

Dimensional Analyzer
and the
Evident Partiality Legal Domain

Dan Tysver
April 27, 1990

In satisfaction of
Harvard Law School's
Written Work
Requirement

ABSTRACT

This paper discusses the usefulness and flexibility of a dimensional index that is used to retrieve relevant legal cases from a case base. This case-based reasoning technique was first used for the study of the law by Kevin Ashley and Edwina Rissland. In his Ph.D. dissertation, Ashley describes the HYPO program used to implement a dimensional index in the area of trade secrets law. His dissertation examines the usefulness of dimensions by showing how the trade secrets area was successfully represented in HYPO.

Using the descriptions of the HYPO program found in Ashley's dissertation, I have duplicated part of its workings in a program entitled Dimensional Analyzer (or DAn). DAn implements the dimensional index and the argumentation algorithms that HYPO used to generate 3-Ply arguments. Cases in DAn are represented at the dimensional level, as the factual predicate and case frame level of case representation were not deemed important to the study of dimensional indexes. DAn does not suggest Hypotheticals, but it does generate generalization rules that are useful in analyzing a domain and in providing a simple weighting of dimensions that is useful in argument production. In addition, DAn implements a query system which allows the user to directly query the case base.

In testing DAn, I have dimensionally analyzed the area of vacating arbitration awards on the grounds of an arbitrator's evident partiality. The implementation of the domain is discussed in this paper, as well as the inherent problems in converting a case to dimensions. The problems encountered by Simon Mendelson in dimensionally analyzing the area of criminal appeals are also considered, and an alternate analysis of the domain is suggested and implemented in DAn. General conclusions are then made about the adaptability of dimensional analysis to other legal domains.

TABLE OF CONTENTS

	<u>Page</u>
1. INTRODUCTION	1
1.1 Artificial Intelligence and the Law--Rule Based Systems	1
1.2 Case Based Reasoning and the HYPO Program	3
1.3 Introduction to Dimensions	4
2. HYPO AND DIMENSIONAL ANALYSIS	6
2.1 Case Representation: Frames, Predicates, and Dimensions	6
2.2 The Structure of HYPO's Dimensions	7
2.3 Dimensions as Indexes to the Case Base	9
3. ANALYSIS OF DIMENSIONS IN HYPO	12
3.1 Session Manager	13
3.2 Case Analyzer	13
3.3 Case Positioner	14
3.4 Best Case Selector	14
3.5 3-Ply Arguer	15
3.6 Hypo Generator	18
3.7 Explainer	18
4. EVIDENT PARTIALITY	18
4.1 Statutory Framework	19
4.2 Commonwealth Coatings	20
4.3 Reactions to Commonwealth Coatings	22
4.4 Evident Partiality Today	22
5. DIMENSIONAL ASSIGNMENT	24
5.1 Assignment Problems in HYPO	24
5.2 Dimensional Assignment in the Evident Partiality Domain	25
5.3 Example Cases	27
5.3.1 <u>Abramovich v. Pennsylvania Liquor Control Board</u>	27
5.3.2 <u>Richco Structures v. Parkside Village, Inc.</u>	29
6. REPRESENTATION IN PROLOG	33
6.1 Prolog	33
6.1.1 Introduction to Prolog	33
6.1.2 Programming in Prolog	34

6.2 Representation of Dimensions in DAn	38
6.3 Representation of Cases in DAn	40
6.4 Representation of Example Cases in Prolog	42
6.4.1 <u>Abramovich</u>	42
6.4.2 <u>Richco</u>	42
6.5 Fullness of Representation: Cook Industries	43
6.6 "No" Magnitude Dimensions	45
 7. OVERVIEW OF THE DIMENSIONAL ANALYZER PROGRAM	 47
7.1 Purpose	47
7.2 Hypotheticals	48
7.3 Menu System	50
7.4 Introduction to DAn's Modules	50
7.4.1 Domain Statistics	50
7.4.2 New Domain	51
7.4.3 New Cases	52
7.4.4 Domain Analysis and Generalization Rules	53
7.4.4.1 Presumption	53
7.4.4.2 Both-Sided Dimensions	54
7.4.4.3 Necessary Dimensions	54
7.4.4.4 Overwhelming Dimensions	55
7.4.4.5 Overwhelming Dimension Pairs	55
7.4.4.6 Dominated Dimension Pairs	56
7.4.5 Argue, and its Supporting Procedures	57
7.4.6 Query System	61
 8. GENERALIZATION RULES	 62
8.1 Learning through Generalization Rules	62
8.2 Weighting through Generalization Rules	64
8.3 Support Needed	65
 9. EXAMPLE ANALYSIS BY DAn	 65
9.1 Analysis of the Evident Partiality Domain	65
9.2 Analysis of the <u>Richco</u> Fact Situation	67
9.2.1 Arguing the Case	67
9.2.2 Query Along Knowledge-by-Party Dimension	67
9.3 Analysis of the <u>Cook</u> Fact Situation	68
9.4 Analysis of <u>Commonwealth Coatings</u> : Upholding Award	68

10. PARAMETERS	68
10.1 Most-On-Point Criteria	69
10.2 Include/Exclude Generalization Rules in Arguments	69
10.3 Include/Exclude "No" Valued Dimensions in Analysis	70
10.4 Include/Exclude Overlapping Counter-Examples	70
10.5 Support Level Needed	70
10.6 Output Control	70
11. SUTTABILITY OF DOMAINS FOR DIMENSIONS	71
11.1 Evident Partiality Domain	71
11.2 Criminal Appeal Domain	72
12. CONCLUSION	75

LIST OF FIGURES

1. HYPO's Argument for the Structural Dynamics Case
2. Flow of Control in HYPO
3. Information in HYPO's Case Analysis Record
4. HYPO's Case Analyzer
5. HYPO's Case Positioner
6. HYPO's Best Case Selector
7. HYPO's 3-Ply Arguer
8. DAn's Analysis of Evident Partiality Domain
9. DAn's Argument for Richco
10. DAn's Query System Output
11. DAn's Argument for Cook Industries
12. Partial Output of DAn's Argument for Commonwealth Coatings
13. DAn's Analysis of Criminal Appeal Domain
14. DAn's Argument for John Doe Hypothetical
15. DAn's Argument for Jane Roe Hypothetical

LIST OF APPENDIXES

- Appendix A: List of Evident Partiality Dimensions (From Query Output)
- Appendix B: List of Evident Partiality Cases (From Query Output)
- Appendix C: Prolog Facts Used to Encode Evident Partiality Domain
- Appendix D: Prolog Facts Used to Encode Evident Partiality Cases
- Appendix E: Prolog Facts used to Encode Criminal Appeal Domain (From New Domain procedure)
- Appendix F: Prolog Facts used to Encode Criminal Appeal Cases (From New Cases procedure)

SECTION 1: INTRODUCTION

1.1 Artificial Intelligence and the Law--Rule Based Systems

The claim that legal expert systems will revolutionize the practice of law is being heard with increasing frequency from both legal and computer professionals from around the country. Even major legal consulting firms are preparing their clients for the coming onslaught of artificial intelligence systems. At a recent seminar on Technology Planning for Law Firms, Jon Klemens from the consulting firm of Altman & Weil suggested that the use of expert systems will be the major change in the practice of law during the 1990s, and will soon be the primary motivator for the placement of terminals and computers on the desks of lawyers.¹ According to Klemens, expert systems have already drastically changed the practice of law in a number of small law firms, who are suddenly able "to compete with anyone."² The academic community is also caught up in the excitement, and research is underway to explore the uses of expert systems.³

Expert systems are often referred to as rule-based systems, in that a set of rules is used to represent a legal domain or issue. Generally, legal situations are analyzed in these programs by asking the user to respond to a series of questions. As the responses are given, the program applies rules and asks additional questions. At the end of the analysis, the system returns "an answer" (ie, "The defendant is liable," or "Form A should be used to create this trust") to the user.

Rule-based systems can easily represent rule-governed activities, as such activities are forced to follow certain rules. The rules need not be obvious or known, yet most people believe that rules in rule-governed activities exist, if only investigators could

¹ Seminar presented by Data General Corporation at their Newton, MA office on March 22, 1990.

² Id.

³ See, e.g., Marilyn T. MacCrimmon, Expert Systems in Case-Based Law: The Hearsay Rule Advisor, in Second International Conference on AI and Law, Boston 1989 (ACM Press) [hereinafter Second International Conference]; Andreas Galtung and Dag Syvert Mæsel, Xcite (an expert system for naturalization cases), in Second International Conference.

accurately distill them from the world and encode them into expert systems. A good example of an expert system in a rule-governed activity is the MYCIN program. The creators of MYCIN believed that the source of an infectious disease could be determined with a high degree of accuracy, if only they had the ability to decipher the rules that exist in nature. Their testing of the system was basically a test of the rules that they had developed, and did not represent a challenge to the assumption that the knowledge being studied was capable of being embodied in a set of rules. As more medical discoveries are made in this area, more rules can be added to the system and the accuracy of the MYCIN system can be improved. Hence, the study of the source of infections is considered a rule-governed activity, and is appropriately the subject of a rule-based system.

However, in spite of the important and useful applications that can be developed with purely rule-based logic, such logic is of limited use for those lawyers and researchers who wish to more accurately represent the law. This is because the legal system is a rule-guided, as opposed to a rule-governed activity. Rule-guided activities such as legal jurisprudence are quite different. In the legal field, the "true nature" of a discipline can never be perfectly distilled simply by studying the way the law works. The rules which govern the legal system are constantly being subjected to scrutiny, debate and change. This has a profound effect on the possibility of accurately reflecting those rules in a rule-based expert system.

Another difficulty of representing the law in an expert system is that the study of legal rules usually involves the struggle to interpret the (natural) language of a statute or a judicial decision. Because human language is subject to so many meanings, and because many statutes and decisions are left purposefully ambiguous, the "true meaning" of legal proclamations can never be completely distilled. In addition, the very nature of the legal profession is to debate the meaning and interpretation of legal rules. To say that a rule-based system accurately encodes a legal principal ignores this reality. There are always

"gaps, conflicts and ambiguities"⁴ which exist in the law that a lawyer must point out and seek to expand upon in the service of her client. Finally, legal rules are subject to constant changes in meaning and can be overruled, overturned and revoked at any time. Thus, even though rule-based systems are capable of providing a first glance at the way a legal domain is shaped, they are not an accurate way to reflect the law and legal reasoning.

1.2 Case Based Reasoning and the HYPO Program

In light of the shortcomings of rule-based or expert systems, the work of Kevin Ashley⁵ appears to be a major advancement in the modeling of legal reasoning and argumentation. As Ashley points out, lawyers tend to argue on the basis of analogy, taking a current situation and analogizing it to previously decided cases. This means that AI programs that wish to model legal thought must also think through analogies, and must be capable of pulling relevant cases from a knowledge base in order to create those analogies. Ashley also recognized the need for legal artificial intelligence systems to create arguments for both sides of a position, rather than simply stating a legal conclusion. This two-sided format reflects the understanding that legal conclusions are not accurately represented as provable facts. In addition, an argument for each side of a proposition is much more useful for a lawyer to receive than a simple statement of which side should win a particular legal issue.

Ashley implements these ideas in the HYPO program that he developed with Edwina Rissland.⁶ HYPO uses a case-based reasoning system in which a knowledge base of cases on a particular legal subject is indexed through the use of "dimensions." By using case-based reasoning to analyze the law, HYPO avoids many of the problems

⁴ A phrase familiar to students of Professor Duncan Kennedy of Harvard Law School.

⁵ Kevin Ashley, *Modelling Legal Argument: Reasoning with Cases and Hypotheticals*, Ph.D. Dissertation, Department of Computer and Information Science, University of Massachusetts, 1988 [hereinafter Dissertation]

⁶ Professor, Computer and Information Sciences, University of Massachusetts, Amherst.

associated with expert systems. Changes in the law are represented in HYPO by adding new cases to the case base. Conflicting rules or minority approaches in an area are represented by separate cases, all of which could be retrieved by HYPO as relevant to a particular situation. In addition, HYPO does not try to produce a mythical "correct" answer, but rather attempts only to develop arguments. Finally, since HYPO analyzes the law through the use of analogies to previously decided cases, as opposed to using a list of programmer created rules, HYPO more closely represents the type of legal analysis done by lawyers.

1.3 Introduction to Dimensions

There are numerous ways to represent a legal case in a computer program. However, it is important that the representation chosen by a programmer allows the system efficient access to the cases in the case base whenever a new situation is analyzed. Not only must every case be accessible, but the meaning and implication of those cases must be capable of computer analysis. Because of this, a "full text" version of a court's decision, such as is available through Westlaw or Lexis, would not be a feasible representation for a computer program which analyzes the implications and relationships of various cases within a legal domain. Although the information necessary to accomplish such an analysis may exist in the court's decision, computers are presently unable to analyze human language to the degree necessary to understand a legal decision. Thus, current computer programs must represent court decisions in a format other than full text.

Ashley's HYPO program has chosen to represent cases through the use of dimensions. A dimension represents the existence or non-existence of a factual situation that a court or commentator has stated to be relevant to deciding the outcome of an actual legal case. The list of dimensions for a particular legal domain should properly include every factual situation that a court has determined to be relevant to deciding cases in that

domain. Facts which are considered irrelevant to making a decision should not be included in the list of dimensions.

Ashley chose to use cases in the legal domain of trade secrets law, paying particular attention to cases involving the appropriation of computer software. A typical scenario in this domain involves a plaintiff corporation suing another corporation for appropriating trade secrets about their software. Often an employee with confidential information about the trade secret switched employment from the plaintiff to the defendant company, giving the defendant access to the secret information. In order to analyze cases such as these, Ashley examined various law review articles and legal cases to determine the appropriate dimensions for the domain. Eventually thirty-three trade secrets cases were implemented, using a total of 13 separate dimensions.

The HYPO program uses these dimensions to do almost all of its analysis dealing with legal cases. It is through the dimensions that:

- 1) cases relevant to the fact situation currently being analyzed (the "current fact situation" or "cfs") are retrieved from the case base,
- 2) retrieved relevant cases are compared to the current fact situation to determine the most on point cases,
- 3) most on point cases are compared and contrasted to determine which cases are appropriate for arguments and counter-examples,
- 4) cases are cited and analogies are made and refuted in the production of arguments, and
- 5) hypothetical variations of fact situations are suggested.

Given the usefulness of dimensions as demonstrated by the HYPO program, it would seem advisable to study further the desirability of representing legal cases through dimensions. One way to do such a study is to create a dimensionally indexed case base in different legal domain. Once a new case base is created, the usefulness of dimensions could be explored by manipulating the new domain's dimensions in a variety of ways. Such manipulation

techniques need not be limited to the type of analysis done by HYPO. In this paper, I will discuss my implementation of a dimensionally indexed case base in the area of vacating arbitration awards on the grounds of an arbitrator's evident partiality. This case base was created on the Dimensional Analyzer program (with the self-aggrandizing acronym of DAn) that I developed for this task. I will also examine some of the problems inherent in dimensional analysis, as well as the alternative analysis procedures that were developed in DAn.

SECTION 2: HYPO AND DIMENSIONAL ANALYSIS

2.1 Case Representation: Frames, Predicates and Dimensions

Although dimensions are the basic analytical tool that HYPO uses to analyze legal situations, they are only the top part of the program's layered representation of cases. Each representational layer can be thought of as a distillation of the facts that are contained in the lower levels. The most basic factual information in HYPO is represented at the lowest representational layer through the use of "case frames." As its name implies, a frame is a structure in a computer program into which data can be stored. The information is placed in the frame's "slots," which reserve space for specific types of information. A particular slot may reserve space for facts such as the name of the parties to a case or a case's reporter citation. HYPO uses a combination of these case frames to represent a single legal case. The "top-level" case frame contains general information about a case, such as the parties, the date of the court's decision, and the prevailing side. Lower-level case frames contain more detailed information about the parties and the factual situation involved in the dispute, and are connected to the top level case frame through the use of "pointers." The case frames are filled out by information entered directly by the user.

After the user fills out the case frames, the facts found therein are analyzed in order to determine which "factual predicates" have been satisfied. These predicates make up the

second layer that HYPO uses to represent a case. Factual predicates are basically commonly factual patterns that exist in a domain. A factual predicate in the trade secrets domain is "employee-paid-to-switch," which represents the factual pattern where a defendant employee received some type of bribe to switch employment. All possible factual predicates (thirty exist in Hypo) are stored in another frame entitled the interpretation frame or "interp-frame." Interp-frames are associated with a single case, and keep track of the factual predicates that are satisfied by the case.

Only after the interp-frame is filled out does HYPO begin to determine which dimensions are applicable to a case. The predominant way that dimensions differ from factual predicates is that every dimension in a domain has been considered important in the decision of an actual legal case. This is not so in the case of factual predicates. Each of HYPO's thirteen dimensions have a list of prerequisite factual predicates that must be satisfied before the dimension is considered applicable to a case. When all the prerequisite predicates are satisfied (or all but the "triggering prerequisites" in the case of "near-miss" dimensions), the dimension is considered applicable. Like the interp frames, once the applicable dimensions are determined, they are permanently associated with the case and do not need to be reassigned each time the program is run.

2.2 The Structure of HYPO's Dimensions

Before looking at how HYPO actually makes use of dimensions, it is important to understand how the program structures its dimensions. HYPO implements the basic information about a dimension through the use of a frame which has slots for three types of information. The first slot contains the list of prerequisite factual predicates that must be satisfied for a dimension to be applicable. As stated above, these prerequisites state the requirements that must be met before a dimension applies to a given fact situation. If all but the "triggering prerequisites" are met, then the dimension is not considered applicable, but

rather is considered a "near-miss" dimension. Near-miss dimensions are used by HYPO to develop hypothetical variations of a fact situation in order to explore the implications of slight changes in the facts of a case.

The second type of information contained in the dimension frames are "focal slots," which define the magnitude that a dimension can have. When a dimension is considered applicable to a case, it is often important to maintain some information indicating how strongly the dimension applies to the facts of that case. For instance, in the trade secrets domain, HYPO uses the dimension Secrets-Voluntarily-Disclosed, to represent the situation where a plaintiff voluntarily disclosed its confidential trade secrets to outsiders. For this dimension, it is important to keep track of how many outsiders received this confidential information. It would make a great deal of difference in arguing the case whether the disclosures were made only to two people as opposed to two thousand. This difference in magnitude is discovered through the use of the directions maintained in the focal slot of the dimension. These directions tell HYPO where to look in the lower level representation of the case in order to determine the appropriate magnitude of the dimension.

The third type of information found in the dimension frames is called the "range information," which tells HYPO how to interpret the values that are retrieved by the focal slot of the dimension. The basic function of the range information is to explain which side in a dispute benefits from the presence of a dimension. Similarly, the range information can be used to determine which side benefits as the magnitude of the dimension varies. For instance, in the Secrets-Voluntarily-Disclosed dimension, the range information would tell HYPO that the defendant benefits from the presence of this dimension. Additionally, the plaintiff is benefited by a lower number of disclosees in the focal slot, and the defendant is benefited by more disclosees. Another way of saying this is that Secrets-Voluntarily-Disclosed dimension is a pro-defendant dimension.

Thus, each of HYPO's thirteen dimensions is represented by a frame containing the dimension's prerequisites, focal slot values, and its range information. This is general

information about the dimension, and is not connected with a specific case. If HYPO finds that a dimension is applicable to a specific case after analyzing the case's interp-frame, then the dimension is added to that case's "dimension list." In this way, each top-level case frame maintains a list of the dimensions applicable to that case.

2.3 Dimensions as Indexes to the Case Base

HYPO uses dimensions primarily as an index to the cases contained in its knowledge base. For instance, if HYPO desired to find all cases which would be relevant to a given fact situation, HYPO would first analyze the situation to see which dimensions were applicable to the case. HYPO would then take that list of dimensions, and retrieve every case that also contained those dimensions. This search would be done by examining the dimension list of every case in the case base, and noting which cases share an applicable dimension with the fact situation. Not only would the names of the relevant cases be retrieved, but so also would all the information known about that case. This information can then be used in argument production.

For example, in Structural Dynamics Research Corp. v. Engineering Mechanics Research Corp., 401 F.Supp. 1102 (E.D. Mich 1975), a case analyzed in HYPO and discussed in-depth in Ashley's dissertation, the following dimensions were considered applicable:

- 1) Common-Employee-Transferred-Product-Tools (plaintiff's former employee brought product development information to the defendant),
- 2) Nondisclosure-Agreement-Re-Defendant-Access (plaintiff and defendant entered into a nondisclosure agreement),
- 3) Nondisclosure-Agreement-Specific (plaintiff's non-disclosure agreement specifically referred to the plaintiff's product),⁷

⁷ The status of this third dimension is not exactly clear. Apparently, the dimension exists in both

- 4) Common-Employee-Sole-Developer (employee defendant was sole developer of plaintiff's product).

According to the range information in the dimensions, the existence of the first three dimensions favor the plaintiff, while the existence of the fourth dimension favors the defendant. In retrieving cases that are relevant to Structural Dynamics, HYPO searches for cases which have any of these dimensions as applicable. A total of twelve relevant cases are retrieved, but none of the retrieved cases contain all four dimensions as applicable.

At this point it is necessary to determine which of the retrieved cases are most relevant to the fact situation being analyzed. This determination is made with the help of dimensions. A case is considered more "on point" than another case when it shares more applicable dimensions with the current fact situation than the other case. This is true, however, only when the set of dimensions that the less-on-point case shares with the fact situation is a subset of the set of dimensions that the more-on-point case shares with the fact situation. In other words, if a case shares two dimensions with a fact situation, then it is less on point than another case only if that other case shares those same two dimensions plus at least one additional dimension. Under this definition, it is possible for more than one case to be "most on point." This explanation of what cases are most on point not only makes sense computationally,⁸ but also logically from a legal point of view. When trying to make an argument to a court that a particular precedent should be followed as opposed to a contrary precedent, a lawyer will try to point out how the fact situation

Structural Dynamics and Amoco, but the argument output based on Amoco does not cite the dimension as a similarity, and instead the existence of the dimension in Structural Dynamics is used as a way to distinguish Amoco in the argument. See Dissertation, p. 195, 197. Apparently, both case have the dimension as "applicable," but in Structural Dynamics it has a positive value for its focal slot, while in Amoco, a negative value in its focal slot indicates that the dimension clearly does not exist. In this example, I have therefore chosen to present Amoco as if it lacked this dimension. This problem of "no value" dimensions stems from HYPO's derivation of dimensions and is discussed in greater detail in section 6.6 of this paper.

⁸ The determination of whether one set or list is a subset of another is a simple algorithm in most computer languages.

involved in the favorable precedent is more similar to the current facts than those in the contrary precedent. Hence, the lawyer will argue that the favorable precedent is more on point.

In the Structural Dynamics example, two contrary cases are considered by HYPO to be most-on-point (as well as "best cases," as explained below in §3.4). The first most-on-point case retrieved was Analogic Corp. v. Data Translation, Inc., 358 N.E.2d 804 (Mass. 1976). Analogic shares two dimensions with Structural Dynamics:

- 1) Nondisclosure-Agreement-Re-Defendant-Access, and
- 2) Common-Employee-Transferred-Product-Tools.

The court in Analogic found in favor of the plaintiff. The other most on point case, Amoco Production Co. v. Lindley, 609 P.2d 733 (Okla. 1980), found in favor of the defendant and also shares two dimensions with Structural Dynamics:

- 1) Nondisclosure-Agreement-Re-Defendant-Access, and
- 2) Common-Employee-Sole-Developer.

In selecting these cases as most on point, HYPO has determined that they are the most important cases in arguing the Structural Dynamics case.

Once the most important cases for a factual situation are determined, HYPO must be able to generate different types of argument propositions to include in its final output. Each of these different types of arguments relies on an analysis of the cases' dimensions. The basic contention that the holding of a specific case should be followed in a factual situation is supported by making an analogy between the situation and the cited case. The analogy in HYPO is made simply by stating the dimensions that the two cases share.

Another argument form is to attack the analogy made between the factual situation and the presented case. This attack is usually made by distinguishing the two situations, and in HYPO this also is accomplished through dimensions. For example, the analogy that the plaintiff in Structural Dynamics would like to make between that case and Analogic

could be attacked by noting that in Structural Dynamics, the employee defendant was the sole developer of the plaintiff's product. This was not so in Analogic, and therefore the cases should not be decided alike. In order to make such a distinction, HYPO need only determine that the Structural Dynamics case contained a dimension that Analogic did not share. A similar distinction could be made according to the magnitude of the dimension. For instance, even though both Structural Dynamics and Analogic involved a situation where an employee brought information from the plaintiff to the defendant corporation, this information was more beneficial and important to the company in Analogic, and therefore the two cases are distinguishable. This type of distinction is found by examining the dimension's magnitude for each case.

A third attack on an analogy would be to develop a counter-example. In this attack, the same factors that are shared by the cited case and the factual situation are shared by the counter-example, but the counter-example resolved the dispute in the opposite manner. Again, the presentation of counter-examples in HYPO is greatly aided by the presence of the dimensional index, as the program need only search its cases along the shared dimensions to discover the existence of any counter-examples.

The final argument production for the Structural Dynamics case that was produced by HYPO can be found in Figure 1. This Figure is copied from Ashley's thesis, and provides a fine example of how useful dimensional analysis can be in analyzing cases.

SECTION 3: ANALYSIS OF DIMENSIONS IN HYPO

Before analyzing the methods that the Dimensional Analyzer program uses to handle dimensions, a summary must be made of the workings of the HYPO program. During the analysis of a given fact situation, different modules are called to analyze the situation and produce the final argument output. The flow of control between the modules is shown in Figure 2. The following is a brief description of each of these modules.

3.1 Session Manager

The Session Manager allows the user to select a fact situation for HYPO to analyze.

The fact situation may be selected from:

- 1) a case already in the case base,
- 2) a fact situation created by the Hypo Generator module as an interesting variation of an existing fact situation, or
- 3) a new fact situation (ie, a new case) that is entered through the Case Editor.

The Case Editor allows the user to input a brand new fact situation. Menus are used to aid in entering the case, and the information entered is placed directly into the slots of a new case frame.

The Session Manager also creates a empty data structure that Ashley calls a Case Analysis Record, or CAR. This CAR is the basic data structure used by HYPO to store the results obtained from each module during the analysis of a fact situation. The information stored in a CAR is shown in Figure 3.

3.2 Case Analyzer

It is the job of the Case Analyzer to take the current fact situation's case frames (given to it by the Session Manager) and turn the information contained therein into factual predicates and dimensions. First the factual predicates are determined, and placed into an interp-frame. As explained above in §2.1, factual predicates are commonly existing factual patterns found in a domain. The interp-frame is then attached to the Case Analysis Record. In the next step, the Case Analyzer takes the factual predicates found in the interp-frame and determines the satisfied and near-miss dimensions. As explained in §2.2, applicable dimensions occur when all of a dimension's prerequisites are satisfied by the factual predicates contained in the case's interp-frame. Near-miss dimensions are those dimensions in which all but the "triggering" prerequisites are satisfied. The list of

applicable and near-miss dimensions are then attached to the Case Analysis Record. (See Figure 4 for a representation of the processes involved in the Case Analyzer).

3.3 Case Positioner

The Case Positioner is faced with the task of taking the list of applicable and near-miss dimensions associated with the fact situation being analyzed, and selecting cases from the case base which are considered on-point. On-point cases are those cases which share a dimension with the current factual situation. Thus, applicable on-point cases have an applicable dimension that is in the CAR's applicable dimension list, while near-miss on-point cases have an applicable dimension that is in the CAR's near-miss dimension slot.

The on-point cases are then arranged by the Case Positioner in a structure called a Claim Lattice. The purpose of a Claim Lattice is to order the on-point cases according to how on-point they are. As explained in §2.3 of this paper, the relative on-pointness of a case is determined by the number of dimensions that a case shares with the current fact situation. HYPO creates two claim lattices, one for applicable cases (the "Regular Claim Lattice") and one for near-miss cases (the "Extended Claim Lattice"). The claim lattices that are created form the shape of a tree, with more-on-point cases appearing above less-on-point cases. Cases whose list of applicable dimensions do not overlap completely are placed side-by-side in the tree. The two claim lattices are then inserted in the Case Analysis Record's Possible-Claims slot. (Figure 5 displays the actions of the Case Positioner).

3.4 Best Case Selector

The Best Case Selector takes the Claim Lattices from the Possible Claims slot and selects the most-on-point applicable and near-miss cases. These most-on-point cases are selected for each side in an argument (ie, plaintiff and defendant). All that is necessary to select these cases is to traverse the Regular and Extended Claim Lattices. Since the Lattices

were created according to the cases' relative on-pointness to the fact situation being analyzed, no additional comparison between cases need be done at this point.

Once the most-on-point cases are selected, the Best Case Selector determines which of these most-on-point cases are appropriate for use in directly making an argument. Such cases are considered "best cases," and are defined to be the most-on-point cases which share at least one dimension with the current fact situation that favors the same side as the case itself. (See Figure 6 for the Best Case Selector diagram).

An example may help explain the Best Case Selector's actions. Suppose HYPO were analyzing a domain concerned with negligence suits against truck drivers. The case being analyzed (the current fact situation), contains three dimensions as applicable:

- 1) truck-driver-was-drunk (pro-plaintiff),
- 2) truck-driver-was-speeding (pro-plaintiff), and
- 3) plaintiff-was-driving-wrong-way (pro-defendant).

The Smith and Jones cases were determined by HYPO to be the most-on-point applicable cases for the plaintiff's position. The Smith case shared the truck-driver-was-drunk, and the truck-driver-was-speeding dimensions with the current fact situation, while Jones shared only the plaintiff-was-driving-wrong-way dimension. Even though both cases were most-on-point (no cases shared these dimensions plus others and held for the plaintiff), only the Smith cases would be considered a best case. An argument for the plaintiff based on the Jones case would sound like: "because the plaintiff was driving the wrong way down the street, the plaintiff should win a negligence case against the truck driver." This clearly would not be a productive argument. However, the Jones case would make an ideal counter-example to a defendant's argument that the plaintiff should lose, because in Jones, the plaintiff was driving the wrong way and still won her negligence suit.

3.5 3-Ply Arguer

The 3-Ply Arguer module of HYPO takes the entire Case Analysis Record and produces a separate argument for each best case. In creating the argument, the 3-Ply Arguer first asserts that the holding in the best case should be followed in the current fact situation. To support this argument, the Arguer analogizes the best case to the current fact situation by stating the dimensions that they share.

The second stage of the 3-Ply Arguer is to argue against following the best case. In doing so, the best case is first distinguished from the current fact situation by noting the dimensions that the two situations do not share. Only differing dimensions which argue *against* the analogy are used to distinguish the best case. In other words, dimensions which favor the conclusion of the best case and are found in the best case and not in the current fact situation are distinguishable dimensions, and so are dimensions which favor the opposite side and are found in the current fact situation and not in the best case. This is best understood through an example. Using the same negligent truck driver example, suppose the Smith case shared the truck-driver-was-drunk, and the truck-driver-was-speeding dimensions with the current fact situation, and also had the truck-driver-admitted-guilt dimension as applicable. The 3-Ply Arguer could distinguish Smith by pointing out that in Smith, the truck driver admitted her guilt, while the truck driver in the current fact situation did not. Smith was a better case for the plaintiff than the current fact situation, and therefore should not be followed in this case. If Smith had contained a pro-defendant dimension that the current fact situation did not contain, such as plaintiff-makes-living-by-suing-truck-drivers, that dimension would not be a good dimension for a making a distinction, because it does not make Smith a better case for the plaintiff. Similarly, the Smith case could be distinguished because of the presence of the plaintiff-was-driving-wrong-way dimension in the current fact situation. Since this dimension did not occur in Smith, the current fact situation is a stronger case for the defense than the Smith case, and therefore Smith should not be followed.

After distinguishing the best case, the 3-Ply Arguer attempts to find counter-

examples. Counter-examples are those cases which share some or all of the dimensions shared by the cited best case and the current fact situation, but which nevertheless held for the opposite side. Counter-examples can easily be retrieved from the most-on-point cases found by the Case Positioner. HYPO implements four types of counter-examples (as in all counter-examples, each of these types involved a case that had a holding opposite of the holding in the cited case):

- 1) more-on-point: in which the counter-example is more-on-point than (or equally on point as) the cited case,
- 2) potentially-more-on-point: in which the counter-example is drawn from the Extended Claim Lattice and is more-on-point in that lattice than the cited case (relies on near-miss dimensions and is useful in developing hypothetical variations on the current fact situation),
- 3) overlapping: in which the counter-example is not as on-point as the cited case, but contains some of the same applicable dimensions as the cited case, and
- 4) boundary: in which the counter-example has a greater magnitude along a dimension shared by the cited case and the fact situation.

After distinguishing the best case and offering any counter examples that were found, the 3-Ply Arguer develops the third stage of the three ply argument, in which the cited best case is defended as an analogy by distinguishing the offered counter-examples from the current fact situation. These cases are distinguished in the same way that the best case was distinguished in stage two.

Although the 3-Ply Module is the module which outputs the final argument to the user, the basic parts of the argument are first stored in a data structure called the Argument Record. The output of the Argument Record to the user is not made until the 3-Ply Arguer is requested to do so by the Explainer, as described below. (See Figure 1 for two sample

3-Ply Arguments produced by HYPO, and Figure 7 for the graphical representation of the steps taken by the 3-Ply Arguer).

3.6 Hypo Generator

The Hypo Generator is the module which generates hypotheticals, or variations of fact patterns, that might be suggested by the Explainer module. Typically, the Hypo Generator is asked to generate a hypothetical in which the current fact situation is changed slightly, so that a potentially more-on-point counter-example will become an applicable more-on-point counter-example. In other words, the Hypo Generator would slightly vary the facts in order to show how such a variation could change the resulting argument.

3.7 Explainer

The Explainer receives the Argument Record from the 3-Ply Arguer and also tells the 3-Ply Arguer when the argument should be displayed to the user (or written to a file). The Explainer also suggests hypotheticals to the Hypo Generator, and then allows the user to repeat the case analysis loop with one of the suggested hypotheticals. The Explainer is also responsible for comparing the strength of both sides' argument in a claim, and then tells the user if one side has a stronger argument. The strength of an argument is determined by the existence of a best case to which the other side cannot respond with a more-on-point counter-example. If both sides or neither side of an argument can produce such cases, then the arguments are considered equal in strength. However, if only one side can produce such a "non-trumped" case, then that side is determined to be the stronger side.

SECTION 4: EVIDENT PARTIALITY

In this project, I explored the usefulness of dimensional analysis by analyzing the domain of evident partiality. The following sections provide a brief introduction to this

legal domain.

4.1 Statutory Framework

Alternative dispute resolution methods have become more popular as the case loads in the courts of the United States grow larger and the corresponding wait for a case to be tried grows longer. In the commercial area, the most popular alternative dispute resolution technique used today is arbitration. Arbitration is a consensual process, in that both parties to an arbitration must agree to settle their dispute in front of an arbitrator (or arbitrators). Once such an agreement is made, however, the parties are bound by the arbitrators decision, as courts are required to enforce the decisions of arbitrators under statutory law.

In the United States, arbitration is governed either by the United States Arbitration Act (9 U.S.C. §§1-14), or by state law. The U.S. Act was enacted in 1925 in response to a reluctance by the federal courts to enforce contract arbitration provisions.⁹ This statute covers most of the commercial arbitration hearings in the country, as courts have read the law to have exclusive authority over all arbitration contracts that Congress could Constitutionally regulate under the Commerce Clause. Snyder v. Smith, 736 F.2d 409, 418 (7th Cir. 1984). Thus state law controls only in those rare occasions when the activity at the heart of the dispute is not under the scope of the U.S. Arbitration Act. Since most states have adopted some version of the Uniform Arbitration Act, this means that practically every commercial arbitration is governed either by the U.S. Act or Uniform Act.

While the Uniform Arbitration Act and U.S. Arbitration Act are separate entities, they are similar in many respects, especially in the area of upholding and vacating the award of an arbitrator. Arbitration awards generally become enforceable only after they are confirmed by a court of law. In most cases, it is not necessary to get the award confirmed because the losing party agrees to abide by the arbitrator's decision. However, losing

⁹ Venue for Motions to Confirm or Vacate Arbitration Awards Under the Federal Arbitration Act, 57 Fordham L. Rev. 653, 653.

parties always have the option to ask the court to vacate the award. This motion is usually made at the same time that the prevailing party moves for confirmation. Since a great deal of emphasis has been put on the finality of arbitration awards, in order that they become a viable alternative to court cases, both the U.S. and Uniform statutes have put severe restrictions on the vacatur of awards. Under both statutes, vacatur is allowed only for the grounds that are explicitly set forth. According to the U.S. Act:

In either of the following cases the United States court in and for the district wherein the award was made may make an order vacating the award upon the application of any party to the arbitration--

- (a) Where the award was procured by corruption, fraud, or undue means.
- (b) Where there was evident partiality or corruption in the arbitrators, or either of them. . . .

9 U.S.C. §10. A similar provision exists in the Uniform Arbitration Act:

Upon application of a party, the court shall vacate an award where:

- (1) The award was procured by corruption, fraud or other undue means;
- (2) There was evident partiality by an arbitrator appointed as a neutral; . . .

Uniform Arbitration Act, §12 (a). The grounds found in the two statutes are nearly identical, and both statutes allow the vacating of arbitration awards where the "evident partiality" of an arbitrator can be shown. However, neither statute further defines this phrase, leaving its definition and application up to the courts.

4.2 Commonwealth Coatings

The preeminent case in the interpretation of the phrase "evident partiality" is the 1968 U.S. Supreme Court case of Commonwealth Coatings, Corp. v. Continental Casualty Co., 393 U.S. 145, 21 L.Ed. 2d 301, 89 S.Ct. 337 (1968). In this case, the Supreme Court was forced to interpret the "evident partiality" clause of the United States Arbitration Act when a neutral arbitrator of a three member arbitration panel had concealed his past business dealing with one of the parties to the arbitration. The court characterized

the relationship as "sporadic," but it lasted over 5 years and involved fees totalling over \$12,000. No evidence suggested that the arbitrator had in fact acted prejudicially, yet the court found that fact to be irrelevant. Justice Black wrote for the majority, and stated that:

It is true that arbitrators cannot sever all their ties with the business world, since they are not expected to get all their income from their work deciding cases, but we should, if anything, be even more scrupulous to safeguard the impartiality of arbitrators than judges, since the former have completely free rein to decide the law as well as the facts and are not subject to appellate review. We can perceive no way in which the effectiveness of the arbitration process will be hampered by the simple requirement that arbitrators disclose to the parties any dealings that might create *an impression of possible bias*.

393 U.S. at 148-149, 89 S.Ct. at 339 (emphasis added). Black concluded his opinion along the same lines:

. . . any tribunal permitted by law to try cases and controversies not only must be unbiased but also must avoid even *the appearance of bias*. We cannot believe that it was the purpose of Congress to authorize litigants to submit their cases and controversies to arbitration boards that might reasonably be thought biased against one litigant and favorable to another.

393 U.S. at 150, 89 S.Ct. at 340 (emphasis added).

The primary emphasis of Black's opinion was that any dealings which might create an "impression of possible bias" must be revealed by the arbitrator before the arbitration. In that way, the parties may choose whether or not to submit their dispute to that particular arbitrator. If any such dealings are not disclosed before the arbitration, then the award made under such conditions must be vacated under the evident partiality provision of the U.S. Arbitration Act.

Justice White, joined by Marshall, wrote a concurring opinion in which he emphasized that certain prior dealings must be revealed, but he cautioned that an arbitrator "cannot be expected to provide the parties with his [or her] complete and unexpurgated business biography."¹⁰ According to this concurrence, if a business relationship between the arbitrator and a party is *trivial*, then there is no reason to disclose those

¹⁰ 393 U.S. at 151, 89 S.Ct. at 340

dealings and such a relationship will not provide the grounds for vacating the award. This concurrence has additional strength given that the "majority" opinion was not a pure majority, as only four justices signed Black's opinion.

The dissenting opinion, written by Justice Fortas and joined by Harlan and Stewart, challenged the majority's disclosure requirement, noting that it is not based on any language found in the statute. Fortas argued that "evident partiality" should be read to mean actual conduct favoring one party over another, and therefore that arbitration awards should not be vacated in the absence of actual partiality, bias or misconduct.¹¹

4.3 Reactions to Commonwealth Coatings

The initial reaction to the Commonwealth Coatings decision was predominantly negative. Several commentators disagreed with the court's holding that arbitrators are to be held accountable to the same standard of conduct as judges.¹² Some complained about the nebulous standard created by the combined majority and concurring opinion, in that all relationships which might create "an impression of possible bias" must be revealed, yet "trivial" relationships may be ignored.¹³ In addition, the fact that the "failure to disclose" test was not found in the statute bothered some commentators as much as it did Justice Fortas.¹⁴

4.4 Evident Partiality Today

In spite of the fact that the statutory authority behind vacating arbitration awards is quite uniform throughout the country, courts today still interpret the phrase "evident partiality" in a variety of ways. This is true even after Commonwealth Coatings, although

¹¹ 393 U.S. at 154, 89 S.Ct. at 342.

¹² Case Notes, 20 Mercer L. Rev. 449. See Recent Developments, 44 St. John's L. Rev. 93 (1969) (arguing that arbitration is a contractual, not judicial, arrangement).

¹³ Recent Developments, 44 St. John's L. Rev. 93,101 (1969); Notes, 47 Texas Law Review 1207 (1969).

¹⁴ See Case Notes, 20 Mercer L. Rev. 449.

this case continues to be the primary source of law for courts faced with the challenge of interpreting the phrase "evident partiality." Even among the cases which cite Commonwealth Coatings as controlling authority, the decision has only slightly narrowed the focus of the current debate, as individual judge's have focused on different parts in the opinion. Some courts regard the "impression of possible bias" test or "appearance of bias" test as governing,¹⁵ while others refer to whether or not the relationship was "trivial."¹⁶ Even courts that use only the "impression of possible bias" test have recognized that the test itself provides little guidance as to when an undisclosed relationship between an arbitrator and a party should be considered to bring about such an impression. For instance, no impression of bias has been found when, although an arbitrator failed to reveal a significant relationship with another party, the complaining party had somehow obtained information about that relationship and failed to act on such information.¹⁷ Also, courts have found that when the arbitration concerns a narrow trade area in which only a limited number of professionals with knowledge of the area exists, then the parties must expect a certain amount of business dealings between the arbitrator/professional and the parties.¹⁸

In addition, several courts have been forced to give meaning to the "trivial" relationships discussed by Justice White. In doing so, courts have treated similar cases

¹⁵ See, e.g., Albion Public Schools v. Albion Education Assoc., 130 Mich. App. 698, 344 N.W.2d 55 (1983), Barcon Associates v. Tri-County Asphalt Corp., 86 N.J. 179, 430 A.2d 214 (1981), Egan & Sons Co. v. Mears Park Development Co., 414 N.W.2d 785 (Minn. Ct. App. 1987).

¹⁶ See, e.g., Isaacson v. Hayden Stone Inc., 359 F.Supp. 1050 (S.D.N.Y. 1973), Merit Ins. Co. v. Leatherby Ins. Co., 714 F.2d 673 (7th Cir. 1983), Ormsbee Development Co. v. Grace, 668 F.2d 1140 (10th Cir. 1982).

¹⁷ See, e.g., Baar & Beards, Inc. v. Oleg Cassini, Inc., 30 N.Y.2d 649, 331 N.Y.S.2d 670 (N.Y. Ct. App. 1972) (party found to have waived objection); Jean A. McCoy & Sons, Inc. v. LaSalle County, 48 Ill. App. 3d 802, 363 N.E.2d 442 (Ill. App. Ct. 1977) (knowledge of relationship without acting on that knowledge constituted waiver of objections); St. Paul Ins. Co. v. Lulis, 6 Wash. App. 205, 492 P.2d 575 (1971) (party could be inferred to have knowledge of a relationship which was notorious and well-known).

