ((perpendicular 6 7)) ((perpendicular 2 8)) ((perpendicular 4 8)) ((perpendicular 7 8)) ((perpendicular 2 9)) ((perpendicular 4 9)) ((perpendicular 7 9)) ((perpendicular 2 10)) ((perpendicular 4 10)) ((perpendicular 7 10)) ((perpendicular 5 11)) ((perpendicular 6 11)) ((perpendicular 8 11)) ((perpendicular 9 11)) ((perpendicular 10 11)) ((perpendicular 2 12)) ((perpendicular 4 12)) ((perpendicular 7 12)) ((perpendicular 11 12)) ((perpendicular 2 13)) ((perpendicular 4 13)) ((perpendicular 7 13)) ((perpendicular 11 13)) ((perpendicula 2 14)) ((perpendicular 4 14)) ((perpendicular 7 14)) ((perpendicular 11 14)) ((perpendicular 2 15)) ((perpendicula 4 15)) ((perpendicular 7 15)) ((perpendicular 11 15)) ((perpendicular 5 16)) ((perpendicular 6 16)) ((perpendicular 8 16)) ((perpendicular 9 16)) ((perpendicular 10 16)) ((perpendicular 12 16)) ((perpendicular 13 16)) ((perpendicul 14 16)) ((perpendicular 15 16)) ((perpendicular 5 17)) ((perpendicular 6 17)) ((perpendicular 8 17)) ((perpendicular 9 17)) ((perpendicular 10 17)) ((perpendicular 12 17)) ((perpendicular 13 17)) ((perpendicular 14 17)) ((perpendicul 15 17)) ((perpendicular 2 18)) ((perpendicular 4 18)) ((perpendicular 7 18)) ((perpendicular 11 18)) ((perpendicula 16 18)) ((perpendicular 17 18)) ((perpendicular 2 19)) ((perpendicular 4 19)) ((perpendicular 7 19)) ((perpendicular 11 19)) ((perpendicular 16 19)) ((perpendicular 17 19)) ((perpendicular 2 20)) ((perpendicular 4 20)) ((perpendicular 7 20)) ((perpendicular 11 20)) ((perpendicular 16 20)) ((perpendicular 17 20)) ((perpendicular 2 21)) ((perpendicula 4 21)) ((perpendicular 7 21)) ((perpendicular 11 21)) ((perpendicular 16 21)) ((perpendicular 17 21)) ((perpendicul 2 22)) ((perpendicular 4 22)) ((perpendicular 7 22)) ((perpendicular 11 22)) ((perpendicular 16 22)) ((perpendicul 17 22)) ((perpendicular 2 23)) ((perpendicular 4 23)) ((perpendicular 7 23)) ((perpendicular 11 23)) ((perpendicula 16 23)) ((perpendicular 17 23)) ((perpendicular 2 24)) ((perpendicular 4 24)) ((perpendicular 7 24)) ((perpendicular 11 24)) ((perpendicular 16 24)) ((perpendicular 17 24)) ((