¹⁸ Garfield & Co. v. Wiest, 432 F.2d 849 (2d. Cir. 1970); Cook Industries, Inc. v. C. Itoh & Co., Inc., 449 F.2d 106 (2d. Cir. 1970); Perl v. General Fire & Casualty Co., 34 A.D.2d 748, 310 N.Y.S.2d 196 (N.Y. Sup. Ct., App. Div. 1970).

differently, and there is as of yet no definitive explanation as to when a past business or social relationship rises to a level requiring vacatur. Thus, the meaning of evident partiality is presently being decided on a case by case basis, taking directions from Commonwealth Coatings and other case which discuss the interpretation problem. Thus, this appears to be a domain ripe for exploration through the use of dimensional analysis.

SECTION 5: DIMENSIONAL ASSIGNMENT

5.1 Assignment Problems in HYPO

In HYPO, cases are entered factually through the use of the program's Case Editor. The person responsible for analyzing a case need only input yes, no or a simple phrase in response to the Case Editor's questions. These responses are then inserted directly into the slots of the case frames. After this factual information is entered, HYPO analyzes the case frames to determine the satisfied factual predicates and applicable dimensions. The main advantage of this method of case input is that the amount of interpretation required to enter a case is minimized. As Ashley put it:

Dimensions cut down on the amount of information needed by the person entering a case in the [case knowledge base]. He [or She] need only record the facts expressly reported by the court, the claims and the court's decision of the claims. Although that in itself is an interpretive task, at least it requires less interpretation than determining what issues the courts decided and what rationales they used in their decisions.¹⁹

Ashley is correct to a point. The major part of the interpretation of facts is done "automatically" by the Hypo program when it analyzes the case frames. However, before this analysis is possible, Ashley first had to determine what combinations of factual predicates were required in order for a dimension to exist. This was a very significant step in the interpretation of the law of trade secrets. Ashley states that he did not develop the dimensions, but rather they were discovered primarily through law review articles and in the cases themselves.²⁰ This fact does not necessarily decrease the amount of

¹⁹ Dissertation, p. 246.

interpretation involved, as Ashley is relying not only upon his own interpretation of a law review author, but also the author's interpretation of the cases. Thus, although Ashley claims that HYPO has minimized the interpretation problem at the input stage, his program relies on:

- 1) the law review author's interpretation of the cases,
- 2) the author's ability to express and organize her/his belief in the article and its footnotes,
- 3) Ashley's own interpretation in creating dimensions from the author's footnotes,
- 4) Ashley's (most likely difficult) implementation of those dimensions as combinations of factual predicates,
- 5) the related problem of implementing factual predicates based upon the facts found in the case frames, and
- 6) the inputer's interpretation of the case and the information requested by the case frames.

Thus there is a great deal of intellectual distance between HYPO's implemented dimensions and the trade secrets cases that the dimensions represent.

5.2 Dimensional Assignment in the Evident Partiality Domain

I do not mention Ashley's interpretation problem to critique the dimensional assignment done in HYPO, but rather it is mentioned to defend the interpretation method used to assign the evident partiality dimensions in the Dimensional Analyzer program. There have been no treatises or law review articles published on the evident partiality standard, so the dimensions were assigned to the cases after analyzing the cases directly.²¹ Dimensions were selected by examining what factual issues or combination of

²⁰ Dissertation, p. 247.

²¹ With some guidance from ALR articles on the subject: Setting Aside Arbitration Award on Ground of

factual issues played a role in a court's decision. Although it was possible that such factors would be very case specific, a significant number of dimensions appeared in a great number of cases. If a factor used by a court in deciding a case was unique to that case, a new dimension was nevertheless created. In this way, every factor which was determined to play a role in a court's decision was represented by a dimension. After analyzing over forty cases, twenty-three dimensions were created (See Appendix A for a list of implemented dimensions, and Appendix B for the implemented cases).

Once the dimensions for the evident partiality claim were created, each case was again analyzed to determine which dimensions were applicable and what magnitude each dimension should have. Contrast this method to that used HYPO, where dimensional assignment is accomplished through a complex algorithm in which data entered into the case frames is converted to factual predicates and then into dimensions. The direct assignment of dimensions is quite different from HYPO's method, and appears on the surface to be more vulnerable to the charge of subjective interpretation. But comparison with Ashley's interpretation difficulty reveals that this is at best an arguable claim. Ashley states that in HYPO, "the interpretation has been performed by the domain expert, the author [of the law review article]" ²² The dimensions, however, were still created through Ashley's interpretation. In contrast, the evident partiality dimensions were created through my interpretation of the cases themselves, and the assignment of the dimensions to the cases was done with an understanding of what the dimensions mean. Thus, although Ashley was more systematic and algorithmic in his interpretation of the trade secret cases, the evident partiality interpretations benefited from a uniform understanding of the meaning of, and the requirements for, the dimensions.

Interest or Bias of Arbitrators, 56 A.L.R.3d. 697; Vacation of Arbitration Awards [under the U.S. Arbitration Act], 20 A.L.R.Fed 295.

²² Dissertation, p. 246.

5.3 Example Cases

In order to understand what is involved in the assignment of dimensions, it is helpful to look at some example evident partiality cases. The Abramovich example was chosen to exemplify the basic technique of dimensional assignment, while the Richco case shows some of the problems and limitations inherent in representing a complex fact situation in a limited set of dimensions.

5.3.1 Abramovich v. Pennsylvania Liquor Control Board

In Abramovich v. Pennsylvania Liquor Control Board, 416 A.2d 474 (Penn. 1980), the Supreme Court of Pennsylvania vacated an arbitration award on the grounds of evident partiality. In that case, Felix Abramovich requested an arbitration hearing before the state Attorney General in a contractual dispute between himself and the state's liquor control board. The arbitrator assigned to the dispute, W. W. Anderson, was also the Deputy Attorney General assigned to serve the liquor control board. In serving the board, Anderson acted as the board members' legal counsel, reviewed and approved all large contracts, and approved all standard contractual language for the board. Anderson in fact had approved the actual contract involved in the Abramovich case.

During the arbitration, Anderson met with various members of the board as well as the counsel representing the board in the arbitration hearing. At these meetings, Anderson discussed the financial status of Abramovich's business, as well as the possibility of settling the dispute. Abramovich was unaware of any of these dealings or meetings until after Anderson concluded the arbitration and ruled in favor of the liquor control board. Abramovich lost his action to vacate the award at the trial level, but the Pennsylvania Supreme Court reversed and vacated the award. The Supreme Court relied on the intimate associations between Anderson and the board, and his consequent direct connection with the contract in dispute. In addition, the court stated that the ex parte discussions on the merits of the case were highly inappropriate to an adjudicative proceeding.

Section 5: Dimensional Assignment

Reducing the facts of this case to dimensions is a relatively straight forward process. The following three dimensions make up the representation of the Abramovich case in DAn's evident partiality domain (note that relationship dimensions deal with relationships between the arbitrator and the party who prevailed in the arbitration proceeding):

- 1) attorney-client-relationship-with-party (magnitude: On-Going),
- 2) improper-communications-with-party-or-lawyer (magnitude: Yes),
- 3) knowledge-by-party (magnitude: No).

The first dimension represents the on-going relationship that existed between Anderson and the liquor control board. The second records the fact that ex-parte communications occurred between Anderson and the members of the board and their attorney. Notice that there is no way to represent which of these two dimensions was the primary motivating factor in the court's decision. The main reason that this type of information is not encoded is that it is very rare for a court to directly state that one factor was the sole reason for a decision. This is true not only in the evident partiality domain, but also in practically every legal domain. Even when a court does make such a statement, law students and lawyers are taught not to put too much emphasis on such statements. Thus, neither HYPO nor DAn attempt to encode or use a court's direct statement of its primary motivation.

The knowledge-by-party dimension is included as an applicable dimension in Abramovich, in spite of the fact that Abramovich did not have any knowledge of the arbitrator's dealings until after the award was made. This lack of knowledge is represented by giving the dimension a magnitude of "No." The use of the "No" magnitude indicates that the court discussed this dimension, but the dimension did actually exist in the case's facts. Compare this to the fact that the financial-interest-in-outcome dimension is not included in Abramovich's dimensional representation. The absence of this dimension indicates that the arbitrator did not have a financial interest in the outcome of the arbitration

as far as could be determined by the court's opinion. If the court had explicitly stated that Anderson did not have such an interest, the dimension would have been assigned to the case with a magnitude of "No." The effect of dimensions with "No" valued magnitudes is discussed below in §6.6.

5.3.2 Richco Structures v. Parkside Village, Inc.

In 1978, the Wisconsin Supreme Court analyzed a construction arbitration award in the case of Richco Structures v. Parkside Village, Inc., 82 Wis.2d 547, 263 N.W.2d 204. The plaintiff, Richco Structures, had sued Parkside Village for payment on a construction contract. Parkside counter-claimed for breach of contract, and the two parties agreed to submit the matter to arbitration. Each side appointed an arbitrator, and those two arbitrators selected Wallace Syring as the third, neutral arbitrator. The award in the case was made in favor of Parkside Village and was signed by Syring.

Prior to the arbitration, Syring had worked for a contractor which had at several times in the last few years worked with a company closely related to Parkside Village, Inc. (the companies were closely related in that both companies shared the same president). Syring himself had hired Parkside's related company as a subcontractor on two jobs, one such job continued through the time of the arbitration. Syring also had hired one of Parkside's lawyers as his own lawyer on three different occasions, one of which was also continuing at the time of the arbitration. In addition, Syring apparently had a business and social relationship with the arbitrator appointed by Parkside. Richco Structures discovered the relationship between Syring and Parkside's attorney ten days before the award came through, and decided not to take any action on it until after the award was made. Richco did not know of the other relationships until after the award.

After the award was made by the arbitration panel, Parkside moved for the trial court to uphold the award, while Richco simultaneously moved for the vacatur of the award

on the grounds of corruption and evident partiality. The trial court ordered an evidentiary hearing in order to determine whether the decision could be vacated on the grounds of "corruption and undue means." After the hearing, the trial court stated that there was no actual corruption or fraud, but it vacated the award nonetheless on the grounds of evident partiality.

The Supreme Court of Wisconsin agreed with the trial court in finding that these facts should have been disclosed at the outset of the arbitration hearing, and therefore ordered the vacatur of the award on evident partiality grounds. In so holding, the Court relied primarily on the undisclosed business dealings between Syring and Parkside's related company. The court noted that a relationship between an arbitrator and a party's attorney might be an independent ground for vacating an arbitration award (citing San Luis Obispo Bay Properties, Inc. v. Pacific Gas and Electric Co., 104 Cal . Rptr. 733 (1972) and Johnston v. Security Ins. Co. of Hartford, 86 Cal. Rptr. 133 (1970)). However, the court implied that any objection to this relationship might have been waived by Richco Structures when it failed to act on its knowledge of this relationship prior to the making of the award (citing Cook Industries, Inc. v. C. Itoh and Co. Inc., 449 F.2d 106 (2d Cir. 1971)). The court did not resolve this issue, as the award was vacated on other grounds.

In assigning dimensions to this case, four dimensions seemed applicable without much difficulty:

- 1) attorney-client-relationship-with-lawyer (magnitude: On-Going);
- 2) relationship-with-party-or-lawyer-through-third-person (magnitude: Past);
- 3) arbitrator-was-not-appointed-as-neutral (magnitude: No); and
- 4) partiality-affected-outcome (magnitude: No).

The first dimension recognizes the on-going attorney-client relationship that existed between the arbitrator (Syring) and a lawyer of the non-complaining party (Parkside's lawyer). The second dimension reflects the fact that the arbitrator (Syring) had a past business or social relationship with a third party (Parkside's appointed arbitrator) who had

a significant relationship with a party to the dispute (Parkside). The third and fourth dimension reflect the fact that the court explicitly stated the the arbitrator was appointed as the neutral, and that the arbitrator's partiality had not been shown to affect the outcome of the award.

The difficult dimensions to assign in this case were the substantial-business-relationship-with-party and the knowledge-by-party dimensions. In deciding whether or not the substantial-business-relationship-with-party dimension applies to the Richco case, two factors must be considered. First, we must distinguish between the substantial-business-relationship-with-party dimension and the minor-business-relationship-with-party dimension. There are no clear cut lines between these two dimensions, but as a general rule of application the substantial business relationship dimension has been applied whenever the relationship between the parties exceeded a value of \$1,000. In this case, no dollar value was placed on the business relationship. All that is known is that the two companies were involved in several construction projects. In order for the person analyzing the case to guess that such a relationship exceed the \$1,000 value, some understanding of the construction world must be assumed. Given that construction contracts usually involve much more than \$1,000, the substantial business relationship dimension was chosen.

The dollar value used to distinguish these two dimensions is a completely arbitrary division, and there is nothing within the evident partiality domain which explicitly justifies this choice. To some extent, this problem would be solved by combining the two dimensions and using the dollar value to determine the magnitude of the new dimension. In this way, there would be no reason for the person assigning the dimensions to distinguish between two different dimensions.

However, using a single dimension would mean that a case with a minor business relationship would be considered an on-point case to a fact situation involving a substantial

business relationship. However, analysis of the evident partiality domain indicates that courts tend to treat minor business relationships between an arbitrator and a party as a unavoidable event in the business world, while more substantial relationships were very significant in finding the existence of evident partiality. Therefore, in order to more effectively represent the cases and to make comparisons between cases more realistic, minor and substantial business relationships were divided into separate dimensions.

Another problem with assigning the substantial-business-relationship-with-party dimension to Richco is that the business relationship involved in that case took place with a company *related* to the arbitration party. The non-arbitrator party involved in the relationship is clearly a significant fact; this is the only distinction between the substantial-business-relationship-with-party dimension and the substantial-business-relationship-with-lawyer dimension. The relationship-with-party dimension is a more apt portrayal of the facts in Richco than the relationship-with-lawyer dimension, but why not create a substantial-business-relationship-with-company-related-to-party dimension? This question comes up repeatedly in assigning dimensions to legal cases: should an existing dimension be stretched to include a new fact situation, or should an new dimension be created to represent it? Usually, there is no obvious way to solve this dilemma. The person responsible for assigning dimensions to a case (or the person encoding the algorithm which assigns dimensions to a particular fact situation) must determine which solution would best represent the spirit of the cases in a domain. In Richco, I therefore chose to assign the substantial-business-relationship-with-party dimension with a magnitude of "On-Going."

The other problem dimension in the Richco case was knowledge-by-party. The case made it clear that Richco Structures did not know of the business relationship between Syring and Parkside's related company. This relationship was the primary reason the court vacated the arbitration award. At the same time, the court stated that Richco knew of the attorney-client relationship between Syring and Parkside's lawyer before the award was

made. Assigning either a "Yes" or a "No" magnitude to this dimension could be troublesome. If the "Yes" magnitude is assigned, this case would then appear to stand for the proposition that an arbitration award can be vacated even when the complaining party knew of the arbitrator's dealings with the other party. If the "No" magnitude is assigned, then the partial knowledge that Richco did have is ignored. One solution is to include a third possible magnitude that could be assigned to this dimension: Partial. A second solution would be to create a new dimension: partial-knowledge-by-party. The creation of a new dimension, however, would not allow a case with the knowledge-by-party dimension to be retrieved as relevant to a case with this new dimension. Therefore, the first solution was adopted, and I assigned knowledge-by-party to Richco with a magnitude of "Partial."

These examples represent some of the common difficulties in assigning dimensions. Other difficulties also exist, including misreading a case, miss-assignment of dimension magnitudes, or the late or non-recognition of important dimensions in a domain. In the evident partiality domain, the unanimous-arbitration-panel dimension (pro uphold) was not noticed until every case had been checked twice for dimensions. In re-analyzing cases for this one dimension, not as much care was taken and at least one case with the dimension was overlooked (Cook Industries, see §6.5).

SECTION 6: REPRESENTATION IN PROLOG

6.1 Prolog

6.1.1 Introduction to Prolog

The HYPO program was written in Zetalisp, a lisp dialect, on a Texas Instruments Explorer computer.²³ The Dimensional Analyzer (DAn) program which was used to implement the evident partiality domain was written in C-Prolog, a Prolog dialect running on a Digital Equipment Corporation's VAX computer. Although HYPO and DAn are

²³ Dissertation, p. 326

similar programs, the languages they are written in are quite different. Lisp is the most popular language used in artificial intelligence programming, and is most well-known for its ability to process lists effectively. It is a traditional programming language in that the programmer tells the computer what steps it should take to solve a problem. Prolog is a backwards chaining theorem prover, in which programs are written as rules for proving relations among objects. In other words, the Prolog programmer tells the computer what needs to be proven and how it can be proven, and the computer figures out the actual steps that are taken.

Part of the process that Prolog uses to prove theorems involves searching through a data base of facts and rules in order to find a way to satisfy a clause. This searching technique was put to use by DAn in order to automate the search for related and on-point cases. In addition, this searching power allows DAn to index a case or a dimension along any fact that is encoded. In comparison, HYPO was limited to accessing relevant case through three methods: the dimensional index, through "typed lists," and through cross references between cases which could be utilized only when a relevant case was already accessed.²⁴ Since DAn represents cases through Prolog fact statements, it is literally possible to index any case or dimension through any fact that has been implemented in the program. This power is partially realized in DAn's Query System.²⁵

Another useful feature of the Prolog language is its ability to assert and retract facts and rules. In DAn, this ability allows the program to change the current knowledge base and to add facts to the knowledge base while the program is running. Thus, new domains can be retrieved in the middle of a run, and the creation and deletion of generalization rules can take place when needed.

6.1.2 Programming in Prolog

²⁴ Dissertation, p. 224.

²⁵ See *supra*, §7.4.6

Prolog programs are written in such a way as to be understandable to a person with only a small understanding of Prolog syntax. For instance, the fact that sue is a lawyer is represented in Prolog by the fact statement:

lawyer(sue).

Such facts are usually placed into Prolog's memory by consulting an outside file which contains this statement. Once the facts are consulted, it is possible to ask Prolog questions about the facts. One can ask if sue is a lawyer:

lawyer(sue).

Yes

(responses by Prolog will be printed in bold in this section) or if mary is a lawyer:

lawyer(mary).

No

Prolog answers "No" because it has no information that mary is a lawyer. She may be a very famous attorney, but since she is not represented in its present knowledge base, Prolog assumes that mary is not a lawyer. Small letters are used to represent names because capital letters are used in Prolog to represent variables. Variables can be used to make Prolog respond in ways other than yes or no. For instance, we can ask "who is a lawyer:"

lawyer(Who).

Prolog responds by stating that sue is a lawyer:

Who = sue

By saying that "Who" equals sue, Prolog is stating that it can satisfy the "lawyer(Who)" clause if it sets the variable Who equal to the object "sue."

Other facts could be added to our makeshift data base, such as the fact that lisa and karen are the daughters of sue, and bob is karen's husband.

daughter(sue, karen).

```
daughter(sue, lisa).
husband(karen, bob).
```

It would be a simple matter to ask Prolog who is karen's husband:

```
husband(karen, Who).
Who = bob
```

We can also define a rule which will allow Prolog to do more complex reasoning for us, rather than simply repeat back what we told it. A rule which determines if someone is the child of another person can be defined as:

```
child(Person, Daughter) :-
    daughter(Person, Daughter).
```

Notice the capital letters that indicate that variables are being used, so that Prolog can fill in the objects needed to fulfill this rule. The ':' symbol indicates that what is on the left is to be defined by the clauses on the right. The ':' could also be thought of as separating the two parts of an "if-then" statement. The part to the right of ':' is the "if" part of the clause, while the clause on the left makes up the then clause. The period indicates that the end of the definition has been reached. Thus, this rule could be read as "if someone is the daughter of a person [daughter(Person, Daughter)], then the daughter is also considered the child of that person [child(Person, Daughter)]. We would also need to define another rule for male children:

```
child(Person, Son) :-
    son(Person, Son).
```

Notice that the variables used in these definitions (Person, Daughter and Son) could have been any words which began with a capital letter. Thus, the definition of a son_in_law can be defined as follows:

```
son_in_law(X, Y) :-
    daughter(X, Z),
```

```
husband(Z, Y).
```

Clauses separated by a comma (','), can be read as joined together by an "and" condition.

Thus, this rule should be understood as stating that the son-in-law of X is Y if: 1) the daughter of X is Z, *and* 2) the husband of Z is Y.

Once these rules are defined, Prolog will answer queries which use these rules:

```
child(karen, Who).
Who = lisa ;
Who = sue ;
No
```

In response to this question, Prolog search through its knowledge of facts and rules.

Although no facts directly define anyone as a child of karen, Prolog finds the rules which allow it to determine children based on the existence of other facts. Thus, when the first child rule is encountered, Prolog begins to try to satisfy the right side (or "if" side) of the rule by looking for facts or rules concerning daughters. It then finds that lisa is the daughter of karen, and it returns lisa as one solution to the query. By entering a semicolon (;), the user asks Prolog to go back and find another solution. Prolog then returns with sue. When the semicolon is entered again, Prolog again tries to satisfy the first child rule, but no more facts exist about the daughters of karen. The second rule concerning sons is then attempted, but no facts or rules exists concerning sons of karen. Prolog therefore fails in trying to find another answer to the query, and returns the answer "No."

We can also ask questions in which Prolog fills in more than one variable:

```
son_in_law(Person_Number_1, Person_Number_2).
Person_Number_1 = karen
Person_Number_2 = bob
```

In solving the son_in_law question, Prolog must satisfy both the daughter(X, Z) requirement and the husband(Z, Y) requirement. The first fact retrieved in trying to satisfy the daughter predicate will be daughter(karen, lisa). At this point, karen was assigned to

X, and lisa is assigned to Z. When Prolog realizes it cannot satisfy husband(lisa, Y), it "backtracks" and tries to find another solution to daughter(X, Z). It finds the daughter(karen, sue) fact, and assigns karen to X, and sue to Z. Now, husband(sue, Y) and be satisfied with bob being assigned to Y. The entire "if" statement has now been satisfied, and Prolog reports its solution to the user.

6.2 Representation of Dimensions in DAn

The Dimensional Analyzer program utilizes Prolog facts similar to the above examples to represent dimensions. Perhaps the most basic dimension information represented is what Ashley referred to as the Range Information. This information tells the system what side in an argument is furthered by the presence of a dimension. DAn implements this information with simple prolog fact statement of the form:

```
pro(Dimension, Side).
```

In place of the word Dimension is the actual name of the dimension, and Side is replaced by the actual side that the dimension favors. In every domain, there are exactly two sides. In evident partiality, the sides are "vacate" and "uphold." In HYPO's trade secrets domain, Ashley used "plaintiff" and "defendant." Using this pro fact statement, the dimensions attorney-client-relationship-with-party and knowledge-by-party are encoded:

```
pro(attorney-client-relationship-with-party, vacate).
pro(knowledge-by-party, uphold).
```

Other information is represented similarly. The description of each dimension that is used to explain the dimension during argumentation output is encoded in the following format:

```
describe(Dimension, Explanation).
```

In our two example dimensions, the Prolog fact statements are:

```
describe(attorney-client-relationship-with-party,
'the arbitrator had an attorney client
relationship with a party').
```

```
describe(knowledge-by-party,
        'the objecting party knew of the
        potential disqualification').
```

The second half of the explain fact is a phrase which is represented in Prolog through the use of single quotation marks.

The possible magnitudes of the dimensions are also implemented through the use of Prolog fact statements:

```
range(Dimension, Range-List).
```

The second part of the range fact statement is a list of phrases. A list in Prolog begins and ends with a bracket, and each element of the list is separated by commas from the other elements. Thus, the two element Prolog list including elements a and b is represented as "[a, b]." In DAN's Range-List, the elements in the list are phrases, which are used in argumentation output when explaining the magnitude of a dimension. For the two example dimensions, the range fact statements are:

```
range(attorney-client-relationship-with-party,
      ['the dimension did not occur',
       'the relationship had ended before the arbitration',
       'the relationship was on-going during the arbitration']).
range(knowledge-by-party,
      ['the dimension did not occur',
       'the knowledge was partial',
       'the knowledge was complete']).
```

The first element in every Range-List included in the Dimensional Analyzer program is always the magnitude value 'the dimension did not occur.' This value, also referred to as a "No" value magnitude, is discussed in more detail below in §6.6. The elements of the Range-List are accessed by the write-range procedure, which takes the magnitude of a dimension in a case represented by an integer, and prints out the appropriate phrase from the range list. (The Prolog facts used to encode the evident partiality domain can be found in Appendix C).

6.3 Representation of Cases in DAn

Cases in DAn are represented in a very similar manner. Various types of information is needed about each case, and this information is stored in Prolog fact statements. The following facts are used to represent a legal case in DAn:

```

citation(Case, Citation).*
year(Case, Year).*
shortname(Case, Shortname).*
longname(Case, Longname).*
holding(Case, Side).*
has_dimension(Case, Dimension, Magnitude).**
support(Case, Case-Cited-For-Support, Dimension-Being-
Supported).#
distinguishes(Case, Case-Distinguished, Dimension-Being-
Distinguished-Along).#
overrules(Case, Overruled-Case).#

```

Not every fact needs to appear in every case representation, only those facts marked by an asterisk are considered mandatory. In addition, the facts marked by a hash mark can appear numerous times in a single case representation, depending on the facts of the case.

The citation fact contains information similar to the Blue Book form of case citation,²⁶ but it does not include the year of the decision. The year, simply enough, is recorded in the year fact. The information relating to a case's jurisdiction is currently encoded along with the citation. However, it would be a simple matter to create a separate "jurisdiction" fact which encodes this information. This fact could be used by DAn to only retrieve relevant cases which have been decided in jurisdiction which governs the current fact situation. For this to work, some type of information must be encoded to show that cases from the Supreme Court and the Second Circuit will govern the Federal District Court of the Southern District of New York. These relationships could also be encoded in Prolog facts, but many facts would be needed to describe every state and federal jurisdiction.

²⁶ A Uniform System of Citation, published annually by the Harvard Law Review Association.

Because of the limited value encoding these fact would have on studying the usefulness of dimensions, I have chosen not to embody jurisdictional information in DAn.

The shortname and longname facts record the name of the case as it will be stated in DAn's output to the user. The holding fact represent the holding of the case, which can either be for one side in the domain or another. There is no middle ground. Thus, if a lower court awarded summary judgment in favor of upholding an award, and the higher court reversed stating that there is an issue of fact that must be determined, then the appellate court's decision would be a holding of "vacate." However, if the appellate court did not indicate a strong likelihood that the award would be vacated, then the decision would be more neutral in tone and would most likely not be encoded into DAn's case base.

The has_dimension fact encodes the applicable dimensions that a case contains. Since a case can have numerous dimensions, an unlimited number of has_dimension facts can be encoded about a single case. Also encoded into this fact is the magnitude that the dimension has in the case. The magnitude is indicated as an integer. A zero value magnitude indicates that "the dimension did not occur."²⁷ Other integers represent different values in a dimension's range list, as described above in §6.2. A magnitude of "1" would retrieve the first element in the range list after "the dimension did not occur," a "2" value would retrieve the second element, and so on. Notice that in this embodiment of a dimension's magnitude, comparisons may be done simply by looking at the integer value. The larger the integer, the greater the magnitude of the dimension.

DAn's representation of a case also stores information about how the case relates to other cases, encoding whether the case cites another case for support, or distinguishes a case, or (at least partially) overrules a case. For example, the in the Richco case discussed above, the court cited Cook for the proposition that a party with knowledge of a potential disqualification cannot remain silent and object to the arbitrator only after the award (Richco, 263 N.W.2d 204, 213, citing Cook, 449 F.2d 106). This information is

²⁷ See infra §6.6 for a complete discussion.

recorded in the following fact:

support(richco, cook, knowledge-by-party).

Similarly, if a case had distinguished another case because of the presence of a particular dimension, then the distinguishes fact would be used. The overrule fact allows the recording of the fact that the current case at least partially overruled another case. (The Prolog facts used to encode the evident partiality cases can be found in Appendix D).

6.4 Representation of Example Cases in Prolog

6.4.1 Abramovich

In Section §5.3.1, the case of Abramovich v. Pennsylvania Liquor Control Board, 416 A.2d 474 (Penn. 1980) was discussed, and the following three dimensions were assigned to it:

- 1) attorney-client-relationship-with-party (magnitude: On-Going),
- 2) improper-communications-with-party-or-lawyer (magnitude: Yes),
- 3) knowledge-by-party (magnitude: No).

Accordingly, the representation of this case in DAn was made with the following Prolog statements:

```
citation(abramovich, '416 A.2d 474 (Penn.)').
shortname(abramovich, 'Abramovich').
longname(abramovich, 'Abramovich v. Penn. Liquor Control
Board').
year(abramovich, 1980).
holding(abramovich, vacate).
has_dimension(abramovich, attorney-client-relationship-with-
party, 2).
has_dimension(abramovich, improper-communications-with-
party-or-lawyer, 2).
has_dimension(abramovich, knowledge-by-party, 0).
```

6.4.2 Richco

Similarly, Richco Structures v. Parkside Village, Inc., 82 Wis. 2d 547, 263 N.W.2d 204 (1978), discussed in §5.3.2 is represented in DAn as follows:

```
citation(richco, '82 Wis. 2d 547, 263 N.W.2d 204').
shortname(richco, 'Richco').
longname(richco, 'Richco Structures v. Parkside Village, Inc.').
year(richco, 1978).
holding(richco, vacate).
has_dimension(richco, relationship-with-party-or-lawyer-
    through-third-person, 1).
has_dimension(richco, substantial-business-relationship-with-
    party, 2).
has_dimension(richco, attorney-client-relationship-with-lawyer,
    2).
has_dimension(richco, knowledge-by-party, 1).
has_dimension(richco, partiality-affected-outcome, 0).
support(richco, cook, knowledge-by-party).
support(richco, san-luis-obispo, relationship-with-party-or-
    lawyer-through-third-person).
support(richco, johnston, minor-business-relationship-with-
    lawyer).
```

Not only are the dimensions mentioned in §5.3.2 included, but so also are the citations the court made to Cook, San Luis Obispo, and Johnston.

6.5 Fullness of Representation: Cook Industries

The above two examples show how facts derived from a case can be represented in Prolog. It is also helpful to do the analysis in the opposite direction. That is, examine a Prolog implementation of a case and see what information about the case can be derived from it. The following is DAn's representation of Cook Industries, Inc. v. C. Itoh & Co. (America) Inc., 449 F.2d 106 (2d Cir. 1971):

```
citation(cook, '449 F.2d 106 (2d Cir.)').
shortname(cook 'Cook').
longname(cook, 'Cook Industries, Inc. v. C. Itoh & Co.').
year(cook, 1971).
holding(cook, uphold).
```

Section 6: Representation in Prolog

```
has_dimension(cook, substantial-business-relationship-with-  
party, 2).  
has_dimension(cook, narrow-business-area, 1).  
has_dimension(cook, knowledge-by-party, 2).  
distinguishes(cook, commonwealth-coatings, knowledge-by-  
party).  
distinguishes(cook, commonwealth-coatings, narrow-business-  
area).  
support(cook, garfield, narrow-business-area).  
support(cook, garfield, knowledge-by-party).
```

The interpretation of the above is mostly self-explanatory. Cook was decided in 1971 and held that the arbitration award should be upheld. There are three applicable dimensions: substantial-business-relationship-with-party (magnitude value 2), narrow-business-area (magnitude value 1), and knowledge-by-party (magnitude value 2). By retrieving information about these dimension (their description from the "describe" fact and their range values from the "range" fact), we know the arbitrator in Cook was involved in an on-going substantial business relationship with one of the parties. We also know that the complaining party to the arbitration knew everything about the relationship before the arbitration award was made. Additionally, the court thought it was important that the arbitration concerned a narrow business area, in that the arbitrator's relationship with a party was therefore less avoidable than in other contexts.²⁸ The court distinguished the Supreme Court's holding in Commonwealth Coatings in two ways: first, by the fact that the complaining party in Commonwealth Coatings did not have any knowledge about the relationship prior to the award; and second, that the business area in Commonwealth Coatings was not as narrow as in this case. In other words, these two dimensions are the court's rationale for not following the Supreme Court's holding in Commonwealth

²⁸ More completely, this dimension indicates that the arbitration in the case required that an expert in the area of the dispute be involved as an arbitrator. Since the subject matter of the arbitration was a "narrow area," there are very few experts who are qualified to serve as arbitrators. Therefore, courts have reasoned, since there are so few people involved in this subject matter, it is only natural that some business dealings between a party and the arbitrator had taken place. When a court recognizes that an arbitration involves such a narrow business area, courts look for business dealings outside the ordinary course of business to indicate evident partiality. See, Garfield & Co v. Wiest, 432 F.2d 849 (2d Cir. 1970).

Coatings. The court then cited Garfield & Co. v. Wiest, 432 F.2d 849 (2d Cir. 1970) (the full name and citation of this case can be retrieved by searching for the longname, citation and year facts associated with garfield), for the propositions that when an arbitration concerns a narrow business area, a relationship between an arbitrator and a party is less avoidable. The court cites Garfield again for support of the idea that when the complaining party has knowledge of the relationship, that relationship cannot form the basis of vacating the award.

DAn's implementation of the Cook case includes most of the important points that are found in the actual court decision. However, no information regarding the dissenting opinion is included, nor is the legal standard that the court used (the "impression of possible bias" test from Commonwealth Coatings) represented. In many cases, the discussion of legal standard to be used by court is at the center of the lawsuit. Even in evident partiality cases, many courts devote the majority of their opinion to a discussion of what test to apply, and the application of the test to the facts is considered secondary and almost self-explanatory.²⁹ By not encoding this information, programs like DAn and HYPO have made a calculated decision on what they choose to analyze and represent, but they abandon hope of encoding a "full" representation of the cases that they encode.

6.6 "No" Magnitude Dimensions

Although a court may have explicitly stated that the fact situation represented by a particular dimension did not occur in the case, both DAn and HYPO would nevertheless assign that dimension to the case. However, the magnitude of the dimension would indicate that the dimension did not occur. In DAn, this "No" valued magnitude is represented in the `has_dimension` fact with a magnitude value of "0."³⁰ For every

²⁹ See, e.g., Andros Compania Maritima v. Marc Rich & Co., A.G., 579 F.2d 691 (2d Cir. 1978); Barcon Associates, Inc. v. Tri-Country Asphalt Corp., 86 N.J. 179, 430 A.2d 214 (1980).

³⁰ See Richco's partiality-affected-outcome dimension, §6.4.2.

dimension in DAn, this zero value is explained in the Range-List by the phrase "the dimension did not occur."

In encoding this information, both HYPO and DAn are distinguishing between a dimension which definitely did not occur, and a dimension which apparently did not occur because the court decision did not mention the necessary facts.³¹ The two programs, however, treat this distinction differently in their analysis. Although Ashley does not discuss the problem, when HYPO retrieves cases relevant to a current fact situation, it treats cases with a "no" magnitude for a dimension the same as cases which have a positive magnitude for the dimension. As a result, when HYPO analyzes a case with a (positive valued) dimension, it will retrieve cases in which the dimension definitely did not occur as on-point. Not only does this make direct comparisons and the creation of analogies more difficult,³² but it also has the consequence of occasionally excluding some cases with positive valued dimensions as a result of the analysis by the most-on-point and best case algorithms. I do not believe this is intentionally encoded into HYPO, but rather it is a consequence of the automated method of dimensional assignment that HYPO uses.³³

In DAn, this problem is avoided by using the get-dimension function to retrieve analogous cases instead of directly calling the has-dimension fact statement.³⁴ The get-dimension function works as if the has-dimension fact were called, except that it ignores

³¹ See *supra*, §5.3.1

³² Apparently, HYPO recognizes the existence of "No" magnitude dimension when it begins to make an argument, and does not include the dimension as part of its analogy between the cited case and the current fact situation. See *supra* note 7.

³³ Although Ashley does not explain why HYPO behaves in this manner, the following is one theory. Since HYPO recognizes the applicability of a dimension through the existence of factual predicates, a dimension would be considered applicable even though the prerequisite factual predicate which determines the magnitude of the dimension has a "No" value. When determining on-point, most-on-point, and best cases, apparently HYPO examines only the applicability of dimensions, and not their magnitudes. The "No" value magnitude is not discovered until argumentation, and at that point it is too late to reassess the on-pointness of the cases. It would certainly be possible to rework HYPO to fix this, but this has not been done. Perhaps Ashley will explain the purpose of using these "No" valued dimensions in this way in the forthcoming book based on his thesis, to be published by the MIT Press.

³⁴ See *supra*, §5.2.

"No" magnitude dimensions. This function is defined:

```
get-dimension(Case-name, Dimension, Size) :-
    has-dimension(Case-name, Dimension, Size),
    if(include-no-dimension-values-as-relevant,
        true,
        Size > 0).
```

The "if" clause tells Prolog to check whether the include-no-dimension-values-as-relevant parameter has been set.³⁵ This parameter decides whether or not to include "No" magnitude dimensions, allowing DAn to emulate HYPO if desired. If the parameter is set, DAn performs the next clause, which in this case is "true." The true clause indicates that nothing further must be done, and the case, dimension and magnitude are returned just as if the has-dimension fact were called directly. Since the parameter is usually not set, DAn will usually skip the true clause and check to see if the Size variable has a value greater than 0. If so, the case, dimension and magnitude are returned. If not, the get-dimension procedure fails, just as if the "No" valued dimension were never implemented.

The only time when DAn makes use of "No" valued dimensions is when a distinction is made between a case which has a positive value and a case with a "No" value for a dimension. In these circumstances, DAn reports that the latter case not have the dimension, plus it indicates that the court in the case directly stated this fact (perhaps indicating its importance). In this way, the distinction along this dimension is made stronger.

SECTION 7: OVERVIEW OF THE DIMENSIONAL ANALYZER

PROGRAM

7.1 Purpose

The Dimensional Analyzer program was written in order to further explore the usefulness of representing legal cases through dimensions in an artificial intelligence

³⁵ See *infra*, §10 for a discussion of DAn's parameters.

program. However, if the only purpose of the program were to *represent* the cases, then little knowledge would be gained about the power of using dimensions *to analyze* a legal domain. Rather, the present discussion would be limited to the fullness of the representation, as discussed above in §6.5.

The HYPO program moved beyond the representational level by exploring the power of dimensional analysis through argument production and hypothetical creation. The Dimensional Analyzer program also develops arguments by using the same basic algorithm implemented in HYPO. However, DAn enhances argument development and moves beyond it through the implementation of a query system and the creation of generalization rules. The query system allows the user to ask questions directly of the program concerning the cases and dimensions found in the domain. The program responds by presenting to the user a summary of the cases or dimensions which satisfy the question. The generalization rules analyze the cases in the domain and propose various generalizations which hold true for every case in the case base. These rules can then either be displayed as part of an analysis of the domain, or be included directly in the production of arguments. Their inclusion in arguments helps strengthen the points and counter-points produced by proposing appropriate generalizations about the domain.

7.2 Hypotheticals

DAn does not attempt to create hypothetical variations of a fact situation. The main reason for this is that HYPO and its progeny have deeply explored the algorithms and theories behind the generation of hypotheticals. Little could be added by duplicating such work. A second explanation is that factual predicates have not been implemented in DAn. Since factual predicates are required in order to implement near-miss dimensions, which are essential for some of HYPO's most interesting hypotheticals, a hypothetical generator in DAn would be severely handicapped.

A third, more interesting reason for not implementing hypotheticals is that the primary hypothetical creation algorithm of HYPO would not effectively work in the evident partiality domain, at least not as the evident partiality dimensions have been implemented in DAn. One of HYPO's algorithms suggests a hypothetical variation of a fact situation in order to force a potentially more-on-point counter-example to become directly on-point. Potentially more-on-point counter-examples are found in the extended claim lattice, which HYPO created by retrieving cases that were relevant to a fact situation's near-miss dimensions. These near-miss dimensions are dimensions whose prerequisites are satisfied except for the "triggering prerequisites." Ashley does not discuss the theory behind the selection of triggering and non-triggering prerequisites, but it appears that non-triggering prerequisites contain the general factual patterns which must exist so that the triggering prerequisites are possible. As an example, the prerequisites for HYPO's Security-Measures-Adopted dimension require

- 1) That a corporate plaintiff exists,
- 2) That the corporate plaintiff makes a product,
- 3) That there is product development information about the product,
- 4) That a corporate defendant exists,
- 5) That the corporate defendant makes a product,
- 6) That the plaintiff and defendant compete over their products, and
- 7) That security measures were taken by the plaintiff for the product.

Only the seventh prerequisite actually requires that security measures be taken. This prerequisite is considered by HYPO to be the triggering prerequisite. The first six simply verify that the general fact pattern is such that it makes sense to discuss the security measures taken by the plaintiff.

The problem is that in the evident partiality domain, every case involves a factual pattern which allows practically every dimension to occur. A substantial business

relationship between an arbitrator and a party can occur regardless of the parties corporate makeup or the subject of their dispute. Thus, there would be no non-triggering prerequisites for the evident partiality dimensions, and practically every dimension would be considered a near-miss to every case. The extended claim lattice would then contain every case in the case base, and the hypotheticals suggested by the program would try to change every new fact situations into the same cases, regardless of the facts of the present situation being considered. Although there still might be some use in such hypotheticals, the information gained by such hypotheticals could be gain just as easily by running a test case through the system in which every dimension is considered applicable.³⁶

7.3 Menu System

The Dimensional Analyzer program is implemented so that all user choices are made through menus. The implementation of a menu system in the backward chaining environment of Prolog was quite simple. Most menus were implemented to call themselves recursively until the user chooses to exit the menu. At that point, in the original version of the program, the parent menu was called. When the user selected to abort the main menu, the system-defined abort procedure was called to abort the run. Unfortunately, this method of menu implementation caused the system stack (which keeps track of the clauses Prolog is trying to satisfy) to overflow, as the nesting became too deep. To avoid this problem, some of the menus were designed to fail when completed, clearing the system stack of the clauses that were attempted while running that menu. In order to do this, the calling procedure has to anticipate failure by the menu and provide for an alternate way to succeed.

7.4 Introduction to DAn Modules

7.4.1 Domain Statistics

The Domain Statistics procedure displays to the user the present configuration of

³⁶ The option of running such a test case is given in the New Cases module described infra, §7.4.3.

the domain, including the number of cases currently in the knowledge base, the number of dimensions, and the breakdown of the cases and dimensions. Cases are broken down by their holdings, including cases which have a holding of unknown. Dimensions are broken down in two ways. First, according to which side they favor. And second, according to the number of overwhelming, both-sided, and necessary dimensions as determined by the generalization rules.³⁷

7.4.2 New Domain

The New Domain procedure is responsible for letting users input a new domain directly into DAn. In order to define a domain, New Domain must ask the user for the following information:

- 1) The name of the domain (ie, 'Evident Partiality'),
- 2) The two sides to a dispute in this domain (vacate and uphold),
- 3) Phrases describing the two sides to be used in argument production ('vacating the arbitration award,' and 'upholding the arbitration award'),
and
- 4) The following information about each dimension in the domain:
 - a) Name (ie, knowledge_by_party),
 - b) Side the dimension favors (uphold),
 - c) Number of non-"No" magnitude values possible (2),
 - d) A phrase for each magnitude value, not including the "No" value phrase 'the dimension did not occur,' ('the knowledge was partial', 'the knowledge was complete'), and finally
 - e) A phrase used to describe the dimension ('the objecting party knew of the potential disqualification').

This information is the total amount of information needed to define a new domain. Once the user inputs the information, data concerning the old domain is erased from the current

³⁷ See *infra*, §7.4.4.

memory, and the new domain data is asserted. The new domain also is written to an external file in Prolog fact statements, allowing the domain to be recalled at any point. This also allows the user to edit the file externally, either to correct a mistake made in entry, or to add an additional dimension. Eventually, these latter actions will also be accomplishable from inside the New Domain procedure.

7.4.3 New Cases

The New Cases procedure works almost identically to New Domain. The user is requested to input information, which is then asserted into the present knowledge base. General information about a case (such as the various names used by DAn, plus the holding, date, and citation) is entered directly by the user in response to questions asked by DAn. The dimensions are assigned to the case through the help of a numbered list of dimensions which appears on the screen. A menu allows the user to:

- 1) Assign a dimension and magnitude value to the case,
- 2) Assign every dimension to the case with a minimum magnitude,
- 3) Ask for a definition of the dimension,
- 4) List the dimensions currently assigned to the case, or
- 5) Remove all the dimensions currently assigned to the case.

The first choice (1) asks the user to enter the number of the dimension that should be assigned. Once assigned, a list of possible magnitudes is display for the user to choose. The second selection (2) allows a user to quickly assign every dimension to the case being entered. Analyzing such a case is a useful way to analyze a domain as a maximum number of cases are retrieved as best cases to be argued. The third choice (3) simply displays the describe parameter of the dimension. The fourth (4) and fifth (5) choices allow for the detection and correction of inputting errors.

Once the dimensions are assigned to the case, the user must enter the cases (if any)

that the new case cites for support, distinguishes along a dimension, or overrules. This data entry is also helped by the displaying of the numbered dimension list.

After entering a case or several cases, the user has the option of saving the cases to a file name. The information is stored in Prolog fact statements, allowing reuse by DAn at any time. The cases are also stored in a temporary file, but this file is eventually erased by DAn.

7.4.4 Domain Analysis and Generalization Rules

The Domain Analysis procedure analyzes the current legal domain using generalization rules, and then outputs the satisfied rules and explanations to the user. The domain analysis need only be done once during a program run, as the results of the analysis are asserted into the present Prolog knowledge base. The results are not stored in a file, so a new domain analysis must be done each time the Dimensional Analyzer program begins. In addition, the analysis is redone if a new domain or case is entered or retrieved from a file. This is because the presence of a new case (and especially a new domain) will most likely alter the rules produced by the analysis.

The effect of generalization rules are explained in more detail in Section 9. Generally, the rules are designed to tell something about the domain and the domain's dimensions that would not be obvious on first examination. Currently five generalization rules have been implemented.

7.4.4.1 Presumption

The presumption rule indicates that there is a presumption within the domain favoring one side of the dispute over the other. This presumption is noticed by finding cases which favor a particular side, but have no applicable dimensions which favor that same side. A presumption can only exist in favor of a single side. Thus, if cases exist on both sides of a domain which don't have any supporting dimension, then there is no

presumption in the domain.

The **Presumption** rule is used in the **Argue** procedure when no other argument could be made due to the lack of best cases. In DAn, the presumption rule takes the form of the Prolog fact statement

presumption(Side, Cases).

where Side indicates the side that the presumption favors, and Cases is a list of cases which support the existence of the presumption. Every case in Cases is a case which held for Side, but did not contain any dimensions in favor of Side.

7.4.4.2 Both-Sided Dimensions

When a dimension is associated with cases which have held for both sides in a domain, then the dimension is considered "both-sided." Such dimensions are not useful in and of themselves, but the existence of the both-sided dimension generalization rule aids in the discovery of other rules. In DAn, the both-sided dimension rule is encoded:

both-sided(Dimension, Cases).

where Dimension is the name of the dimension and Cases is a list of all the cases containing Dimension as applicable.

7.4.4.3 Necessary Dimensions

The necessary dimension rule indicates the existence of a dimension which appears to be a necessary condition for a particular holding. The rule is discovered by searching for dimensions which appear in every case that holds for a particular side.

Necessary dimensions are used in the **Argue** procedure when a distinction is being made between the current fact situation which lacks the necessary dimension and a cited case or counter-example which contains it. DAn implements this rule as:

necessary(Dimension, Side, Cases).

where Dimension is the necessary dimension, Side is the side that requires Dimension, and Cases is the list of all cases holding for Side (and also having Dimension as applicable).

7.4.4.4 Overwhelming Dimensions

The overwhelming dimension rule indicates that certain dimensions exist which overwhelm the other dimensions. This rule is found by noticing that certain dimensions only occur in cases which have held for the same side that the dimension favors. In other words, whenever the dimension occurs, the case holds for the side that the dimension favors.

Overwhelming dimensions are used by Argue to support an analogy between the cited cases and the current fact situation. This type of argument states basically that the cited cases should be followed because they share an overwhelming dimension with the fact situation, and every case which has this dimension has been decided in this way. Overwhelming dimensions are implemented through the following Prolog fact:

overwhelming(Dimension, Cases).

where Dimension is the overwhelming dimension and Cases is the list of cases which have Dimension as an applicable dimension.

7.4.4.5 Overwhelming Dimension Pairs

The overwhelming dimension pair rule is similar to the overwhelming dimension rule, except that this rule notices instances where two both-sided dimensions acting together have an overwhelming effect. In order for this rule to be satisfied, the two dimensions in the pair must both favor the same side, and be found together only in cases which have held for that side.

This rule is used in Argue identically to the way the overwhelming dimension rule is used. It is implemented as follows:

```
overwhelming_dimension_pair(Dimension1, Dimension2,
                             Cases).
```

where Dimension1 and Dimension2 are the two dimensions of the overwhelming dimension pair, and Cases is a list of all cases which contain both dimensions as applicable.

7.4.4.6 Dominated Dimension Pairs

The dominated dimension pair rule also finds pairs of dimensions that appear to have a special effect on each other. In this instance, a certain, "dominated" dimension may have an overwhelming effect on all cases, except when the dimension appears with another, opposite-sided "dominating" dimension. When the two dimensions appear together, the dominated dimension has no effect and the cases always hold for the opposite-side, consistent with the dominating dimension. This rule will find dimensions that are very influential, except when countered by a particular, dominating dimension. Dominating dimension pairs are also used by the Argue procedure to strengthen an analogy between a cited case and the current fact situation.

To find dominating dimension pairs, several requirements must be met. First, the dominated dimension must be both-sided. Second, whenever the dominated and dominating dimensions appear together, the case must hold for the side that the dominating dimension favors. Third, whenever the dominated dimension appears alone, it must be overwhelming.

DAn implements the dominated dimension pair in a simple Prolog fact statement:

```
dominated-dimension-pair(Dominated-Dimension, Dominating-
                          Dimension, Cases).
```

where Dominated-Dimension is the dominated dimension, Dominating-Dimension is the dominating dimension, and Cases is the list of all cases in which the two dimensions appeared together.

7.4.5 Argue, and its Supporting Procedures

Argument production in DAn is very similar to the modules which produce arguments in HYPO. Both programs retrieve relevant cases, analyze those cases by determining which ones are the "most-on-point" and "best" cases, and then produce a 3-ply argument based on the best cases. However, the DAn's Argue procedure improves upon HYPO in several ways. First, it does not treat "No" valued dimensions as on-point.³⁸ Second, it implements two-different most-on-point criteria.³⁹ Third, it allows the user to choose whether less-on-point counter-arguments will be made.⁴⁰ Fourth, it groups similar best cases together into a single argument.⁴¹ And finally, it includes the use of generalization rules which create stronger arguments for a position.⁴²

The first step in analyzing a fact situation is to choose the situation that is to be analyzed. The selection can be entered directly, or by choosing a fact situation from a list created by DAn. The possible situations that may be analyzed include all cases currently in the case base for a domain, plus any case the user has entered from the New Cases procedure. Once selected, DAn displays a summary of the fact situation on the screen and gives the user the following choices:

- 1) Argue the case from the side for which the case held,
- 2) Argue the opposite side,

³⁸ As explained in §6.6.

³⁹ See *infra*, §10.1.

⁴⁰ See *infra*, §10.4.

⁴¹ As explained further in this section.

⁴² *Id.*

- 3) Argue both sides,
- 4) Argue the case using a case chosen by the user.

Once the choice is made, DAn begins to analyze and argue the fact situation.

To find relevant cases, DAn creates a list of the supporting and opposing cases. These lists include both the name of the case and the dimensions that each case shares with the current fact situation. The lists of supporting and opposing cases are then sent to the most-on-point procedure to determine which cases are to be considered most-on-point. Currently two algorithms can be used to compare cases for on-pointness, both of which are explained below in Section 10.1. The best cases are then determined using the same technique implemented in HYPO.⁴³

The most-on-point and best-cases are sent to DAn's Argue procedure, which does the actual argumentation of the current fact situation. If no best cases exists, then Argue attempts to make an argument based on the Presumption generalization rule. The lack of best cases indicates that there are no cases which share with the current fact situations even a single dimension which favors the side being argued. Under HYPO's algorithm, no argument could then be made. In DAn, however, it is still possible to argue that even in the absence of a dimension favoring a particular side, that side should win because a Presumption rule may exist in favor of that side. If the rule does exist, Argue presents and explains the presumption, and cites the cases which support the presumption. If possible, Argue will try to cite cases which support the presumption and are also found in the most-on-point cases for the current fact situation. If no such cases exists, then all of the cases which support the presumption are cited.

If best cases do exist for a side, then DAn's Argue procedure will make a 3-Ply style argument based on those cases. Unlike HYPO, Argue does not argue each "best case" separately, but rather it combines cases which share a common set of dimensions into a single argument. This choice was made because the basic argument for each case in such

⁴³ Explained in §3.4.

a set is very similar (for instance, the counter-examples generated in response to each such case are identical). Argue asserts the proposition that this set of cases should be followed, and uses the shared dimensions to create the necessary analogy to the present fact situation.

Before allowing the other side to distinguish the cases and present counter-examples, the Argue procedure will attempt to further this particular proposition by arguing through the use of generalization rules. The first step in creating a generalization rule argument is to search for applicable rules. This is done by examining the dimensions that the current cases share with the present fact situation. If any of the dimensions is an Overwhelming dimension, then Argue includes this fact in the argument. In addition, the Argue procedure searches for Overwhelming Dimension Pairs and Dominated Dimension Pairs among the common dimensions. If either rule is applicable, then the procedure outputs an argument based on that rule. It would never be necessary to determine which side the rule favors, for if the present cases fall under the rule, then the rule will favor the same side as the cases. This is inherent in generalization rules (there are no exceptions). The argument produced simply states that all cases in the case-base which fall under this rule have been decided only in favor of the current argument.

This method of arguing rules connects the rules with the cases that supported the creation of the rule in the first place. If the rules were argued separately, the complete argumentation output would be repetitive, as the same combination of dimensions would appear in the rule argument as well as the case argument. In addition, these rules are best seen as factors which make an argument based on cases with certain dimensions stronger. In effect, they allow more weight to be placed on the fact that these particular dimensions are shared or not shared with the current fact situation.

After arguing the affirmative rules, the Argue procedure begins stage two of the 3-Ply argument, and attempts to distinguish each of the cases proposed by the first side. Distinctions are made in a manner identical to that done by HYPO. The Argue procedure

notes the existence of additional dimensions in either the cited case or the current fact situation, as well as the existence of a difference in magnitude along a shared dimension. As in HYPO, the procedure makes these distinctions only when they can form the basis of an argument that the cited case should not be followed (See §3.5 above). Unlike HYPO, however, DAn's Argue procedure attempts to determine when the existence of an additional dimension in either the cited case or current fact situation can be considered an "important" distinction. Argue considers a distinction important if a court used the same distinction when deciding its case. Thus, Argue checks every distinction made on the basis of an additional dimension to see if that dimension has ever been used by a court to distinguish another case. If it has, Argue points out the importance of the distinction and cites the appropriate cases for support.

Argue also checks to see if the additional dimension being distinguished has been determined to be necessary dimension by the Domain Analyzer. If so, and the cited case being distinguished favors the same side as the necessary dimension rule, Argue points out that the current fact situation is very different from the cited case in that the fact situation is lacking a dimension which every other case which held for that side contained. Note that this distinction will only occur when the cited case contains the dimension as applicable and the current fact situation does not. If the cited case did not contain the dimension, then it could not have held for the same side that the necessary dimension favors.

The next step for the Argue procedure is to find counter-examples to the cited cases. The procedure finds more-on-point, equally-on-point, and overlapping counter-examples in the sense that those concepts are defined in HYPO.⁴⁴ It cannot find potentially-more-on-point counter examples as near-miss dimensions are not implemented in DAn. Additionally, I have chosen not to implement boundary counter-examples, because the evident partiality dimensions have very limited variance in possible magnitudes and boundary counter-examples would therefore have been of little practical use.

⁴⁴ See *supra* §3.5.

When the counter-examples have been proposed, Argue then moves into the third stage of the argument by defending the analogy initially proposed. To do this, Argue simply distinguishing the cited counter-examples from the current fact situation by using the same distinction procedure described above. Once this is done, argument production is complete.

7.4.6 Query System

The Query System built into DAn allows the user to directly access the case base and retrieve information through the use of menu directed questions or queries. The Query System can handle queries about cases and queries about dimensions. The two types of queries are handled separately and cannot be mixed. The initial query menu allows the user to choose the type of query desired. In addition, the user can choose to find every case or every dimension in the case base.

The queries are created by the Query System in the form of a Prolog rule. The rule is constructed in parts, saved to a temporary file, and then consulted into Prolog's knowledge base. The Query System then finds all cases or dimensions which meet the query's conditions, and prints out a summary of each case or dimension retrieved.

The Case Query Menu handles queries aimed at finding cases. The menu allows the user to search for cases that:

- 1) have or do not have a particular dimension,
- 2) held for a particular side,
- 3) distinguish or are distinguished by a case along a dimension,
- 4) cites or is cited by a case for support along a dimension, or
- 5) overrules or is overruled by a case.

The first, third and fourth choices then allow the user to either select a particular dimension, or let the Query System define the clause in such a way that any dimension would satisfy it.

After selecting a choice from the Case Query Menu, the user can either select another clause from the menu or end the query. If another clause is selected, it is joined to the previously created clauses by either an 'Or' or an 'And' conjunction. The conjunctions are implemented in the Prolog query through the Prolog 'And' (comma) and 'Or' (semi-colon) punctuation.

The Dimension Query menu works in a similar manner, and gives the user the option to search for dimensions that:

- 1) are or are not found in a particular case,
- 2) favor a particular side,
- 3) are found in cases that have held for both sides of the issue (ie, are both-sided dimensions),
- 4) are overwhelming,
- 5) dominate or are dominated by another dimensions (ie, are part of a dominated dimension pair), or
- 6) are part of an overwhelming dimension pair.

Once a clause is selected, the user can either end the query, or can select another clause which will be connected to the previous clauses by an 'Or' or an 'And' clause.

SECTION 8: GENERALIZATION RULES

8.1 Learning through Generalization Rules

Generalization rules provide DAN with the ability to draw general information about a domain from the cases that exist in its case base. The information discovered is stored in the form of rules which hold true for every known case. An example generalization rule from the legal domain of negligent truck drivers might be: "If a case contains the dimension truck-driver-admitted-guilt, then it will hold for the plaintiff."⁴⁵ Another generalization rule might state that "the dimension plaintiff-was-injured-or-incurred-damages must occur

⁴⁵ This is an overwhelming dimension rule.

in a case in order for the plaintiff to win."⁴⁶ These rules tell us it is very damaging for a truck driver to admit her guilt. Indeed, no truck driver has ever overcome such an admission according to our cases. The rules also tell us that if a plaintiff did not receive any injuries or damages as a result of the truck driver's actions, then she cannot collect in her suit, at least according to the known cases.

These two negligent truck-drivers generalization rules provide us with some very useful information about the domain without ever having to analyze a particular case. A student of tort law might tell us that the first rule involves the admission of negligence, so it is not surprising that truck drivers rarely or never win in such situations. The second rule involves cases where the plaintiff did not sustain any damages, so her negligence claim could not win. These rules are therefore not surprising, but they do represent useful information that has been "learned" about the domain from the cases in the case base.

Although both of these example generalization rules are consistent with common understanding about the law, note that they are also contradictory in the situation where the truck driver admits guilt, yet the plaintiff was not hurt in any way. Tort law tells us the plaintiff loses in this situation (damages are a required element of a negligence claim), but DAn can only argue the conflicting rules. Notice also that if the plaintiff-was-injured-or-incurred-damages dimension was defined negatively (plaintiff-was-uninjured-and-did-not-incur-damages), the generalization rule would become similar to the confessing truck driver rule: "If a case contains the dimension plaintiff-was-uninjured-and-did-not-incur-damages, then it will hold for the defendant."⁴⁷

DAn presents what it has learned from generalization rules such as these in its Analyze Domain procedure. Analyze Domain outputs every existing generalization rule which holds true for the domain, along with an explanation about the purpose and meaning of those rules. Aside from the overwhelming and necessary rules used in the above

⁴⁶ This is a necessary dimension rule.

⁴⁷ Thus being changed from a necessary dimension rule to an overwhelming dimension rule.

examples, DAN will output any presumption, overwhelming dimension pair and the dominated dimension pair rules that exist for the domain. Each of these rules provide clues to the workings of a domain. If a presumption is found, for instance, this may indicate that courts have explicitly concluded that one side in a dispute should prevail unless the other side can overwhelm that presumption. On the other hand, the existence of a presumption rule may indicate only the presence of an undiscovered dimension which is influencing the court in favor of a particular side.⁴⁸ Only further legal research can indicate the true meaning of a generalization rule, but DAN will at least point out the fact that some relationships exists that needs explaining.

8.2 Weighting through Generalization Rules

Generalization rules are also used by DAN to aid in argument production. The presumption rule allows DAN to make an argument for a position even in the absence of any best cases to cite.⁴⁹ The overwhelming dimension, overwhelming dimension pair, and dominated dimension pair rules allow DAN's Argue procedure to supplement its analogy argument with generalization rules which hold true for all the cases in the case base.⁵⁰ The necessary dimension rule is used to help bolster a distinction that is being made by pointing out the importance of the dimension that is lacking in one of the cases.⁵¹

In the above uses of generalization rules, all but the presumption rule are used by Argue to bolster the impact of certain dimensions in its argument. In other words, the use of generalization rules in argument production allows certain dimensions to take on more weight than other dimensions. This use of generalization rules is therefore a cautious method to add weighting to dimensional analysis. The weighting does not change the

⁴⁸ Ashley describes the presumption situation as an "anomaly," and he suggests that the presence of such an anomaly may allow an extended version of HYPO to realize when it needs to learn a new dimension. Dissertation, p. 339.

⁴⁹ See *supra* §7.4.5

⁵⁰ *Id.*

⁵¹ *Id.*

criteria used to select cases, nor does it even change structure of the argument produced. Rather, the weights are used to allow the user to rely more confidently on the power and impact of certain dimensions.

As an example, suppose the knowledge-by-party dimension was determined to be an overwhelming dimension in the evident partiality domain. When this fact is displayed by the Argue procedure in analyzing a fact situation, that dimension is going to take on crucial importance to the user of the system who plans to argue the case before a court. The existence of other dimensions in this fact situation will still be interesting, but the dimension noted by the generalization rule will most likely be at the heart of any argument made by either side.

8.3 Support Needed

In determining the existence of generalization rules in a domain, DAn looks not only at the basic requirements for the rules as explained in Section 7.4, but also at the number of cases which support the existence of the rule. Each Prolog fact that defines a generalization rule contains a Cases value which contains a list of the support cases for a rule. Before a rule is asserted into Prolog's knowledge base, the number of cases in this list is compared with the number of cases currently required by the support level needed parameter. If there are fewer support cases than allowed by the parameter, then the generalization rule is not asserted. Only if enough support exists is the rule asserted. The support level needed parameter exists to prevent rules from forming when the rule is only supported by only one (or several) case(s). The default level of support needed is two cases, but this parameter can be changed by the user.

SECTION 9: EXAMPLE ANALYSIS BY DAn

9.1 Analysis of the Evident Partiality Domain

The Analyze Domain procedure in DAn allows a newcomer to the evident partiality domain to learn something about the domain even without analyzing a particular case. This procedure is chosen directly from the Main Menu of DAn. Once chosen, DAn will determine what generalization rules exist in the evident partiality domain, and then present the results to the user. The actual output made by DAn for this domain is shown in Figure 8.

The first section of this domain analysis reminds the user that the applicable generalization rules depend upon the setting of the support level needed parameter, and that the level of support needed at the time of this run was two cases (the default).

Next, the presumption rule is presented and explained. Here, DAn explains that a presumption exists in the evident partiality domain in favor of upholding the award. The number of cases cited in support of this rule indicates that this presumption is a strong one, and the analysis of cases in this domain supports this conclusion.

Four overwhelming dimensions exist in the evident partiality domain, three of which (potential-conflicts-waived-in-contract, relationship-unknown-to-arbitrator, and unanimous-arbitration-panel) cite two cases for support. The fourth dimension (narrow-business-area) cites three supporting cases. All four dimensions favor the uphold side of the domain. Each of these overwhelming dimensions can be characterized as very strong dimensions and would be very difficult to refute in an argument, yet it is true that none of them can cite a large number of cases for support.

Only one overwhelming dimension pair exists in the evident partiality domain, that being substantial-business-relationship-with-party paired with attorney-client-relationship-with-lawyer. It is hard to imagine any actual synergy taking place between these two dimensions, so it is likely that this generalization rule reflects only a statistical quirk.

Finally, two dominated dimension pairs have been found in the domain: substantial-business-relationship-with-party being dominated by narrow-business-area, and

attorney-client-relationship-with-party dominated by knowledge-by-party. These two rules are very interesting, in that they represent perhaps the two strongest pro-vacate dimensions being dominated by some of the strongest pro-uphold dimensions. Much could be learned about the evident partiality domain simply by studying the way these dominated dimension pairs were worked out in the cases.

9.2 Analysis of the Richco Fact Situation

9.2.1 Arguing the Case

DAn's argument output for the facts of the Richco case⁵² is shown in Figure 9. This figure includes arguments for both the uphold and vacate positions, and shows the basic format of DAn's argumentation output. The parameters during this analysis of Richco were:

- 1) Use the COVERAGE-1 most-on-point technique,
- 2) Generalization rules are used in argument production,
- 3) Exclude "No" valued dimensions in the analysis,
- 4) Use overlapping counter-examples, and
- 5) Two support cases are needed to form a generalization rule.⁵³

Comparison with HYPO's 3-Ply argument in Figure 1 shows how much DAn has borrowed from HYPO. The primary distinctions between these two arguments are:

- 1) Various changes in the language,
- 2) The use of previous distinctions to increase the weight given to distinctions based on certain dimensions, and
- 3) The use of the overwhelming dimension pair generalization rule.

9.2.2 Query Along Knowledge-by-Party Dimension

After analyzing the Richco situation, it is very likely that a lawyer trying to argue

⁵² These facts are discussed in details in §5.3.2 and §6.4.2.

⁵³ See infra §10 for a complete discussion of these parameters.

for the pro-vacate position would be concerned about the knowledge her client had of the arbitrator's relationships with the opposing party. Although DAn offered the Barcon and Freeport cases as examples of cases where the award was vacated even though the complaining party had knowledge of the relationship, the attorney may want to know of every such case. Comprehensive knowledge about how to overcome this dimension may be crucial to her case.

To gain this knowledge, the attorney could use DAn's Query System to retrieve the appropriate cases. The cases found are shown in Figure 10. Notice that the first lines in the query output represents the Prolog format of the query that was entered. The query searched for cases which contained the knowledge-by-party dimension and yet held for the pro-vacate position. Four cases were retrieved, including the Richco decision.

9.3 Analysis of the Cook Fact Situation

DAn's argumentation of the Cook Industries fact situation is found in Figure 11. In addition to the features shown in the Richco analysis, this argument also shows how overwhelming dimensions and dominated dimension pairs are argued by DAn. The output found in Figure 11 used the same parameters as the Richco output did.⁵⁴

9.4 Analysis of Commonwealth Coatings: Upholding Award

Figure 12 is DAn's analysis of the pro-uphold position given the facts of Commonwealth Coatings. The argument produced is the presumption generalization rule argument, because Commonwealth Coatings contains no pro-uphold dimensions. If generalization rules were not used, no argument could be produced for this position.

SECTION 10: PARAMETERS

DAn allows the user to set several parameters that are used to control the

⁵⁴ See *infra* §10.1.

functioning of the Argue, Analyze Domain, and Query System procedures. To change parameters, the user selects the change parameters option from the Main Menu. The current values of the parameters are then display and the user may select to change any or all parameters.

10.1 Most-On-Point Criteria

Currently, two most-on-point criteria are implemented in DAn, both of which are based on the most-on-point technique used by HYPO. The COVERAGE technique is basically the same algorithm used in HYPO. It determines whether one case is "more on point" than another by examining the number of dimensions that each case shares with the current fact situation. If one case shares the same dimensions that another case shares, plus at least one more dimension, then that case is considered more on point.⁵⁵

DAn's other most-on-point technique is COVERAGE-1. It uses the same technique as COVERAGE but it further analyzes cases that are considered equally-on-point with the current fact situation. If a case is considered equally-on-point by COVERAGE, but has fewer non-shared dimensions than another case, then that case is more on point according to COVERAGE-1. This technique is useful in domains such as evident partiality where a large number of cases have been implemented. COVERAGE-1 is the default most-on-point technique.

10.2 Include/Exclude Generalization Rules in Arguments

This parameter determines whether DAn will use the generalization rules it develops in argument production. If generalization rules are not used, then the argument production will look similar to the 3-Ply arguments of HYPO. The default is to include generalization rules in arguments.

⁵⁵ See supra § 3.3 for a more in-depth discussion of this technique.

10.3 Include/Exclude "No" Valued Dimensions in Analysis

DAn was implemented in such a way that dimensions which had a "No" magnitude would not be considered when determining on-point cases. In addition, such dimensions are ignored when DAn creates generalization rules and retrieves cases in response to a query. However, this parameter allows the user to override this default and to include such dimensions in DAn's analysis. When "No" valued dimensions are included in analysis, DAn operates more like HYPO.

10.4 Include/Exclude Overlapping Counter-Examples

Overlapping counter-examples are counter-examples which are not as on-point as the cited case, but contain some of the same shared dimensions and hold for the opposite side. Usually, these counter-examples will be useful in arguing against an analogy, but a user may wish to stop the use of these examples if only equally or more-on-point counter-examples are desired. The default is to include overlapping counter-examples.

10.5 Support Level Needed

As explained in §8.3, generalization rules are considered to exist only if they have enough support from the cases in the case base. The number of cases needed to create a generalization rule can be set by this parameter. The default is that two cases are needed to create a generalization rule.

10.6 Output Control

This parameter allows the user to direct the output of the Argue, Analyze Domain, and Query System procedures to a file. If the value of this parameter is "user," all output is to be sent to the terminal screen. Even when output is being sent to a file, DAn directs all menus and input requests to the terminal screen. The default is that output is directed to

"user."

SECTION 11: SUITABILITY OF DOMAINS FOR DIMENSIONS

11.1 Evident Partiality Domain

The trade secrets domain analyzed Ashley's HYPO program is highly suited for dimensional analysis. Courts and commentators had listed numerous factors or dimensions as important in determining the existence of a trade secret. This meant not only that much of the selection of dimensions been done by domain professionals,⁵⁶ but also that experts had legitimated the analysis of trade secrets using a set of dimension-like factors. Courts faced with trade secret issues discuss them in a way very similar to grouping dimensions on one side of an issue and the comparing them to the dimensions on the other side. Because the trade secrets domain was so well suited to this type of analysis, many have asked whether or not dimensions can be effectively implemented in other domains which do not share these features. After my experience in implementing the evident partiality domain, I believe that they can.

The evident partiality domain is similar to the trade secrets domain in that the primary issue involves a mushy, ill-defined legal concept. In trade secrets cases, courts are faced with trying to determine when a trade secret exists and when it is infringed upon. In the evident partiality domain, courts attempt to add meaning to phrases such as "evident partiality," "impression of possible bias," "appearance of bias," and "trivial relationship." In both domains, the courts have attempted to define and limit the scope of these concepts, but one cannot help but feel that much of what guides a court is the basic underlying facts, and the court's beliefs as to which party is more deserving of sympathy. At the same time, courts have been aware of precedent, and have avoided holding in direct contradiction with cases involving similar facts. Thus, on the surface, the two domains appear quite similar.

However, the domains differ in several important ways. First of all, no court has

⁵⁶ As Ashley claims. See *supra*, §5.1.

ever spoken of the evident partiality domain in terms of weighing a set of factors for each side. Instead, most courts try to divine inspiration from the words of the Commonwealth Coatings decision, and then directly apply their inspiration to the facts of the case. Second, it appears that the dimensions that Ashley created in HYPO were given fairly equal in weight by the courts. This is not true in the evident partiality domain, where certain pro-uphold dimensions are given much more importance by the courts. Finally, there is a strong presumption in the evident partiality domain in favor of upholding the arbitration award. No such presumption existed in the trade secrets domain.

In spite of these differences, dimensions were successfully implemented in the evident partiality domain. The representation of cases included all of the factual factors considered important by the court, and the resulting arguments were both reasonable and impressive. In addition, the generalization rules were able to develop rules which noted some important trends and conclusions about the domain.

On the other hand, no attempt was made by Ashley or myself to embody a court's reasoning, in that none of the standards or tests used by a judge were encoded. Ashley was not bothered by this lack in his trade secrets domain, referring to such standards only when discussing the problems of creating a program which deals with abstract legal principals.⁵⁷ In the evident partiality domain, however, several courts placed a great deal of emphasis on the standard which should be applied to a given fact situation. This type of information would be difficult to represent, and even more difficult to reason with, but it would be an important part of representing cases in the evident partiality domain.

11.2 Criminal Appeal Domain

In a 1989 paper, Simon Mendelson of Harvard Law School attempted a dimensional analysis "of the law governing appeals in criminal cases."⁵⁸ In his paper,

⁵⁷ Dissertation, p. 333.

⁵⁸ Simon Mendelson, "An Attempted Dimensional Analysis of the Law Governing Government Appeals

Mendelson commented on the inability of his dimensions to encode what he considered to be important factual details of this domain. He concluded, somewhat surprisingly, that the failure of dimensions to fully capture the true emphasis of the decisions was due to the domain's fact sensitivity.⁵⁹ The true source of the problem was the selection of dimensions.

Mendelson analyzed seven Supreme Court cases on the subject of when a prosecutor is allowed to appeal a criminal case without violating the defendant's Fifth Amendment right against Double Jeopardy. After his analysis, Mendelson developed 10 dimensions for the domain:

When-Motion-Raised,
When-Motion-Decided,
Decision-Below-Label,
Type-of-Error-Below,
Jury-Verdict-Below,
Further-Proceedings-Required,
Mistrial-Character,
Grounds-of-Decision,
Basis-of-Motion, and
Who-Initiated-Motion.

At first glance, it is obvious that these dimensions could not be used for the creation of arguments by either HYPO or DAn. This is because the dimensions do not favor either side of an argument. Rather, each dimension contains information which could favor both sides. Thus the existence of these dimensions could not be used as the basis for an argument.

After analyzing the Supreme Court cases in this domain, another problem appears to be that Mendelson did not create dimensions for factual events which clearly played important roles in the Court's decisions. Mendelson complained that to do so "seems like

in Criminal Cases," in Second International Conference (1989).

⁵⁹ Mendelson also suggests that the problem might stem from the domain's "conflicting nature" and the fact that it was an unsettled area of the law. *Id.*

cheating," but the representation of relevant facts is what dimension are designed to do. Additionally, several of Mendelson's dimensions represented facts which the Court never considered relevant.

In attempting to understand the limits of dimensional analysis, I spent an afternoon analyzing the seven Supreme Court cases selected by Mendelson. The following ten dimensions seemed more appropriate to the domain:

further-proceedings-required (pro: disallowed),
labelled-acquittal-by-judge (pro: disallowed),
trial-held-before-judge (pro: disallowed),
defendant-initiates-motion (pro: allowed),
final-verdict-reached (pro: allowed),
hung-jury (pro: allowed),
motion-unrelated-to-guilt (pro: allowed),
reprosecution-was-contemplated-by-judge (pro: allowed),
pre-trial-type-motion-not-timely (pro: allowed), and
trial-judge-implied-guilt (pro: allowed),

These dimensions and other domain information was entered into DAN through the New Domain procedure. The seven Supreme Court cases were entered through DAN's New Cases procedure. (See Appendix E for the Prolog representation of criminal appeal domain and Appendix F for the representation of the criminal appeal cases).

The Analyze Domain procedure was run on this new domain, and the results were impressive. The generalization rules realized that the motion-unrelated-to-guilt and the defendant-initiates-motion dimensions were necessary dimensions required to allow the government to appeal. This is the exact requirement put forth by the Supreme Court in U.S. v. Scott, 437 U.S. 82 (1978). The further-proceedings-required dimension was considered necessary for the defendant to keep the government from appealing, which also accurately represents the rules developed by the Court. In addition, the power of the labeled-acquittal-by-judge dimension was discovered as the Analyze Domain procedure labeled it an overwhelming dimension. Even the dominated dimension pair rules developed

an accurate statement of the law when it found that the further-proceedings-required dimension is dominated by the motion-unrelated-to-guilt requirement. This Domain Analysis can be found in Figure 13.

The implementation was then tested by Arguing the John Doe hypothetical proposed by Mendelson in his paper with the implemented cases. Not only did the Argue procedure correctly pick the Scott decision as being most relevant, but it also argued through the generalization rules the exact holding of the Scott decision. (See Figure 14). Another hypothetical was later created and argued in order to exemplify the use of the necessary rule in argumentation. (See Figure 15).

Thus, the problem with Mendelson's example domain was not its factual nature, but rather the dimensions which Mendelson chose to represent it. Every problem described in his paper has been solved using these new dimensions. In response to the possibility that his dimensions might be inappropriate, Mendelson asked the question "Are Ashley's dimensions so difficult to select properly that only a top expert in a field can do it?" The answer is clearly not, as I analyzed and implemented the domain on DAN in one long afternoon. Rather, the dimensions must be chosen with an understanding of their purpose and use. The dimensions should represent the factual (and procedural) information in a case that a judge (or commentator) has stated will affect the outcome of the case. Technically, then, one could never successfully argue Mendelson's main point, that the dimensions used in a domain did not capture important factual details in a case. If they did not, then the dimensions were not chosen properly. Not every important fact will be recognized by the creator of the dimensions as important, and this is an inherent problem in representing cases through dimensions. But Mendelson purposefully chose not to represent facts known to be important, and that shortcoming is not attributable to dimensional analysis.

SECTION 12: CONCLUSION

In this paper, I have shown some of the strengths of representing legal cases through dimensions. I successfully implemented the evident partiality domain, which indicates that dimensions could be used to represent cases in many areas of the law. The successful encoding of the criminal appeal domain first analyzed by Simon Mendelson supports this conclusion. In addition, I introduced the Dimensional Analyzer (DAn) program which is capable of analyzing a legal domain through the use of dimensions. Not only can DAn develop 3-Ply arguments in a style very similar to that used in Ashley's HYPO program, but it is also capable of answering queries about a system and can develop and reason using generalization rules.

The representation of the evident partiality cases was a straight-forward example of dimensional assignment. Although no implementation was made of the rationales or standards that were used in the cases, every major factual detail which a court labeled significant was included in the dimensions. In addition, various relationships between the cases were encoded, including the citation of a case for support along a dimension, the distinguishing of a case along a dimension, and the overruling of a case. Unlike HYPO, DAn makes use of these relationships by using them to strengthen the arguments produced in the Argue procedure.

The program was written in Prolog, which had the advantage of creating methods of access to every fact stored in the program. DAn produces 3-Ply arguments similar to that produced by HYPO, except that DAn's arguments are capable of providing a simple weighting scheme to put more emphasis on some dimensions in an argument. These weights do not effect the analysis of the 3-Ply argument, rather they allow the user to put more faith in particular dimensions and pairs of dimensions that appear in the output. The generalization rules created by DAn provide an additional method of giving weight to arguments based on certain dimensions. In addition, the generalization rules effectively provide a way to learn general facts about a domain through the analysis of the dimensional

representation of cases.

Finally, DAn implemented a query procedure which allows the user to directly access information about the cases and dimensions found in the case base. Although the Query System uses Prolog to make the actual queries, the user does not need to understand Prolog as queries are created through the selection of menu choices.

I implemented the evident partiality on the Dimensional Analyzer program in an effort to study the domain as well as to study the usefulness of analyzing cases with dimensions. It is clear from this analysis that a very useful study of a domain can take place through dimensional analysis. The domain can be effectively represented by the dimensions that are created in DAn. The DAn program not only argues the new domain, but it can also generalize and query the domain. In this way, this study of dimensions and the creation of DAn has increased the usefulness of dimensional representation.

[2]

==> POINT for DEFENDANT as Side-1:

WHERE:

Employee defendant was sole developer of plaintiff's product.

EVEN THOUGH:

Plaintiff and defendant entered into a nondisclosure agreement.

DEFENDANT should win a claim for TRADE-SECRETS-MISAPPROPRIATION.

CITE: Amoco Production Co. V. Lindley
609 P.2d 733.

<== RESPONSE for PLAINTIFF as Side-2:

Amoco Production Co. V. Lindley is distinguishable because:

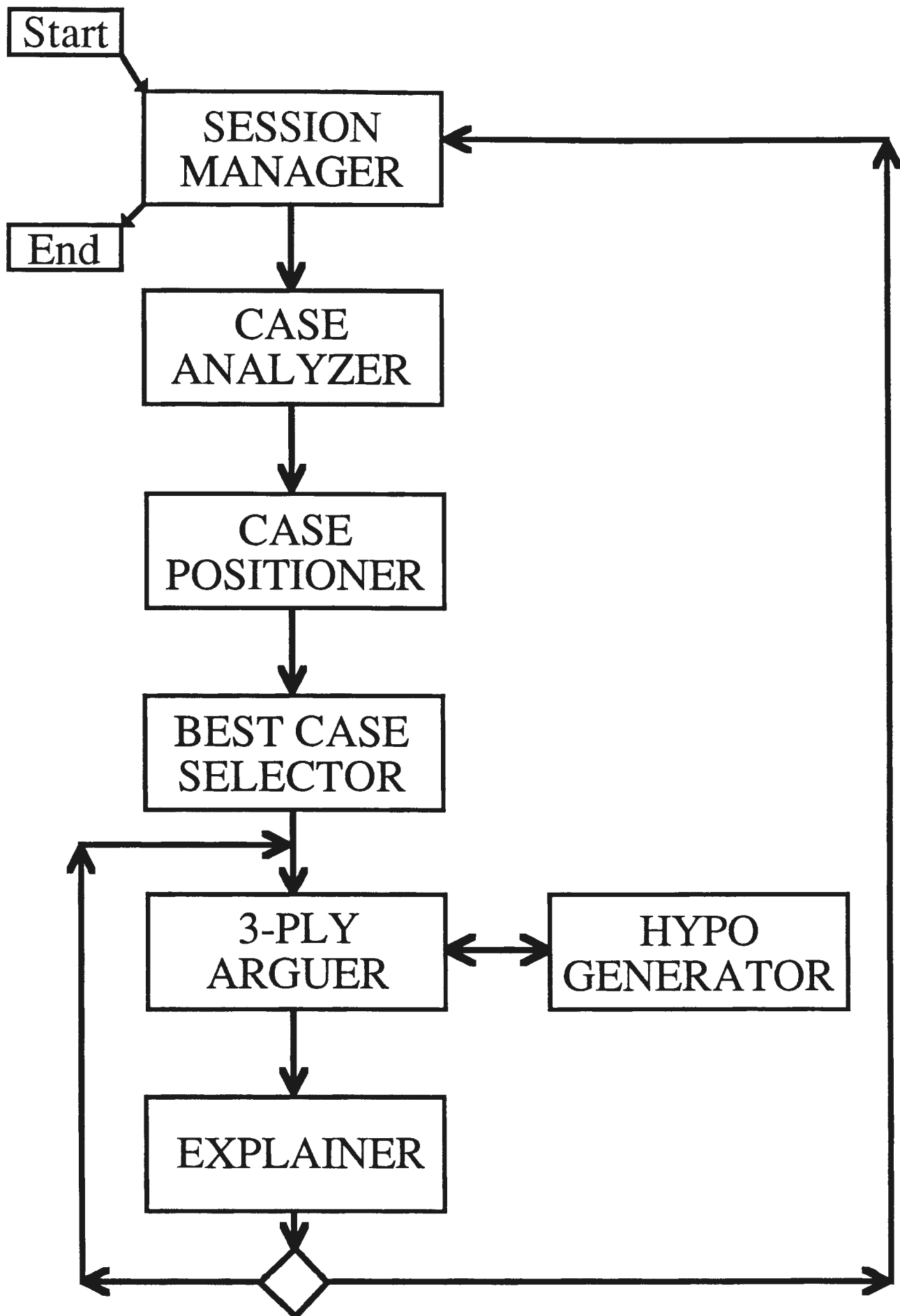
In Structural Dynamics, defendant received something of value for entering into the agreement. Not so in Amoco.

In Structural Dynamics, plaintiff's former employee brought product development information to Defendant. Not so in Amoco.

In Structural Dynamics, the nondisclosure agreement specifically referred to plaintiff's product. Not so in Amoco.

==> REBUTTAL for DEFENDANT as Side-1:

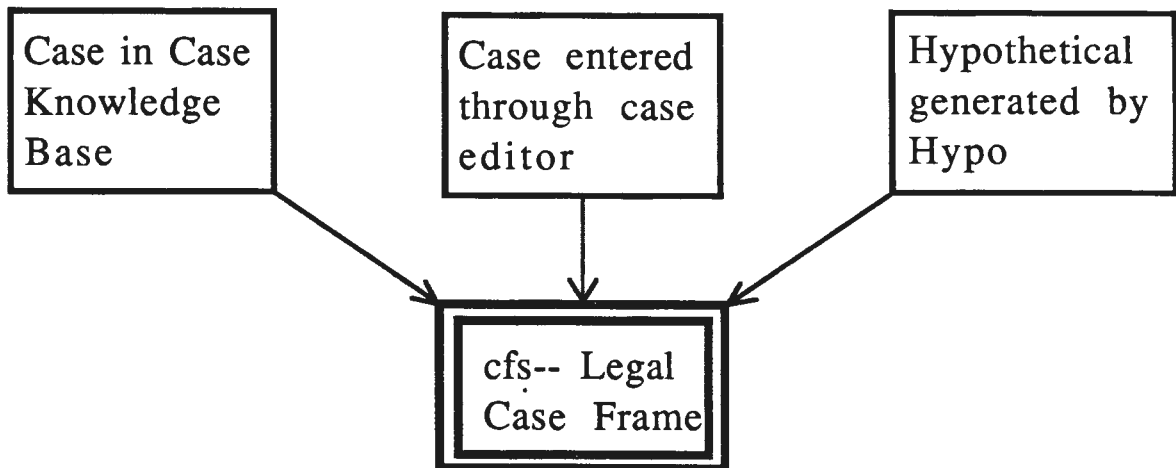
None.



Session Manager/Case Analysis

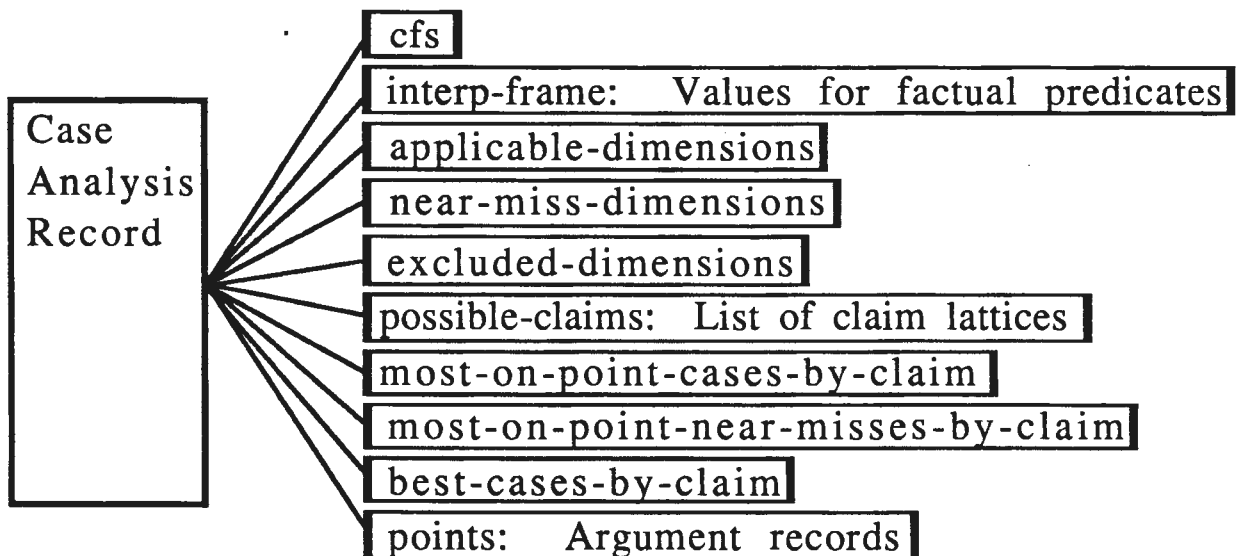
Session

Primary purpose is the selection of the current fact situation (cfs) and the creation of a Case Analysis Record.