perpendicular 2 25)) ((perpendicular 4 25)) ((perpendicular 7 25)) ((perpendicular 11 25)) ((perpendicular 16 25)) ((perpendicular 17 25)) ((perpendicular 2 26)) ((perpendicula 4 26)) ((perpendicular 7 26)) ((perpendicular 11 26)) ((perpendicular 16 26)) ((perpendicular 17 26)) ((perpendicul 5 13)) ((perpendicular 6 13)) ((perpendicular 8 13)) ((perpendicular 9 13)) ((perpendicular 10 13)) ((perpendicula 12 13)) ((perpendicular 5 14)) ((perpendicular 6 14)) ((perpendicular 8 14)) ((perpendicular 9 14)) ((perpendicular 10 14)) ((perpendicular 12 14)) ((perpendicular 13 15)) ((perpendicular 14 15)) ((perpendicular 13 18)) ((perpendicul 14 18)) ((perpendicular 5 19)) ((perpendicular 6 19)) ((perpendicular 8 19)) ((perpendicular 9 19)) ((perpendicular 10 19)) ((perpendicular 12 19)) ((perpendicular 15 19)) ((perpendicular 18 19)) ((perpendicular 5 20)) ((perpendicula 6 20)) ((perpendicular 8 20)) ((perpendicular 9 20)) ((perpendicular 10 20)) ((perpendicular 12 20)) ((perpendicul 15 20)) ((perpendicular 18 20)) ((perpendicular 13 21)) ((perpendicular 14 21)) ((perpendicular 19 21)) ((perpendicul 20 21)) ((perpendicular 13 22)) ((perpendicular 14 22)) ((perpendicular 19 22)) ((perpendicular 20 22)) ((perpendicul 13 23)) ((perpendicular 14 23)) ((perpendicular 19 23)) ((perpendicular 20 23)) ((perpendicular 13 24)) ((perpendicul 14 24)) ((perpendicular 19 24)) ((perpendicular 20 24)) ((perpendicular 5 25)) ((perpendicular 6 25)) ((perpendicular 8 25)) ((perpendicular 9 25)) ((perpendicular 10 25)) ((perpendicular 12 25)) ((perpendicular 15 25)) ((perpendicul 18 25)) ((perpendicular 21 25)) ((perpendicular 22 25)) ((perpendicular 23 25)) ((perpendicular 24 25)) ((perpendicul 5 26)) ((perpendicular 6 26)) ((perpendicular 8 26)) ((perpendicular 9 26)) ((perpendicular 10 26)) ((perpendicula 12 26)) ((perpendicular 15 26)) ((perpendicular 18 26)) ((perpendicular 21 26)) ((perpendicular 22 26)) ((perpendicul 23 26)) ((perpendicular 24 26)) ((perpendicular 13 19)) ((perpendicular 14 19)) ((perpendicular 13 20)) ((perpendicul 14 20)) ((perpendicular 13 25)) ((perpendicular 14 25)) ((perpendicular 13 26)) ((perpendicular 14 26)) ((perpendicul 19 25)) ((perpendicular 20 25)) ((perpendicular 19 26)) ((perpendicular 20 26))) (((circle)) ((circle 1)) ((circle 2)))

124 applications of lp

```
number of true solutions 1
number of total solutions 1
cpu time 21836.61133 seconds
```