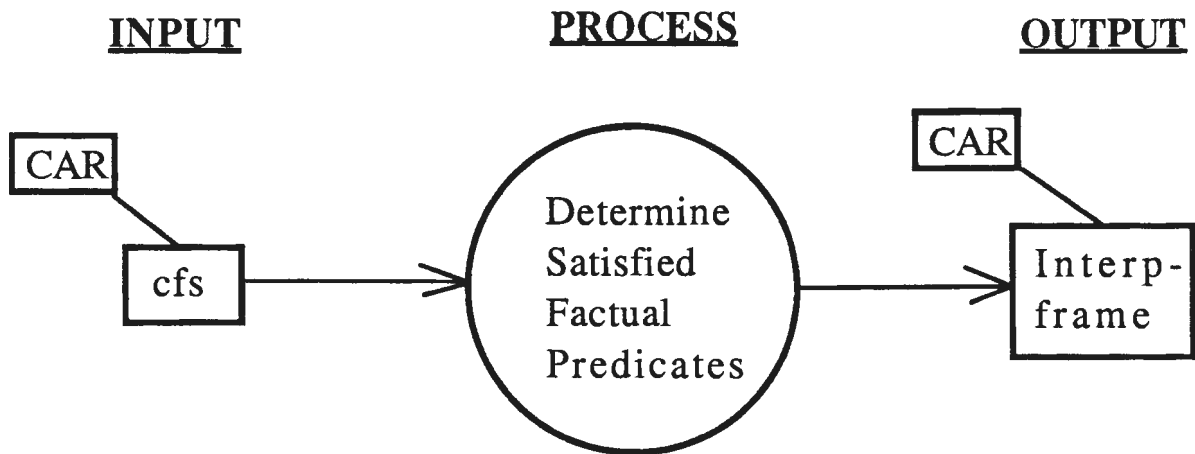
Case Analysis

Primary storage record for Hypo

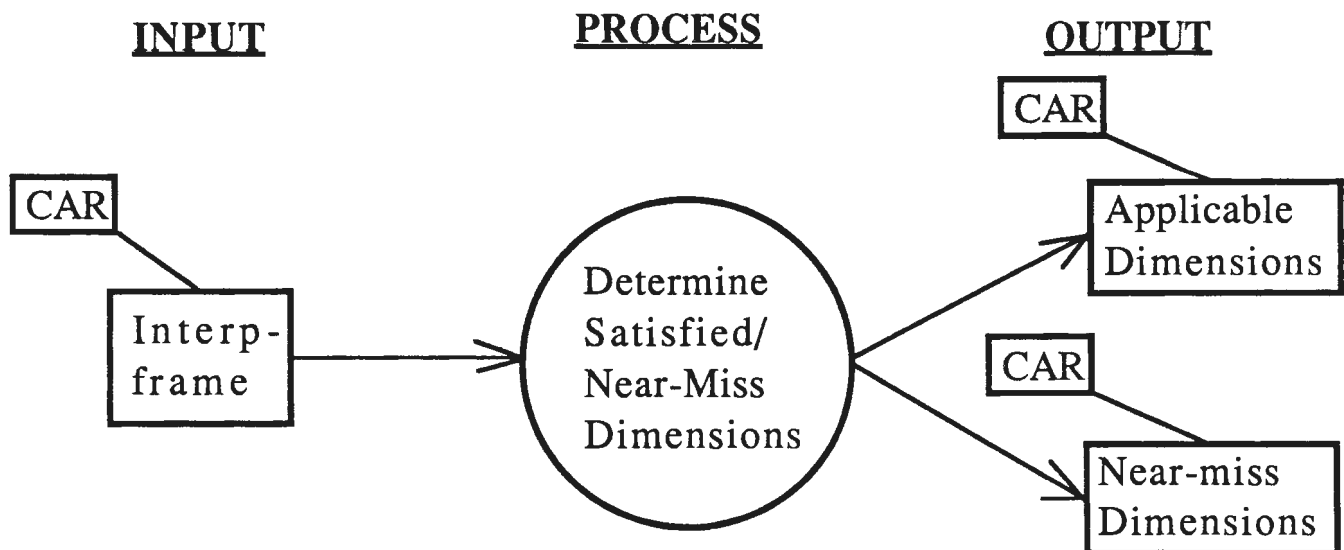


Case Analyzer

Dimensionally analyze



Determine satisfied

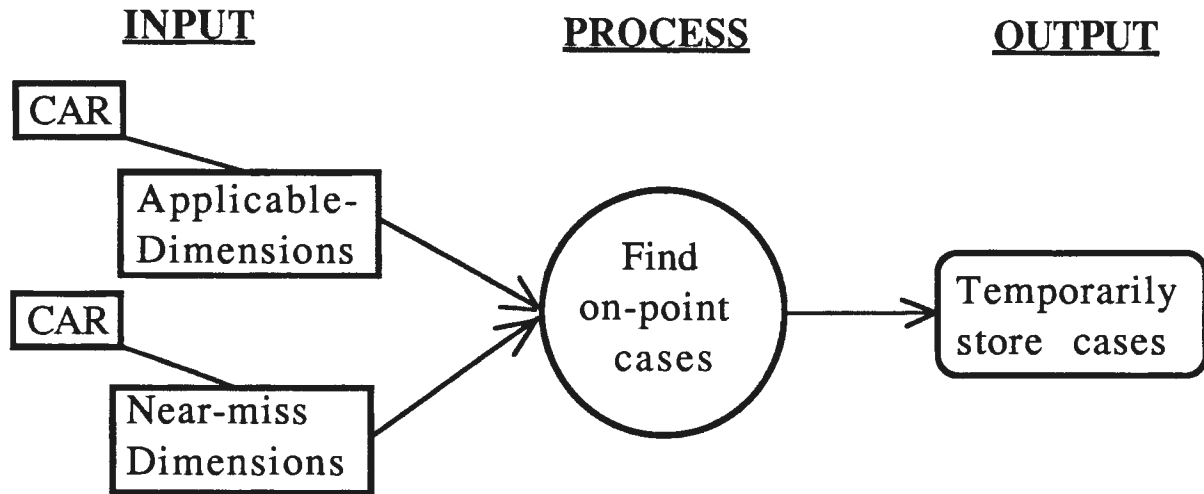


If all prerequisites are satisfied, then dimension is applicable

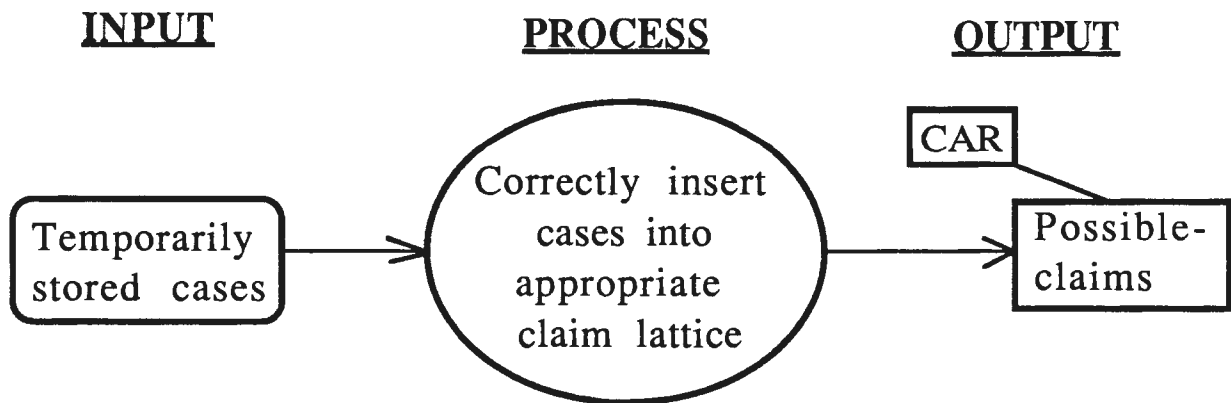
If all but triggering/focal pt. prereq's are met, then near-miss

Case

Retrieve on-point cases



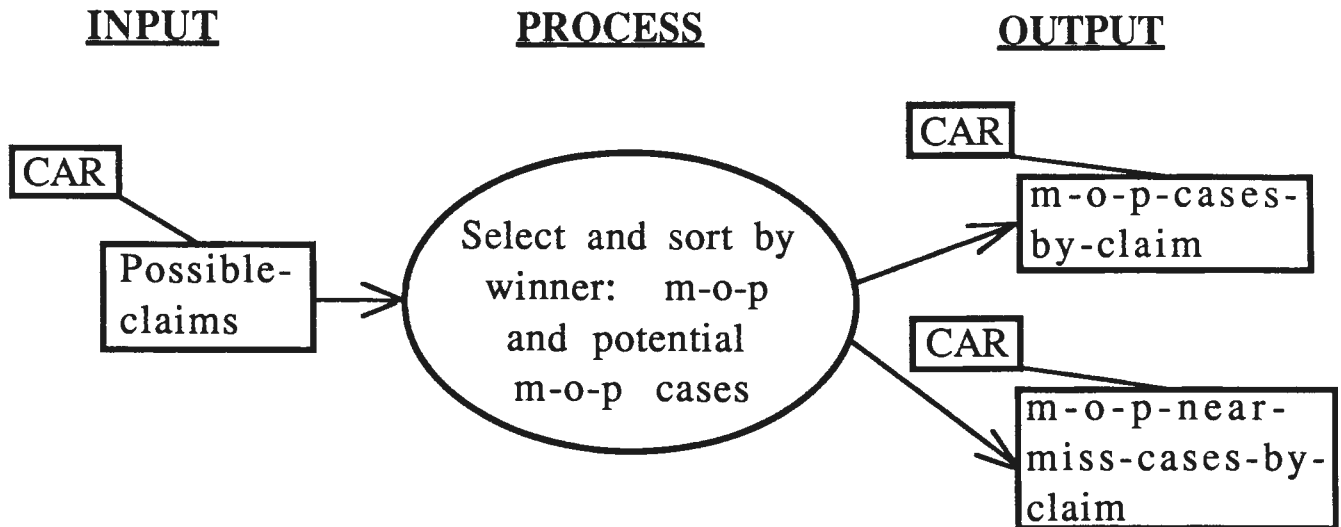
Organize Cases in Claim



A regular claim lattice and an extended claim lattice (for near-misses) are created. Cases are inserted into the lattices through a insertion algorithm which uses the "coverage" concept.

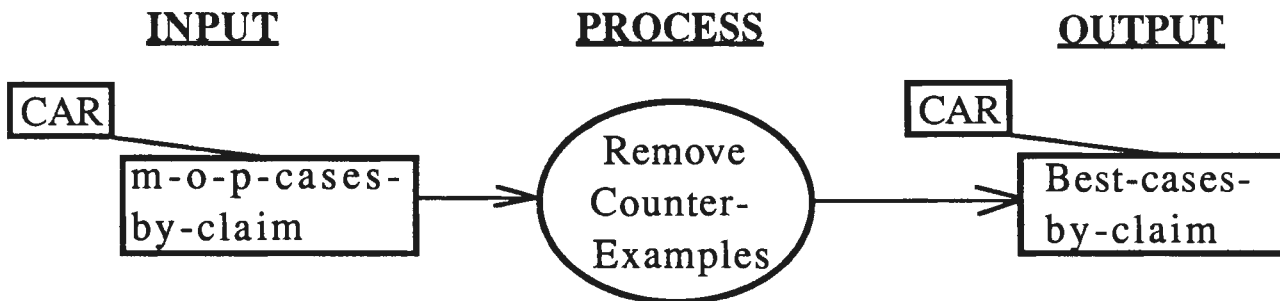
Best Case

Choose most-on-point (m-o-p)



M-O-P Algorithm: (1) Add children of root to "new roots" list. (2) For each node on list: (a) if cases exist, add to m-o-p cases for appropriate side; (b) else add node's children to list.

Separate out counter-example cases



If a case that holds for a side does not share with the cfs even one Dimension that favors that side, then that case is most appropriate for counter-examples.

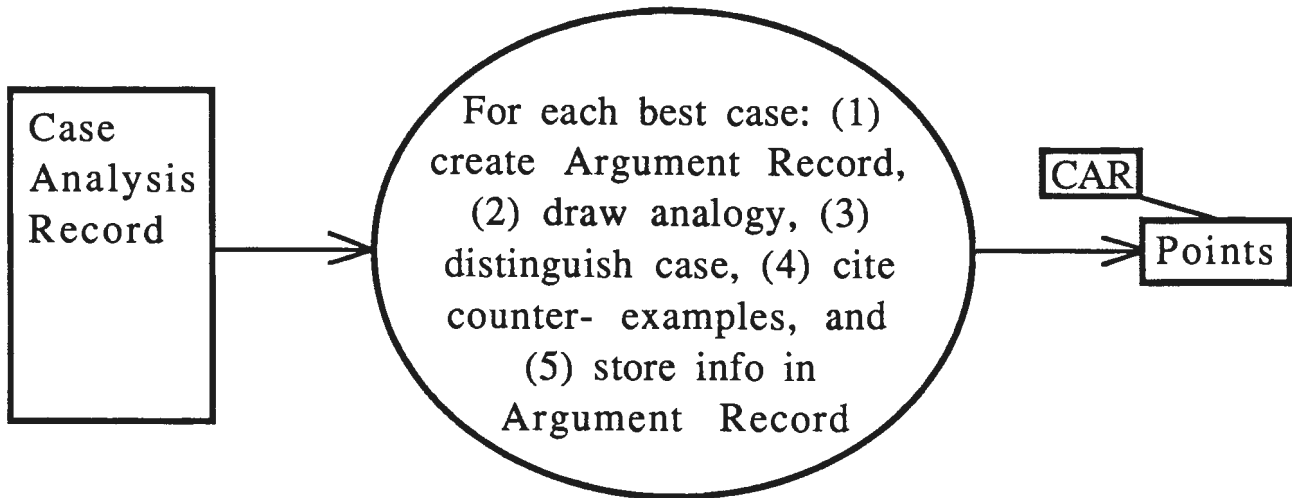
3-Ply Arguer

Create Argument

INPUT

PROCESS

OUTPUT

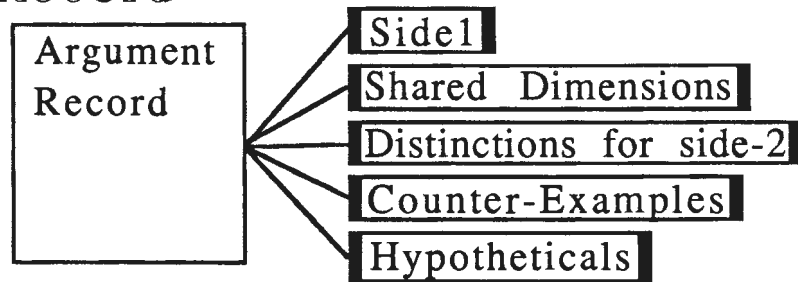


Analogy: Showing the Dimensions shared by cfs and cited case

Distinguishing: Factors which make cfs stronger for side-2 than cited case. Compare the Dimensions, and point out unshared Dimensions and different positions along shared Dimensions

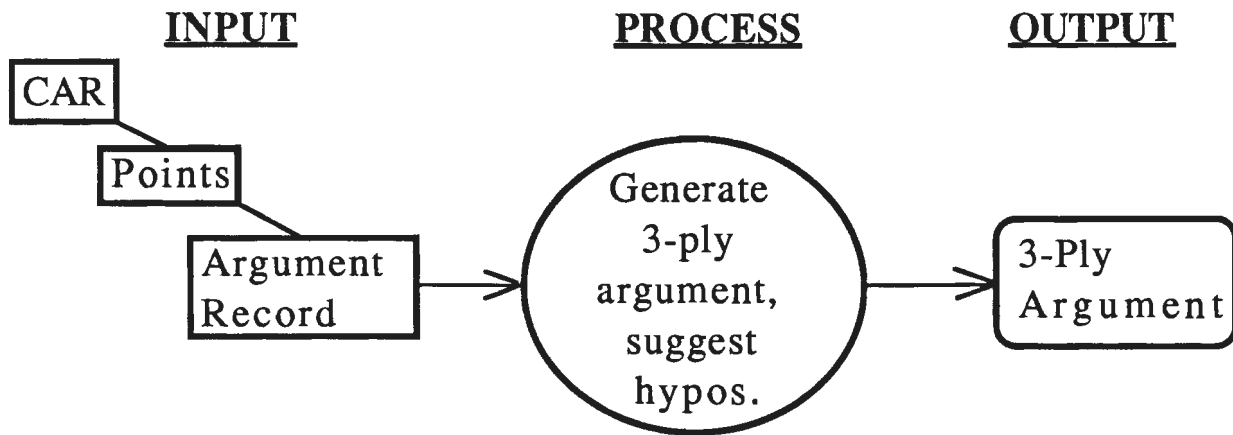
Counter-examples: Cases sharing some or all of the Dimensions shared by cited case and cfs, but which nevertheless held for side-2. Four types: More-on-point, Potentially-more-on-point, Overlapping, Boundary.

Argument Record



3-Ply Arguer

When Requested, Generate 3-Ply



3-Ply argument consists of:

- (1) Making a point for side-1 by citing case and drawing analogy.**
- (2) Making a response for side-2 by reciting distinctions, and generating another argument record for more-on-point and boundary counter-examples.**
- (3) Making rebuttal by responding to cases cited by side-2.**

Hypotheticals are suggested for each potentially more-on-point counter-example

This analysis was done when the required level of support in order for a generalization rule to be satisfied was 2 cases.

PRESUMPTION:

According to the cases, there is a presumption within the Evident Partiality domain in favor of upholding the arbitration award.

This presumption means that this position is capable of winning a dispute on the merits without even a single dimension which supports that side.

SEE:

Andros, Chevron, Country Mutual Insurance, Del Bianco, Morelite, Ormsbee, Perl, Reed Martin, San

There are no such cases on the vacating the arbitration award position.

OVERWHELMING DIMENSIONS:

The following dimensions have special importance in the field of Evident Partiality in that they appear only in cases which were decided in favor of the side that the dimension favors. Such dimensions are called overwhelming dimensions, and may be of crucial importance in arguing a case or understanding how a case may be decided in this area of the law.

The fact that a dimension is overwhelming is especially important if it appears in a great number of cases. The following overwhelming dimensions are therefore followed by the cases in which they are found.

narrow_business_area

Found in :Cook, Garfield, Isaacson

potential_conflicts_waived_in_contract

Found in :Garfield, Isaacson

relationship_unknown_to_arbitrator

Found in :House Grain, R. E. Bean

unanimous_arbitration_panel

Found in :Coughlan, Weinrott

OVERWHELMING DIMENSION PAIRS:

Sometimes two dimensions in a domain are closely linked together in that when they appear together, they have more persuasive power than if they appeared alone. In this analysis, dimension pairs that overwhelmed other dimensions (in that two dimensions that favor the same side always "win-out" whenever they are together) are considered examples of such closely linked dimensions. In the Evident Partiality domain, the following dimensions are overwhelming pairs:

substantial_business_relationship_with_party

Paired with : attorney_client_relationship_with_lawyer

SEE: Richco, Sanko

DOMINATED DIMENSION PAIRS:

Certain dimension may be overwhelming in all cases except when the dimension occurs with a particular, opposite-sided dimensions.

This situation may indicate a peculiar connection between the two dimensions, where one dimension may be very persuasive except in the presence of another, dominating dimension. If one dimension is found to be dominating over numerous other dimensions, then it may be that there is not a particularly strong relationship between the dimensions, but rather that that dimension is (or is nearly) overwhelming.

In the Evident Partiality domain, the following dimensions are dominated pairs:

attorney_client_relationship_with_party is dominated

by knowledge_by_party

SEE: Baar & Beards, R. E. Bean

substantial_business_relationship_with_party is dominated

by narrow_business_area

SEE: Cook, Garfield, Isaacson

+++++
---> ARGUMENT FOR VACATING THE ARBITRATION AWARD:
+++++

Cases suggest that the presence of the following factors should lead the court in Richco to hold in favor of vacating the arbitration award:

the arbitrator had a substantial business
relationship with a party.

Even Though:
the objecting party knew of the
potential disqualification.

SEE:
Barcon Assoc., Inc. v. Tri-County Asphalt Co.

<--- RESPONSE FOR UPHOLDING THE ARBITRATION AWARD:

Barcon is distinguishable from Richco because:
In Barcon, a party was indebted to the arbitrator.
This was not true in Richco.

COUNTER-EXAMPLES:

Cook Industries, Inc. v. C. Itoh & Co.,
449 F.2d 106 (2d Cir.) (1971), is as on point and held for the
pro uphold position.

+++++
---> REBUTTAL FOR VACATING THE ARBITRATION AWARD:
+++++

Cook is distinguishable from Richco because:
In Richco, the arbitrator had an attorney client
relationship with a party's lawyer.
This was not true in Cook.

In Richco, the arbitrator had a relationship with an entity or a
person who was closely connected to a party or their lawyer.
This was not true in Cook.

In Cook, the arbitration involved a narrow business area.
This was not true in Richco.
Courts have acknowledged the importance of this distinction.
SEE: Cook (distinguishing Commonwealth Coatings)
Sanko (distinguishing Cook)

In both cases, the objecting party knew of the
potential disqualification.
However, in Cook, the knowledge was complete
while in Richco, the knowledge was partial.

+++++
---> ARGUMENT FOR VACATING THE ARBITRATION AWARD:
+++++

Cases suggest that the presence of the following factors should lead the court in Richco to hold in favor of vacating the arbitration award:

the arbitrator had a relationship with an entity or a
person who was closely connected to a party or their lawyer.

Even Though:

the objecting party knew of the
potential disqualification.

SEE:

Freeport Construction Co. v. Star Forge, Inc.

<--- RESPONSE FOR UPHOLDING THE ARBITRATION AWARD:

Freeport is distinguishable from Richco because:

In Freeport, the arbitrator knew of a party's objection.
This was not true in Richco.

In Freeport, the objecting party objected to the
arbitrator before the award.

This was not true in Richco.

Courts have acknowledged the importance of this distinction.

SEE: Morelite (distinguishing Cook)

In both cases, the arbitrator had a relationship with an entity or a
person who was closely connected to a party or their lawyer.

However, in Freeport, the relationship was on-going during the arbitration
while in Richco, the relationship had ended before the arbitration.

COUNTER-EXAMPLES:

Cook Industries, Inc. v. C. Itoh & Co.,
449 F.2d 106 (2d Cir.) (1971) is not as on point, but it held for
the pro uphold position, where it was the case that:
the objecting party knew of the
potential disqualification.

San Luis Obispo Bay Properties v. Pacific Gas and Elect. Co.,
28 Cal. App. 3d 556, 104 Cal. Rptr. 733 (1972) is not as on point, but it held for
the pro uphold position, where it was the case that:
the arbitrator had a relationship with an entity or a
person who was closely connected to a party or their lawyer.

+++++
---> REBUTTAL FOR VACATING THE ARBITRATION AWARD:
+++++

Cook is distinguishable from Richco because:

In Richco, the arbitrator had an attorney client

relationship with a party's lawyer.
This was not true in Cook.

In Richco, the arbitrator had a relationship with an entity or a person who was closely connected to a party or their lawyer.
This was not true in Cook.

In Cook, the arbitration involved a narrow business area.
This was not true in Richco.
Courts have acknowledged the importance of this distinction.
SEE: Cook (distinguishing Commonwealth Coatings)
Sanko (distinguishing Cook)

In both cases, the objecting party knew of the potential disqualification.
However, in Cook, the knowledge was complete while in Richco, the knowledge was partial.

San Luis Obispo is distinguishable from Richco because:

In Richco, the arbitrator had an attorney client relationship with a party's lawyer.
This was not true in San Luis Obispo.

In Richco, the arbitrator had a substantial business relationship with a party.
This was not true in San Luis Obispo.
Courts have acknowledged the importance of this distinction.
SEE: Reed Martin (distinguishing Commonwealth Coatings)

+++++
---> ARGUMENT FOR VACATING THE ARBITRATION AWARD:
+++++

Cases suggest that the presence of the following factors should lead the court in Richco to hold in favor of vacating the arbitration award:

the arbitrator had an attorney client
relationship with a party's lawyer,
the arbitrator had a substantial business
relationship with a party.

SEE:

Sanko S.S. Co., Ltd., v. Cook Industries, Inc.

In addition, cases in the casebase show that the following
rules support this conclusion:

RULE: In all cases where
the arbitrator had a substantial business
relationship with a party as well as
the arbitrator had an attorney client
relationship with a party's lawyer, the court
has ruled in favor of vacating the arbitration award.

<--- RESPONSE FOR UPHOLDING THE ARBITRATION AWARD:

Sanko is distinguishable from Richco because:

In Richco, the objecting party knew of the
potential disqualification.

This was not true in Sanko.

Courts have acknowledged the importance of this distinction.

SEE: Cook (distinguishing Commonwealth Coatings)

Sanko (distinguishing Garfield)

Sanko (distinguishing Cook)

In Sanko, the arbitrator had a minor business
relationship with a party.

This was not true in Richco.

COUNTER-EXAMPLES:

Cook Industries, Inc. v. C. Itoh & Co.,
449 F.2d 106 (2d Cir.) (1971) is not as on point, but it held for
the pro uphold position, where it was the case that:
the arbitrator had a substantial business
relationship with a party.

Franke v. Farm Bureau Mut. Ins. Co.,
421 N.W.2d 406 (Minn.) (1988) is not as on point, but it held for
the pro uphold position, where it was the case that:
the arbitrator had an attorney client
relationship with a party's lawyer.

++++++
---> REBUTTAL FOR VACATING THE ARBITRATION AWARD:
++++++

Cook is distinguishable from Richco because:

In Richco, the arbitrator had an attorney client
relationship with a party's lawyer.

This was not true in Cook.

In Richco, the arbitrator had a relationship with an entity or a
person who was closely connected to a party or their lawyer.

This was not true in Cook.

In Cook, the arbitration involved a narrow business area.

This was not true in Richco.

Courts have acknowledged the importance of this distinction.

SEE: Cook (distinguishing Commonwealth Coatings)

Sanko (distinguishing Cook)

In both cases, the objecting party knew of the
potential disqualification.

However, in Cook, the knowledge was complete
while in Richco, the knowledge was partial.

Franke is distinguishable from Richco because:

In Richco, the arbitrator had a relationship with an entity or a
person who was closely connected to a party or their lawyer.

This was not true in Franke.

In Richco, the arbitrator had a substantial business
relationship with a party.

This was not true in Franke.

Courts have acknowledged the importance of this distinction.

SEE: Reed Martin (distinguishing Commonwealth Coatings)

In Franke, the arbitrator objected to was not a neutral arbitrator.

This was not true in Richco.

+++++
---> ARGUMENT FOR UPHOLDING THE ARBITRATION AWARD:
+++++

Cases suggest that the presence of the following factors should lead the court in Richco to hold in favor of upholding the arbitration award:

the objecting party knew of the
potential disqualification.

Even Though:

the arbitrator had a substantial business
relationship with a party.

SEE:

Cook Industries, Inc. v. C. Itoh & Co.

<--- RESPONSE FOR VACATING THE ARBITRATION AWARD:

Cook is distinguishable from Richco because:

In Richco, the arbitrator had an attorney client
relationship with a party's lawyer.

This was not true in Cook.

In Richco, the arbitrator had a relationship with an entity or a
person who was closely connected to a party or their lawyer.

This was not true in Cook.

In Cook, the arbitration involved a narrow business area.

This was not true in Richco.

Courts have acknowledged the importance of this distinction.

SEE: Cook (distinguishing Commonwealth Coatings)

Sanko (distinguishing Cook)

In both cases, the objecting party knew of the
potential disqualification.

However, in Cook, the knowledge was complete
while in Richco, the knowledge was partial.

COUNTER-EXAMPLES:

Barcon Assoc., Inc. v. Tri-County Asphalt Co.,
86 N.J. 179, 430 A.2d 214 (1981), is as on point and held for the
pro vacate position.

+++++
---> REBUTTAL FOR UPHOLDING THE ARBITRATION AWARD:
+++++

Barcon is distinguishable from Richco because:

In Barcon, a party was indebted to the arbitrator.

This was not true in Richco.

QUERY:

query(Case) :- get_dimension(Case, knowledge_by_party, _), holding(Case, vacate).

RESULT:

Barcon Assoc., Inc. v. Tri-County Asphalt Co.

86 N.J. 179, 430 A.2d 214 (1981)

Short Name : Barcon

Internal Name : barcon

Holding : vacate

Dimensions : substantial_business_relationship_with_party (pro: vacate)
indebtedness_with_party (pro: vacate)
arbitrator_was_not_appointed_as_neutral (pro: uphold)
knowledge_by_party (pro: uphold)

Freeport Construction Co. v. Star Forge, Inc.

61 Ill. App. 3d 999, 378 N.E.2d 558 (1978)

Short Name : Freeport

Internal Name : freeport

Holding : vacate

Dimensions : objection_to_arbitrator_made_prior_to_award (pro: vacate)
arbitrator_had_knowledge_of_objection (pro: vacate)
knowledge_by_party (pro: uphold)
relationship_with_party_or_lawyer_through_third_person (pro: vacate)

Citing McKinney Drilling, San Luis Obispo for support along:

relationship_with_party_or_lawyer_through_third_person.

Morelite Construction Corp. v. N.Y. City Dist. Council

748 F.2d 79 (2d Cir.) (1984)

Short Name : Morelite

Internal Name : morelite

Holding : vacate

Dimensions : objection_to_arbitrator_made_prior_to_award (pro: vacate)
social_or_family_relationship_with_party (pro: vacate)
knowledge_by_party (pro: uphold)

Distinguishing Cook along:

objection_to_arbitrator_made_prior_to_award.

Richco Structures v. Parkside Village, Inc.

82 Wis.2d 547, 263 N.W.2d 204 (1978)

Short Name : Richco

Internal Name : richco

Holding : vacate

Dimensions : relationship_with_party_or_lawyer_through_third_person (pro: vacate)
substantial_business_relationship_with_party (pro: vacate)
attorney_client_relationship_with_lawyer (pro: vacate)
knowledge_by_party (pro: uphold)

Citing Cook for support along:

knowledge_by_party.

Citing Johnston for support along:

minor_business_relationship_with_lawyer.

Citing San Luis Obispo for support along:

relationship_with_party_or_lawyer_through_third_person.

+++++
---> ARGUMENT FOR UPHOLDING THE ARBITRATION AWARD:
+++++

Cases suggest that the presence of the following factors should lead the court in Cook to hold in favor of upholding the arbitration award:

the objecting party knew of the
potential disqualification,
the arbitration involved a narrow business area.

Even Though:

the arbitrator had a substantial business
relationship with a party.

SEE:

Isaacson v. Hayden, Stone Inc.

In addition, cases in the casebase show that the following
rules support this conclusion:

RULE: All known cases where
the arbitration involved a narrow business area
have held in favor of upholding the arbitration award.

RULE: According to the cases, the fact that
the arbitrator had a substantial business
relationship with a party
is not important where it is also true that
the arbitration involved a narrow business area.

<--- RESPONSE FOR VACATING THE ARBITRATION AWARD:

Isaacson is distinguishable from Cook because:

In Isaacson, the objecting party waived potential conflicts of the
arbitrator in the contract requiring arbitration.

This was not true in Cook.

Courts have acknowledged the importance of this distinction.

SEE: Garfield (distinguishing Commonwealth Coatings)
Sanko (distinguishing Garfield)

COUNTER-EXAMPLES:

Richco Structures v. Parkside Village, Inc.,
82 Wis.2d 547, 263 N.W.2d 204 (1978) is not as on point, but it held for
the pro vacate position, where it was the case that:

the objecting party knew of the
potential disqualification,
the arbitrator had a substantial business
relationship with a party.

Barcon Assoc., Inc. v. Tri-County Asphalt Co.,
86 N.J. 179, 430 A.2d 214 (1981) is not as on point, but it held for
the pro vacate position, where it was the case that:

the objecting party knew of the
potential disqualification,

the arbitrator had a substantial business
relationship with a party.

+++++
---> REBUTTAL FOR UPHOLDING THE ARBITRATION AWARD:
+++++

Richco is distinguishable from Cook because:

In Cook, the arbitration involved a narrow business area.

This was not true in Richco.

Courts have acknowledged the importance of this distinction.

SEE: Cook (distinguishing Commonwealth Coatings)

Sanko (distinguishing Cook)

In Richco, the arbitrator had an attorney client

relationship with a party's lawyer.

This was not true in Cook.

In Richco, the arbitrator had a relationship with an entity or a

person who was closely connected to a party or their lawyer.

This was not true in Cook.

In both cases, the objecting party knew of the

potential disqualification.

However, in Cook, the knowledge was complete

while in Richco, the knowledge was partial.

Barcon is distinguishable from Cook because:

In Cook, the arbitration involved a narrow business area.

This was not true in Barcon.

Courts have acknowledged the importance of this distinction.

SEE: Cook (distinguishing Commonwealth Coatings)

Sanko (distinguishing Cook)

In Barcon, a party was indebted to the arbitrator.

This was not true in Cook.

+++++
----> ARGUMENT FOR VACATING THE ARBITRATION AWARD:
+++++

Cases suggest that the presence of the following factors should lead the court in Cook to hold in favor of vacating the arbitration award:

the arbitrator had a substantial business
relationship with a party.

Even Though:
the objecting party knew of the
potential disqualification.

SEE:
Richco Structures v. Parkside Village, Inc.
Barcon Assoc., Inc. v. Tri-County Asphalt Co.

<---- RESPONSE FOR UPHOLDING THE ARBITRATION AWARD:

Richco is distinguishable from Cook because:
In Cook, the arbitration involved a narrow business area.
This was not true in Richco.
Courts have acknowledged the importance of this distinction.
SEE: Cook (distinguishing Commonwealth Coatings)
Sanko (distinguishing Cook)

In Richco, the arbitrator had an attorney client
relationship with a party's lawyer.
This was not true in Cook.

In Richco, the arbitrator had a relationship with an entity or a
person who was closely connected to a party or their lawyer.
This was not true in Cook.

In both cases, the objecting party knew of the
potential disqualification.
However, in Cook, the knowledge was complete
while in Richco, the knowledge was partial.

Barcon is distinguishable from Cook because:
In Cook, the arbitration involved a narrow business area.
This was not true in Barcon.
Courts have acknowledged the importance of this distinction.
SEE: Cook (distinguishing Commonwealth Coatings)
Sanko (distinguishing Cook)

In Barcon, a party was indebted to the arbitrator.
This was not true in Cook.

COUNTER-EXAMPLES:

Isaacson v. Hayden, Stone Inc.,
359 F.Supp. 1050 (DC NY) (unknown) is more on point and held for the

pro uphold position, where it was also true that:
the arbitration involved a narrow business area.

+++++
---> REBUTTAL FOR VACATING THE ARBITRATION AWARD:
+++++

Isaacson is distinguishable from Cook because:

In Isaacson, the objecting party waived potential conflicts of the
arbitrator in the contract requiring arbitration.

This was not true in Cook.

Courts have acknowledged the importance of this distinction.

SEE: Garfield (distinguishing Commonwealth Coatings)

Sanko (distinguishing Garfield)

+++++
---> ARGUMENT FOR UPHOLDING THE ARBITRATION AWARD:
+++++

According to the cases, there is a presumption in the
Evident Partiality domain in favor of upholding the arbitration award.
Thus, even though no dimensions exist in Commonwealth Coatings in favor of
upholding the arbitration award, the presumption that this
position normally prevails must be overcome.

SEE GENERALLY:

Andros, Chevron, Country Mutual Insurance, Del Bianco, Morelite, Ormsbee,
Perl, Reed Martin, San Luis Obispo, Saville, Turner

<--- RESPONSE FOR VACATING THE ARBITRATION AWARD:

None.

This analysis was done when the required level of support in order for a generalization rule to be satisfied was 2 cases.

NECESSARY DIMENSIONS:

The following dimensions are special in the Criminal Appeal domain in that every cases which holds for a particular side contains that dimension. In other words, these dimensions appear to be necessary, but not necessarily sufficient conditions for the court to hold for the side that the dimension favors.

motion_unrelated_to_guilt

Is a necessary dimension for allowing the government to appeal.

Found in :Lee, Scott, Wilson

defendant_initiates_motion

Is a necessary dimension for allowing the government to appeal.

Found in :Lee, Scott, Wilson

further_proceedings_required

Is a necessary dimension for not allowing an appeal.

Found in :Finch, Jenkins, Martin Linen, Sanabria

OVERWHELMING DIMENSIONS:

The following dimensions have special importance in the field of Criminal Appeal in that they appear only in cases which were decided in favor of the side that the dimension favors. Such dimensions are called overwhelming dimensions, and may be of crucial importance in arguing a case or understanding how a case may be decided in this area of the law.

The fact that a dimension is overwhelming is especially important if it appears in a great number of cases. The following overwhelming dimensions are therefore followed by the cases in which they are found.

motion_unrelated_to_guilt

Found in :Lee, Scott, Wilson

labelled_acquittal_by_judge

Found in :Martin Linen, Sanabria

DOMINATED DIMENSION PAIRS:

Certain dimension may be overwhelming in all cases except when the dimension occurs with a particular, opposite-sided dimensions.

This situation may indicate a peculiar connection between the two dimensions, where one dimension may be very persuasive except in the presence of another, dominating dimension. If one dimension is found to be dominating over numerous other dimensions, then it may be that there is not a particularly strong relationship between the dimensions, but rather that that dimension is (or is nearly) overwhelming.

In the Criminal Appeal domain, the following dimensions are dominated pairs:

further_proceedings_required is dominated

by motion_unrelated_to_guilt

SEE: Lee, Scott

defendant_initiates_motion is dominated

by labelled_acquittal_by_judge
SEE: Martin Linen, Sanabria

+++++
---> ARGUMENT FOR ALLOWING THE GOVERNMENT TO APPEAL:
+++++

Cases suggest that the presence of the following factors should lead the court in John Doe Hypo to hold in favor of allowing the government to appeal:

the defendant initiated the motion,
the motion to dismiss was unrelated to guilt.

Even Though:
further proceedings will be required.

SEE:
U.S. v. Scott

In addition, cases in the casebase show that the following rules support this conclusion:

RULE: All known cases where
the motion to dismiss was unrelated to guilt
have held in favor of allowing the government to appeal.

RULE: According to the cases, the fact that
further proceedings will be required
is not important where it is also true that
the motion to dismiss was unrelated to guilt.

<--- RESPONSE FOR NOT ALLOWING AN APPEAL:

Scott is distinguishable from John Doe Hypo because:
In both cases, further proceedings will be required.
However, in John Doe Hypo, further proceedings are definite
while in Scott, further proceedings are unlikely.

COUNTER-EXAMPLES:

Sanabria v. U.S.,
437 U.S. 54 (1978) is not as on point, but it held for
the pro disallowed position, where it was the case that:
the defendant initiated the motion,
further proceedings will be required.
U.S. v. Martin Linen,
430 U.S. 564 (1977) is not as on point, but it held for
the pro disallowed position, where it was the case that:
the defendant initiated the motion,
further proceedings will be required.

+++++
---> REBUTTAL FOR ALLOWING THE GOVERNMENT TO APPEAL:
+++++

Sanabria is distinguishable from John Doe Hypo because:
In John Doe Hypo, the motion to dismiss was unrelated to guilt.

This was not true in Sanabria as was directly stated in the case.

In Sanabria, the trial judge labeled her action an acquittal.
This was not true in John Doe Hypo.

In both cases, further proceedings will be required.
However, in Sanabria, a new trial will be required
while in John Doe Hypo, further proceedings are definite.

Martin Linen is distinguishable from John Doe Hypo because:

In John Doe Hypo, the motion to dismiss was unrelated to guilt.
This was not true in Martin Linen as was directly stated in the case.

In Martin Linen, the trial judge labeled her action an acquittal.
This was not true in John Doe Hypo.

+++++
---> ARGUMENT FOR NOT ALLOWING AN APPEAL:
+++++

Cases suggest that the presence of the following factors should lead the court in John Doe Hypo to hold in favor of not allowing an appeal:

further proceedings will be required.

Even Though:
the defendant initiated the motion.

SEE:
Sanabria v. U.S.
U.S. v. Martin Linen

<--- RESPONSE FOR ALLOWING THE GOVERNMENT TO APPEAL:

Sanabria is distinguishable from John Doe Hypo because:
In John Doe Hypo, the motion to dismiss was unrelated to guilt.
This was not true in Sanabria as was directly stated in the case.

In Sanabria, the trial judge labeled her action an acquittal.
This was not true in John Doe Hypo.

In both cases, further proceedings will be required.
However, in Sanabria, a new trial will be required
while in John Doe Hypo, further proceedings are definite.

Martin Linen is distinguishable from John Doe Hypo because:
In John Doe Hypo, the motion to dismiss was unrelated to guilt.
This was not true in Martin Linen as was directly stated in the case.

In Martin Linen, the trial judge labeled her action an acquittal.
This was not true in John Doe Hypo.

COUNTER-EXAMPLES:

U.S. v. Scott,
437 U.S. 82 (1978) is more on point and held for the
pro allowed position, where it was also true that:
the motion to dismiss was unrelated to guilt.

+++++
---> REBUTTAL FOR NOT ALLOWING AN APPEAL:
+++++

Scott is distinguishable from John Doe Hypo because:
In both cases, further proceedings will be required.
However, in John Doe Hypo, further proceedings are definite
while in Scott, further proceedings are unlikely.

+++++
---> ARGUMENT FOR ALLOWING THE GOVERNMENT TO APPEAL:
+++++

Cases suggest that the presence of the following factors should lead the court in Jane Roe Hypo to hold in favor of allowing the government to appeal:

the motion to dismiss was unrelated to guilt,
the judge contemplated the defendant would be reprosecuted.

SEE:
Lee v. U.S.

In addition, cases in the casebase show that the following rules support this conclusion:

RULE: All known cases where
the motion to dismiss was unrelated to guilt
have held in favor of allowing the government to appeal.

<--- RESPONSE FOR NOT ALLOWING AN APPEAL:

Lee is distinguishable from Jane Roe Hypo because:
In Jane Roe Hypo, the trial judge labeled her action an acquittal.
This was not true in Lee.

In Lee, the defendant initiated the motion.
This was not true in Jane Roe Hypo as was directly stated in the case.
According to the cases in the case base, this distinction is
crucial as no case has found in favor of allowing an appeal without
the presence of this dimension.
SEE: Lee, Scott, Wilson

In Lee, the defendant was not timely in a pretrial type of motion.
This was not true in Jane Roe Hypo.

No counter-examples.

+++++
---> REBUTTAL FOR ALLOWING THE GOVERNMENT TO APPEAL:
+++++

None

+++++
---> ARGUMENT FOR NOT ALLOWING AN APPEAL:
+++++

Cases suggest that the presence of the following factors should lead the court in Jane Roe Hypo to hold in favor of not allowing an appeal:

the trial judge labeled her action an acquittal.

SEE:

Sanabria v. U.S.

U.S. v. Martin Linen

In addition, cases in the casebase show that the following rules support this conclusion:

RULE: All known cases where
the trial judge labeled her action an acquittal
have held in favor of not allowing an appeal.

<--- RESPONSE FOR ALLOWING THE GOVERNMENT TO APPEAL:

Sanabria is distinguishable from Jane Roe Hypo because:

In Jane Roe Hypo, the motion to dismiss was unrelated to guilt.
This was not true in Sanabria as was directly stated in the case.

In Jane Roe Hypo, the judge contemplated the defendant would be reprosecuted.
This was not true in Sanabria.
Courts have acknowledged the importance of this distinction.
SEE: Sanabria (distinguishing Lee)

In Sanabria, further proceedings will be required.
This was not true in Jane Roe Hypo.

Martin Linen is distinguishable from Jane Roe Hypo because:

In Jane Roe Hypo, the motion to dismiss was unrelated to guilt.
This was not true in Martin Linen as was directly stated in the case.

In Jane Roe Hypo, the judge contemplated the defendant would be reprosecuted.
This was not true in Martin Linen.
Courts have acknowledged the importance of this distinction.
SEE: Sanabria (distinguishing Lee)

In Martin Linen, further proceedings will be required.
This was not true in Jane Roe Hypo.

No counter-examples.

+++++
---> REBUTTAL FOR NOT ALLOWING AN APPEAL:
+++++

None

QUERY:

query(Dimension) :- pro(Dimension, _).

RESULT:

arbitrator_had_knowledge_of_objection

Description : the arbitrator knew of a party's objection

Overwhelming : No

Both-Sided : Yes

Range :
the dimension did not occur
the dimension did occur

Found in : baar_and_beards freeport garfield

arbitrator_was_not_appointed_as_neutral

Description : the arbitrator objected to was not a neutral arbitrator

Overwhelming : No

Both-Sided : Yes

Range :
the dimension did not occur
the dimension did occur

Found in : baar_and_beards barcon foley franke

attorney_client_relationship_with_lawyer

Description : the arbitrator had an attorney client

relationship with a party's lawyer

Overwhelming : No

Both-Sided : Yes

Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration

Part of Overwhelming Pair with :

substantial_business_relationship_with_party

Found in : franke richco sanko

attorney_client_relationship_with_party

Description : the arbitrator had an attorney client

relationship with a party

Overwhelming : No

Both-Sided : Yes

Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration

Dominated by :

knowledge_by_party

Found in : abramovich baar_and_beards colony_liquor egan northwest_mechanical r_e_bean

comments_by_arbitrator_during_arbitration

Description : the arbitrator made comments indicating
her or his partiality

Overwhelming : No

Both-Sided : Yes

Range :
the dimension did not occur
the dimension did occur

Found in : del_bianco holodnak saville turner

employee_employer_relationship_with_lawyer

Description : the arbitrator had an employment relationship

with a party's lawyer
Overwhelming : No
Both-Sided : No
Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration
Found in : No Cases.

employee_employer_relationship_with_party
Description : the arbitrator had an employment
relationship with a party
Overwhelming : No
Both-Sided : Yes
Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration
Found in : albion merit_ins

excessive_or_inadequate_award
Description : the arbitration award was excessive or inadequate
Overwhelming : No
Both-Sided : No
Range :
the dimension did not occur
the dimension did occur
Found in : country_mutual_insurance

financial_interest_in_outcome
Description : the arbitrator had a financial interest
in the arbitration's outcome
Overwhelming : No
Both-Sided : No
Range :
the dimension did not occur
the dimension did occur
Found in : No Cases.

improper_communications_with_party_or_lawyer
Description : the arbitrator had improper communication
with a party or lawyer
Overwhelming : No
Both-Sided : Yes
Range :
the dimension did not occur
the dimension did occur
Found in : abramovich jean_a_mccoy

indebtedness_with_party
Description : a party was indebted to the arbitrator
Overwhelming : No
Both-Sided : No
Range :
the dimension did not occur
the dimension did occur
Found in : barcon

inferred_knowledge_by_party
Description : the objecting party should have known

of the potential disqualification
Overwhelming : No
Both-Sided : No
Range :
the dimension did not occur
the dimension did occur
Found in : st_paul_ins

knowledge_by_party

Description : the objecting party knew of the
potential disqualification
Overwhelming : No
Both-Sided : Yes
Range :
the dimension did not occur
the knowledge was partial
the knowledge was complete
Dominating :
attorney_client_relationship_with_party
Found in : baar_and_beards barcon cook freeport garfield isaacson jean_a_mccoy milliken morelit

minor_business_relationship_with_lawyer

Description : the arbitrator had a minor business
relationship with a party's lawyer
Overwhelming : No
Both-Sided : No
Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration
Found in : johnston

minor_business_relationship_with_party

Description : the arbitrator had a minor business
relationship with a party
Overwhelming : No
Both-Sided : Yes
Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration
Found in : foley house_grain milliken sanko

narrow_business_area

Description : the arbitration involved a narrow business area
Overwhelming : Yes
Both-Sided : No
Range :
the dimension did not occur
the dimension did occur
Dominating :
substantial_business_relationship_with_party
Found in : cook garfield isaacson

objection_to_arbitrator_made_prior_to_award

Description : the objecting party objected to the
arbitrator before the award
Overwhelming : No
Both-Sided : Yes
Range :

the dimension did not occur
the dimension did occur
Found in : baar_and_beards freeport garfield isaacson milliken morelite

partiality_affected_outcome

Description : the arbitrator's partiality affected
the outcome of the arbitration

Overwhelming : No

Both-Sided : No

Range :
the dimension did not occur
the dimension did occur

Found in : mckinney_drilling

potential_conflicts_waived_in_contract

Description : the objecting party waived potential conflicts of the
arbitrator in the contract requiring arbitration

Overwhelming : Yes

Both-Sided : No

Range :
the dimension did not occur
the dimension did occur

Found in : garfield isaacson

relationship_unknown_to_arbitrator

Description : the disqualification was unknown to the arbitrator

Overwhelming : Yes

Both-Sided : No

Range :
the dimension did not occur
the dimension did occur

Found in : house_grain r_e_bean

relationship_with_party_or_lawyer_through_third_person

Description : the arbitrator had a relationship with an entity or a
person who was closely connected to a party or their lawyer

Overwhelming : No

Both-Sided : Yes

Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration

Found in : freeport mckinney_drilling richco san_luis_obispo weinrott

relationship_with_persons_similarly_situated

Description : the arbitrator had a significant relationship with
a similarly situated person

Overwhelming : No

Both-Sided : Yes

Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration

Found in : hartman ormsbee perl reed_martin turner

social_or_family_relationship_with_lawyer

Description : the arbitrator had a social or family
relationship with a party's lawyer

Overwhelming : No

Both-Sided : Yes

Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration
Found in : coughlan johnston r_e_bean st_paul_ins

social_or_family_relationship_with_party

Description : the arbitrator had a social or family
relationship with a party

Overwhelming : No

Both-Sided : Yes

Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration

Found in : andros chevron morelite r_e_bean

substantial_business_relationship_with_lawyer

Description : the arbitrator had a substantial business relationship
with a party's lawyer

Overwhelming : No

Both-Sided : No

Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration

Found in : No Cases.

substantial_business_relationship_with_party

Description : the arbitrator had a substantial business
relationship with a party

Overwhelming : No

Both-Sided : Yes

Range :
the dimension did not occur
the relationship had ended before the arbitration
the relationship was on-going during the arbitration

Dominated by :

narrow_business_area

Part of Overwhelming Pair with :

attorney_client_relationship_with_lawyer

Found in : barcon commonwealth_coatings cook garfield isaacson j_p_stevens james_d_morrisey

unanimous_arbitration_panel

Description : the arbitrator panel was unanimous in its decision

Overwhelming : Yes

Both-Sided : No

Range :
the dimension did not occur
the dimension did occur

Found in : coughlan weinrott

QUERY:

query(Case) :- holding(Case, _).

RESULT:

Abramovich v. Penn. Liquor Control Board

416 A.2d 474 (Penn.) (1980)

Short Name : Abramovich

Internal Name : abramovich

Holding : vacate

Dimensions : attorney_client_relationship_with_party (pro: vacate)

improper_communications_with_party_or_lawyer (pro: vacate)

Albion Pub. Schools v. Albion Ed. Assoc.

130 Mich. App. 698, 344 N.W.2d 55 (1983)

Short Name : Albion

Internal Name : albion

Holding : vacate

Dimensions : employee_employer_relationship_with_party (pro: vacate)

Andros Compania Maritima v. Marc Rich & Co.

579 F.2d 691 (2d. Cir.) (1978)

Short Name : Andros

Internal Name : andros

Holding : uphold

Dimensions : social_or_family_relationship_with_party (pro: vacate)

Baar & Beards, Inc. v. Oleg Cassini, Inc.

331 N.Y.S.2d 670 (1972)

Short Name : Baar & Beards

Internal Name : baar_and_beards

Holding : uphold

Dimensions : attorney_client_relationship_with_party (pro: vacate)

knowledge_by_party (pro: uphold)

objection_to_arbitrator_made_prior_to_award (pro: vacate)

arbitrator_had_knowledge_of_objection (pro: vacate)

arbitrator_was_not_appointed_as_neutral (pro: uphold)

Barcon Assoc., Inc. v. Tri-County Asphalt Co.

86 N.J. 179, 430 A.2d 214 (1981)

Short Name : Barcon

Internal Name : barcon

Holding : vacate

Dimensions : substantial_business_relationship_with_party (pro: vacate)

indebtedness_with_party (pro: vacate)

arbitrator_was_not_appointed_as_neutral (pro: uphold)

knowledge_by_party (pro: uphold)

Chevron Transport Corp. v. Astro Vencedor Compania Naviera

300 F.Supp. 179 (1969)

Short Name : Chevron

Internal Name : chevron
Holding : uphold
Dimensions : social_or_family_relationship_with_party (pro: vacate)

Colony Liquor Distributers, Inc. v. Local 669, I.B.T.
34 App. Div. 2d 1060, 312 N.Y.S.2d 403 (1970)
Short Name : Colony Liquor
Internal Name : colony_liquor
Holding : vacate
Dimensions : attorney_client_relationship_with_party (pro: vacate)

Commonwealth Coatings Corp. v. Contenental Casualty Co.
393 U.S.145, 21 L.Ed.2d. 301, 89 S.Ct. 337 (1968)
Short Name : Commonwealth Coatings
Internal Name : commonwealth_coatings
Holding : vacate
Dimensions : substantial_business_relationship_with_party (pro: vacate)

Distinguished by Cook along:
knowledge_by_party.
Distinguished by Cook along:
narrow_business_area.
Distinguished by Garfield along:
potential_conflicts_waived_in_contract.
Distinguished by Reed Martin along:
substantial_business_relationship_with_party.

Cook Industries, Inc. v. C. Itoh & Co.
449 F.2d 106 (2d Cir.) (1971)
Short Name : Cook
Internal Name : cook
Holding : uphold
Dimensions : substantial_business_relationship_with_party (pro: vacate)
narrow_business_area (pro: uphold)
knowledge_by_party (pro: uphold)

Distinguishing Commonwealth Coatings along:
knowledge_by_party.
Distinguishing Commonwealth Coatings along:
narrow_business_area.
Distinguished by Sanko along:
knowledge_by_party.
Distinguished by Sanko along:
narrow_business_area.
Distinguished by Morelite along:
objection_to_arbitrator_made_prior_to_award.
Citing Garfield for support along:
knowledge_by_party.
Citing Garfield for support along:
narrow_business_area.
Cited by Richco for support along:
knowledge_by_party.

Coughlan Construction Co., Inc. v. Town of Rockport

23 Mass. App. Ct. 994, 505 N.E.2d 203 (1987)
Short Name : Coughlan
Internal Name : coughlan
Holding : uphold
Dimensions : social_or_family_relationship_with_lawyer (pro: vacate)
unanimous_arbitration_panel (pro: uphold)

Citing Weinrott for support along:
unanimous_arbitration_panel.

Country Mutual Ins. Co. v. Kuzmickas
2 Ill. App. 2d 313, 276 N.E.2d 357 (1971)
Short Name : Country Mutual Insurance
Internal Name : country_mutual_insurance
Holding : uphold
Dimensions : excessive_or_inadequate_award (pro: vacate)

Del Bianco & Assoc., Inc. v. Adam
6 Ill. App. 3d 286, 285 N.E.2d 480 (1972)
Short Name : Del Bianco
Internal Name : del_bianco
Holding : uphold
Dimensions : comments_by_arbitrator_during_arbitration (pro: vacate)

Egan & Sons Co. v. Mears Park Development Co.
414 N.W.2d 785 (Minn.) (1987)
Short Name : Egan
Internal Name : egan
Holding : vacate
Dimensions : attorney_client_relationship_with_party (pro: vacate)

Foley Co. v. Grindsted Products, Inc.
233 Kan. 339, 662 P.2d 1254 (1983)
Short Name : Foley
Internal Name : foley
Holding : uphold
Dimensions : minor_business_relationship_with_party (pro: vacate)
arbitrator_was_not_appointed_as_neutral (pro: uphold)

Citing Morelite for support along:
arbitrator_was_not_appointed_as_neutral.
Citing Perl for support along:
narrow_business_area.

Franke v. Farm Bureau Mut. Ins. Co.
421 N.W.2d 406 (Minn.) (1988)
Short Name : Franke
Internal Name : franke
Holding : uphold
Dimensions : attorney_client_relationship_with_lawyer (pro: vacate)
arbitrator_was_not_appointed_as_neutral (pro: uphold)

Freeport Construction Co. v. Star Forge, Inc.

61 Ill. App. 3d 999, 378 N.E.2d 558 (1978)

Short Name : Freeport

Internal Name : freeport

Holding : vacate

Dimensions : objection_to_arbitrator_made_prior_to_award (pro: vacate)

arbitrator_had_knowledge_of_objection (pro: vacate)

knowledge_by_party (pro: uphold)

relationship_with_party_or_lawyer_through_third_person (pro: vacate)

Citing McKinney Drilling, San Luis Obispo for support along:

relationship_with_party_or_lawyer_through_third_person.

Garfield & Co. v. Wiest

432 F.2d 849 (1970)

Short Name : Garfield

Internal Name : garfield

Holding : uphold

Dimensions : substantial_business_relationship_with_party (pro: vacate)

knowledge_by_party (pro: uphold)

potential_conflicts_waived_in_contract (pro: uphold)

objection_to_arbitrator_made_prior_to_award (pro: vacate)

arbitrator_had_knowledge_of_objection (pro: vacate)

narrow_business_area (pro: uphold)

Distinguishing Commonwealth Coatings along:

potential_conflicts_waived_in_contract.

Distinguished by Sanko along:

knowledge_by_party.

Distinguished by Sanko along:

potential_conflicts_waived_in_contract.

Cited by Cook for support along:

knowledge_by_party.

Cited by Cook for support along:

narrow_business_area.

Hartman v. Cooper

474 A.2d 959 (unknown)

Short Name : Hartman

Internal Name : hartman

Holding : vacate

Dimensions : relationship_with_persons_similarly_situated (pro: vacate)

Holodnak v. Avco Corp.

381 F.Supp. 191 (unknown)

Short Name : Holodnak

Internal Name : holodnak

Holding : vacate

Dimensions : comments_by_arbitrator_during_arbitration (pro: vacate)

House Grain Co. v. Obst

659 S.W.2d 903 (Texas) (1983)

Short Name : House Grain
Internal Name : house_grain
Holding : uphold
Dimensions : minor_business_relationship_with_party (pro: vacate)
relationship_unknown_to_arbitrator (pro: uphold)

Isaacson v. Hayden, Stone Inc.
359 F.Supp. 1050 (DC NY) (unknown)
Short Name : Isaacson
Internal Name : isaacson
Holding : uphold
Dimensions : substantial_business_relationship_with_party (pro: vacate)
narrow_business_area (pro: uphold)
knowledge_by_party (pro: uphold)
objection_to_arbitrator_made_prior_to_award (pro: vacate)
potential_conflicts_waived_in_contract (pro: uphold)

J. P. Stevens & Co., Inc. v. Rytex Corp.
34 N.Y.2d 123, 356 N.Y.S.2d 278, 312 N.E.2d 466 (1974)
Short Name : J. P. Stevens
Internal Name : j_p_stevens
Holding : vacate
Dimensions : substantial_business_relationship_with_party (pro: vacate)

James D. Morrissey, Inc. v. Gross Construction Co., Inc.
296 Pa.Super. 151, 443 A.2d 344 (1982)
Short Name : James D. Morrissey
Internal Name : james_d_morrissey
Holding : vacate
Dimensions : substantial_business_relationship_with_party (pro: vacate)

Jean A. McCoy & Sons, Inc. v. LaSalle Cnty.
48 Ill. App. 3d. 802, 363 N.E.2d 442 (1977)
Short Name : Jean A. McCoy
Internal Name : jean_a_mccoy
Holding : uphold
Dimensions : improper_communications_with_party_or_lawyer (pro: vacate)
knowledge_by_party (pro: uphold)

Johnston v. Security Ins. Co. of Hartford
6 Cal. App. 3d. 839, 86 Cal. Rptr. 133 (1970)
Short Name : Johnston
Internal Name : johnston
Holding : vacate
Dimensions : social_or_family_relationship_with_lawyer (pro: vacate)
minor_business_relationship_with_lawyer (pro: vacate)

Cited by Richco for support along:
minor_business_relationship_with_lawyer.

McKinney Drilling v. Mach I Ltd.

32 Md. App. 205, 359 A.2d 100 (unknown)

Short Name : McKinney Drilling

Internal Name : mckinney_drilling

Holding : vacate

Dimensions : relationship_with_party_or_lawyer_through_third_person (pro: vacate)
partiality_affected_outcome (pro: vacate)

Cited by Freeport for support along:

relationship_with_party_or_lawyer_through_third_person.

Morelite Construction Corp. v. N.Y. City Dist. Council

714 F.2d 673 (2d Cir.) (1983)

Short Name : Morelite

Internal Name : merit_ins

Holding : uphold

Dimensions : employee_employer_relationship_with_party (pro: vacate)

Milliken & Co. v. Tiffany Loungewear, Inc.

99 App. Div. 2d 993, 473 N.Y.S.2d 443 (1984)

Short Name : Milliken

Internal Name : milliken

Holding : uphold

Dimensions : objection_to_arbitrator_made_prior_to_award (pro: vacate)
knowledge_by_party (pro: uphold)
minor_business_relationship_with_party (pro: vacate)

Morelite Construction Corp. v. N.Y. City Dist. Council

748 F.2d 79 (2d Cir.) (1984)

Short Name : Morelite

Internal Name : morelite

Holding : vacate

Dimensions : objection_to_arbitrator_made_prior_to_award (pro: vacate)
social_or_family_relationship_with_party (pro: vacate)
knowledge_by_party (pro: uphold)

Distinguishing Cook along:

objection_to_arbitrator_made_prior_to_award.

Northwest Mechanical, Inc. v. Public Utilities Comm.

283 N.W.2d 522 (Minn.) (1979)

Short Name : Northwest Mechanical

Internal Name : northwest_mechanical

Holding : vacate

Dimensions : attorney_client_relationship_with_party (pro: vacate)
substantial_business_relationship_with_party (pro: vacate)

Ormsbee Development Co. v. Grace

668 F.2d 1140 (10th Cir.) (1982)

Short Name : Ormsbee

Internal Name : ormsbee

Holding : uphold
Dimensions : relationship_with_persons_similarly_situated (pro: vacate)

Perl v. General Fire and Cas. Co.
34 App. Div. 2d. 748, 310 N.Y.S.2d 196 (1970)
Short Name : Perl
Internal Name : perl
Holding : uphold
Dimensions : relationship_with_persons_similarly_situated (pro: vacate)

Cited by Foley for support along:
narrow_business_area.

R. E. Bean Construction Co. v. Middlebury Assoc.
139 Vt. 200, 428 A.2d 306 (1980)
Short Name : R. E. Bean
Internal Name : r_e_bean
Holding : uphold
Dimensions : relationship_unknown_to_arbitrator (pro: uphold)
attorney_client_relationship_with_party (pro: vacate)
social_or_family_relationship_with_lawyer (pro: vacate)
social_or_family_relationship_with_party (pro: vacate)
knowledge_by_party (pro: uphold)

Reed & Martin, Inc. v. Westinghouse Electric Corp.
439 F.2d 1268 (1971)
Short Name : Reed Martin
Internal Name : reed_martin
Holding : uphold
Dimensions : relationship_with_persons_similarly_situated (pro: vacate)

Distinguishing Commonwealth Coatings along:
substantial_business_relationship_with_party.

Richco Structures v. Parkside Village, Inc.
82 Wis.2d 547, 263 N.W.2d 204 (1978)
Short Name : Richco
Internal Name : richco
Holding : vacate
Dimensions : relationship_with_party_or_lawyer_through_third_person (pro: vacate)
substantial_business_relationship_with_party (pro: vacate)
attorney_client_relationship_with_lawyer (pro: vacate)
knowledge_by_party (pro: uphold)

Citing Cook for support along:
knowledge_by_party.
Citing Johnston for support along:
minor_business_relationship_with_lawyer.
Citing San Luis Obispo for support along:
relationship_with_party_or_lawyer_through_third_person.

San Luis Obispo Bay Properties v. Pacific Gas and Elect. Co.
28 Cal. App. 3d 556, 104 Cal. Rptr. 733 (1972)

Short Name : San Luis Obispo
Internal Name : san_luis_obispo
Holding : uphold
Dimensions : relationship_with_party_or_lawyer_through_third_person (pro: vacate)

Cited by Freeport, Richco for support along:
relationship_with_party_or_lawyer_through_third_person.

Sanko S.S. Co., Ltd., v. Cook Industries, Inc.

495. F.2d 1260 (2d Cir.) (1973)

Short Name : Sanko
Internal Name : sanko
Holding : vacate
Dimensions : attorney_client_relationship_with_lawyer (pro: vacate)
substantial_business_relationship_with_party (pro: vacate)
minor_business_relationship_with_party (pro: vacate)

Distinguishing Cook, Garfield along:
knowledge_by_party.
Distinguishing Cook along:
narrow_business_area.
Distinguishing Garfield along:
potential_conflicts_waived_in_contract.

Saville International, Inc. v. Galanti Group, Inc.

107 Ill. App. 3d 799, 438 N.W.2d 509 (1982)

Short Name : Saville
Internal Name : saville
Holding : uphold
Dimensions : comments_by_arbitrator_during_arbitration (pro: vacate)

St. Paul Ins. Cos. v. Lulis

6 Wash. App. 205, 492 P.2d 575 (1971)

Short Name : St. Paul Insurance
Internal Name : st_paul_ins
Holding : uphold
Dimensions : social_or_family_relationship_with_lawyer (pro: vacate)
inferred_knowledge_by_party (pro: uphold)

Turner Fisheries, Inc. v. Seafood Workers Union

19 Mass. App. 925, 471 N.E.2d 770 (1984)

Short Name : Turner
Internal Name : turner
Holding : uphold
Dimensions : relationship_with_persons_similarly_situated (pro: vacate)
comments_by_arbitrator_during_arbitration (pro: vacate)

Weinrott v. Carp

32 N.Y.2d 190, 344 N.Y.S.2d 848, 298 N.E.2d 42 (1973)

Short Name : Weinrott
Internal Name : weinrott
Holding : uphold

Dimensions : relationship_with_party_or_lawyer_through_third_person (pro: vacate)
unanimous_arbitration_panel (pro: uphold)

Cited by Coughlan for support along:
unanimous_arbitration_panel.

/****** DOMAIN INFORMATION *****/

**
** This file contains the general information needed to define a domain.
** This information includes:
** domain(Domain_Name).
** write_side(Side1_Description).
** write_side(Side2_Description).
** write_side_caps(Side1_Description).
** write_side_caps(Side2_Description).
** get_opposite_side(Side1, Side2).
** get_opposite_side(Side2, Side1).
**
** Where Domain_Name is the name of the domain to be used in output,
** Side1 and Side2 are the two sides to the domain, and the SideX_
** Descriptions are short phrases defining the sides also to be used
** in output.
**
*****/

domain('Evident Partiality').

write_side(vacate) :-
 write('vacating the arbitration award').
write_side(uphold) :-
 write('upholding the arbitration award').
write_side_caps(vacate) :-
 write('VACATING THE ARBITRATION AWARD').
write_side_caps(uphold) :-
 write('UPHOLDING THE ARBITRATION AWARD').

get_opposite_side(vacate, uphold).
get_opposite_side(uphold, vacate).

/****** DIMENSIONS *****/

**
** The dimensions are currently implemented by stating the pro,
** describe and range facts for a dimension:
**
** pro(Dimension, Side).
** describe(Dimension, Dimension_Description).
** range(Dimension, Range_List)
**
** where Dimension is the name of the dimension, Side is the side of the
** domain the dimension favors, Dimension_Description is the description
** phrase which is used to describe the dimension in output, and Range_List
** is the list of phrases to represent the possible magnitudes of the
** dimension.
**
** In the Evident Partiality domain, since many of the dimension ranges
** were identical, as short cut was used in which each dimension was
** assigned a size_type, and an additional range function was inserted at
** the end of the dimension list to simulate the range facts.
**
*****/

size_type(substantial_business_relationship_with_party, 2).

describe(substantial_business_relationship_with_party,
'the arbitrator had a substantial business
relationship with a party').
pro(substantial_business_relationship_with_party, vacate).

size_type(substantial_business_relationship_with_lawyer, 2).
describe(substantial_business_relationship_with_lawyer,
'the arbitrator had a substantial business relationship
with a party's lawyer').
pro(substantial_business_relationship_with_lawyer, vacate).

size_type(minor_business_relationship_with_party, 2).
describe(minor_business_relationship_with_party,
'the arbitrator had a minor business
relationship with a party').
pro(minor_business_relationship_with_party, vacate).

size_type(minor_business_relationship_with_lawyer, 2).
describe(minor_business_relationship_with_lawyer,
'the arbitrator had a minor business
relationship with a party's lawyer').
pro(minor_business_relationship_with_lawyer, vacate).

size_type(employee_employer_relationship_with_party, 2).
describe(employee_employer_relationship_with_party,
'the arbitrator had an employment
relationship with a party').
pro(employee_employer_relationship_with_party, vacate).

size_type(employee_employer_relationship_with_lawyer, 2).
describe(employee_employer_relationship_with_lawyer,
'the arbitrator had an employment relationship
with a party's lawyer').
pro(employee_employer_relationship_with_lawyer, vacate).

size_type(social_or_family_relationship_with_party, 2).
describe(social_or_family_relationship_with_party,
'the arbitrator had a social or family
relationship with a party').
pro(social_or_family_relationship_with_party, vacate).

size_type(social_or_family_relationship_with_lawyer, 2).
describe(social_or_family_relationship_with_lawyer,
'the arbitrator had a social or family
relationship with a party's lawyer').
pro(social_or_family_relationship_with_lawyer, vacate).

size_type(attorney_client_relationship_with_party, 2).
describe(attorney_client_relationship_with_party,
'the arbitrator had an attorney client
relationship with a party').
pro(attorney_client_relationship_with_party, vacate).

size_type(attorney_client_relationship_with_lawyer, 2).
describe(attorney_client_relationship_with_lawyer,
'the arbitrator had an attorney client
relationship with a party's lawyer').
pro(attorney_client_relationship_with_lawyer, vacate).

size_type(partiality_affected_outcome, 1).

describe(partiality_affected_outcome,
'the arbitrator's partiality affected
the outcome of the arbitration').
pro(partiality_affected_outcome, vacate).

size_type(improper_communications_with_party_or_lawyer, 1).
describe(improper_communications_with_party_or_lawyer,
'the arbitrator had improper communication
with a party or lawyer').
pro(improper_communications_with_party_or_lawyer, vacate).

size_type(comments_by_arbitrator_during_arbitration, 1).
describe(comments_by_arbitrator_during_arbitration,
'the arbitrator made comments indicating
her or his partiality').
pro(comments_by_arbitrator_during_arbitration, vacate).

size_type(financial_interest_in_outcome, 1).
describe(financial_interest_in_outcome,
'the arbitrator had a financial interest
in the arbitration's outcome').
pro(financial_interest_in_outcome, vacate).

size_type(relationship_with_persons_similarly_situated, 2).
describe(relationship_with_persons_similarly_situated,
'the arbitrator had a significant relationship with
a similarly situated person').
pro(relationship_with_persons_similarly_situated, vacate).

size_type(relationship_with_party_or_lawyer_through_third_person, 2).
describe(relationship_with_party_or_lawyer_through_third_person,
'the arbitrator had a relationship with an entity or a
person who was closely connected to a party or their lawyer').
pro(relationship_with_party_or_lawyer_through_third_person, vacate).

size_type(objection_to_arbitrator_made_prior_to_award, 1).
describe(objection_to_arbitrator_made_prior_to_award,
'the objecting party objected to the
arbitrator before the award').
pro(objection_to_arbitrator_made_prior_to_award, vacate).

size_type(arbitrator_had_knowledge_of_objection, 1).
describe(arbitrator_had_knowledge_of_objection,
'the arbitrator knew of a party's objection').
pro(arbitrator_had_knowledge_of_objection, vacate).

size_type(indebtedness_with_party, 1).
describe(indebtedness_with_party,
'a party was indebted to the arbitrator').
pro(indebtedness_with_party, vacate).

size_type(excessive_or_inadequate_award, 1).
describe(excessive_or_inadequate_award,
'the arbitration award was excessive or inadequate').
pro(excessive_or_inadequate_award, vacate).

size_type(unanimous_arbitration_panel, 1).
describe(unanimous_arbitration_panel,
'the arbitrator panel was unanimous in its decision').
pro(unanimous_arbitration_panel, uphold).

```
size_type(substantial_business_relationship_with_party, 2).
describe(substantial_business_relationship_with_party,
'the arbitrator had a substantial business
    relationship with a party').
pro(substantial_business_relationship_with_party, vacate).
```

```
size_type(arbitrator_was_not_appointed_as_neutral, 1).
describe(arbitrator_was_not_appointed_as_neutral,
'the arbitrator objected to was not a neutral arbitrator').
pro(arbitrator_was_not_appointed_as_neutral, uphold).
```

```
size_type(knowledge_by_party, 1).
describe(knowledge_by_party,
'the objecting party knew of the
    potential disqualification').
pro(knowledge_by_party, uphold).
range(knowledge_by_party,
    ['the dimension did not occur',
     'the knowledge was partial',
     'the knowledge was complete']).
```

```
size_type(inferred_knowledge_by_party, 1).
describe(inferred_knowledge_by_party,
'the objecting party should have known
    of the potential disqualification').
pro(inferred_knowledge_by_party, uphold).
```

```
size_type(relationship_unknown_to_arbitrator, 1).
describe(relationship_unknown_to_arbitrator,
'the disqualification was unknown to the arbitrator').
pro(relationship_unknown_to_arbitrator, uphold).
```

```
size_type(potential_conflicts_waived_in_contract, 1).
describe(potential_conflicts_waived_in_contract,
'the objecting party waived potential conflicts of the
    arbitrator in the contract requiring arbitration').
pro(potential_conflicts_waived_in_contract, uphold).
```

```
size_type(narrow_business_area, 1).
describe(narrow_business_area,
'the arbitration involved a narrow business area').
pro(narrow_business_area, uphold).
```

```
range(Dimension, Range_List) :-
    size_type(Dimension, Size_Type),
    do_range(Dimension, Size_Type, Range_List),!.
```

```
do_range(Dimension, 1,
    ['the dimension did not occur', 'the dimension did occur']).
do_range(Dimension, 2,
    ['the dimension did not occur',
     'the relationship had ended before the arbitration',
     'the relationship was on-going during the arbitration']).
```

***** CASES *****

**

** Cases are implemented by stating the following facts:

**

** citation(Case, Citation).

** shortname(Case, ShortName).

** longname(Case, LongName).

** year(Case, Year).

** holding(Case, Side).

** has_dimension(Case, Dimension1, Value1).

** has_dimension(Case, Dimension2, Value2).

** support(Case, Case cited for support, Dimension being supported).

** distinguishes(Case, Case distinguished, Dimension distinguished along).

** overrules(Case, Overruled_Case).

**

** where Case is a shortened case name used to represent the case internally,

** Short_Name is the shortened version of the named used in output, Long_

** Name is the official name of the case used in output, Citation is

** something approximating a partial Blue Book citation for the case, Side is

** either "uphold" or "vacate," Dimension is a name of a dimension, and Value

** is the magnitude of that particular dimension as an integer. Translating the

** magnitude integer into its actual magnitude translation is accomplished

** on output in the print_range procedure using a dimension's range_list.

**

** This case list was entered manually.

**

*****/

/*** case **/

citation(commonwealth_coatings, '393 U.S.145, 21 L.Ed.2d. 301, 89 S.Ct. 337').

shortname(commonwealth_coatings, 'Commonwealth Coatings').

longname(commonwealth_coatings,

'Commonwealth Coatings Corp. v. Contenental Casualty Co.').

year(commonwealth_coatings, 1968).

holding(commonwealth_coatings, vacate).

has_dimension(commonwealth_coatings, substantial_business_relationship_with_party, 1).

has_dimension(commonwealth_coatings, partiality_affected_arbitration, 0).

has_dimension(commonwealth_coatings, knowledge_by_party, 0).

/*** case **/

citation(reed_martin, '439 F.2d 1268').

shortname(reed_martin, 'Reed Martin').

longname(reed_martin,

'Reed & Martin, Inc. v. Westinghouse Electric Corp.').

year(reed_martin, 1971).

holding(reed_martin, uphold).

has_dimension(reed_martin, partiality_affected_arbitration, 0).

has_dimension(reed_martin, relationship_with_persons_similarly_situated, 1).

has_dimension(reed_martin, knowledge_by_party, 0).

distinguishes(reed_martin, commonwealth_coatings, substantial_business_relationship_with_party).

/*** case **/

citation(cook, '449 F.2d 106 (2d Cir.)').

shortname(cook, 'Cook').

longname(cook,

'Cook Industries, Inc. v. C. Itoh & Co.').

year(cook, 1971).

holding(cook, uphold).

has_dimension(cook, substantial_business_relationship_with_party, 2).

has_dimension(cook, narrow_business_area, 1).

has_dimension(cook, knowledge_by_party, 2).
distinguishes(cook, commonwealth_coatings, knowledge_by_party).
distinguishes(cook, commonwealth_coatings, narrow_business_area).
support(cook, garfield, narrow_business_area).
support(cook, garfield, knowledge_by_party).

/** case **/

citation(johnston, '6 Cal. App. 3d. 839, 86 Cal. Rptr. 133').
shortname(johnston, 'Johnston').
longname(johnston,
 'Johnston v. Security Ins. Co. of Hartford').
year(johnston, 1970).
holding(johnston, vacate).
has_dimension(johnston, arbitrator_was_not_appointed_as_neutral, 0).
has_dimension(johnston, partiality_affected_arbitration, 0).
has_dimension(johnston, social_or_family_relationship_with_lawyer, 2).
has_dimension(johnston, minor_business_relationship_with_lawyer, 1).
has_dimension(johnston, knowledge_by_party, 0).

/** case **/

citation(colony_liquor, '34 App. Div. 2d 1060, 312 N.Y.S.2d 403').
shortname(colony_liquor, 'Colony Liquor').
longname(colony_liquor,
 'Colony Liquor Distributers, Inc. v. Local 669, I.B.T.').
year(colony_liquor, 1970).
holding(colony_liquor, vacate).
has_dimension(colony_liquor, attorney_client_relationship_with_party, 1).
has_dimension(colony_liquor, knowledge_by_party, 0).

/** case **/

citation(perl, '34 App. Div. 2d. 748, 310 N.Y.S.2d 196').
shortname(perl, 'Perl').
longname(perl,
 'Perl v. General Fire and Cas. Co.').
year(perl, 1970).
holding(perl, uphold).
has_dimension(perl, relationship_with_persons_similarly_situated, 2).
has_dimension(perl, knowledge_by_party, 0).

/** case **/

citation(st_paul_ins, '6 Wash. App. 205, 492 P.2d 575').
shortname(st_paul_ins, 'St. Paul Insurance').
longname(st_paul_ins,
 'St. Paul Ins. Cos. v. Lusia').
year(st_paul_ins, 1971).
holding(st_paul_ins, uphold).
has_dimension(st_paul_ins, social_or_family_relationship_with_lawyer, 1).
has_dimension(st_paul_ins, financial_interest, 0).
has_dimension(st_paul_ins, inferred_knowledge_by_party, 1).

/** case **/

citation(garfield, '432 F.2d 849').
shortname(garfield, 'Garfield').
longname(garfield,
 'Garfield & Co. v. Wiest').
year(garfield, 1970).
holding(garfield, uphold).
has_dimension(garfield, substantial_business_relationship_with_party, 2).
has_dimension(garfield, knowledge_by_party, 1).

has_dimension(garfield, potential_conflicts_waived_in_contract, 1).
has_dimension(garfield, objection_to_arbitrator_made_prior_to_award, 1).
has_dimension(garfield, arbitrator_had_knowledge_of_objection, 1).
has_dimension(garfield, narrow_business_area, 1).
distinguishes(garfield, commonwealth_coatings, potential_conflicts_waived_in_contract).

/** case **/

citation(sanko, '495. F.2d 1260 (2d Cir.)').
shortname(sanko, 'Sanko').
longname(sanko,
 'Sanko S.S. Co., Ltd., v. Cook Industries, Inc.').
year(sanko, 1973).
holding(sanko, vacate).
has_dimension(sanko, attorney_client_relationship_with_lawyer, 1).
has_dimension(sanko, substantial_business_relationship_with_party, 1).
has_dimension(sanko, minor_business_relationship_with_party, 1).
distinguishes(sanko, garfield, knowledge_by_party).
distinguishes(sanko, garfield, potential_conflicts_waived_in_contract).
distinguishes(sanko, cook, narrow_business_area).
distinguishes(sanko, cook, knowledge_by_party).

/** case **/

citation(san_luis_obispo, '28 Cal. App. 3d 556, 104 Cal. Rptr. 733').
shortname(san_luis_obispo, 'San Luis Obispo').
longname(san_luis_obispo,
 'San Luis Obispo Bay Properties v. Pacific Gas and Elect. Co.').
year(san_luis_obispo, 1972).
holding(san_luis_obispo, uphold).
has_dimension(san_luis_obispo, relationship_with_party_or_lawyer_through_third_person, 1).

/** case **/

citation(j_p_stevens, '34 N.Y.2d 123, 356 N.Y.S.2d 278, 312 N.E.2d 466').
shortname(j_p_stevens, 'J. P. Stevens').
longname(j_p_stevens,
 'J. P. Stevens & Co., Inc. v. Rytex Corp.').
year(j_p_stevens, 1974).
holding(j_p_stevens, vacate).
has_dimension(j_p_stevens, substantial_business_relationship_with_party, 1).
has_dimension(j_p_stevens, partiality_affected_outcome, 0).
has_dimension(j_p_stevens, knowledge_by_party, 0).

/** case **/

citation(andros, '579 F.2d 691 (2d. Cir.)').
shortname(andros, 'Andros').
longname(andros,
 'Andros Compania Maritima v. Marc Rich & Co.').
year(andros, 1978).
holding(andros, uphold).
has_dimension(andros, social_or_family_relationship_with_party, 2).
has_dimension(andros, financial_interest_in_outcome, 0).

/** case **/

citation(ormsbee, '668 F.2d 1140 (10th Cir.)').
shortname(ormsbee, 'Ormsbee').
longname(ormsbee,
 'Ormsbee Development Co. v. Grace').
year(ormsbee, 1982).
holding(ormsbee, uphold).
has_dimension(ormsbee, relationship_with_persons_similarly_situated, 2).
has_dimension(ormsbee, financial_interest, 0).

```
/** case **/
citation(foley, '233 Kan. 339, 662 P.2d 1254').
shortname(foley, 'Foley').
longname(foley,
    'Foley Co. v. Grindsted Products, Inc.').
year(foley, 1983).
holding(foley, uphold).
has_dimension(foley, minor_business_relationship_with_party, 1).
has_dimension(foley, partiality_affected_arbitration, 0).
has_dimension(foley, arbitrator_was_not_appointed_as_neutral, 1).
support(foley, statute, arbitrator_was_not_appointed_as_neutral).
support(foley, perl, narrow_business_area).
```

```
/** case **/
citation(weinrott, '32 N.Y.2d 190, 344 N.Y.S.2d 848, 298 N.E.2d 42').
shortname(weinrott, 'Weinrott').
longname(weinrott,
    'Weinrott v. Carp').
year(weinrott, 1973).
holding(weinrott, uphold).
has_dimension(weinrott, relationship_with_party_or_lawyer_through_third_person, 2).
has_dimension(weinrott, unanimous_arbitration_panel, 1).
has_dimension(weinrott, knowledge_by_party, 0).
has_dimension(weinrott, partiality_affected_arbitration, 0).
```

```
/** case **/
citation(milliken, '99 App. Div. 2d 993, 473 N.Y.S.2d 443').
shortname(milliken, 'Milliken').
longname(milliken,
    'Milliken & Co. v. Tiffany Loungewear, Inc.').
year(milliken, 1984).
holding(milliken, uphold).
has_dimension(milliken, objection_to_arbitrator_made_prior_to_award, 1).
has_dimension(milliken, knowledge_by_party, 1).
has_dimension(milliken, minor_business_relationship_with_party, 1).
has_dimension(milliken, partiality_affected_arbitration, 0).
```

```
/** case **/
citation(house_grain, '659 S.W.2d 903 (Texas)').
shortname(house_grain, 'House Grain').
longname(house_grain,
    'House Grain Co. v. Obst').
year(house_grain, 1983).
holding(house_grain, uphold).
has_dimension(house_grain, minor_business_relationship_with_party, 1).
has_dimension(house_grain, knowledge_by_party, 0).
has_dimension(house_grain, relationship_unknown_to_arbitrator, 1).
has_dimension(house_grain, partiality_affected_arbitration, 0).
```

```
/** case **/
citation(r_e_bean, '139 Vt. 200, 428 A.2d 306').
shortname(r_e_bean, 'R. E. Bean').
longname(r_e_bean,
    'R. E. Bean Construction Co. v. Middlebury Assoc.').
year(r_e_bean, 1980).
holding(r_e_bean, uphold).
has_dimension(r_e_bean, relationship_unknown_to_arbitrator, 1).
has_dimension(r_e_bean, attorney_client_relationship_with_party, 2).
has_dimension(r_e_bean, social_or_family_relationship_with_lawyer, 2).
```

has_dimension(r_e_bean, social_or_family_relationship_with_party, 2).
has_dimension(r_e_bean, knowledge_by_party, 1).

/** case **/

citation(isaacson, '359 F.Supp. 1050 (DC NY)').
shortname(isaacson, 'Isaacson').
longname(isaacson,
 'Isaacson v. Hayden, Stone Inc.').
year(isaacson, unknown).
holding(isaacson, uphold).
has_dimension(isaacson, substantial_business_relationship_with_party, 2).
has_dimension(isaacson, narrow_business_area, 1).
has_dimension(isaacson, knowledge_by_party, 1).
has_dimension(isaacson, objection_to_arbitrator_made_prior_to_award, 1).
has_dimension(isaacson, potential_conflicts_waived_in_contract, 1).
supports(isaacson, garfield, potential_conflicts_waived_in_contract).

/** case **/

citation(barcon, '86 N.J. 179, 430 A.2d 214').
shortname(barcon, 'Barcon').
longname(barcon,
 'Barcon Assoc., Inc. v. Tri-County Asphalt Co.').
year(barcon, 1981).
holding(barcon, vacate).
has_dimension(barcon, substantial_business_relationship_with_party, 2).
has_dimension(barcon, indebtedness_with_party, 1).
has_dimension(barcon, arbitrator_was_not_appointed_as_neutral, 1).
has_dimension(barcon, partiality_affected_arbitration, 0).
has_dimension(barcon, knowledge_by_party, 2).

/** case **/

citation(abramovich, '416 A.2d 474 (Penn)').
shortname(abramovich, 'Abramovich').
longname(abramovich,
 'Abramovich v. Penn. Liquor Control Board').
year(abramovich, 1980).
holding(abramovich, vacate).
has_dimension(abramovich, attorney_client_relationship_with_party, 2).
has_dimension(abramovich, improper_communications_with_party_or_lawyer, 1).
has_dimension(abramovich, knowledge_by_party, 0).

/** case **/

citation(james_d_morrissey, '296 Pa.Super. 151, 443 A.2d 344').
shortname(james_d_morrissey, 'James D. Morrissey').
longname(james_d_morrissey,
 'James D. Morrissey, Inc. v. Gross Construction Co., Inc.').
year(james_d_morrissey, 1982).
holding(james_d_morrissey, vacate).
has_dimension(james_d_morrissey, substantial_business_relationship_with_party, 2).
has_dimension(james_d_morrissey, arbitrator_was_not_appointed_as_neutral, 0).
has_dimension(james_d_morrissey, knowledge_by_party, 0).

/** case **/

citation(richco, '82 Wis.2d 547, 263 N.W.2d 204').
shortname(richco, 'Richco').
longname(richco,
 'Richco Structures v. Parkside Village, Inc.').
year(richco, 1978).
holding(richco, vacate).
has_dimension(richco, relationship_with_party_or_lawyer_through_third_person, 1).

has_dimension(richco, substantial_business_relationship_with_party, 2).
has_dimension(richco, attorney_client_relationship_with_lawyer, 2).
has_dimension(richco, knowledge_by_party, 1).
has_dimension(richco, partiality_affected_outcome, 0).
support(richco, cook, knowledge_by_party).
support(richco, san_luis_obispo, relationship_with_party_or_lawyer_through_third_person).
support(richco, johnston, minor_business_relationship_with_lawyer).