C.8 RESULTS FOR THE TOWER IMAGE

In this section, we show the results for the tower image presented in Appendix B. Again note that although there were 468 levels in the tree, only 168 applications of the consistency checker were required.

```
Computed focal distance     3.00000


POINT 1
 ....... projection at    -0.60000    -6.00000
 ....... space location unknown unknown unknown

POINT 2
 ....... projection at    -0.60000    -1.80000
 ....... space location unknown unknown unknown

POINT 3
 ....... projection at    -0.89736    -5.25661
 ....... space location unknown unknown unknown

POINT 4
 ....... projection at    -0.89736    -1.57698
 ....... space location unknown unknown unknown

POINT 5
 ....... projection at    -0.46771    -1.40314
 ....... space location unknown unknown unknown

POINT 6
 ....... projection at    -0.15397    -1.57698
 ....... space location unknown unknown unknown

POINT 7
 ....... projection at    -0.15397    -5.25661
 ....... space location unknown unknown unknown

POINT 8
 ....... projection at    -0.46773    -2.80636
 ....... space location unknown unknown unknown

POINT 9
 ....... projection at     0.21593    -2.25000
 ....... space location unknown unknown unknown

POINT 10
 ....... projection at    -0.15397    -3.15397
 ....... space location unknown unknown unknown

POINT 11
 ....... projection at     0.53185    -2.46815
 ....... space location unknown unknown unknown

POINT 12
 ....... projection at    -0.15397    -3.67963
 ....... space location unknown unknown unknown
```

```
POINT 13
 ....... projection at    0.53185    -2.87951
 ....... space location unknown unknown unknown

POINT 14
 ....... projection at    0.21593    -1.12500
 ....... space location unknown unknown unknown

POINT 15
 ....... projection at    0.21593    -3.75000
 ....... space location unknown unknown unknown

POINT 16
 ....... projection at    0.53185    -4.11359
 ....... space location unknown unknown unknown

POINT 17
 ....... projection at    0.53185    -1.23408
 ....... space location unknown unknown unknown

POINT 18
 ....... projection at    0.75000    -3.75000
 ....... space location unknown unknown unknown

POINT 19
 ....... projection at    0.75000    -1.12500
 ....... space location unknown unknown unknown

POINT 20
 ....... projection at    0.44546    -1.03364
 ....... space location unknown unknown unknown
```

```
LINE 1
 ....... from 1 to 2
 ....... equation    1.00000 x +     0.00000 z +     0.60000
 ....... vanishing point infinity infinity
 ....... direction cosines     0.00000     0.00000     1.00000

LINE 2
 ....... from 1 to 3
 ....... equation    1.00000 x +     0.40000 z +     3.00000
 ....... vanishing point    -3.00000 -0.34060E-06
 ....... direction cosines    -0.70711     0.70711 -0.80280E-07

LINE 3
 ....... from 3 to 4
 ....... equation    1.00000 x +     0.00000 z +     0.89736
 ....... vanishing point infinity infinity
 ....... direction cosines     0.00000     0.00000     1.00000

LINE 4
 ....... from 4 to 2
 ....... equation    0.75000 x +     1.00000 z +     2.25000
 ....... vanishing point    -3.00000 -0.34060E-06
 ....... direction cosines    -0.70711     0.70711 -0.80280E-07

LINE 5
 ....... from 4 to 5
 ....... equation   -0.40463 x +     1.00000 z +     1.21389
 ....... vanishing point     3.00000 -0.34060E-06
 ....... direction cosines     0.70711     0.70711 -0.80280E-07

LINE 6
 ....... from 5 to 6
 ....... equation    0.55410 x +     1.00000 z +     1.66230
 ....... vanishing point    -3.00000 -0.34060E-06
 ....... direction cosines    -0.70711     0.70711 -0.80280E-07

LINE 7
 ....... from 6 to 2
 ....... equation   -0.50000 x +     1.00000 z +     1.50000
 ....... vanishing point     3.00000 -0.34060E-06
 ....... direction cosines     0.70711     0.70711 -0.80280E-07

LINE 8
 ....... from 6 to 7
 ....... equation    1.00000 x +     0.00000 z +     0.15397
 ....... vanishing point infinity infinity
 ....... direction cosines     0.00000     0.00000     1.00000

LINE 9
 ....... from 7 to 1
```

```
 ....... equation    1.00000 x +    -0.60000 z +    -3.00000
 ....... vanishing point    3.00000 -0.34060E-06
 ....... direction cosines    0.70711    0.70711 -0.80280E-07

LINE 10
 ....... from 8 to 9
 ....... equation   -0.81381 x +     1.00000 z +     2.42572
 ....... vanishing point    2.92163  -0.048075
 ....... direction cosines    0.69764    0.71636  -0.011480

LINE 11
 ....... from 10 to 11
 ....... equation   -1.00000 x +     1.00000 z +     3.00000
 ....... vanishing point    3.00000 -0.0000036041
 ....... direction cosines    0.70711    0.70711 -0.84949E-06

LINE 12
 ....... from 12 to 13
 ....... equation    1.00000 x +    -0.85714 z +    -3.00000
 ....... vanishing point    3.00000    0.00000
 ....... direction cosines    0.70711    0.70711    0.00000

LINE 13
 ....... from 14 to 15
 ....... equation    1.00000 x +     0.00000 z +    -0.21593
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 14
 ....... from 9 to 11
 ....... equation    0.69053 x +     1.00000 z +     2.10090
 ....... vanishing point unknown unknown
 ....... direction cosines unknown unknown unknown

LINE 15
 ....... from 15 to 16
 ....... equation    1.00000 x +     0.86889 z +     3.04243
 ....... vanishing point unknown unknown
 ....... direction cosines unknown unknown unknown

LINE 16
 ....... from 16 to 17
 ....... equation    1.00000 x +     0.00000 z +    -0.53185
 ....... vanishing point infinity infinity
 ....... direction cosines    0.00000    0.00000    1.00000

LINE 17
 ....... from 16 to 18
 ....... equation    1.00000 x +    -0.60000 z +    -3.00000
 ....... vanishing point    3.00000 -0.0000023614
```

```
 ....... direction cosines     0.70711     0.70711 -0.55659E-06

LINE 18
 ....... from 18 to 19
 ....... equation     1.00000 x +     0.00000 z +    -0.75000
 ....... vanishing point infinity infinity
 ....... direction cosines     0.00000     0.00000     1.00000

LINE 19
 ....... from 17 to 19
 ....... equation    -0.50000 x +     1.00000 z +     1.50000
 ....... vanishing point     3.00000     0.00000
 ....... direction cosines     0.70711     0.70711     0.00000

LINE 20
 ....... from 14 to 20
 ....... equation    -0.39802 x +     1.00000 z +     1.21094
 ....... vanishing point     2.92163    -0.048075
 ....... direction cosines     0.69764     0.71636    -0.011480

LINE 21
 ....... from 14 to 17
 ....... equation     0.34527 x +     1.00000 z +     1.05045
 ....... vanishing point    -3.32368     0.097106
 ....... direction cosines    -0.74215     0.66988     0.021683

LINE 22
 ....... from 19 to 20
 ....... equation     0.30000 x +     1.00000 z +     0.90000
 ....... vanishing point    -3.32368     0.097106
 ....... direction cosines    -0.74215     0.66988     0.021683
```

```
ARC 1
 ....... from 0 to 0
 ....... equation  1.00000 x2 + -0.66667 xz + 145.3687 z2
 .......         + 0.00000 x  + 11.42090 z  + 9.00000

ARC 2
 ....... from 0 to 0
 ....... equation  1.00000 x2 + 0.86293 xz + 7.2691 z2
 .......         + 0.00000 x  + 16.0000 z  + 9.00000

ARC 3
 ....... from 0 to 0
 ....... equation  1.00000 x2 + -0.09763 xz + 6.92063 z2
 .......         + 0.00000 x  + 6.4142 z + 9.00000

ARC 4
 ....... from 0 to 0
 ....... equation  1.00000 x2 + 0.19527 xz + 1.47999 z2
 .......         + 0.00000 x  + 7.2929 z + 9.00000


PLANE 1
 ....... lines ( 1 2 3 4 )
 ....... vanishing trace
 .......      1.00000 x +    0.00000 z +    3.00000
 ....... direction cosines of normal
 .......      0.70711    0.70711    0.00000
 ....... distance from origin unknown

PLANE 2
 ....... lines ( 1 7 8 9 )
 ....... vanishing trace
 .......      1.00000 x +    0.00000 z +   -3.00000
 ....... direction cosines of normal
 .......      0.70711   -0.70711    0.00000
 ....... distance from origin unknown

PLANE 3
 ....... lines ( 4 5 6 7 )
 ....... arcs  ( 3 )
 ....... vanishing trace
 .......      0.00000 x +    1.00000 z + 0.34060E-06
 ....... direction cosines of normal
 .......      0.00000 0.11353E-06    1.00000
 ....... distance from origin unknown

PLANE 4
 ....... lines ( 8 11 12 16 )
 ....... vanishing trace
 .......       1.00000 x +    0.00000 z +   -3.00000
```

```
 ....... direction cosines of normal
 .......       0.70711   -0.70711    0.00000
 ....... distance from origin unknown

PLANE 5
 ....... lines ( 10 11 14 )
 ....... arcs  ( 3 4 )
 ....... vanishing trace unknown x + unknown z + unknown
 ....... direction cosines of normal unknown unknown unknown
 ....... distance from origin unknown

PLANE 6
 ....... lines ( 13 14 15 16 )
 ....... vanishing trace unknown x + unknown z + unknown
 ....... direction cosines of normal unknown unknown unknown
 ....... distance from origin unknown

PLANE 7
 ....... lines ( 16 17 18 19 )
 ....... vanishing trace
 .......       1.00000 x +    0.00000 z +   -3.00000
 ....... direction cosines of normal
 .......       0.70711   -0.70711    0.00000
 ....... distance from origin unknown

PLANE 8
 ....... lines ( 19 20 21 22 )
 ....... arcs  ( 2 )
 ....... vanishing trace
 .......       0.00000 x +    1.00000 z +    0.00000
 ....... direction cosines of normal
 .......       0.00000    0.00000    1.00000
 ....... distance from origin unknown
```