/** case **/

citation(chevron, '300 F.Supp. 179').
shortname(chevron, 'Chevron').
longname(chevron,
 'Chevron Transport Corp. v. Astro Vencedor Compania Naviera').
year(chevron, 1969).
holding(chevron, uphold).
has_dimension(chevron, social_or_family_relationship_with_party, 1).

/** case **/

citation(holodnak, '381 F.Supp. 191').
shortname(holodnak, 'Holodnak').
longname(holodnak,
 'Holodnak v. Avco Corp.').
year(holodnak, unknown).
holding(holodnak, vacate).
has_dimension(holodnak, comments_by_arbitrator_during_arbitration, 1).

/** case **/

citation(northwest_mechanical, '283 N.W.2d 522 (Minn.)').
shortname(northwest_mechanical, 'Northwest Mechanical').
longname(northwest_mechanical,
 'Northwest Mechanical, Inc. v. Public Utilities Comm.').
year(northwest_mechanical, 1979).
holding(northwest_mechanical, vacate).
has_dimension(northwest_mechanical, attorney_client_relationship_with_party, 1).
has_dimension(northwest_mechanical, substantial_business_relationship_with_party, 1).
has_dimension(northwest_mechanical, partiality_affected_outcome, 0).

/** case **/

citation(jean_a_mccoy, '48 Ill. App. 3d. 802, 363 N.E.2d 442').
shortname(jean_a_mccoy, 'Jean A. McCoy').
longname(jean_a_mccoy,
 'Jean A. McCoy & Sons, Inc. v. LaSalle Cnty.').
year(jean_a_mccoy, 1977).
holding(jean_a_mccoy, uphold).
has_dimension(jean_a_mccoy, improper_communications_with_party_or_lawyer, 1).
has_dimension(jean_a_mccoy, knowledge_by_party, 1).

/** case **/

citation(mckinney_drilling, '32 Md. App. 205, 359 A.2d 100').
shortname(mckinney_drilling, 'McKinney Drilling').
longname(mckinney_drilling,
 'McKinney Drilling v. Mach I Ltd.').
year(mckinney_drilling, unknown).
holding(mckinney_drilling, vacate).
has_dimension(mckinney_drilling, relationship_with_party_or_lawyer_through_third_person, 1).
has_dimension(mckinney_drilling, partiality_affected_outcome, 1).

/** case **/

citation(baar_and_beards, '331 N.Y.S.2d 670').
shortname(baar_and_beards, 'Baar & Beards').

longname(baar_and_beards,
 'Baar & Beards, Inc. v. Oleg Cassini, Inc.').
year(baar_and_beards, 1972).
holding(baar_and_beards, uphold).
has_dimension(baar_and_beards, attorney_client_relationship_with_party, 1).
has_dimension(baar_and_beards, knowledge_by_party, 2).
has_dimension(baar_and_beards, objection_to_arbitrator_made_prior_to_award, 1).
has_dimension(baar_and_beards, partiality_affected_outcome, 0).
has_dimension(baar_and_beards, arbitrator_had_knowledge_of_objection, 1).
has_dimension(baar_and_beards, arbitrator_was_not_appointed_as_neutral, 1).

/** case **/
citation(saville, '107 Ill. App. 3d 799, 438 N.W.2d 509').
shortname(saville, 'Saville').
longname(saville,
 'Saville International, Inc. v. Galanti Group, Inc.').
year(saville, 1982).
holding(saville, uphold).
has_dimension(saville, comments_by_arbitrator_during_arbitration, 1).

/** case ** Double check mckinney, third party **/
citation(freeport, '61 Ill. App. 3d 999, 378 N.E.2d 558').
shortname(freeport, 'Freeport').
longname(freeport,
 'Freeport Construction Co. v. Star Forge, Inc.').
year(freeport, 1978).
holding(freeport, vacate).
has_dimension(freeport, objection_to_arbitrator_made_prior_to_award, 1).
has_dimension(freeport, arbitrator_had_knowledge_of_objection, 1).
has_dimension(freeport, potential_conflicts_waived_in_contract, 0).
has_dimension(freeport, knowledge_by_party, 2).
has_dimension(freeport, relationship_with_party_or_lawyer_through_third_person, 2).
support(freeport, san_luis_obispo, relationship_with_party_or_lawyer_through_third_person).
support(freeport, mckinney_drilling, relationship_with_party_or_lawyer_through_third_person).

/** case **/
citation(del_bianco, '6 Ill. App. 3d 286, 285 N.E.2d 480').
shortname(del_bianco, 'Del Bianco').
longname(del_bianco,
 'Del Bianco & Assoc., Inc. v. Adam').
year(del_bianco, 1972).
holding(del_bianco, uphold).
has_dimension(del_bianco, comments_by_arbitrator_during_arbitration, 1).

/** case **/
citation(country_mutual_insurance, '2 Ill. App. 2d 313, 276 N.E.2d 357').
shortname(country_mutual_insurance, 'Country Mutual Insurance').
longname(country_mutual_insurance,
 'Country Mutual Ins. Co. v. Kuzmickas').
year(country_mutual_insurance, 1971).
holding(country_mutual_insurance, uphold).
has_dimension(country_mutual_insurance, excessive_or_inadequate_award, 1).
has_dimension(country_mutual_insurance, partiality_affected_outcome, 0).

/** case **/
citation(albion, '130 Mich. App. 698, 344 N.W.2d 55').
shortname(albion, 'Albion').
longname(albion,
 'Albion Pub. Schools v. Albion Ed. Assoc.').
year(albion, 1983).

holding(albion, vacate).
has_dimension(albion, employee_employer_relationship_with_party, 1).
has_dimension(albion, knowledge_by_party, 0).

/** case **/
citation(hartman, '474 A.2d 959').
shortname(hartman, 'Hartman').
longname(hartman,
 'Hartman v. Cooper').
year(hartman, unknown).
holding(hartman, vacate).
has_dimension(hartman, partiality_affected_outcome, 0).
has_dimension(hartman, relationship_with_persons_similarly_situated, 1).
has_dimension(hartman, knowledge_by_party, 0).

/** case **/
citation(coughlan, '23 Mass. App. Ct. 994, 505 N.E.2d 203').
shortname(coughlan, 'Coughlan').
longname(coughlan,
 'Coughlan Construction Co., Inc. v. Town of Rockport').
year(coughlan, 1987).
holding(coughlan, uphold).
has_dimension(coughlan, social_or_family_relationship_with_lawyer, 1).
has_dimension(coughlan, unanimous_arbitration_panel, 1).
has_dimension(coughlan, knowledge_by_party, 0).
support(coughlan, weinrott, unanimous_arbitration_panel).

/** case **/
citation(turner, '19 Mass. App. 925, 471 N.E.2d 770').
shortname(turner, 'Turner').
longname(turner,
 'Turner Fisheries, Inc. v. Seafood Workers Union').
year(turner, 1984).
holding(turner, uphold).
has_dimension(turner, financial_interest_in_outcome, 0).
has_dimension(turner, arbitrator_was_not_appointed_as_neutral, 0).
has_dimension(turner, relationship_with_persons_similarly_situated, 1).
has_dimension(turner, comments_by_arbitrator_during_arbitration, 1).

/** case **/
citation(egan, '414 N.W.2d 785 (Minn.)').
shortname(egan, 'Egan').
longname(egan,
 'Egan & Sons Co. v. Mears Park Development Co.').
year(egan, 1987).
holding(egan, vacate).
has_dimension(egan, attorney_client_relationship_with_party, 1).
has_dimension(egan, knowledge_by_party, 0).

/** case **/
citation(franke, '421 N.W.2d 406 (Minn.)').
shortname(franke, 'Franke').
longname(franke,
 'Franke v. Farm Bureau Mut. Ins. Co.').
year(franke, 1988).
holding(franke, uphold).
has_dimension(franke, attorney_client_relationship_with_lawyer, 2).
has_dimension(franke, unanimous_arbitration_panel, 0).
has_dimension(franke, arbitrator_was_not_appointed_as_neutral, 1).
has_dimension(franke, partiality_affected_outcome, 0).

/*** case **/

citation(morelite, '748 F.2d 79 (2d Cir.)').

shortname(Morelite, 'Morelite').

longname(Morelite,

'Morelite Construction Corp. v. N.Y. City Dist. Council').

year(morelite, 1984).

holding(morelite, vacate).

has_dimension(morelite, objection_to_arbitrator_made_prior_to_award, 1).

has_dimension(morelite, social_or_family_relationship_with_party, 2).

has_dimension(morelite, knowledge_by_party, 2).

distinguishes(morelite, cook, objection_to_arbitrator_made_prior_to_award).

/*** case **/

citation(merit_ins, '714 F.2d 673 (2d Cir.)').

shortname(merit_ins 'Merit Insurance Co.').

longname(merit_ins

'Merit Ins. Co. v. Leatherby Ins. Co.').

year(merit_ins, 1983).

holding(merit_ins, uphold).

has_dimension(merit_ins, arbitrator_was_not_appointed_as_neutral, 0).

has_dimension(merit_ins, knowledge_by_party, 0).

has_dimension(merit_ins, employee_employer_relationship_with_party, 1).

```

domain('Criminal Appeal').
get_opposite_side(allowed, disallowed).
get_opposite_side(disallowed, allowed).
write_side(allowed) :-
    write('allowing the government to appeal').
write_side(disallowed) :-
    write('not allowing an appeal').
write_side_caps(allowed) :-
    write('ALLOWING THE GOVERNMENT TO APPEAL').
write_side_caps(disallowed) :-
    write('NOT ALLOWING AN APPEAL').
pro(further_proceedings_required, disallowed).
range(further_proceedings_required,
['the dimension did not occur',
'further proceedings are unlikely',
'further proceedings are possible',
'further proceedings are definite',
'a new trial will be required']).
describe(further_proceedings_required,
'further proceedings will be required').

pro(hung_jury, allowed).
range(hung_jury,
['the dimension did not occur',
'the dimension did occur']).
describe(hung_jury,
'the trial jury was hung').

pro(labelled_acquittal_by_judge, disallowed).
range(labelled_acquittal_by_judge,
['the dimension did not occur',
'the dimension did occur']).
describe(labelled_acquittal_by_judge,
'the trial judge labeled her action an acquittal').

pro(trial_held_before_judge, disallowed).
range(trial_held_before_judge,
['the dimension did not occur',
'the dimension did occur']).
describe(trial_held_before_judge,
'the trial was held before a judge').

pro(trial_judge_implied_guilt, allowed).
range(trial_judge_implied_guilt,
['the dimension did not occur',
'dimension did occur']).
describe(trial_judge_implied_guilt,
'the trial judge implied that the defendant was guilty').

pro(reprosecution_was_contemplated_by_judge, allowed).
range(reprosecution_was_contemplated_by_judge,
['the dimension did not occur',
'the dimension did occur']).
describe(reprosecution_was_contemplated_by_judge,
'the judge contemplated the defendant would be reprosecuted').

pro(pre_trial_type_motion_not_timely, allowed).
range(pre_trial_type_motion_not_timely,
['the dimension did not occur',
'the dimension did occur']).

```

describe(pre_trial_type_motion_not_timely,
 'the defendant was not timely in a pretrial type of motion').

pro(defendant_initiates_motion, allowed).
range(defendant_initiates_motion,
['the dimension did not occur',
'the dimension did occur']).
describe(defendant_initiates_motion,
 'the defendant initiated the motion').

pro(motion_unrelated_to_guilt, allowed).
range(motion_unrelated_to_guilt,
['the dimension did not occur',
'the dimension did occur']).
describe(motion_unrelated_to_guilt,
 'the motion to dismiss was unrelated to guilt').

pro(final_verdict_reached, allowed).
range(final_verdict_reached,
['the dimension did not occur',
'the defendant was found not guilty',
'the defendant was found guilty']).
describe(final_verdict_reached,
 'a final verdict was reached at the trial level').

citation(sanabria, '437 U.S. 54').
shortname(sanabria, 'Sanabria').
longname(sanabria, 'Sanabria v. U.S.').
year(sanabria, 1978).
holding(sanabria, disallowed).
has_dimension(sanabria, final_verdict_reached, 1).
has_dimension(sanabria, defendant_initiates_motion, 1).
has_dimension(sanabria, further_proceedings_required, 4).
has_dimension(sanabria, motion_unrelated_to_guilt, 0).
has_dimension(sanabria, labelled_acquittal_by_judge, 1).
distinguishes(sanabria, lee, reprosecution_was_contemplated_by_judge).
distinguishes(sanabria, lee, final_verdict_reached).
citation(lee, '432 U.S. 23').
shortname(lee, 'Lee').
longname(lee, 'Lee v. U.S.').
year(lee, 1977).
holding(lee, allowed).
has_dimension(lee, motion_unrelated_to_guilt, 1).
has_dimension(lee, defendant_initiates_motion, 1).
has_dimension(lee, pre_trial_type_motion_not_timely, 1).
has_dimension(lee, further_proceedings_required, 4).
has_dimension(lee, reprosecution_was_contemplated_by_judge, 1).
has_dimension(lee, trial_held_before_judge, 1).
citation(finch, '433 U.S. 676').
shortname(finch, 'Finch').
longname(finch, 'Finch v. U.S.').
year(finch, 1978).
holding(finch, disallowed).
has_dimension(finch, defendant_initiates_motion, 0).
has_dimension(finch, further_proceedings_required, 2).
has_dimension(finch, trial_judge_implied_guilt, 0).
has_dimension(finch, final_verdict_reached, 0).
distinguishes(finch, wilson, final_verdict_reached).
citation(scott, '437 U.S. 82').
shortname(scott, 'Scott').
longname(scott, 'U.S. v. Scott').
year(scott, 1978).
holding(scott, allowed).
has_dimension(scott, further_proceedings_required, 1).
has_dimension(scott, motion_unrelated_to_guilt, 1).
has_dimension(scott, defendant_initiates_motion, 1).
overrule(scott, jenkins).
citation(wilson, '420 U.S. 332').
shortname(wilson, 'Wilson').
longname(wilson, 'U.S. v. Wilson').
year(wilson, 1975).
holding(wilson, allowed).
has_dimension(wilson, motion_unrelated_to_guilt, 1).
has_dimension(wilson, defendant_initiates_motion, 1).
has_dimension(wilson, final_verdict_reached, 2).
has_dimension(wilson, further_proceedings_required, 0).
citation(jenkins, '420 U.S. 358').
shortname(jenkins, 'Jenkins').
longname(jenkins, 'U.S. v. Jenkins').
year(jenkins, 1975).
holding(jenkins, disallowed).
has_dimension(jenkins, motion_unrelated_to_guilt, 0).
has_dimension(jenkins, further_proceedings_required, 3).
has_dimension(jenkins, trial_held_before_judge, 1).
citation(martin_linen, '430 U.S. 564').

shortname(martin_linen, 'Martin Linen').
longname(martin_linen, 'U.S. v. Martin Linen').
year(martin_linen, 1977).
holding(martin_linen, disallowed).
has_dimension(martin_linen, motion_unrelated_to_guilt, 0).
has_dimension(martin_linen, defendant_initiates_motion, 1).
has_dimension(martin_linen, further_proceedings_required, 2).
has_dimension(martin_linen, labelled_acquittal_by_judge, 1).
has_dimension(martin_linen, hung_jury, 1).
citation(doe, 'unknown').
shortname(doe, 'John Doe Hypo').
longname(doe, 'John Doe Hypo').
year(doe, unknown).
holding(doe, unknown).
has_dimension(doe, defendant_initiates_motion, 1).
has_dimension(doe, motion_unrelated_to_guilt, 1).
has_dimension(doe, further_proceedings_required, 3).

```

/***** Dimensional ANalysis Program (DAn) *****/
**
** AUTHOR:   Daniel A. Tysver
** DATE: April 25, 1990
**
** In satisfaction of Harvard Law School's Written Work Requirement.
**
** The basic purpose of DAn is to allow users to analyze a legal
** domain through the use of dimensions, as was first described
** by Kevin Ashley and Edwina Rissland in their HYPO program.
**
** DAn allows users to input a legal domain through the use of a
** comprehensive menuing system. All of the data required by the
** program can be input through menus. Once a user inputs a domain,
** DAn is capable of analyzing the domain in a variety of ways:
**
** - Argumentation output similar to HYPO
**   + Including the use of Generalization Rules
**   + Plus allowing the user to select a variety of parameters,
** - Generalization analysis
**   + Including the creation of generalization rules
**   + Plus an output of the rules and their explanations,
** - Query analysis
**   + Including retrieval of cases or dimensions
**   + Allowing users to develop queries through menus,
** - General domain statistics.
**
*****/

```

```

/***** INTERFACE or START *****/

```

```

**

```

```

** This is the start up procedure and main menu.

```

```

**

```

```

** It initializes the terminal to return to the top of the scrollable
** region on a home command, writes the main banner (the screen heading
** and associated scrolling region), and calls the implementation of the
** main menu. After a selection is implemented, the procedure calls
** itself. Therefore, the only way to exit the program is by an
** abort command, executed by menu selection 11.

```

```

**

```

```

*****/

```

```

start :-

```

```

    scrolling_region_home,
    write_main_banner,
    menu_1,
    start.

```

```

/***** MENU 1 *****/

```

```

**

```

```

** Implements the Main Menu.

```

```

**

```

```

*****/

```

```

menu_1 :-

```

```

    home,
    write_menu_1,
    get_choice(Choice),!,
    do_menu_1(Choice).

```

```

/***** WRITE MENU 1 *****/

```

```

**

```

```

**

```

```

**

```

```

*****/

```

```

write_menu_1 :-

```

```

    wide, write('      MAIN MENU'),nl,nl,
    write('You may take any of the following actions:'),nl,nl,
    write('      1> Enter and argue a case from the casebase,')nl,
write('      2> Argue a case selected from a list of cases in the casebase,')nl,
write('      3> Input a new case for analysis or storage in the casebase,')nl,
    write('      4> Analyze the domain through generalization rules,')nl,
    write('      5> Change the program's parameters,')nl,
    write('      6> Retrieve a new domain,')nl,
    write('      7> Create a new domain,')nl,
    write('      8> Query the present domain,')nl,
    write('      9> Tell summary of domain statistics,')nl,
    write('     10> Consult an outside file for more domain information,')nl,
    write('or     11> Quit.').nl,nl.

```

```

/***** MAIN MENU CHOICE 1: ARGUE A CASE *****/

```

```

**

```

```

** Get-a-valid-case is called from the utilities file.

```

```

** Once the case is selected, Menu-2 is called to

```

```
** argue the case.
```

```
**
```

```
*****/
```

```
do_menu_1(1):-
```

```
    home,
```

```
    write('Enter the name of the case that you wish to analyze. '),nl,nl,
```

```
    get_a_valid_case(Case),
```

```
    menu_2(Case).
```

```
/** MAIN MENU CHOICE 2: ARGUE A CASE FROM LIST **
```

```
**
```

```
** Similar to Choice 1, except that the case list
```

```
** is printed immediately through the call to
```

```
** get-case-through-menu (found in utilities).
```

```
**
```

```
*****/
```

```
do_menu_1(2):-
```

```
    get_case_through_menu(Case),!,
```

```
    menu_2(Case).
```

```
/** MAIN MENU CHOICE 3: NEW CASE FOR ANALYSIS OR STORAGE **
```

```
**
```

```
** Calls the create-new-cases procedure in the NewCases file.
```

```
**
```

```
*****/
```

```
do_menu_1(3):-
```

```
    create_new_cases.
```

```
***** MAIN MENU CHOICE 4: ANALYZE DOMAIN *****
```

```
**
```

```
** Changes the banner to the analyze domain banner,
```

```
** and then calls the analyze-domain procedures
```

```
** from the analyze file. Output is directed
```

```
** to the file defined in the get-output parameter.
```

```
**
```

```
*****/
```

```
do_menu_1(4):-
```

```
    write_analyze_banner,
```

```
    analyze_domain,
```

```
    get_output(Output),
```

```
    tell(Output),
```

```
    write_domain_analysis,
```

```
    told,
```

```
    tell(user),
```

```
    ready.
```

```
/** MAIN MENU CHOICE 5: CHANGE PARAMETERS **
```

```
**
```

```
** Calls Menu-3, the parameters menu, and then recalls
```

```
** menu-1 (if execution stopped without recalling
```

```
** menu-1, then it would be necessary to re-write
```

```
** the main banner).
```

```

**
*****/

do_menu_1(5):-
    home,
    menu_3,
    menu_1.

/*** MAIN MENU CHOICE 6: RETRIEVE NEW DOMAIN ***
**
** Double checks the choice with the user, and then
** resets all domain and analysis information and
** the executes reconsult-appropriate-cases, which
** simply consults the cases chosen by the user
** which contains the new domain information.
**
*****/

do_menu_1(6):-
    home,
    write('Are you sure you want to retrieve a new domain?'),nl,
    get_yes_or_no(Choice),
    if(Choice = no,
        menu_1,
        true),
    nl,
    reset_domain_knowledge,
    reset_analysis_parameters,
    consult_appropriate_cases,
    start.

/*** MAIN MENU CHOICE 7: CREATE A NEW DOMAIN ***
**
** Calls the create-new-domain procedure
** from the NewDomain file.
**
*****/

do_menu_1(7):-
    create_new_domain.

/***** MAIN MENU CHOICE 8: QUERY THE DOMAIN *****/
**
** Calls query from the Query file. Query
** ends by failing, so it is necessary to
** provide an alternate way to satisfy this call.
**
*****/

do_menu_1(8):-
    query.
do_menu_1(8).

/*** MAIN MENU CHOICE 9: TELL SUMMARY OF DOMAIN STATS ***
**

```

```

** This procedure outputs a general summary of
** the statistics concerning the present domain.
** It is necessary that domain analysis has occurred
** in order to retrieve all the necessary statistics.
**
*****/

```

```

do_menu_1(9) :-
    home,
    get_and_write_statistics.
do_menu_1(9) :-
    ready,
    menu_1.

```

```

get_and_write_statistics :-
    domain(Domain),
    analyze_domain,
    number_of(Dimension1, X^pro(Dimension1, X), Dim_Number),
    number_of(Dimension2, X^both_sided(Dimension2, X), B_S_Dim_Number),
    number_of(Dimension3, X^overwhelming(Dimension3, X), O_Dim_Number),
    number_of(Dimension4, X^Y^necessary(Dimension4, X, Y), N_Dim_Number),
    pro(Dimension, Side),
    get_opposite_side(Side, Opp_Side),
    number_of(Dimension5, pro(Dimension5, Side), Dim_Number_Side),
    number_of(Dimension6, pro(Dimension6, Opp_Side), Dim_Number_Opp_Side),
    number_of(Case1, X^holding(Case1, X), Case_Number),
    number_of(Case2, holding(Case2, Side), Side_Case_Number),
    number_of(Case3, holding(Case3, Opp_Side), Opp_Side_Case_Number),
    number_of(Case4, holding(Case4, unknown), Unknown_Case_Number),
    write('    Summary Statistics of '), write(Domain), write(' Domain:'),nl,
    nl,write('Number of Cases: '), write(Case_Number),nl,
    write('Cases pro-'),write(Side), write(': '),
    write(Side_Case_Number),nl,
    write('Cases pro-'), write(Opp_Side), write(': '),
    write(Opp_Side_Case_Number),nl,
    write('Undecided Cases (Hypos): '),
    write(Unknown_Case_Number),nl,nl,
    write('Dimensions: '), write(Dim_Number),nl,
    write('Dimensions pro-'),write(Side), write(': '),
    write(Dim_Number_Side),nl,
    write('Dimensions pro-'),write(Opp_Side), write(': '),
    write(Dim_Number_Opp_Side),nl,
    write('Number of Both-Sided Dimensions: '), write(B_S_Dim_Number),nl,
    write('Number of Overwhelming Dimensions: '), write(O_Dim_Number),nl,
    write('Number of Necessary Dimensions: '), write(N_Dim_Number),nl,
    nl,nl,!fail.

```

```

number_of(Element, Test, Number) :-
    setof(Element, Test, List),
    length(List, Number).
number_of(Element, Test, 0).

```

```

/***** MAIN MENU CHOICE 10: CONSULT OUTSIDE FILE *****/
**
** Allows the user to specify an external file which contains
** information about the domain without resetting all domain
** information. Analysis information must be reset.
**

```

```
*****/
```

```
do_menu_1(10):-  
    home,  
    write('Are you sure you want to consult another file?'),nl,  
    get_yes_or_no(Choice),  
    if(Choice = no,  
        menu_1,  
        true),  
    nl,  
    reset_analysis_parameters,  
    consult_appropriate_cases,  
    start.
```

```
/***** MAIN MENU CHOICE 11: QUIT *****/
```

```
**  
** Calls end, which resets the terminal  
** screen window and aborts.  
**  
*****/
```

```
do_menu_1(11):- end.
```

```
/***** MAIN MENU INAPPROPRIATE RESPONSE *****/
```

```
**  
**  
**  
*****/
```

```
do_menu_1(_):-  
    inappropriate_response(New_Choice),  
    do_menu_1(New_Choice).
```

```
/****** MENU 2: ARGUE A CASE *****/
```

```
**
** This menu outputs a case banner and general case information, and
** then allows users to argue one or both sides of the case, or
** to argue this case against another case chosen by the user. The
** menu calls itself, so a choice is provided to return to the main menu.
**
** When a case which has not held for either side (ie, it is a hypo or
** and undecided case), it is necessary to temporarily assign it a
** definite holding in order for the argue procedures to work correctly.
** Thus, it was necessary to include the get-argue-for-side and
** restore-holding procedures.
**
*****/
```

```
menu_2(Case) :-
    holding(Case, Side_of_Case),
    write_case_banner(Case, Side_of_Case),
    if(include_generalization_rules_in_arguments,
        analyze_domain,
        true),
    home,
    write_implementation(Case, Side_of_Case),
    if(Side_of_Case == unknown,
        get_argue_for_side(Case, Side),
        Side = Side_of_Case),
    get_opposite_side(Side, Opp_Side),
    write_menu_2(Side, Opp_Side),
    get_choice(Choice),!,
    do_menu_2(Choice, Case, Side, Opp_Side, Side_of_Case),
    if(Side_of_Case == unknown,
        restore_holding(Case),
        true),
    ready,
    menu_2(Case).
```

```
get_argue_for_side(Case, New_Side) :-
    get_opposite_side(New_Side, Its_Opposite),
    retract(holding(Case, unknown)),
    asserta(holding(Case, New_Side)).
```

```
restore_holding(Case) :-
    retract(holding(Case, Current_Side)),
    asserta(holding(Case, unknown)).
```

```
write_implementation(Case, Side) :-
    write(' Holding   : '), write(Side),nl,
    write__dimensions(Case),nl.
```

```
/****** WRITE MENU 2 *****/
```

```
**
**
**
*****/
```

```
write_menu_2(Side, Opp_Side) :-
    write(' ANALYZE CASE MENU'), nl,
    write('This case can be analyzed in a several ways:'),nl,nl,
    write(' 1> Analyze the pro-'), write(Side), write(' position, '), nl,
```

```

write('    2> Analyze the pro-'), write(Opp_Side), write(' position, '),
nl, write('    3> Analyze both sides, '), nl,
write('    4> Analyze the case against another case chosen by you, '), nl,
write('or    5> Return to Main Menu. '), nl, nl.

```

```

/***** ARGUE MENU CHOICE 1 & 2: ANALYZE SIDE *****/

```

```

**
** Choice 1 always argues for the side the case holds for,
** while choice 2 argues for the opposite side.
**
** Using analyze-side from the Argue file.
**
*****/

```

```

do_menu_2(1, Case, Side, Opp_Side, Tag) :-
    !,
    get_output(Output),
    tell(Output),
    analyze_side(Case, Side, Opp_Side),
    told,
    tell(user).

```

```

do_menu_2(2, Case, Side, Opp_Side, Tag) :-
    !,
    get_output(Output),
    tell(Output),
    analyze_side(Case, Opp_Side, Side),
    told,
    tell(user).

```

```

/***** ARGUE MENU CHOICE 3: ARGUE BOTH SIDES *****/

```

```

**
** Using Analyze from the Argue file.
**
*****/

```

```

do_menu_2(3, Case, Side, Opp_Side, Tag) :-
    !,
    get_output(Output),
    tell(Output),
    analyze(Case, Side, Opp_Side),
    told,
    tell(user).

```

```

/**** ARGUE MENU CHOICE 4: ARGUE AGAINST CASE *****/

```

```

**
** Argue against a case chosen by the user.
** Get-a-valid-case is from the Utilities file.
** Analyze-against-case is from the Argue file.
**
*****/

```

```

do_menu_2(4, Case, Side, Opp_Side, Tag) :-
    get_a_valid_case(Against_Case), !,
    write_case_banner(Case, Side),
    get_output(Output),
    tell(Output),

```

```
analyze_against_case(Case, Side, Opp_Side, Against_Case),  
told,  
tell(user).
```

```
/**** ARGUE MENU CHOICE 4: ARGUE AGAINST CASE ***  
**  
**  
**  
*****/  

```

```
do_menu_2(5, Case, Side, Opp_Side, Tag) :-  
    if(Tag == unknown,  
        restore_holding(Case),  
        true),  
    start.
```

```
/**** ARGUE MENU CHOICE 4: ARGUE AGAINST CASE ***  
**  
**  
**  
*****/  

```

```
do_menu_2(_, Case, Side, Opp_Side, Tag) :-  
    inappropriate_response(New_Choice),  
    do_menu_2(New_Choice, Case, Side, Opp_Side, Tag).
```

```
/****** MENU 3: CHANGE PARAMETERS *****/
```

```
**  
** Writes out current parameters and allows the user to change  
** each paramter. This menu also calls itself, thus it has  
** a return to Main Menu choice.  
**  
****/
```

```
menu_3 :-  
    wide,write('CHANGE PARAMETERS MENU'),nl,nl,  
    write_out_parameters,  
    write_menu_3,  
    get_choice(Choice),  
    do_menu_3(Choice).
```

```
redo_menu_3 :-  
    home,  
    menu_3.
```

```
/****** WRITE OUT PARAMETERS *****/
```

```
**  
**  
**  
****/  
  
write_out_parameters :-  
    m_o_p_criteria(M_O_P_Type),  
    write('Most On Point Criteria Now Used: '), write(M_O_P_Type),nl,  
    write('Include Generalization Rules: '),  
    if(include_generalization_rules_in_arguments,  
        write('Yes'),  
        write('No')),nl,  
    write('Analyze "No" Dimension Values: '),  
    if(include_no_dimension_values_as_relevant,  
        write('Yes'),  
        write('No')),nl,  
    write('Include Overlapping Counter-Examples: '),  
    if(use_overlapping_counter_examples,  
        write('Yes'),  
        write('No')),nl,  
    support_needed(Support_Level),  
    get_output(File_Name),  
    write('Level of Support Needed for Rules: '),  
    write(Support_Level),nl,  
    write('Output is sent to: '),  
    write(File_Name),nl,nl.
```

```
/****** WRITE MENU 3 *****/
```

```
**  
**  
**  
****/  
  
write_menu_3 :-  
    write('You have the following choices:'),nl,  
    write('    1> Change Most-On-Point criteria, '),nl,  
    write('    2> Include/exclude generalization rules in arguments, '),nl,
```

```

        write('    3> Include/exclude "no" dimension values in analysis,')nl,
        write('    4> Include/exclude overlapping counter-examples,')nl,
write('    5> Change level of support needed to create generalization rule,')nl,
write('    6> Send Argumentation and Analysis to external file or screen,')nl,
        write('or    7> Return to Main Menu.').nl,nl.

```

```

/** CHANGE PARAMETERS MENU CHOICE 1: CHANGE M_O_P CRITERIA **
**

```

```

** This choice calls its own menu system, which allows the
** implemented most-on-point parameters to be displayed and
** explained to the user if desired.
**

```

```

*****/

```

```

do_menu_3(1):-
    home,
    m_o_p_criteria(M_O_P_Type),
    write('The current Most on Point algorithm is '), write(M_O_P_Type), nl,
    m_o_p_menu,
    redo_menu_3.

```

```

/***** MOST ON POINT PARAMETER MENU *****/

```

```

**
**
**
*****/

```

```

m_o_p_menu:-
    write_m_o_p_menu,
    get_choice(Choice),
    do_m_o_p_menu(Choice).

```

```

write_m_o_p_menu:-
    write('You may choose between the following options:'),nl,nl,
    write('    1> Change M_O_P algorithm to coverage'),nl,
    write('    2> Change M_O_P algorithm to coverage_1 (Minimum overlap)'),
nl,write('    3> Explain M_O_P algorithms,')nl,
    write('or    4> Exit menu without changing M_O_P algorithm.').nl,nl.

```

```

/**** M-O-P MENU CHOICE 1: COVERAGE ****

```

```

**
**
**
*****/

```

```

do_m_o_p_menu(1):-
    abolish(m_o_p_criteria, 1),
    assert(m_o_p_criteria(coverage)).

```

```

/**** M-O-P MENU CHOICE 2: COVERAGE-1 ****

```

```

**
**
**
*****/

```

```

do_m_o_p_menu(2):-
    abolish(m_o_p_criteria, 1),

```

```
assert(m_o_p_criteria(coverage_1)).
```

```
/** M-O-P MENU CHOICE 3: EXPLAIN M-O-P CRITERIA **  
**  
**  
**  
*****/
```

```
do_m_o_p_menu(3) :-  
    home,  
    write('There are currently two implemented most on point criteria:'),nl,  
    write('coverage and coverage_1. As could be guessed from their'),nl,  
    write('titles, both algorithms are based on the coverage technique'),nl,  
    write('described by Ashley in his Hypo program. '),nl,nl,  
    write('COVERAGE is basically the same algorithm used in Hypo. It'),nl,  
    write('determines whether one case is "more on point" than another'),nl,  
    write('by looking at the number of dimensions that each case shares'),  
    nl,write('with the current fact situation. The more dimensions'),nl,  
    write('shared, the more on point a case is. If two cases have the'),nl,  
    write('the same number of dimensions in common, then both cases are'),  
    nl,write('equally on point'),nl,nl,  
    write('COVERAGE_1 uses the same technique, but it further analyzes'),nl,  
    write('cases that share an equal number of dimensions with the'),nl,  
    write('current fact situation. If a case shares the same number'),nl,  
    write('of dimensions, but has fewer non-shared dimensions than'),nl,  
    write('another case, then that case is more on point. '),nl,nl,  
    ready,  
    home,  
    m_o_p_menu.
```

```
/** M-O-P MENU CHOICE 4: MAIN MENU ****  
**  
**  
**  
*****/
```

```
do_m_o_p_menu(4).
```

```
/** M-O-P MENU INAPPROPRIATE CHOICE ****  
**  
**  
**  
*****/
```

```
do_m_o_p_menu(_) :-  
    inappropriate_response(New_Choice),  
    do_m_o_p_menu(New_Choice).
```

```
/** CHANGE PARAMETERS MENU CHOICE 2: GENERALIZATION RULES **  
**  
**  
**  
*****/
```

```
do_menu_3(2) :-
```

```

home,
write('Currently, generalization rules '),
if(include_generalization_rules_in_arguments,
    write('are'),
    write('are not')),
write(' included in argument production.').nl,
write('Would you like them to be included?').nl,
get_yes_or_no(Choice),
if(Choice = yes,
    asserta(include_generalization_rules_in_arguments),
    abolish(include_generalization_rules_in_arguments,0)),
redo_menu_3.

```

/** CHANGE PARAMETERS MENU CHOICE 3: INCLUDE NO DIMENSION VALUES **

```

**
**
**
*****/

```

```

do_menu_3(3) :-
    home,
    write('Currently, "no" dimension values '),
    if(include_no_dimension_values_as_relevant,
        write('are'),
        write('are not')),
    write(' considered relevant in determining').nl,
write('supporting and opposing cases. Would you like them to be included?').nl,
nl, get_yes_or_no(Choice),
if(Choice = yes,
    asserta(include_no_dimension_values_as_relevant),
    abolish(include_no_dimension_values_as_relevant, 0)),
redo_menu_3.

```

/** CHANGE PARAMETERS MENU CHOICE 4: INCLUDE OVERLAPPING COUNTER EXAMPLES **

```

**
**
**
*****/

```

```

do_menu_3(4) :-
    home,
    write('Currently, overlapping counter examples '),
    if(include_no_dimension_values_as_relevant,
        write('are'),
        write('are not')),
    write(' use in argument production.').nl,
    write('Would you like them to be included?').nl,nl,
    get_yes_or_no(Choice),
    if(Choice = yes,
        asserta(use_overlapping_counter_examples),
        abolish(use_overlapping_counter_examples, 0)),
redo_menu_3.

```

/** CHANGE PARAMTERS MENU CHOICE 5: LEVEL OF SUPPORT ***

```
**
**
**
***** /
```

```
do_menu_3(5) :-
    get_support_needed(Old_Level, New_Level),
    check_reset_analysis_parameters(Old_Level, New_Level),
    redo_menu_3.
```

```
get_support_needed(Old_Level, New_Level) :-
    home,
    support_needed(Old_Level),
    write('Currently, at least '), write(Old_Level),
    write(' case[s] must support a rule in order'),nl,
write('for the rule to be used in the domain analysis or a case argument.').nl,
nl,write('What would you like the new level of support to be?').nl,nl,
get_a_number(New_Level).
```

```
check_reset_analysis_parameters(Old_Level, New_Level) :-
    Old_Level == New_Level.
check_reset_analysis_parameters(Old_Level, New_Level) :-
    reset_analysis_parameters,
    abolish(support_needed,1),
    asserta(support_needed(New_Level)).
```

```
/***** CHANGE PARAMETERS MENU CHOICE 6: EXPORT TO FILE *****/
**
**
**
***** /
```

```
do_menu_3(6) :-
    home,
    write('Please enter the name of the file to which you wish the'),nl,
    write('analysis output to be sent. If you wish all output to go'),nl,
    write('to the screen, enter "user"'),nl,
    get_unused_file_name(File_Name),
    abolish(get_output,1),
    asserta(get_output(File_Name)),
    redo_menu_3.
```

```
/***** CHANGE PARAMETERS MENU CHOICE 7: RETURN TO MAIN MENU *****/
**
**
**
***** /
```

```
do_menu_3(7).
```

```
/***** CHANGE PARAMETERS MENU INAPPROPRIATE RESPONSE *****/
**
**
**
***** /
```

```

do_menu_3(_):-
    inappropriate_response(New_Choice),
    do_menu_3(New_Choice).

/***** WRITE BANNERS *****/
**
** These procedures write screen banners across the
** top of the terminal screen, and then set the
** scrollable region so as to protect those banners.
**
** The current banner (or screen condition) is recorded
** through the screen_type parameter. Thus, if a banner
** is ever to be written while it is currently on the screen,
** then the writing procedure is skipped.
**
*****/

write_main_banner :-
    screen_type(main),!, home.
write_main_banner :-
    domain(Domain),
    narrow_screen,
    set_screen_type(main),
    set_window(1,24),
    home,
    big_top, tab(4), write('DAN WORLD--'), write(Domain), nl,
    big_bottom, tab(4), write('DAN WORLD--'), write(Domain), nl,nl,
    set_window(4,23).

write_case_banner(Case, Side) :-
    screen_type(case), !, home.
write_case_banner(Case, Side) :-
    narrow_screen,
    set_screen_type(case),
    set_window(1,24),
    home,
    write('*****'),nl,
    wide, write(Case), nl,
    tab(5), write_long_name(Case), nl,
    tab(5), write_citation(Case),nl,
    write('*****'),nl,
    set_window(7,24).

write_analyze_banner :-
    screen_type(analyze), !, home.
write_analyze_banner :-
    narrow_screen,
    set_screen_type(analyze),
    set_window(1,24),
    home,
    domain(Domain),
    wide, write('Analysis of '), write(Domain), write(' Domain'),
    set_window(3,24).

/***** END *****/
**
** Restores window to entire screen before termination.
**
*****/

```

end :-

save_cursor,
set_window(1,24),
restore_cursor,
abort.

/***** INTERFACE PROCEDURES *****/

**

** Procedures used in more than one location in the interface.

**

*****/

reset_analysis_parameters :-

abolish(analysis_done,0),
abolish(presumption,2),
abolish(both_sided,2),
abolish(overwhelming,2),
abolish(necessary, 3),
abolish(overwhelming_dimension_pair,3),
abolish(dominated_dimension_pair,3).

reset_domain_knowledge :-

abolish(write_side, 1),
abolish(write_side_caps, 1),
abolish(domain,1),
abolish(get_opposite_side, 2),
abolish(dimension_number, 2),
abolish(pro, 2),
abolish(describe, 2),
abolish(range, 2),
abolish(citation, 2),
abolish(shortname, 2),
abolish(holding, 2),
abolish(year, 2),
abolish(jurisdiction, 2),
abolish(has_dimension, 3),
abolish(support, 3),
abolish(distinguishes, 3),
abolish(override, 3).

/***** PARAMETERS *****/

**

** Not Included in list (ie, not effective at start of program):

** include_no_dimension_values_as_relevant.

**

*****/

screen_type(plain).

get_output(user).

support_needed(2).

use_overlapping_counter_examples.

include_generalization_rules_in_arguments.

m_o_p_criteria(coverage_1).

/***** CONSULT APPROPRIATE CASES *****/

**

**** Consults existing file chosen by the user.**

*******/**

consult_appropriate_cases :-

nl,

write('Enter the name of a file containing information on the domain. '),

nl,

get_existing_file(File_Name),

home,

consult(File_Name),

nl,write('Is information contained in any more files?'),nl,

get_yes_or_no(Answer),

if(Answer = yes,

consult_appropriate_cases,

true).

```

/***** CREATE NEW DOMAIN *****/
**
** This procedure allows the user to create a new domain through a menu
** system. The domain consists of information about the domain in
** general:
**
** - Name of domain,
** - Possible sides for which a case could hold,
** - Descriptions of the possible sides,
**
** as well as information about each dimension which exists in the domain:
**
** - Name of the dimension,
** - Which side the dimension favors,
** - How many magnitudes the dimension can have,
** - The names of each magnitude of the dimension,
** - A phrase that can be used to describe the dimension.
**
** Present domain and analysis information must be reset, as
** any analysis must be redone on this new domain.
**
*****/

```

```

create_new_domain :-
    home,
    write('Are you sure you want to create a new domain?'),nl,nl,
    get_yes_or_no(Answer),
    if(Answer = yes,
        true,
        menu_1),
    reset_domain_knowledge,
    reset_analysis_parameters,
    get_unused_file_name(File_Name),
    get_new_domain_info(File_Name),
    get_new_dimension_info(File_Name).

```

```

/***** GET NEW DOMAIN INFORMATION *****/
**
** Gets the domain information and writes it to a file.
**
*****/

```

```

get_new_domain_info(File_Name) :-
    write('Please enter the name of the domain in single quotes. '),nl,
    get_choice(Domain_Name),nl,
    write('Please name one of the two sides that may win a dispute. '),nl,
    get_choice(Side1),nl,
    write('Please enter the other side that may win a dispute. '),nl,
    get_choice(Side2),nl,nl,
    write('Please describe in a short phrase using only small letters. '),nl,
    write('how you want the pro-'), write(Side1),
    write(' position to be stated by the '),nl,
    write('program. Please put the phrase in single quotation marks. '),nl,
    get_choice(Side1_Description),nl,
    write('Now enter the same phrase using only capital letters. '),nl,
    get_choice(Side1_Description_Caps),nl,nl,nl,
    write('Please describe in a short phrase using only small letters. '),nl,
    write('how you want the pro-'), write(Side2),
    write(' position to be stated by the '),nl,

```

```

write('program. Please put the phrase in single quotation marks.').nl,
get_choice(Side2_Description).nl,
write('Now enter the same phrase using only capital letters.').nl,
get_choice(Side2_Description_Caps).nl,
tell(File_Name),
write('domain(''),write(Domain_Name), write(')').nl,
write('get_opposite_side('), write(Side1), write(', '), write(Side2),
write(')').nl,
write('get_opposite_side('), write(Side2), write(', '), write(Side1),
write(')').nl,
define_write_side(Side1, Side1_Description),
define_write_side(Side2, Side2_Description),
define_write_side_caps(Side1, Side1_Description_Caps),
define_write_side_caps(Side2, Side2_Description_Caps),
asserta(domain(Domain_Name)),
asserta(get_opposite_side(Side1, Side2)),
asserta(get_opposite_side(Side2, Side1)),
tell(user).