(((parallel)) ((parallel 1 3)) ((parallel 2 4)) ((parallel 1 8)) ((parallel 3 8)) ((parallel 7 9)) ((parallel 1 13)) ((parallel 3 13)) ((parallel 8 13)) ((parallel 1 16)) ((parallel 3 16)) ((parallel 8 16)) ((parallel 13 16)) ((parallel 1 18)) ((parallel 3 18)) ((parallel 8 18)) ((parallel 13 18)) ((parallel 16 18)) ((parallel 5 11)) ((parallel 5 12)) ((parallel 11 12)) ((parallel 5 17)) ((parallel 11 17)) ((parallel 12 17)) ((parallel 10 20)) ((parallel 21 22))) (((perpendicular)) ((perpendicular 1 2)) ((perpendicular 2 3)) ((perpendicular 1 4)) ((perpendicular 3 4)) ((perpendicular 1 5)) ((perpendicular 3 5)) ((perpendicular 1 6)) ((perpendicular 3 6)) ((perpendicular 1 7)) ((perpendicular 3 7)) ((perpendicular 2 8)) ((perpendicular 4 8)) ((perpendicular 5 8)) ((perpendicular 6 8)) ((perpendicular 7 8)) ((perpendicular 1 9)) ((perpendicular 3 9)) ((perpendicular 8 9)) ((perpendicular 1 10)) ((perpendicular 3 10)) ((perpendicular 8 10)) ((perpendicular 1 11)) ((perpendicular 3 11)) ((perpendicular 8 11)) ((perpendicular 1 12)) ((perpendicular 3 12)) ((perpendicular 8 12)) ((perpendicular 2 13)) ((perpendicular 4 13)) ((perpendicular 5 13)) ((perpendicular 6 13)) ((perpendicular 7 13)) ((perpendicular 9 13)) ((perpendicular 10 13)) ((perpendicular 11 13)) ((perpendicular 12 13)) ((perpendicul 1 14)) ((perpendicular 3 14)) ((perpendicular 8 14)) ((perpendicular 13 14)) ((perpendicular 1 15)) ((perpendicula 3 15)) ((perpendicular 8 15)) ((perpendicular 13 15)) ((perpendicular 2 16)) ((perpendicular 4 16)) ((perpendicular 5 16)) ((perpendicular 6 16)) ((perpendicular 7 16)) ((perpendicular 9 16)) ((perpendicular 10 16)) ((perpendicula 11 16)) ((perpendicular 12 16)) ((perpendicular 14 16)) ((perpendicular 15 16)) ((perpendicular 1 17)) ((perpendicula 3 17)) ((perpendicular 8 17)) ((perpendicular 13 17)) ((perpendicular 16 17)) ((perpendicular 2 18)) ((perpendicula 4 18)) ((perpendicular 5 18)) ((perpendicular 6 18)) ((perpendicular 7 18)) ((perpendicular 9 18)) ((perpendicular 10 18)) ((perpendicular 11 18)) ((perpendicular 12 18)) ((perpendicular 14 18)) ((perpendicular 15 18)) ((perpendicul 17 18)) ((perpendicular 1 19)) ((perpendicular 3 19)) ((perpendicular 8 19)) ((perpendicular 13 19)) ((perpendicula 16 19)) ((perpendicular 18 19)) ((perpendicular 1 20)) ((perpendicular 3 20)) ((perpendicular 8 20)) ((perpendicular 13 20)) ((perpendicular 16 20)) ((perpendicular 18 20)) ((perpendicular 1 21)) ((perpendicular 3 21

)) ((perpendicular 8 21)) ((perpendicular 13 21)) ((perpendicular 16 21)) ((perpendicular 18 21)) ((perpendicular 1 22)) ((perpendicula 3 22)) ((perpendicular 8 22)) ((perpendicular 13 22)) ((perpendicular 16 22)) ((perpendicular 18 22)) ((perpendicul 2 5)) ((perpendicular 4 5)) ((perpendicular 2 6)) ((perpendicular 4 6)) ((perpendicular 2 7)) ((perpendicular 4 7)) ((perpendicular 2 9)) ((perpendicular 2 10)) ((perpendicular 4 10)) ((perpendicular 2 11)) ((perpendicular 4 11)) ((perpendicular 2 12)) ((perpendicular 4 12)) ((perpendicular 2 14)) ((perpendicular 4 14)) ((perpendicular 2 15)) ((perpendicular 4 15)) ((perpendicular 2 17)) ((perpendicular 4 17)) ((perpendicular 2 19)) ((perpendicular 4 19)) ((perpendicular 2 20)) ((perpendicular 4 20)) ((perpendicular 2 21)) ((perpendicular 4 21)) ((perpendicular 2 22)) ((perpendicular 4 22)) ((perpendicular 4 9)) ((perpendicular 5 10)) ((perpendicular 10 11)) ((perpendicula 10 12)) ((perpendicular 5 14)) ((perpendicular 11 14)) ((perpendicular 12 14)) ((perpendicular 5 15)) ((perpendicula 11 15)) ((perpendicular 12 15)) ((perpendicular 10 17)) ((perpendicular 14 17)) ((perpendicular 15 17)) ((perpendicul 5 19)) ((perpendicular 11 19)) ((perpendicular 12 19)) ((perpendicular 17 19)) ((perpendicular 5 20)) ((perpendicula 11 20)) ((perpendicular 12 20)) ((perpendicular 17 20)) ((perpendicular 5 21)) ((perpendicular 11 21)) ((perpendicula 12 21)) ((perpendicular 17 21)) ((perpendicular 5 22)) ((perpendicular 11 22)) ((perpendicular 12 22)) ((perpendicul 17 22)) ((perpendicular 6 10)) ((perpendicular 6 14)) ((perpendicular 6 15)) ((perpendicular 6 19)) ((perpendicular 6 20)) ((perpendicular 6 21)) ((perpendicular 6 22)) ((perpendicular 7 10)) ((perpendicular 9 10)) ((perpendicular 7 14)) ((perpendicular 9 14)) ((perpendicular 7 15)) ((perpendicular 9 15)) ((perpendicular 7 19)) ((perpendicular 9 19)) ((perpendicular 7 20)) ((perpendicular 9 20)) ((perpendicular 7 21)) ((perpendicular 9 21)) ((perpendicular 7 22)) ((perpendicular 9 22)) ((perpendicular 10 15)) ((perpendicular 10 19)) ((perpendicular 15 20)) ((perpendicul 19 20)) ((perpendicular 10 21)) ((perpendicular 20 21)) ((perpendicular 10 22)) ((perpendicular 20 22)) ((perpendicul 14 19)) ((perpendicular 14 21)) ((perpendicular 14 22)) ((perpendicular 15 19)) ((perpendicular 15 21)) ((perpendicul 15 22))) (((circle)) ((circle 1)) ((circle 2)) ((circle 3)) ((circle 4)))
168 applications of lp
number of true solutions 1

```
number of total solutions 1
cpu time 70237.77344 seconds
```