```

```

define_write_side(Side, Side_Description) :-
    write('write_side('), write(Side), write(') :-'),nl,write(' '),
    write('write(''), write(Side_Description), write(')').nl,
define_write_side_caps(Side, Side_Description) :-
    write('write_side_caps('), write(Side), write(') :-'),nl,
    write(' '),write('write(''), write(Side_Description), write(')').nl.

```

```

/***** GET NEW DIMENSION INFORMATION *****/
**
** Get information on one or more dimension, assert that information
** to the present prolog knowledge base, and write the information
** to a file.
**
*****/

```

```

get_new_dimension_info(File_Name) :-
    create_a_dimension(File_Name).nl,
    write('Would you like to enter another dimension?').nl,
    get_yes_or_no(Answer),
    if(Answer = yes,
        get_new_dimension_info(File_Name),
        true).

```

```

create_a_dimension(File_Name) :-
    home,
    write('Please enter the name of the Dimension. Use only small').nl,
    write('letters and connect words with underlines (_), not spaces.').nl,
    get_choice(Dimension_Name).nl,
    write('What side does '), write(Dimension_Name), write(' favor?').nl,
    get_ok_side(Side).nl,nl,
    write('How many positive magnitude values does this dimension have').nl,
    write('(do not include the "dimension did not occur" value)').nl,nl,
    get_a_number(Number_of_Values),
    create_dimension_values(Number_of_Values, Dimension_Values),
    nl,nl,write('Please enter a complete phrase which can be used to').nl,
    write('describe the dimension in argumentation.').nl,
    get_choice(Description),
    tell(File_Name),
    write('pro('),write(Dimension_Name), write(', '), write(Side),
    write(')').nl,

```

```

write('range('),write(Dimension_Name), write(',') ,nl, write('['),
write_dimension_values(Dimension_Values),write(']').'),nl,
write('describe('), write(Dimension_Name), write(',') , nl,
write('      '), write(Description), write(')').'),nl,nl,
tell(user).

```

```

/***** CREATE DIMENSION VALUES *****/

```

```

**
** Used by create-a-dimension to allow input of a set
** number of dimension magnitudes.
**
*****/

```

```

create_dimension_values(Num_of_Values, ['the dimension did not occur'|Rest]) :-
do_create_dimension_values(Num_of_Values, Rest).

```

```

do_create_dimension_values(0, []).
do_create_dimension_values(Number_of_Values, [Value|Tail]) :-
write('Enter the value of the dimension in single quotes'),nl,
get_choice(Value),nl,
Values_Left is Number_of_Values - 1,
do_create_dimension_values(Values_Left, Tail).

```

```

/***** CREATE DIMENSION VALUES *****/

```

```

**
** Used by create-a-dimension to output a set
** number of dimension magnitudes to the file.
**
*****/

```

```

write_dimension_values([Dimension_Value|[]]) :-
write(''),write(Dimension_Value), write('').
write_dimension_values([Dimension_Value|Tail]) :-
write(''),write(Dimension_Value), write(' ', ),nl,
write_dimension_values(Tail).

```

```

/***** NEW CASE *****/
**
** Adds a new case (or cases) to the present Prolog knowledge base, and
** allows the user to save the case(s) to a new file or append the case(s)
** to an existing file.
**
** The following information can make up the definition of a case:
** - holding(Case, Holding).*
** - shortname(Case, Short_Name).*
** - longname(Case, Long_Name).
** - citation(Case, Citation).*
** - year(Case, Year).*
** - has_dimension(Case, Dimension1, Size1).*
** - has_dimension(Case, Dimension2, Size2).
** - support(Case, Support_Case, Along_Dimension).
** - distinguishes(Case, Distinguished_Case, Along_Dimension).
** - overrule(Case, Overruled_Case).
**
** Information (prolog fact statements) followed by an asterix
** is necessary information, all others are optional.
**
** The purpose of the create-new-cases procedure is to double-
** check the users intention, call the get-new-cases procedure,
** and allow the user to save the cases to an external file.
**
*****/

```

```

create_new_cases :-
    home,
    write('Are you sure you want to enter a new case?'),nl,
    get_yes_or_no(Choice),
    if(Choice = no,
        fail,
        true),
    write_new_case_banner,
    get_new_cases,
    home,
    write_new_case_banner,
    write('Would you like to save these cases to a file?'),nl,
    get_yes_or_no(Answer2),
    if(Answer2 = no,
        fail,
        true),
    write('Please enter either the name of a file to which the cases'),nl,
    write('should be appended, or enter a new file name for the cases'),nl,
    get_file_name(File_Name),
    if(exists(File_Name),
        append_temp_to(File_Name),
        rename_temp_to(File_Name)),
    fail.
create_new_cases :-
    if(exists(temp),
        system("del temp.*;"),
        true).

```

```

/***** WRITE NEW CASE BANNER *****/
**
** Writes a banner telling user that currently a case is being entered,
** and sets scrolling region to protect the banner.
**

```

```
*****/
```

```
write_new_case_banner :-  
    screen_type(new_case), !, home.  
write_new_case_banner :-  
    narrow_screen,  
    set_screen_type(new_case),  
    set_window(1,24),  
    home,  
    domain(Domain),  
    wide, write('          New Cases'),nl,  
    wide, pp([' ', Domain, ' Domain']),nl,  
    set_window(4, 24).
```

```
/***** GET NEW CASES *****/
```

```
**  
** This procedure actually gets the new cases by calling create-  
** a-case. The case is then written to a temporary file, and the  
** user may choose to enter another case.  
**  
*****/
```

```
get_new_cases :-  
    home,  
    create_a_case(Case_Name),  
    write_case_to_temp_file(Case_Name),  
    write_new_case_banner,  
    write('Do you want to enter another case?'),nl,  
    get_yes_or_no(Answer),  
    if(Answer == yes,  
        get_new_cases,  
        close(temp)),!.
```

```
/***** CREATE A CASE *****/
```

```
**  
** Create a case asks the user for the information needed  
** to create a case, and asserts that information to  
** present Prolog knowledge base. In addition, if the  
** new case has been decided for one side or another (ie,  
** its holding is not unknown), then the analysis parameters  
** are reset to indicate that the generalization rules must  
** be reconsidered in order to take the new case into account.  
**  
*****/
```

```
create_a_case(Case) :-  
    write('Please input the name of the case (small letters, no spaces).'),  
    nl, get_choice(Case), !, nl,  
    write('Enter a short name for the case (using single quotes).'),nl,  
    get_choice(Short_Name), !, nl,  
    asserta(shortname(Case, Short_Name)),  
    write('Enter a long name for the case (using single quotes).'),nl,  
    get_choice(Long_Name), !, nl,  
    asserta(longname(Case, Long_Name)),nl,  
    write('For what side did the court in '),  
    write_short_name(Case),  
    write(' hold (if hypo, enter unknown).'),nl,
```

```

    get_ok_side(Side),
    if(Side == unknown,
        true,
        reset_analysis_parameters),
    asserta(holding(Case, Side)),
write('Please enter the citation (enter "n." for none if no cite exists).'),nl,
get_choice(Citation),!,nl,
if(Citation = n,
    asserta(citation(Case, unknown)),
    asserta(citation(Case, Citation))),
write('Please enter the year the case was decided in (n. if none).'),nl,
get_choice(Year),!,
if(Year = n,
    asserta(year(Case, unknown)),
    asserta(year(Case, Year))),
write('Next, the dimensions must be entered. '),nl,
if(screen_type(dimension_list),
    home,
    ready),
get_dimensions(Case),
write_new_case_banner,
get_support_and_distinguishes(Case).

```

```

/***** GET DIMENSIONS *****/

```

```

**
** Gets the dimensions for a case by writing out
** a dimension list and calling the dimension menu.
**
*****/

```

```

get_dimensions(Case) :-
    home,
    write_all_dimensions,
    dimension_menu(Case).

```

```

/***** DIMENSION MENU *****/

```

```

**
** Allows the user to assign one, several,
** or all dimensions to a case. Dimensions
** are referred to by an assigned dimension number.
**
*****/

```

```

dimension_menu(Case) :-
    home,
    write(' You have the following choices:'), nl,
    write(' 1> Assign a dimension and value to '),
    write_short_name(Case), write(','), nl,
    write(' 2> Assign every dimension to '),
    write_short_name(Case), write(','), nl,
    write(' 3> Ask for a definition of a dimension, '),nl,
    write(' 4> List Dimensions now assigned to '),
    write_short_name(Case), write(','), nl,
    write(' 5> Remove all dimensions currently assigned to '),
    write_short_name(Case), write(','), nl,
    write('or 6> End entering dimensions, '),nl,nl,
    get_choice(Choice),!,

```

```
do_dimension_menu(Case, Choice).
```

```
/** DIMENSION MENU CHOICE 1: ASSIGN A DIMENSION **  
**  
** Assigns the chosen dimension to the new case.  
** When a dimension is assigned, the procedure then  
** lists the possible magnitudes that the dimension may have  
** and then requires the user to select a value.  
**  
*****/
```

```
do_dimension_menu(Case, 1) :-  
    home,  
    write('Please enter the number of the dimension you wish to assign'),  
    write(' to '), write_short_name(Case), nl, nl,  
    get_a_number(Number),  
    dimension_number(Dimension, Number),  
    get_range(Dimension, Range_Number),  
    asserta(has_dimension(Case, Dimension, Range_Number)),  
    dimension_menu(Case).
```

```
do_dimension_menu(Case, 1) :-  
    write_message('No Dimension of that number.'),  
    do_dimension_menu(Case, 1).
```

```
get_range(Dimension, Range_Number) :-  
    range(Dimension, Range),  
    home,  
    write(Dimension),nl,  
    write('Enter the number assigned to the correct value for this dimension.'),  
    nl,write_all_ranges(Range, -1, Last_Number),  
    get_choice(Range_Number),  
    Range_Number =< Last_Number.  
get_range(Dimension, Range_Number) :-  
    write_message('No value of that number.'),  
    get_range(Dimension, Range_Number).
```

```
write_all_ranges([], Final_Number, Final_Number).  
write_all_ranges([Range|Tail], Previous_Number, Final_Number) :-  
    Current_Number is Previous_Number + 1,  
    write(' '), write(Current_Number), write('> '),  
    write(Range), nl,  
    write_all_ranges(Tail, Current_Number, Final_Number).
```

```
/** DIMENSION MENU CHOICE 2: ASSIGN ALL DIMENSIONS ***  
**  
** Assigns all the possible dimensions to the newcase. The  
** assigned dimensions are all given the value of one (1).  
**  
** This is useful in special, limited circumstances, so  
** before the action is completed the user is asked to  
** verify their choice.  
**  
*****/
```

```
do_dimension_menu(Case, 2) :-  
    home,  
    write('Are you sure you want to assign ALL DIMENSIONS to '),  
    write_short_name(Case), write('?'), nl,  
    get_yes_or_no(Answer),
```

```

        if(Answer == no,
            fail,
            true),
        pro(Dimension, _),
        asserta(has_dimension(Case, Dimension, 1)),
        fail.
do_dimension_menu(Case, 2) :- dimension_menu(Case).

/** DIMENSION MENU CHOICE 3: DESCRIBE A DIMENSION ***
**
** Describes a dimension chosen by the user.
**
*****/

do_dimension_menu(Case, 3) :-
    home,
    write('Please enter the number of the dimension you wish described.'),
    nl,
    get_a_number(Number),
    dimension_number(Dimension, Number),
    describe(Dimension, Description),nl,
    write(Description),nl,nl,
    ready,
    fail.
do_dimension_menu(Case, 3) :- dimension_menu(Case).

```

```

/** DIMENSION MENU CHOICE 4: LIST CASE'S DIMENSIONS ***
**
** Lists the dimensions and their values currently assigned
** to the new case.
**
*****/

do_dimension_menu(Case, 4) :-
    home,
    write('Dimensions assigned to '),
    write_short_name(Case),write(':'),nl,nl,
    write('Dimension Numbers (Range Number)> '),
    write_out_dimension_numbers(Case),nl,nl,
    ready,
    dimension_menu(Case).

```

```

write_out_dimension_numbers(Case) :-
    has_dimension(Case, Dimension, Size),
    dimension_number(Dimension, Number),
    pp([Number, '(', Size, ') ']),
    fail.
write_out_dimension_numbers(Case).

```

```

/** DIMENSION MENU CHOICE 5: REMOVE ALL ASSIGNED DIMENSIONS ***
**
** Allows the user to remove all currently assigned dimensions
** and start assigning dimensions again from scratch.
**
*****/

```

```

do_dimension_menu(Case, 5) :-
    home,
    write('Are you sure you want to REMOVE the dimensions assigned to '),
    write_short_name(Case), write('?'), nl,
    get_yes_or_no(Answer),
    if(Answer == no,
        fail,
        true),
    has_dimension(Case, Dimension, Size),
    retract(has_dimension(Case, Dimension, Size)),
    fail.
do_dimension_menu(Case, 5) :- dimension_menu(Case).

```

```

/** DIMENSION MENU CHOICE 6: END DIMENSION MENU **
**
** Return to creating the rest of the new case.
**
*****/

```

```

do_dimension_menu(Case, 6).

```

```

/** DIMENSION MENU INAPPROPRIATE CHOICE ***
**
**
**
*****/

```

```

do_dimension_menu(Case, _) :-
    inappropriate_response(New_Choice),
    do_dimension_menu(Case, New_Choice).

```

```

/***** GET SUPPORT AND DISTINGUISHES *****/
**
** Allows the user to assign zero or several relationships
** between the current cases and other cases presently in,
** or to be added to the casebase. Such relations are:
** - support
** - distinguishes
** - overrule.
**
*****/

```

```

get_support_and_distinguishes(Case) :-
    write_support_and_distinguishes_menu(Case),
    get_choice(Choice),
    do_support_and_distinguishes_menu(Choice, Case).

```

```

/***** WRITE SUPPORT/DISTINGUISH MENU ****
**
**
**
*****/

```

```

write_support_and_distinguishes_menu(Case) :-
    write_new_case_banner,
    shortname(Case, Case_Name),
    write('You may now enter any of the following relationships between'),

```

```

nl, write('this new case and other cases.').nl,nl,
pp([' 1> ', Case_Name, ' cites a case for support along a dimension,']),nl,
pp([' 2> ', Case_Name, ' distinguishes a case along a dimension,']),nl,
pp([' 3> ', Case_Name, ' overrules (at least partially) a case,']),nl,
write('or    4> Finish entering this case.').nl,nl.

```

```

/** SUPPORT/DISTINGUISH MENU CHOICE 1: SUPPORT **

```

```

**
** Asserts a case for support to the newcase. The
** case chosen need not be in the present casebase.
**
*****/

```

```

do_support_and_distinguishes_menu(1, Case) :-
    home,
    pp(['What is the case that is cited by ', Case, ' for support?']),nl,nl,
    get_choice(Support_Case),nl,
    pp(['You must now enter the dimension along which ', Support_Case,
        ' offers support.']),nl,
    ready,
    get_a_valid_dimension(Dimension),
    asserta(support(Case, Support_Case, Dimension)),
    get_support_and_distinguishes(Case).

```

```

/** SUPPORT/DISTINGUISH MENU CHOICE 2: DISTINGUISH **

```

```

**
** Asserts a case that is distinguished by the newcase. The
** case chosen need not be in the present casebase.
**
*****/

```

```

do_support_and_distinguishes_menu(2, Case) :-
    home,
    pp(['What is the case that is distinguished by ', Case, '?']),nl,nl,
    get_choice(Distinguished_Case),nl,nl,
    pp(['You must now enter the dimension along which ',
        Distinguished_Case, ' is distinguished.']),nl,nl,
    ready,
    get_a_valid_dimension(Dimension),
    asserta(distinguishes(Case, Distinguished_Case, Dimension)),
    get_support_and_distinguishes(Case).

```

```

/** SUPPORT/DISTINGUISH MENU CHOICE 3: OVERRULE **

```

```

**
** Asserts a case that is overruled the newcase. The
** case chosen need not be in the present casebase.
**
*****/

```

```

do_support_and_distinguishes_menu(3, Case) :-
    home,
    pp(['What is the case that is overruled by ', Case, '?']),nl,nl,
    get_choice(Overruled_Case),
    asserta(overrule(Case, Overruled_Case)),
    get_support_and_distinguishes(Case).

```

```
/** SUPPORT/DISTINGUISH MENU CHOICE 4: END **
```

```
**
```

```
** Finish entering information for a case.
```

```
**
```

```
*****/
```

```
do_support_and_distinguishes_menu(4, Case).
```

```
/** SUPPORT/DISTINGUISH MENU: INAPPROPRIATE RESPONSE **
```

```
**
```

```
**
```

```
**
```

```
*****/
```

```
do_support_and_distinguishes_menu(_, Case) :-
```

```
    inappropriate_response(New_Choice),
```

```
    do_support_and_distinguishes_menu(New_Choice, Case).
```

```
/****** WRITE CASE TO TEMP FILE *****/
```

```
**
```

```
** Writes the new case to a file named 'temp;'.  
** The information that has been entered for the case  
** is retrieved from the Prolog knowledge base.  
**
```

```
*****/
```

```
write_case_to_temp_file(Case_Name) :-
```

```
    tell(temp),
```

```
    citation(Case_Name, Citation),
```

```
    shortname(Case_Name, Short_Name),
```

```
    longname(Case_Name, Long_Name),
```

```
    year(Case_Name, Year),
```

```
    holding(Case_Name, Holding),
```

```
    pp(['citation(', Case_Name, ', ', '"', Citation, '"').']),nl,
```

```
    pp(['shortname(', Case_Name, ', ', '"', Short_Name, '"').']),nl,
```

```
    pp(['longname(', Case_Name, ', ', '"', Long_Name, '"').']),nl,
```

```
    pp(['year(', Case_Name, ', ', ' ', Year, ' ').']),nl,
```

```
    pp(['holding(', Case_Name, ', ', ' ', Holding, ' ').']),nl,
```

```
    write_dimension_to_temp_file(Case_Name),
```

```
    write_support_to_temp_file(Case_Name),
```

```
    write_distinguish_to_temp_file(Case_Name),
```

```
    write_overrule_to_temp_file(Case_Name),
```

```
    tell(user).
```

```
/** WRITE RELATIONSHIP CHARACTERISTICS TO TEMP **
```

```
**
```

```
** These procedures are separate from write case to temp file
```

```
** procedure because a single case may have zero to many of
```

```
** each of these relationships.
```

```
**
```

```
*****/
```

```
write_dimension_to_temp_file(Case) :-
```

```
    has_dimension(Case, Dimension, Size),
```

```
    pp(['has_dimension(', Case, ', ', ' ', Dimension, ', ', ' ', Size, ' ').']),nl,
```

```
    fail.
```

```
write_dimension_to_temp_file(Case).
```

```

write__support_to__temp_file(Case):-
    support(Case, Other__Case, Dimension),
    pp(['support(', Case, ', ', ', Other__Case, ', ', ', Dimension, ').']),nl,
    fail.
write__support_to__temp_file(Case).

```

```

write__distinguish_to__temp_file(Case):-
    distinguishes(Case, Other__Case, Dimension),
    pp(['distinguishes(', Case, ', ', ', Other__Case, ', ', ', Dimension, ').']),nl,
    fail.
write__distinguish_to__temp_file(Case).

```

```

write__override_to__temp_file(Case):-
    override(Case, Other__Case),
    pp(['override(', Case, ', ', ', Other__Case, ').']),nl,
    fail.
write__override_to__temp_file(Case).

```

```

/***** APPEND TEMP TO FILE *****/
**
** Appends the 'temp.:' file to a named file.
**
*****/

```

```

append__temp_to(File_Name):-
    name('append temp ', First_Part_of_Command_String),
    name(File_Name, File_Name_String),
    append(First_Part_of_Command_String, File_Name_String, Command_String),
    system(Command_String).

```

```

/***** RENAME TEMP TO FILE *****/
**
** Renames the 'temp.:' file to a chosed file name.
**
*****/

```

```

rename__temp_to(File_Name):-
    name('rename temp ', First_Part_of_Command_String),
    name(File_Name, File_Name_String),
    append(First_Part_of_Command_String, File_Name_String, Command_String),
    system(Command_String).

```

```

/***** ANALYZE DOMAIN *****/

```

```

**

```

```

** This procedure determines the satisfied Generalization Rules,
** and then asserts the satisfied rules for use by other procedures.
** Five types of generalization rules are discovered:
**   - Both-Sided dimension (not that useful by itself, but
**     it is used in the definition of other rules),
**   - Overwhelming dimension (this dimension only appears in cases
**     holding the same way that the dimension favors),
**   - Necessary dimension (this dimension is found in every case
**     in the case base which hold the same way that the dimension favors),
**   - Overwhelming-Dimension-Pair (where two both-sided dimensions
**     are overwhelming in nature when they appear together),
**   - Dominating-Dimension-Pair (where the first [both-sided]
**     dimension is overwhelming except in the presence of the second
**     dominating dimension, in which case it is ineffective),
**   - Presumption (defines the nature of the whole domain, where cases
**     exists favoring one side even though no dimensions exist in those
**     cases which favor that side).
**
** Because of the length of time often associated with completing a
** domain analysis, this procedure also prints out a 'Please Wait' message
** and an occasional period (.) to keep the user interested.
**
** If the analysis has already been done, the procedure simply returns true.
**
** The asserted rules and associated with the cases which provide support
** for the rule.

```

```

*****/

```

```

analyze_domain :-
    not(analysis_done),
    telling(File),
    tell(user),
    write('Analyzing the domain (please wait)'),
    assert_all_both_sided,
    write(' bs'),
    assert_presumption,
    write(' p'),
    assert_all_overwhelming,
    write(' o'),
    assert_all_necessary,
    write(' n'),
    assert_all_overwhelming_dimension_pairs,
    write(' op'),
    assert_all_dominated_dimension_pairs,
    write(' dp'),
    asserta(analysis_done),!,
    nl,write('--Done.'),nl,nl, tell(File).
analyze_domain.

```

```

/***** BOTH SIDED DIMENSIONS *****/

```

```

**

```

```

** Finds and asserts both-sided dimensions.

```

```

**

```

```

*****/

```

```

assert_all_both_sided :-

```

```

    pro(Dimension, Side),
    assert_both_sided(Dimension), fail.
assert_all_both_sided.

assert_both_sided(Dimension):-
    get_case_with_dimension(Case1, Dimension),
    holding(Case1, Side),
    get_opposite_side(Side, Opp_Side),
    get_case_with_dimension(Case2, Dimension),
    holding(Case2, Opp_Side),
    setof(Case, get_case_with_dimension(Case, Dimension), Cases),!,
    enough_support(Cases),
    asserta(both_sided(Dimension, Cases)),!.

```

```

/***** PRESUMPTION *****/
**
** Finds and asserts a presumption in the domain.
**
*****/

```

```

assert_presumption :-
    pro(Dimension, Side),
    get_opposite_side(Side, Opposite_Side),
    do_assert_presumption(Side),
    do_assert_presumption(Opposite_Side).

do_assert_presumption(Side) :-
    setof(Case, validating_presumption_case(Case, Side), Cases),
    get_opposite_side(Side, Opposite_side),
    not(validating_presumption_case(Any_Case, Opposite_side)),!,
    enough_support(Cases),
    asserta(presumption(Side, Cases)),!.
do_assert_presumption(Side).

validating_presumption_case(Case, Side) :-
    holding(Case, Side),
    no_dimensions_for_side(Case, Side).

no_dimensions_for_side(Case, Side) :-
    get_case_with_dimension(Case, Dimension),
    pro(Dimension, Side), !, fail.
no_dimensions_for_side(Case, Side).

```

```

/***** OVERWHELMING DIMENSIONS *****/
**
** Finds and asserts overwhelming dimensions.
**
*****/

```

```

assert_all_overwhelming :-
    pro(Dimension, Side),
    assert_overwhelming(Dimension), fail.
assert_all_overwhelming.

assert_overwhelming(Dimension) :-
    pro(Dimension, Side),
    get_opposite_side(Side, Opposite_Side),
    get_case_with_dimension(Case1, Dimension),

```

```

        holding(Case1, Opposite_Side),!,
        fail.
assert_overwhelming(Dimension) :-
    setof(Case, get_case_with_dimension(Case, Dimension), Cases),!,
    enough_support(Cases),
    asserta(overwhelming(Dimension, Cases)), !.

```

```

/***** NECESSARY DIMENSIONS *****/

```

```

**
** Finds and asserts all necessary dimensions
**
*****/

```

```

assert_all_necessary :-
    pro(Dimension, Side),
    not(presumption(Side, _)),
    assert_necessary(Dimension, Side), fail.
assert_all_necessary.

```

```

assert_necessary(Dimension, Side) :-
    holding(Case, Side),
    not(get_case_with_dimension(Case, Dimension)), !, fail.
assert_necessary(Dimension, Side) :-
    setof(Case, get_case_on_side(Case, Dimension), Cases),
    enough_support(Cases),
    asserta(necessary(Dimension, Side, Cases)).

```

```

/***** OVERWHELMING DIMENSION PAIR *****/

```

```

**
** Finds and asserts overwhelming-dimension-pairs
**
*****/

```

```

assert_all_overwhelming_dimension_pairs :-
    pro(Dimension1, Side),
    both_sided(Dimension1, _),
    pro(Dimension2, Side),
    not(Dimension1 = Dimension2),
    both_sided(Dimension2, _),
    assert_overwhelming_dimension_pair(Dimension1, Dimension2, Side), fail.
assert_all_overwhelming_dimension_pairs.

```

```

assert_overwhelming_dimension_pair(Dimension1, Dimension2, Side) :-
    not(overwhelming_dimension_pair(Dimension2, Dimension1, _)),
    no_opposite_cases(Dimension1, Dimension2, Side),
    setof(Case, appear_together(Dimension1, Dimension2, Case), Cases),!,
    write(' op'),
    enough_support(Cases),
    asserta(overwhelming_dimension_pair(Dimension1, Dimension2, Cases)),!.

```

```

/***** DOMINATED DIMENSION PAIR *****/

```

```

**
** Finds and asserts dominated-dimension-pairs.
**
*****/

```

```

assert_all_dominated_dimension_pairs :-
    both_sided(Dimension1, _),
    pro(Dimension1, Side1),
    get_opposite_side(Side1, Side2),
    pro(Dimension2, Side2),
    assert_dominated_dimension_pair(Dimension1, Dimension2, Side1, Side2),
    fail.
assert_all_dominated_dimension_pairs.

```

```

assert_dominated_dimension_pair(Dimension1, Dimension2, Side1, Side2) :-
    no_opposite_cases(Dimension1, Dimension2, Side2),
    all_opposing_cases_have(Dimension1, Dimension2, Side1),
    setof(Case, appear_together(Dimension1, Dimension2, Case), Cases),!,
    write(' dp'),
    enough_support(Cases),
    asserta(dominated_dimension_pair(Dimension1, Dimension2, Cases)),!.

```

```

all_opposing_cases_have(Dimension1, Dimension2, Side1) :-
    get_case_with_dimension(Case, Dimension1),
    get_opposite_side(Side1, Side2),
    holding(Case, Side2),
    not(get_case_with_dimension(Case, Dimension2)),!,fail.
all_opposing_cases_have(Dimension1, Dimension2, Side1).

```

/*****ANALYZE UTILITIES *****/

**

** These definitions are used by more than one of
 ** the analyze procedures.

**

*****/

```

get_case_with_dimension(Case, Dimension) :-
    get_dimension(Case, Dimension, Size),
    holding(Case, Side),
    Side \== unknown.

```

```

get_case_on_side(Case, Dimension) :-
    get_dimension(Case, Dimension, Size),
    holding(Case, Side),
    pro(Dimension, Side).

```

```

no_opposite_cases(Dimension1, Dimension2, Side) :-
    appear_together(Dimension1, Dimension2, Case),
    get_opposite_side(Side, Opp_Side),
    holding(Case, Opp_Side), !, fail.

```

```

no_opposite_cases(Dimension1, Dimension2, Side) :-
    appear_together(Dimension1, Dimension2, Case).

```

```

appear_together(Dimension1, Dimension2, Case) :-
    get_case_with_dimension(Case, Dimension1),
    get_case_with_dimension(Case, Dimension2).

```

```

enough_support(Cases) :-
    length(Cases, Length),
    support_needed(Support),
    not(Support > Length).

```

```
/****** WRITE DOMAIN ANALYSIS *****/
```

```
**  
** Prints out and explains the Generalization Rules which have  
** been found to exist in the domain.  
**  
*****/
```

```
write_domain_analysis :-  
    domain(Domain),  
    tell_current_support_level,  
    explain_presumption(Domain),  
    explain_necessary_dimensions(Domain),  
    explain_overwhelming_dimensions(Domain),  
    explain_overwhelming_dimension_pairs(Domain),  
    explain_dominated_dimension_pairs(Domain).
```

```
/****** TELL CURRENT SUPPORT LEVEL *****/
```

```
**  
** Tells the user the current support level that was used to  
** determine satisfied generalization rules.  
**  
*****/
```

```
tell_current_support_level :-  
    support_needed(Support),  
    write('This analysis was done when the required level of support'),nl,  
    write('in order for a generalization rule to be satisfied was'),nl,  
    pp([Support, ' cases.']),nl,nl.
```

```
/****** EXPLAIN PRESUMPTION *****/
```

```
**  
** Explains the implications of the presumption  
** and outputs supporting cases.  
**  
*****/
```

```
explain_presumption(Domain) :-  
    presumption(Side, Cases),  
    get_opposite_side(Side, Opposite_Side),  
    write('PRESUMPTION:'),nl,nl,  
    write('According to the cases, there is a presumption within the '),nl,  
    write(Domain),  
    write(' domain in favor of '), write_side(Side), write(' '),nl,nl,  
    write('This presumption means that this position is capable of winning'),nl,  
    write('a dispute on the merits without even a single dimension which'),nl,  
    write('supports that side. '),nl,nl,  
    write('SEE: '),nl,  
    tab(5), write_list_of_cases(Cases, sp),nl,nl,  
    write('There are no such cases on the '), write_side(Opposite_Side),  
    write(' position. '),nl,nl,nl,  
    explain_presumption(Domain).
```

```
/****** EXPLAIN NECESSARY DIMENSIONS *****/
```

```
**  
** Explains the implications of any necessary dimensions  
** and outputs the dimensions and supporting cases to the user.
```

**

***** /

```
explain_necessary_dimensions(Domain) :-
    necessary(A_Dimension, A_Side, Cases),
    nl, write('NECESSARY DIMENSIONS:'), nl,
    pp(['The following dimensions are special in the ', Domain, ' domain in']), nl,
    write('that every cases which holds for a particular side contains that'), nl,
    write('dimension. In other words, these dimensions appear to be necessary,'),
    nl, write('but not necessarily sufficient conditions for the court to hold'), nl,
    write('for the side that the dimension favors.'), nl, nl,
    write_necessary_dimensions.
explain_necessary_dimensions(Domain).
```

```
write_necessary_dimensions :-
    necessary(Dimension, Side, Cases),
    write(Dimension), nl,
    tab(3),
    write('Is a necessary dimension for '),
    write_side(Side), write(' '), nl,
    tab(3), write('Found in :'),
    write_list_of_cases(Cases, sp), nl, nl, fail.
write_necessary_dimensions.
```

/***** EXPLAIN OVERWHELMING DIMENSIONS *****/

**

** Explains the implications of the overwhelming dimensions
** and outputs supporting cases.

**

***** /

```
explain_overwhelming_dimensions(Domain) :-
    overwhelming(A_Dimension, Cases),
    nl, write('OVERWHELMING DIMENSIONS:'), nl,
    write('The following dimensions have special importance in the field of'), nl,
    write(Domain),
    write(' in that they appear only in cases which were decided in'), nl,
    write('favor of the side that the dimension favors. Such dimensions are'), nl,
    write('called overwhelming dimensions, and may be of crucial importance'), nl,
    write('in arguing a case or understanding how a case may be decided in'), nl,
    write('this area of the law.'), nl,
    write('The fact that a dimension is overwhelming is especially important'), nl,
    write('if it appears in a great number of cases. The following overwhelming'),
    nl, write('dimensions are therefore followed by the cases in which they are found.'),
    nl, nl, write_overwhelming_dimensions.
explain_overwhelming_dimensions(Domain).
```

```
write_overwhelming_dimensions :-
    overwhelming(Dimension, Cases),
    write(Dimension), nl,
    tab(3), write('Found in :'),
    write_list_of_cases(Cases, sp), nl, nl, fail.
write_overwhelming_dimensions.
```

/***** EXPLAIN OVERWHELMING DIMENSION PAIRS *****/

**

** Explains the implications of the Overwhelming-Dimension-Pairs
** and outputs supporting cases.

**

*****/

```
explain_overwhelming_dimension_pairs(Domain):-
    overwhelming_dimension_pair(Dimension1, Dimension2, Cases),
    nl, write('OVERWHELMING DIMENSION PAIRS:'), nl,
    write('Sometimes two dimensions in a domain are closely linked together'), nl,
    write('in that when they appear together, they have more persuasive power'), nl,
    write('than if they appeared alone. In this analysis, dimension pairs'), nl,
    write('that overwhelmed other dimensions (in that two dimensions)'), nl,
    write('that favor the same side always "win-out" whenever they are'), nl,
    write('together) are considered examples of such closely linked'), nl,
    write('dimensions. In the '), write(Domain),
    write(' domain, the following dimensions are overwhelming pairs:'), nl, nl,
    write_overwhelming_dimension_pairs, nl, nl.
explain_overwhelming_dimension_pairs(Domain).
```

```
write_overwhelming_dimension_pairs:-
    overwhelming_dimension_pair(Dimension1, Dimension2, Cases),
    write(Dimension1), nl,
    write('Paired with : '), write(Dimension2), nl,
    tab(3), write('SEE: '), write_list_of_cases(Cases, sp), nl, nl, fail.
write_overwhelming_dimension_pairs.
```

/***** EXPLAIN DOMINATED DIMENSION PAIRS *****/

**

** Explains the implications of the Dominated-Dimension-Pairs
** and outputs supporting cases.

**

*****/

```
explain_dominated_dimension_pairs(Domain):-
    dominated_dimension_pair(Dimension1, Dimension2, Cases),
    nl, write('DOMINATED DIMENSION PAIRS:'), nl,
    write('Certain dimension may be overwhelming in all cases except when the'), nl,
    write('dimension occurs with a particular, opposite-sided dimensions. '), nl,
    write('This situation may indicate a peculiar connection between the two'), nl,
    write('dimensions, where one dimension may be very persuasive except'), nl,
    write('in the presence of another, dominating dimension. If one dimension'),
    nl, write('is found to be dominating over numerous other dimensions, then'), nl,
    write('it may be that there is not a particularly strong relationship'), nl,
    write('between the dimensions, but rather that that dimension is (or is)'), nl,
    write('nearly) overwhelming. '), nl,
    write('In the '), write(Domain),
    write(' domain, the following dimensions are dominated pairs:'), nl, nl,
    write_dominating_dimension_pairs.
explain_dominated_dimension_pairs(Domain).
```

```
write_dominating_dimension_pairs:-
    dominated_dimension_pair(Dimension1, Dimension2, Cases),
    write(Dimension1), write(' is dominated'), nl,
    tab(3), write('by '), write(Dimension2), nl,
    tab(3), write('SEE: '), write_list_of_cases(Cases, sp), nl, nl, fail.
write_dominating_dimension_pairs.
```

```
/****** ARGUE *****/
```

```
**
```

```
** Argue does the 3-ply argumentation output as described in Hypo.  
** The procedure is given the Best Cases (Argument_Cases) for the  
** side for which it is arguing (Side1), as well as the most-on-  
** point favoring and opposing (aka Against_Cases) Side1. The  
** most_on_point favoring cases are used in the argue-presumption  
** procedure, which enables it to cite to similar cases [if they  
** exists] in support of the presumption.  
**
```

```
**
```

```
** Argue not only argues cases in a Hypo manner, but also includes  
** procedures to argue by generalization rules if the  
** include_generalization_rules_in_arguments parameter has been  
** set in the parameters menu of the interface.  
**
```

```
**
```

```
*****/
```

```
argue(CFS, [], M_O_P_Favoring_Cases, Against_Cases, Side1) :-  
    include_generalization_rules_in_arguments,!,  
    argue_presumption(CFS, M_O_P_Favoring_Cases, Side1).  
argue(CFS, [], M_O_P_Favoring_Cases, Against_Cases, Side1) :-  
    !,write('No argument can be made for '),  
    write_side(Side1), write('.') ,nl,nl.  
argue(CFS, Argument_Cases, M_O_P_Favoring_Cases, Against_Cases, Side1) :-  
    do_argue(CFS, Argument_Cases, Against_Cases, Side1).
```

```
do_argue(CFS, [], Against_Cases, Side1).
```

```
do_argue(CFS, [[Cases, Dimensions]]Tail, Against_Cases, Side1) :-  
    nl,nl,nl,  
    write('+++++'),nl,  
    write('---> ARGUMENT FOR '),  
    write_side_caps(Side1), write('.') ,nl,  
    write('+++++'),nl,  
    nl, write('Cases suggest that the presence of the following factors'),  
    write(' should lead a'),nl,  
    write('court to hold in favor of '), write_side(Side1),  
    write('.') , nl, nl,  
    write_list_of_dimensions(Dimensions, Side1), nl,  
    write('SEE:'),nl,  
    write_list_of_cases(Cases, 5, nl),nl,!,  
    get_opposite_side(Side1, Side2),  
    if(include_generalization_rules_in_arguments,  
        argue_rules(CFS, Dimensions, Side1),  
        true),  
    respond(CFS, Cases, Dimensions, Against_Cases, Response_Cases, Side2),  
    write('+++++'),nl,  
    write('---> REBUTTAL FOR '),  
    write_side_caps(Side1), write('.') ,nl,  
    write('+++++'),nl,  
    nl,answer_response(Response_Cases, CFS, Side1),  
    do_argue(CFS, Tail, Against_Cases, Side1).
```

```
/****** ARGUE WRITING PROCEDURES *****/
```

```
**
```

```
**
```

```
**
```

```
*****/
```

```

write_list_of_dimensions(Dimensions, Side) :-
    dimensions_supporting(Dimensions, Supporting_Dimensions, Side),
    dimensions_opposing(Dimensions, Opposing_Dimensions, Side),
    write_dimension_list(Supporting_Dimensions, 5),nl,
    write_opposing(Opposing_Dimensions).

dimensions_supporting([], [], Side).
dimensions_supporting([Dimension|Tail], [Dimension|New_Tail], Side) :-
    pro(Dimension, Side),
    dimensions_supporting(Tail, New_Tail, Side).
dimensions_supporting([Dimension|Tail], New_Tail, Side) :-
    dimensions_supporting(Tail, New_Tail, Side).

dimensions_opposing([], [], Side).
dimensions_opposing([Dimension|Tail], [Dimension|New_Tail], Side) :-
    not(pro(Dimension, Side)),
    dimensions_opposing(Tail, New_Tail, Side).
dimensions_opposing([Dimension|Tail], New_Tail, Side) :-
    dimensions_opposing(Tail, New_Tail, Side).

write_dimension_list([], _).
write_dimension_list([Dimension|_], Tab_Amount) :-
    describe(Dimension, Description),
    tab(Tab_Amount), write(Description), write(' '),nl,
    write_dimension_list(Tail, Tab_Amount).
write_dimension_list([Dimension|Tail], Tab_Amount) :-
    describe(Dimension, Description),
    tab(Tab_Amount), write(Description), write(' '),nl,
    write_dimension_list(Tail, Tab_Amount).

write_opposing([]).
write_opposing(Opposing_Dimensions) :-
    write('Even Though:'),nl,
    write_dimension_list(Opposing_Dimensions, 5),nl.

/***** RESPOND *****/
**
** Respond responds to the Argue_cases by distinguishing
** them and citing counter examples. Rules that might
** have been cited in favor of Side1 are not responded to,
** as no cases in the case_base could refute a generalization
** rule. Contrary generalization rules also cannot exist
** for this particular set of dimensions, for if they did,
** the rules would be in contradiction with the Argue_cases.
**
** The only additional feature added beyond HYPO's algorithm is a very
** short procedure which makes sure that none of the cited
** cases have been overruled by a case in the casebase.
**
*****/

respond(CFS, Side1_Cases, Dimensions, Against_Cases, Side2_Cases, Side2) :-
    write('-----'),nl,
    write('<--- RESPONSE FOR '), write_side_caps(Side2), write(' '), nl,
    write('-----'),nl,
    nl, check_overrule(Side1_Cases),
    distinguish(Side1_Cases, CFS, Side2),nl,nl,
    find_counter_examples(Dimensions, Against_Cases, Counter_Exs),
    write_counter_examples(Counter_Exs, Dimensions, Side2, Side2_Cases).

```

```
/**** CHECK FOR OVERRULING CASES ***
```

```
**  
**  
**
```

```
*****/
```

```
check_ouerrule([]).
```

```
check_ouerrule([Case|Tail]) :-
```

```
    ouerrule(Ouerruling_Case, Case),  
    write_short_name(Case), write(' has been ouerruled by '),  
    write_short_name(Ouerruling_Case), write('.') ,nl,nl,  
    check_ouerrule(Tail).
```

```
check_ouerrule([Case|Tail]) :-
```

```
    check_ouerrule(Tail).
```

```
/****** FIND COUNTER EXAMPLES *****/
```

```
**
```

```
** Overlapping counter examples are only used if the parameter  
** use_overlapping_counter_examples has been set in the set parameter  
** menu of the interface.  
**
```

```
*****/
```

```
find_counter_examples(Dimensions, Against_Cases, Counter_Exs) :-
```

```
    more_o_p_counter_examples(Dimensions, Against_Cases, M_O_P_Counter_Exs),  
    if(use_overlapping_counter_examples,  
        overlap_counter_examples(Dimensions,  
                                Against_Cases,  
                                Counter_Exs,  
                                M_O_P_Counter_Exs),  
        Counter_Exs = M_O_P_Counter_Exs).
```

```
/****** MORE ON POINT COUNTER EXAMPLES *****/
```

```
**
```

```
** More on point counter examples are found by looking at the Against-  
** Case List (which is a collection of the Most On Point Cases favoring  
** Side-2), and looking for instances where the Dimension list of the  
** currently argued Side-1 cases are a sublist of the Against Cases'  
** Dimension list.  
**
```

```
*****/
```

```
more_o_p_counter_examples(Dims, [], []).
```

```
more_o_p_counter_examples(Dims, [[Case_List, Dim_List]|Tail], [Case_List,  
                                                                Dim_List]) :-
```

```
    sublist(Dims, Dim_List),!
```

```
more_o_p_counter_examples(Dims, [[Case_List, Dim_List]|Tail], Counter_Exs) :-
```

```
    more_o_p_counter_examples(Dims, Tail, Counter_Exs).
```

```
/****** OVERLAPPING COUNTER EXAMPLES *****/
```

```
**
```

```
** If more-on-point counter examples exists, then do not try to find  
** overlapping counter examples and set Counter_Exs to the list of  
** more-on-point counter examples. If they do not, then set Counter_Exs  
** to the list of overlapping counter examples.
```

```

**
** Overlapping counter examples are found by finding cases in the
** Against-Case list whose dimension list is a subset of the dimensions
** of the Side-1 cases currently being argued.
**
*****/

overlap_counter_examples(Dims, [], [], []).
overlap_counter_examples(Dims, [[Cases,Dim_List]]Tail],
    [[Cases,Dim_List]]New_Tail], []) :-
    intersection(Dims, Dim_List, Dims_in_Common),
    Dims_in_Common \== [],
    overlap_counter_examples(Dims, Tail, New_Tail, []), !.
overlap_counter_examples(Dims, [[Cases,Dim_List]]Tail],
    New_Tail, []) :-
    overlap_counter_examples(Dims, Tail, New_Tail, []), !.
overlap_counter_examples(Dims, Cases, M_O_P_Counter_Exs, M_O_P_Counter_Exs).

/****** WRITE COUNTER EXAMPLES *****/
**
** Different writing subprocedures are used depending on the type
** of counter example currently being written (ie, more-on-point,
** equally-on-point, overlapping).
**
*****/

write_counter_examples([], Dimensions, Side2, Case_List) :-
    write('No counter-examples.').nl, nl, !.
write_counter_examples([Case_List, Dim_List], Dimensions,
    Side2, Case_List) :-
    sublist(Dim_List, Dimensions),
    sublist(Dim_List, Dimensions),
    write('COUNTER-EXAMPLES:').nl,nl,
    write_the_cases(Case_List, Side2), nl, !.
write_counter_examples([[Case_List, Dim_List]]Tail], Dimensions,
    Side2, Case_List) :-
    sublist(Dim_List, Dimensions),
    write('COUNTER-EXAMPLES:').nl,nl,
    do_write_counter_examples([[Case_List, Dim_List]]Tail], Dimensions,
        Side2, Case_List), nl, !.
write_counter_examples([Case_List, Dim_List], Dimensions, Side2, Case_List) :-
    write('COUNTER-EXAMPLES:').nl,nl,
    get_extras(Dim_List, Dimensions, Extra_Dimensions),
    write_the_cases(Case_List, Side2, Extra_Dimensions, more_on_point), nl.

do_write_counter_examples([], Dimensions, Side2, Case_List).
do_write_counter_examples([[A, B]]C], Dimensions, Side2, Case_List) :-
    intersection(B, Dimensions, Common_Dimensions),
    write_the_cases(A, Side2, Common_Dimensions, overlapping), nl,
    do_write_counter_examples(C, Dimensions, Side2, Case_List).

write_the_cases([], Side2).
write_the_cases([Case|Tail], Side2) :-
    write_long_name(Case), write(' is as on point and held for the').nl,
    write('pro '),
    write(Side2), write(' position.').nl,
    write_the_cases(Tail, Side2).

write_the_cases([], Side2, Extra_Dimensions, _).