Appendix D

DATA FOR TIMING EXPERIMENTS

The image used to generate the graphs presented in Chapter V is shown in Figure 43. The corresponding preprocessed data is also shown.

PREPROCESSED DATA

```
( assert
( ( line 1 ( 1 2 )
    (    0.00000    1.00000    1.81985 ) ) )
( ( line3d 1 ( unknown unknown )
    ( unknown unknown unknown ) ) )
( ( line 2 ( 2 3 )
    (    1.00000  -0.024133    0.33154 ) ) )
( ( line3d 2 ( unknown unknown )
    ( unknown unknown unknown ) ) )
( ( line 3 ( 4 1 )
    (    1.00000    0.00000  -0.044119 ) ) )
( ( line3d 3 ( unknown unknown )
    ( unknown unknown unknown ) ) )
( ( line 4 ( 4 3 )
    (    1.00000    0.53740  -0.039773 ) ) )
( ( line3d 4 ( unknown unknown )
    ( unknown unknown unknown ) ) )
( ( point 1 (   0.049900   -1.81985 ) ) )
( ( point3d 1 unknown unknown unknown ) )
( ( point 2 (   -0.37546   -1.81985 ) ) )
( ( point3d 2 unknown unknown unknown ) )
( ( point 3 (   -0.31558    0.66124 ) ) )
( ( point3d 3 unknown unknown unknown ) )
( ( point 4 (   0.044119 -0.0080863 ) ) )
( ( point3d 4 unknown unknown unknown ) )
( ( line_intersection ( 1 2 )
    (   -0.37546   -1.81985 ) ) )
( ( line_intersection ( 2 1 ) yes ) )
( ( line_intersection ( 1 3 )
    (   0.049900   -1.81985 ) ) )
( ( line_intersection ( 3 1 ) yes ) )
( ( line_intersection ( 1 4 )
    (    1.01777   -1.81985 ) ) )
( ( line_intersection ( 4 1 ) no ) )
( ( line_intersection ( 2 3 )
    (   0.044119   15.56638 ) ) )
```

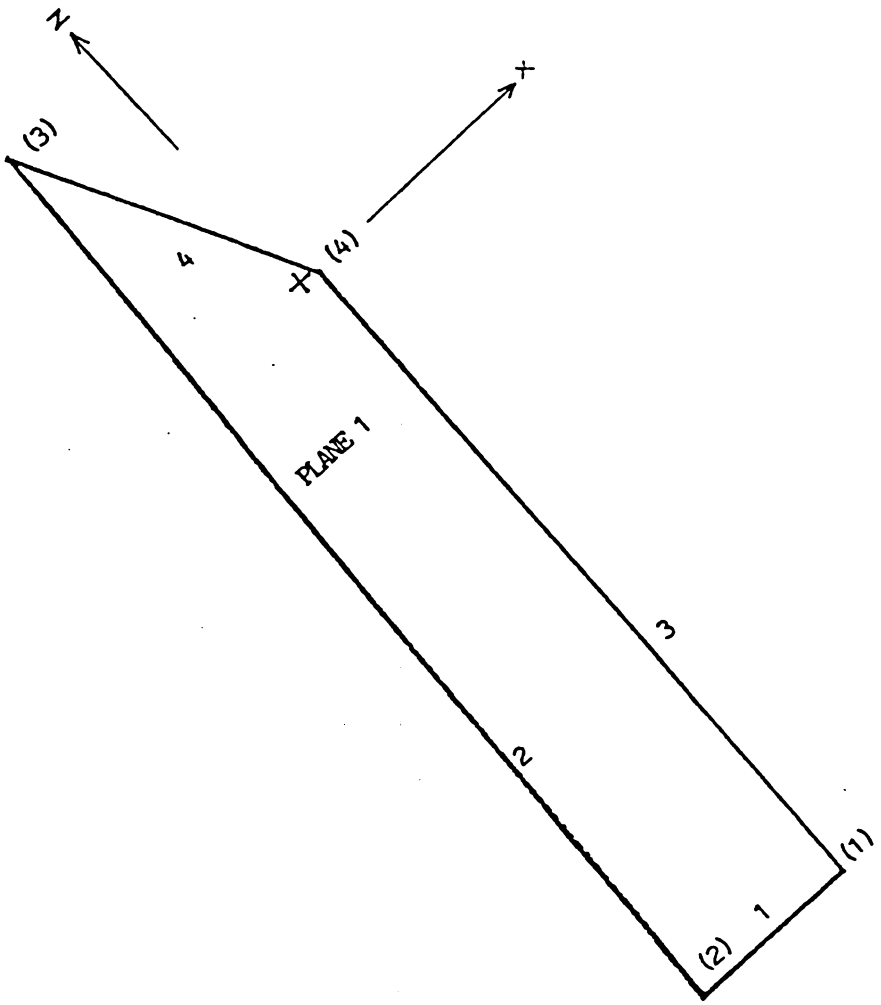


Figure 43: Test data for timing experiments

```
( ( line_intersection ( 3 2 ) no ) )
( ( line_intersection ( 2 4 )
    (   -0.31558    0.66124 ) ) )
( ( line_intersection ( 4 2 ) yes ) )
( ( line_intersection ( 3 4 )
    (   0.044119 -0.0080863 ) ) )
( ( line_intersection ( 4 3 ) yes ) )
( ( plane 1 ( 1 2 3 4 ) ( ) ) )
( ( plane3d 1 (unknown unknown unknown)
              (unknown unknown unknown)
               unknown) )
 )
( <-  %numpts 4 )
( <-  %numlins 4 )
( <-  %numplane 1)
```

The vita has been removed from the scanned document