```

```

write_the_cases([Case|Tail], Side2, Extra_Dimensions, more_on_point) :-
    write_long_name(Case), write(' is more on point and held for the '),nl,
    write('pro '),
    write(Side2), write(' position, where it was also true that:'),nl,
    write_dimension_list(Extra_Dimensions, 5),
    write_the_cases(Tail, Side2, Extra_Dimensions, more_on_point).
write_the_cases([Case|Tail], Side2, Dim_List, overlapping) :-
    write_long_name(Case), write(' is not as on point, but it held for'),nl,
    write('the pro '),
    write(Side2), write(' position, where it was the case that:'),nl,
    write_dimension_list(Dim_List, 5),
    write_the_cases(Tail, Side2, Dim_List, overlapping).

```

```

/***** GET EXTRAS *****/

```

```

**
** Used to get extra dimension found in more-on-
** point counter examples.
**
*****/

```

```

get_extras([], List, []).
get_extras([Element|Tail], List, NewTail) :-
    member(Element, List),
    get_extras(Tail, List, NewTail).
get_extras([Element|Tail], List, [Element|NewTail]) :-
    get_extras(Tail, List, NewTail).

```

```

/***** ANSWER RESPONSE *****/

```

```

**
** The only rebuttal response is to distinguish any
** counter-examples that were cited.
**
*****/

```

```

answer_response([], CFS, Side1) :-
    write('None'),nl,!.
answer_response(Response_Cases, CFS, Side1) :-
    distinguish(Response_Cases, CFS, Side1).

```

```

/***** DISTINGUISH *****/

```

```

**
** First the CFS is distinguished from a case along one
** Side, and then the case is distinguished from the CFS
** along the Opposite Side, using the same procedure.
**
** Do-Distinguish searches for both additional dimensions
** favoring a side, and also greater values in common
** dimensions which favor that side.
**
*****/

```

```

distinguish([], CFS, Side).
distinguish([Case|Tail], CFS, Side) :-
    do_distinguish(CFS, Case, Side, Add_Dims1, Greater_Dims1),
    get_opposite_side(Side, Opp_Side),
    do_distinguish(Case, CFS, Opp_Side, Add_Dims2, Greater_Dims2),

```

```

write_distinguish(CFS, Case, Add_Dims1, Add_Dims2, Greater_Dims1,
Greater_Dims2),
distinguish(Tail, CFS, Side).

do_distinguish(Case1, Case2, Side, Add_Dims, Greater_Dims) :-
    additional_positive_dims(Case1, Case2, Side, Add_Dims),
    greater_positive_dims(Case1, Case2, Side, Greater_Dims).

```

```

/** DISTINGUISH SUB-PROCEDURES **
**
**
**
*****/

```

```

additional_positive_dims(Case1, Case2, Side, Dim_List) :-
    setof(Dimension,
        addition_positive_dim(Case1, Case2, Side, Dimension),
        Dim_List), !.
additional_positive_dims(Case1, Case2, Side, []).

```

```

addition_positive_dim(Case1, Case2, Side, Dimension) :-
    get_dimension(Case1, Dimension, X),
    pro(Dimension, Side),
    not(get_dimension(Case2, Dimension, Y)).

```

```

greater_positive_dims(Case1, Case2, Side, Dim_List) :-
    setof(Dimension,
        greater_positive_dim(Case1, Case2, Side, Dimension),
        Dim_List), !.
greater_positive_dims(Case1, Case2, Side, []).

```

```

greater_positive_dim(Case1, Case2, Side, Dimension) :-
    get_dimension(Case1, Dimension, Size1),
    get_dimension(Case2, Dimension, Size2),
    pro(Dimension, Side),
    Size1 > Size2.

```

```

/** DISTINGUISH WRITING PROCEDURES **
**
** These procedures write out the distinctions that
** can be made. In the write-additional-dimensions
** procedure, a tag value if passed which explains when
** additional dimension should be checked for a
** Necessary rule involving the dimension.
**
** This is done because it is not always required to
** check this rule. In fact, if the rule is used in
** other situations, then the resulting argument would
** be nonsensical.
**
** The Necessary rule is used only when the current fact
** situation is distinguished from a support case or
** a counter-example
**
*****/

```

```

write_distinguish(CFS, Case, [], [], [], []) :-
    write(''), write_short_name(Case),

```

```

        write(' is not distinguishable from '),
        write_short_name(CFS), write(').'),nl,nl.
write__distinguish(CFS, Case,Add_Dims1, Add_Dims2, Greater_Dims1,Greater_Dims2):-
    write_short_name(Case),
    write(' is distinguishable from '),
    write_short_name(CFS),
    write(' because:'),nl,
    write_additional_dims(CFS, Case, Add_Dims1, do_not_check),
    write_additional_dims(Case, CFS, Add_Dims2, check_for_necessary),
    write_greater_dims(CFS, Case, Greater_Dims1),
    write_greater_dims(Case, CFS, Greater_Dims2).

write_additional_dims(Case1, Case2, [], Tag).
write_additional_dims(Case1, Case2, [Dimension|Tail], Tag) :-
    write_additional_dim(Case1, Case2, Dimension, Tag),
    write_additional_dims(Case1, Case2, Tail, Tag).

write_additional_dim(Case1, Case2, Dimension, Tag) :-
    tab(3), write('In '), write_short_name(Case1), write(', '),
    write_dimension_description(Dimension), write('.'), nl,
    tab(3), write('This was not true in '), write_short_name(Case2),
    if(has_dimension(Case2, Dimension, 0),
        write(' as was directly stated in the case.'),
        write('.')'),nl,
    if(involves_necessary(Dimension, Case1, Case2, Tag),
        write_necessary_distinction(Dimension,
            true),
    if(distinguishes(A_Case, Another_Case, Dimension),
        write_important_distinction(Dimension,
            nl).

write_greater_dims(Case1, Case2, []).
write_greater_dims(Case1, Case2, [Dimension|Tail]) :-
    write_greater_dim(Case1, Case2, Dimension),
    write_greater_dims(Case1, Case2, Tail).

write_greater_dim(Case1, Case2, Dimension) :-
    get_dimension(Case1, Dimension, Size1),
    get_dimension(Case2, Dimension, Size2),
    tab(3), write('In both cases, '),
    write_dimension_description(Dimension), write('.'),nl,
    tab(3), write('However, in '), write_short_name(Case1), write(', '),
    write_range(Dimension, Size1), nl,
    tab(3), write('while in '), write_short_name(Case2), write(', '),
    write_range(Dimension, Size2), write('.'), nl, nl.

/***** INVOLVES NECESSARY *****/
**
** Determines whether the dimension being distinguished as
** an additional positive dimension is of the type where it
** is important to point out that the dimension is a
** necessary dimension according to the generalization rules.
**
*****/

involves_necessary(Dimension, Case1, Case2, Tag) :-
    Tag == check_for_necessary,
    include_generalization_rules_in_arguments,
    necessary(Dimension, Side, _),

```

```
holding(Case1, Side),
holding(Case2, Side).
```

```
write_necessary_distinction(Dimension) :-
    necessary(Dimension, Side, Cases),
    tab(3),
    write('According to the cases in the case base, this distinction is'),
    nl, tab(3),
    pp(['crucial as no case has found in favor of', Side, ' without']), nl,
    tab(3), write('the presence of this dimension. '), nl,
    tab(7), write('SEE: '),
    write_list_of_cases(Cases, sp), nl, nl.

write_important_distinction(Dimension) :-
    tab(3),
    write('Courts have acknowledged the importance of this distinction. '),
    nl, tab(7), write('SEE: '),
    write_cases_distinguishing_dimension(Dimension).

write_cases_distinguishing_dimension(Dimension) :-
    distinguishes(Case1, Case2, Dimension),
    write_short_name(Case1), write(' (distinguishing '),
    write_short_name(Case2), write(')'), nl, tab(12), fail.
write_cases_distinguishing_dimension(Dimension) :- nl.
```

```
/****** ARGUE WITH RULES *****/
**
** Arguing with rules involves finding relevant rules that have
** been asserted by the analyze_domain procedure. Rules are relevant
** if they rely only on the dimensions that the current Argue_Cases
** share with the CFS. Since all dimension combinations that the
** CFS shares with cases in the casebase will be found in the
** Aruge_Cases (as can be assured by the selection of most_on_point
** cases), there is no danger of a relevant rule not being proposed
** during an argument run.
**
*****/
```

```
argue_rules(CFS, Dims, Side) :-
    find_overwhelming_dim_rules(CFS, Dims, O_Ds, Side),
    find_overwhelming_dim_pair_rules(CFS, Dims, O_D_Ps, Side),
    find_dominated_dim_pair_rules(CFS, Dims, D_D_Ps, Side),
    do_rule_introduction(O_Ds, O_D_Ps, D_D_Ps),
    argue_overwhelming_dim_rules(O_Ds, Side),
    argue_overwhelming_dim_pair_rules(O_D_Ps, Side),
    argue_dominated_dim_pair_rules(D_D_Ps, Side).
```

```
/****** FIND OVERWHELMING DIMENSION RULES *****/
**
**
**
*****/
```

```
find_overwhelming_dim_rules(CFS, [], [], Side).
find_overwhelming_dim_rules(CFS, [Dim|Tail], [Dim|New_Tail], Side) :-
    overwhelming(Dim, Cases),
    pro(Dim, Side),
    find_overwhelming_dim_rules(CFS, Tail, New_Tail, Side).
```

```

find_overwhelming_dim_rules(CFS, [Dim|Tail], New_Tail, Side) :-
    find_overwhelming_dim_rules(CFS, Tail, New_Tail, Side).

```

```

/***** FIND OVERWHELMING DIMENSION PAIR RULES *****/
**
**
**
*****/

```

```

find_overwhelming_dim_pair_rules(CFS, Dims, O_D_Ps, Side):-
    setof([Dim1, Dim2],
        included_overwhelming_dimension_pairs(Dim1, Dim2, Dims, Side),
        O_D_Ps).
find_overwhelming_dim_pair_rules(CFS, Dimensions, [], Side).

```

```

included_overwhelming_dimension_pairs(Dim1, Dim2, Dims, Side) :-
    overwhelming_dimension_pair(Dim1, Dim2, Cases),
    member(Dim1, Dims),
    member(Dim2, Dims),
    pro(Dim1, Side).

```

```

/***** FIND DOMINATED DIMENSION PAIR RULES *****/
**
**
**
*****/

```

```

find_dominated_dim_pair_rules(CFS, Dims, D_D_Ps, Side) :-
    setof([Dim1, Dim2],
        included_dominated_dimension_pairs(Dim1, Dim2, Dims, Side),
        D_D_Ps).
find_dominated_dim_pair_rules(CFS, Dims, [], Side).

```

```

included_dominated_dimension_pairs(Dim1, Dim2, Dims, Side) :-
    dominated_dimension_pair(Dim1, Dim2, Cases),
    member(Dim1, Dims),
    member(Dim2, Dims),
    pro(Dim2, Side).

```

```

/***** RULE INTRODUCTION *****/
**
**
**
*****/

```

```

do_rule_introduction([], [], []).
do_rule_introduction(O_Ds, O_D_Ps, D_D_Ps) :-
    write('In addition, cases in the casebase show that the following'),nl,
    write('rules support this conclusion:'),nl,nl.

```

```

/***** ARGUE OVERWHELMING DIMENSION RULES *****/
**
**
**
*****/

```

```

argue_overwhelming_dim_rules([], Side).
argue_overwhelming_dim_rules([O_D|Tail], Side) :-
    write('RULE: All known cases where '),nl,
    tab(7), write_dimension_description(O_D), nl,
    tab(7), write('have held in favor of '),
    write_side(Side), write('.'), nl, nl,
    argue_overwhelming_dim_rules(Tail, Side).

/***** ARGUE OVERWHELMING DIMENSION PAIR RULES *****/
**
**
**
*****/

argue_overwhelming_dim_pair_rules([], Side).
argue_overwhelming_dim_pair_rules([[Dim1, Dim2]|Tail], Side) :-
    write('RULE: In all cases where '),nl,
    tab(7), write_dimension_description(Dim1), write(' as well as '),nl,
    tab(7), write_dimension_description(Dim2), write(', the court'),nl,
    tab(7), write('has ruled in favor of '), write_side(Side), write('.'),nl,nl,
    argue_overwhelming_dim_pair_rules(Tail, Side).

/***** ARGUE DOMINATED DIMENSION PAIR RULES *****/
**
**
**
*****/

argue_dominated_dim_pair_rules([], Side).
argue_dominated_dim_pair_rules([[Dim1, Dim2]|Tail], Side) :-
    write('RULE: According to the cases, the fact that '),nl,
    tab(7), write_dimension_description(Dim1), nl,
    tab(7), write('is not important where it is also true that '),nl,
    tab(7), write_dimension_description(Dim2), write('.'),nl,nl,
    argue_dominated_dim_pair_rules(Tail, Side).

/***** ARGUE PRESUMPTION *****/
**
** Argue presumption is different than arguing with
** other generalization rules as the presumption rule
** is invoked only when the best case algorithm has
** ruled out every case because no supporting dimensions
** are shared with the CFS.
**
** If a presumption exists in the domain, than Argue
** presumption explains how the presumption effects
** the current argument. A list of cases which support
** the presumption rule is then cited for support. This
** list is the intersection of the total number of cases
** supporting the presumption and the most_on_point
** cases supporting side-1. If there are no cases in
** the intersection, then every case supporting the
** presumption is outputted for support.
**
*****/

```

```

argue_presumption(CFS, M_O_P_Favoring_Cases, Side1) :-
    presumption(Side1, Cases),
    domain(Domain),
    nl,nl,nl,
    write('+++++'),nl,
    write('---> ARGUMENT FOR '),
    write_side_caps(Side1), write(':'),nl,
    write('+++++'),nl,
    nl,write('According to the cases, there is a presumption in the'),nl,
    write(Domain), write(' domain in favor of '),write_side(Side1),
    write(''),nl,
    write('Thus, even though no dimensions exist in '),
    write_short_name(CFS), write(' in favor of '),nl,
    write_side(Side1), write(', the presumption that this'),
    nl,write('position normally prevails must be overcome.'),nl,nl,
    write('SEE GENERALLY:'),nl,
    write_presumption_cases(Cases, M_O_P_Favoring_Cases),nl,
    write('-----'),nl,
    write('<--- RESPONSE FOR '), write_side_caps(Side2), write(':'), nl,
    write('-----'),nl,
    nl, write('None.'),nl,nl.

argue_presumption(CFS, M_O_P_Favoring_Cases, Side1) :-
    write('No argument can be made for '),
    write_side(Side1), write(''),nl,nl.

write_presumption_cases(Cases, M_O_P_Favoring_Cases) :-
    get_overlap_cases(Cases, M_O_P_Favoring_Cases, [], Overlap_Cases),
    do_write_presumption_cases(Cases, Overlap_Cases).

get_overlap_cases(Cases, [], Overlap_Cases, Overlap_Cases).
get_overlap_cases(Cases, [[M_O_P_Cases, Dims]|Tail], Temp, Overlap_Cases):-
    intersection(Cases, M_O_P_Cases, Temp_Cases),
    append(Temp_Cases, Temp, New_Temp),
    get_overlap_cases(Cases, Tail, New_Temp, Overlap_Cases).

do_write_presumption_cases(Cases, []) :-
    write_list_of_cases(Cases, sp),nl,nl.
do_write_presumption_cases(Cases, Overlap_Cases) :-
    write_list_of_cases(Overlap_Cases, 5, nl),nl.

```

```
/****** PROCEDURES USED TO IMPLEMENT ARGUMENTATION *****/
```

```
**
```

```
** These are the basic procedures used to implement the Argue procedure.
```

```
** Included in these procedures are the calling procedures used to call
```

```
** Argue, and the Most-On-Point and Best-Cases procedures.
```

```
**
```

```
*****/
```

```
/****** ANALYZE *****/
```

```
**
```

```
** Calls argue for both sides of a cases after finding
```

```
** the most-on-point and best-cases for each side.
```

```
**
```

```
*****/
```

```
analyze(Case, Side, Opp_Side) :-
```

```
    home,
```

```
    supporting_cases_with_dimensions(Case, Sup_Cases),
```

```
    opposing_cases_with_dimensions(Case, Opp_Cases),
```

```
    m_o_p(Case, Sup_Cases, M_O_P_Sup_Cases),
```

```
    m_o_p(Case, Opp_Cases, M_O_P_Opp_Cases),
```

```
    best_cases(M_O_P_Sup_Cases, Best_Sup_Cases, Side),
```

```
    best_cases(M_O_P_Opp_Cases, Best_Opp_Cases, Opp_Side),
```

```
    argue(Case, Best_Sup_Cases, M_O_P_Sup_Cases, M_O_P_Opp_Cases, Side),
```

```
    argue(Case, Best_Opp_Cases, M_O_P_Opp_Cases, M_O_P_Sup_Cases, Opp_Side).
```

```
/****** ANALYZE SIDE *****/
```

```
**
```

```
** Calls the Argue procedure when only
```

```
** one side of the argument is desired.
```

```
**
```

```
*****/
```

```
analyze_side(Case, Side, Opp_Side) :-
```

```
    holding(Case, Side),
```

```
    home,
```

```
    supporting_cases_with_dimensions(Case, Sup_Cases),
```

```
    opposing_cases_with_dimensions(Case, Opp_Cases),
```

```
    m_o_p(Case, Sup_Cases, M_O_P_Sup_Cases),
```

```
    m_o_p(Case, Opp_Cases, M_O_P_Opp_Cases),
```

```
    best_cases(M_O_P_Sup_Cases, Best_Sup_Cases, Side),
```

```
    argue(Case, Best_Sup_Cases, M_O_P_Sup_Cases, M_O_P_Opp_Cases, Side).
```

```
analyze_side(Case, Side, Opp_Side) :-
```

```
    holding(Case, Opp_Side),
```

```
    home,
```

```
    supporting_cases_with_dimensions(Case, Sup_Cases),
```

```
    opposing_cases_with_dimensions(Case, Opp_Cases),
```

```
    m_o_p(Case, Sup_Cases, M_O_P_Sup_Cases),
```

```
    m_o_p(Case, Opp_Cases, M_O_P_Opp_Cases),
```

```
    best_cases(M_O_P_Opp_Cases, Best_Opp_Cases, Side),
```

```
    argue(Case, Best_Opp_Cases, M_O_P_Opp_Cases, M_O_P_Sup_Cases, Side).
```

```
/****** ANALYZE AGAINST CASE *****/
```

```
**
```

```
** Calls the argue procedure when the only supporting
```

```
** case for a side is a case chosen by the user.
```

```

**
** Two seperate procedures are implemented. Only one
** is actually used during a call, depending on whether
** or not the chosen case holds for the same side as
** the Current Fact Situation.
**
*****/

analyze_against_case(Case, Side, Opp_Side, Chosen_Case) :-
    holding(Chosen_Case, Side),!,
    home,
    shared_dimensions(Case, Chosen_Case, Dimensions),
    opposing_cases_with_dimensions(Case, Opp_Cases),
    m_o_p(Case, [[Chosen_Case, Dimensions]], M_O_P_Sup_Cases),
    m_o_p(Case, Opp_Cases, M_O_P_Opp_Cases),
    best_cases(M_O_P_Sup_Cases, Best_Sup_Cases, Side),
    argue(Case, Best_Sup_Cases, M_O_P_Sup_Cases, M_O_P_Opp_Cases, Side).
analyze_against_case(Case, Side, Opp_Side, Chosen_Case) :-
    home,
    supporting_cases_with_dimensions(Case, Sup_Cases),
    shared_dimensions(Case, Chosen_Case, Dimensions),
    m_o_p(Case, Sup_Cases, M_O_P_Sup_Cases),
    m_o_p(Case, [[Chosen_Case, Dimensions]], M_O_P_Opp_Cases),
    best_cases(M_O_P_Opp_Cases, Best_Opp_Cases, Opp_Side),
    argue(Case, Best_Opp_Cases, M_O_P_Opp_Cases, M_O_P_Sup_Cases, Opp_Side).

/**** SUPPORTING CASES WITH SHARED DIMENSIONS ****
**
**
**
*****/

supporting_cases_with_dimensions(Case, Supporting_Case_List) :-
    setof([Support_Case, Shared_Dimensions],
        supporting_case_and_dimensions(Case, Support_Case, Shared_Dimensions),
        Supporting_Case_List),!.
supporting_cases_with_dimensions(Case, []).

supporting_case_and_dimensions(Case, Support_Case, Shared_Dimensions) :-
    get_supporting_case(Case, Support_Case),
    Case \== Support_Case,
    shared_dimensions(Case, Support_Case, Shared_Dimensions).

/**** OPPOSING CASES WITH SHARED DIMENSIONS ****
**
**
**
*****/

opposing_cases_with_dimensions(Case, Opposing_Case_List) :-
    setof([Opposing_Case, Shared_Dimensions],
        opposing_case_and_dimensions(Case, Opposing_Case, Shared_Dimensions),
        Opposing_Case_List),!.
opposing_cases_with_dimensions(Case, []).

opposing_case_and_dimensions(Case, Opposing_Case, Shared_Dimensions) :-
    get_opposing_case(Case, Opposing_Case),
    shared_dimensions(Case, Opposing_Case, Shared_Dimensions).

```

/***** MOST ON POINT CASES *****/

**

** Determines which cases from a list of cases and common
** dimensions is most-on-point with the Current Fact Situation
** (CFS). The case list consists of a list of Case/Common-
** Dimensions pairs represented as a two element list (ie,
** [Case, [Dimension1, Dimension2]]). The returned M-O-P-Case-
** List is a list of Cases/Common-Dimensions pairs (ie,
** [[Case1, Case2], [Dimension1, Dimension2, Dimension3]]).
** These pairs represent and contain the same information
** as would be found in the HYPO's highest order claim lattice
** nodes. Lower level nodes are not used by Hypo, and are
** discarded at this point in DAn.

**

** Two different M-O-P criteria are presently implemented:
** - COVERAGE: The same criteria used in Hypo, looks
** for greatest number of common dimensions,
** - COVERAGE-1: Like COVERAGE, this searches for the
** greatest number of common dimensions, but once those
** cases are discovered, they are compared among them-
** selves and only those with the fewest non-matching
** dimensions are considered most-on-point.

**

** The call to m-o-p-technique and all of its subprocedures
** would not be necessary if only COVERAGE was implemented.

**

*****/

m_o_p(CFS, Case_List, Most_On_Point_Case_List) :-
 most_on_point(Case_List, [], Temp_M_O_P_Case_List),
 m_o_p_technique(Temp_M_O_P_Case_List, Most_On_Point_Case_List, CFS).

most_on_point([], List, List).

most_on_point([New_Case_with_Dims|Tail], Current_M_O_P_Case_List, New_List) :-
 insert(New_Case_with_Dims, Current_M_O_P_Case_List, Temp_List),
 most_on_point(Tail, Temp_List, New_List).

/***** INSERT *****/

**

** This implements COVERAGE and the first stage of
** COVERAGE-1 by determining what to do with a
** Case/Common-Dimensions pair given the current state
** of the most-on-point list.

**

*****/

insert([Case,Dims], [], [[Case], Dims]).
insert([Case,Dims], [[Old_Cases, Dims]|Tail], [[Case|Old_Cases], Dims]|Tail)).
insert([Case,Dims], [[Old_Cases,Old_Dims]|Tail], [[Old_Cases,Old_Dims]|Tail]) :-
 sublist(Dims, Old_Dims).
insert([Case,Dims], [[Old_Cases,Old_Dims]|Tail], [[Case],Dims]|Tail):-
 sublist(Old_Dims, Dims).
insert(New_Case, [[Old_Cases,Old_Dims]|Tail], [[Old_Cases,Old_Dims]|NewTail]):-
 insert(New_Case, Tail, NewTail).

/***** M-O-P TECHNIQUE *****/

```

**
** If the COVERAGE technique is used, nothing further
** is done here. If COVERAGE-1 is being used, then
** cases with more extra dimensions than the fewest
** number of extra dimensions in the group are discarded.
**
** The procedure runs through the list twice, first
** to determine the minimum number of extra dimensions,
** and then to discard the less on point cases.
**
*****/

m_o_p_technique([], [], CFS).
m_o_p_technique(Temp_M_O_P_List, Temp_M_O_P_List, CFS) :-
    m_o_p_criteria(coverage).
m_o_p_technique([[Cases, Dims]|Tail], [[New_Cases, Dims]|NewTail], CFS) :-
    m_o_p_criteria(coverage_1),
    remove_cases_with_excess_dims(CFS, Cases, New_Cases),
    m_o_p_technique(Tail, NewTail, CFS).

/***** REMOVE CASES WITH EXCESS DIMENSIONS *****/
**
** Implements the extra step in COVERAGE-1.
**
*****/

remove_cases_with_excess_dims(CFS, Cases, New_Cases) :-
    get_min_number_of_excess_dims(CFS, Cases, Min_Num),
    remove_cases_with_more_excess_dims(CFS, Min_Num, Cases, New_Cases).

/***** COUNTING EXCESS DIMENSIONS PROCEDURES *****/
**
** These procedures determine the number of excess
** dimensions a case has relative to the Current
** Fact Situation, and the minimum such number.
**
*****/

get_number_of_excess_dims(CFS, Case, Number) :-
    list_of_dimensions(Case, Dimensions),
    dimensions_not_used(CFS, Dimensions, Excess_Dimensions),
    length(Excess_Dimensions, Number),!.

list_of_dimensions(Case, Dimensions) :-
    do_list_of_dimensions(Case, [], Dimensions).

do_list_of_dimensions(Case, Temp_List, Dimensions) :-
    get_dimension(Case, Dimension, Size),
    not(member(Dimension, Temp_List)),
    do_list_of_dimensions(Case, [Dimension|Temp_List], Dimensions).
do_list_of_dimensions(Case, Dimensions, Dimensions).

dimensions_not_used(CFS, [], []).
dimensions_not_used(CFS, [Dimension|Tail], NewTail) :-
    get_dimension(CFS, Dimension, Size),
    dimensions_not_used(CFS, Tail, NewTail).
dimensions_not_used(CFS, [Dimension|Tail], [Dimension|NewTail]) :-
    dimensions_not_used(CFS, Tail, NewTail).

```

```

get_min_number_of_excess_dims(CFS, Cases, Min_Num) :-
    do_get_min_number_of_excess_dims(CFS, Cases, 1000, Min_Num).

do_get_min_number_of_excess_dims(CFS, [], Min_Num, Min_Num).
do_get_min_number_of_excess_dims(CFS, [Case|Tail], Temp_Min_Num, Min_Num) :-
    get_number_of_excess_dims(CFS, Case, Number),
    if(Number < Temp_Min_Num,
        do_get_min_number_of_excess_dims(CFS, Tail, Number, Min_Num),
        do_get_min_number_of_excess_dims(CFS, Tail, Temp_Min_Num, Min_Num)).

```

```

/***** REMOVE CASES WITH MORE EXCESS DIMENSIONS **

```

```

**
** Does the actual discarding of less on point cases
** in the COVERAGE-1 technique.
**
*****/

```

```

remove_cases_with_more_excess_dims(CFS, Min_Num, [], []).
remove_cases_with_more_excess_dims(CFS, Min_Num, [Case|Tail], [Case|NewTail]) :-
    get_number_of_excess_dims(CFS, Case, Number),
    Number == Min_Num,
    remove_cases_with_more_excess_dims(CFS, Min_Num, Tail, NewTail).
remove_cases_with_more_excess_dims(CFS, Min_Num, [Case|Tail], NewTail) :-
    remove_cases_with_more_excess_dims(CFS, Min_Num, Tail, NewTail).

```

```

/***** BEST CASES *****/

```

```

**
** Determines the Best Cases to cite for a side in the same way that
** Hypo did: See if the cases have at least one dimension in
** common with the Current Fact Situation that favors the same side
** for which the case held.
**
*****/

```

```

best_cases([], [], Side).
best_cases([[Cases, Dimensions]|Tail], [[Cases, Dimensions]|NewTail], Side) :-
    at_least_one_dimension_on_side(Dimensions, Side),
    best_cases(Tail, NewTail, Side).
best_cases([[Cases, Dimensions]|Tail], NewTail, Side) :-
    best_cases(Tail, NewTail, Side).

```

```

at_least_one_dimension_on_side([], Side) :- !, fail.
at_least_one_dimension_on_side([Dimension|Tail], Side) :-
    pro(Dimension, Side);
    at_least_one_dimension_on_side(Tail, Side).

```

```

/***** QUERY *****/

```

```

**
** This procedure allows the user to query the current case base,
** using a menu system of possible search restrictions and either the
** 'and' or the 'or' conjunction between the restrictions. The
** queries are constructed as prolog strings (a list of ascii codes
** for each character). Once the entire query is constructed, the
** query is exported to a file and then consulted in order to add
** that query to the current prolog knowledge base. The query
** is then printed, and the results are displayed. The result of
** a query will either be a list of cases or a list of dimensions
** which satisfy the query. The results are displayed in a
** comprehensive format, so that all information about the satisfying
** case(s) or dimension(s) in the casebase is displayed.
**
** The two types of query (case query and dimension query) are presented
** in the menu called by the query procedure, as well as queries that
** will retrieve all possible cases and dimensions in the casebase.
** Further refinement of the queries is done by the subprocedures case-query
** (or do_query_menu(1)) and dimension-query (or do_query_menu(2)).
**
*****/

```

```

query :-

```

```

    write_query_banner,
    write_query_menu,
    get_choice(Choice),
    do_query_menu(Choice),!,
    query.

```

```

write_query_banner :-
    screen_type(query), !, home.

```

```

write_query_banner :-
    domain(Domain),
    narrow_screen,
    set_screen_type(query),
    set_window(1, 24),
    home,
    wide, write('    Query of'),nl,
    wide, write(Domain), write(' Domain'),nl,
    set_window(4, 24).

```

```

write_query_menu :-
    write('You have the following choices:'),nl,nl,
    write('    1> Get a case [or cases] where ...'),nl,
    write('    2> Get all cases, '),nl,
    write('    3> Get a dimensions [or dimensions] where ...'),nl,
    write('    4> Get all dimensions, '),nl,
    write('or    5> Return to Main Menu. '),nl,nl.

```

```

/***** QUERY MENU CHOICE 1: CASE QUERIES *****/

```

```

**
** This procedure creates the head of the query, and then sends control
** to get-rest-of-query for further refinement of the query.
**
** When the query has been completed, this procedure combines the tail and
** the head of the query, and sends control to analyze-query-list which
** turns the query list into a prolog procedure. This procedure is then
** called and the output is sent to the current output path.

```

```

**
*****/

do_query_menu(1) :-
    get_output(Output),
    get_rest_of_case_query(Rest_of_Query_List),
    name('query(Case) :- ', Head_List),
    append(Head_List, Rest_of_Query_List, Query_List),
    analyze_query_list(Query_List, Query_Atom),
    tell(Output),
    home,
    write('QUERY: '),nl,
    write(Query_Atom),nl,nl,
    write('RESULT:'),nl,
    get_results_of_query(Cases),
    write_out_cases_satisfying_query(Cases),
    tell(user),
    ready.

/***** GET REST OF QUERY LIST *****/
**
** Further refines case queries according to the choices made by the user.
** After a choice has been made, the user must select a conjunction
** from the conjunction menu, which will either call this menu recursively
** for further refinement of the query, or will terminate the query.
**
** When the query has been terminated (which is done at the conjunction
** menu), then the query string up to this point is created.
**
*****/

get_rest_of_case_query(Rest_of_Query_List) :-
    home,
    write_case_query_menu,
    get_choice(Choice),
    do_case_query_menu(Choice, Phrase),
    get_conjunction(case, Rest_of_Query),
    append(Phrase, Rest_of_Query, Rest_of_Query_List).

write_case_query_menu :-
    write('You may add one of the following clauses to your queries. '),nl,
    write('CASE or CASES that:'),nl,nl,
    write('    1> have a particular dimension, '),nl,
    write('    2> do not have a particular dimension, '),nl,
    write('    3> held for a particular side, '),nl,
    write('    4> distinguish some case along a dimension, '),nl,
    write('    5> are distinguished by some case along a dimension, '),nl,
    write('    6> cites some case for support along a dimension, '),nl,
    write('    7> are cited by some case for support along a dimension, '),nl,
    write('    8> overrule a case, '),nl,
    write('or    9> have been overruled by some case. '),nl,nl.

/***** CASE QUERY CHOICE 1 & 2: [NOT] HAVE A DIMENSION ****/
**
**
**
*****/

do_case_query_menu(1, Phrase) :-

```

```
get_a_dimension(Dimension),
insert('get_dimension(Case, ', Dimension, ', _)', Phrase).
```

```
do_case_query_menu(2, Phrase) :-
    get_a_dimension(Dimension),
    insert('not(get_dimension(Case, ', Dimension, ', _))', Phrase).
```

```
/****** CASE QUERY CHOICE 3: HELD FOR A SIDE *****/
**
**
**
*****/
```

```
do_case_query_menu(3, Phrase) :-
    get_ok_side(Side),
    insert('holding(Case, ', Side, ')', Phrase).
```

```
/** CASE QUERY CHOICE 4 & 5: DISTINGUISH[ED BY] ALONG DIMENSION **
**
**
**
*****/
```

```
do_case_query_menu(4, Phrase) :-
    get_a_dimension(Dimension),
    insert('distinguishes(Case, _, ', Dimension, ')', Phrase).
```

```
do_case_query_menu(5, Phrase) :-
    get_a_dimension(Dimension),
    insert('distinguishes(_, Case, ', Dimension, ')', Phrase).
```

```
/** CASE QUERY CHOICE 6 & 7: SUPPORT[ED BY] A DIMENSION ***
**
**
**
*****/
```

```
do_case_query_menu(6, Phrase) :-
    get_a_dimension(Dimension),
    insert('support(Case, _, ', Dimension, ')', Phrase).
do_case_query_menu(7, Phrase) :-
    get_a_dimension(Dimension),
    insert('support(_, Case, ', Dimension, ')', Phrase).
```

```
/****** CASE QUERY CHOICE 8 & 9: OVERRULE[ED BY] A CASE *****
**
**
**
*****/
```

```
do_case_query_menu(8, Phrase) :-
    name('overrule(Case, _)', Phrase).
```

```
do_case_query_menu(9, Phrase) :-
    name('overrule(_, Case)', Phrase).
```

```
/****** WRITE OUT CASES SATISFYING QUERY *****/
```

```
**
```

```
** This procedure writes out the cases which satisfy the query.
```

```
**
```

```
*****/
```

```
write_out_cases_satisfying_query([]) :-
```

```
    write('No cases satisfy your query.').nl,nl.
```

```
write_out_cases_satisfying_query(Cases) :-
```

```
    do_write_out_cases_satisfying_query(Cases).
```

```
do_write_out_cases_satisfying_query([]).
```

```
do_write_out_cases_satisfying_query([statute|Tail]) :-
```

```
    write('A state statute was cited by at least one case').nl,
```

```
    write_distinguished(statute),
```

```
    write_overruled(statute),
```

```
    write_cited_for_support_by(statute).nl,nl,
```

```
    do_write_out_cases_satisfying_query(Tail).
```

```
do_write_out_cases_satisfying_query([Case|Tail]) :-
```

```
    holding(Case, Holding),
```

```
    write_long_name(Case).nl,
```

```
    tab(2), write_citation(Case).nl,
```

```
    tab(2), write('Short Name : '), write_short_name(Case).nl,
```

```
    tab(2), write('Internal Name : '), write(Case).nl,
```

```
    tab(2), write('Holding : '), write(Holding).nl,
```

```
    write_dimensions(Case).nl,
```

```
    write_distinguishing(Case),
```

```
    write_distinguished(Case),
```

```
    write_overruling(Case),
```

```
    write_overruled(Case),
```

```
    write_citing_for_support(Case),
```

```
    write_cited_for_support_by(Case).nl,nl,
```

```
    do_write_out_cases_satisfying_query(Tail).
```

```
do_write_out_cases_satisfying_query([Case|Tail]) :-
```

```
    not(holding(Case, Holding)),
```

```
    write('Insufficient information on '),
```

```
    write_long_name(Case).nl,nl,
```

```
    do_write_out_cases_satisfying_query(Tail).
```

```
write_distinguishing(The_Case) :-
```

```
    setof(Case, distinguishes(The_Case, Case, Dimension), Cases),
```

```
    tab(2), write('Distinguishing '),
```

```
    write_list_of_cases(Cases, sp), write(' along:').nl,
```

```
    tab(6), write(Dimension), write('.').nl,
```

```
    fail.
```

```
write_distinguishing(The_Case).
```

```
write_distinguished(The_Case) :-
```

```
    setof(Case, distinguishes(Case, The_Case, Dimension), Cases),
```

```
    tab(2), write('Distinguished by '),
```

```
    write_list_of_cases(Cases, sp),
```

```
    write(' along:').nl,
```

```
    tab(6), write(Dimension), write('.').nl,
```

```
    fail.
```

```
write_distinguished(The_Case).
```

```
write_citing_for_support(The_Case) :-
```

```
    setof(Case, support(The_Case, Case, Dimension), Cases),
```

```
    tab(2), write('Citing '),
```

```

    write_list_of_cases(Cases, sp),
    write(' for support along:'),nl,
    tab(6), write(Dimension), write('.'), nl,
    fail.
write_citing_for_support(The_Case).

write_cited_for_support_by(The_Case):-
    setof(Case, support(Case, The_Case, Dimension), Cases),
    tab(2), write('Cited by '),
    write_list_of_cases(Cases, sp),
    write(' for support along:'),nl,
    tab(6), write(Dimension), write('.'), nl,
    fail.
write_cited_for_support_by(The_Case).

write_overruling(The_Case):-
    setof(Case, overrule(The_Case, Case), Cases),
    tab(2), write('Overruling '),
    write_list_of_cases(Cases, sp), write('.'), nl,
    fail.
write_overruling(The_Case).

write_overruled(The_Case):-
    setof(Case, overrule(Case, The_Case), Cases),
    tab(2), write('Overruled by '),
    write_list_of_cases(Cases, sp), write('.'), nl,
    fail.
write_overruled(The_Case).

```

```
/****** QUERY MENU CHOICE 2: RETRIEVE ALL CASES *****/
**
** Retrieves all cases in the casebase.
**
*****/
```

```
do_query_menu(2) :-
    get_output(Output),
    name('query(Case) :- holding(Case, _).', Query_List),
    analyze_query_list(Query_List, Query_Atom),
    home,
    tell(Output),
    write('QUERY: '),nl,
    write(Query_Atom),nl,nl,
    write('RESULT:'),nl,
    get_results_of_query(Cases),
    write_out_cases_satisfying_query(Cases),
    tell(user),
    ready.
```

```
/****** QUERY MENU CHOICE 3: DIMENSION QUERIES *****/
```

```
**
```

```
** Allows the query of dimensions, through a technique identical to  
** that used to query cases.
```

```
**
```

```
*****/
```

```
do_query_menu(3) :-
```

```
    get_output(Output),  
    analyze_domain,  
    get_rest_of_dimension_query(Rest_of_Query_List),  
    name('query(Dimension) :- ', Head_List),  
    append(Head_List, Rest_of_Query_List, Query_List),  
    analyze_query_list(Query_List, Query_Atom),  
    home,  
    tell(Output),  
    write('QUERY: '),nl,  
    write(Query_Atom),nl,nl,  
    write('RESULT:'),nl,  
    get_results_of_query(Dimensions),  
    write_out_dimensions_satisfying_query(Dimensions),  
    tell(user),  
    ready.
```

```
get_rest_of_dimension_query(Rest_of_Query_List) :-
```

```
    home,  
    write_dimension_query_menu,  
    get_choice(Choice),  
    do_dimension_query_menu(Choice, Phrase),  
    get_conjunction(dimension, Rest_of_Query),  
    append(Phrase, Rest_of_Query, Rest_of_Query_List).
```

```
write_dimension_query_menu :-
```

```
    write('You may add one of the following clauses to your queries. '),nl,  
    write('DIMENSION or DIMENSIONS that:'),nl,nl,  
    write('    1> are found in a particular case. '),nl,  
    write('    2> are not found in a particular case. '),nl,  
    write('    3> favor a particular side. '),nl,  
    write('    4> are both-sided. '),nl,  
    write('    5> are overwhelming. '),nl,  
    write('    6> dominate another dimension. '),nl,  
    write('    7> are dominated by another dimension. '),nl,  
    write('or    8> are part of a overwhelming dimension pair. '),nl,nl.
```

```
/****** DIMENSION QUERY CHOICE 1 & 2: [NOT] FOUND IN A CASE *****/
```

```
**
```

```
**
```

```
**
```

```
*****/
```

```
do_dimension_query_menu(1, Phrase) :-
```

```
    get_a_valid_case(Case),  
    insert('get_dimension(', Case, ', Dimension, __)', Phrase).
```

```
do_dimension_query_menu(2, Phrase) :-
```

```
    get_a_valid_case(Case),  
    insert('not(get_dimension(', Case, ', Dimension, __))', Phrase).
```

```

/***** DIMENSION QUERY CHOICE 3: FAVOR A PARTICULAR SIDE *****/
**
**
**
*****/

```

```

do_dimension_query_menu(3, Phrase) :-
    get_ok_side(Side),
    insert('pro(Dimension, ', Side, ')', Phrase).

```

```

/***** DIMENSION QUERY CHOICE 4: BOTH-SIDED *****/
**
**
**
*****/

```

```

do_dimension_query_menu(4, Phrase) :-
    name('both_sided(Dimension, _)', Phrase).

```

```

/***** DIMENSION QUERY CHOICE 5: OVERWELMING *****/
**
**
**
*****/

```

```

do_dimension_query_menu(5, Phrase) :-
    name('overwhelming(Dimension, _)', Phrase).

```

```

/***** DIMENSION QUERY CHOICE 6 & 7: DOMINATE[D BY] ANOTHER *****/
**
**
**
*****/

```

```

do_dimension_query_menu(6, Phrase) :-
    name('dominated_dimension_pair(_, Dimension, _)', Phrase).

```

```

do_dimension_query_menu(7, Phrase) :-
    name('dominated_dimension_pair(Dimension, _, _)', Phrase).

```

```

/*** DIMENSION QUERY CHOICE 8: PART OF OVERWHELMING PAIR ***
**
**
**
*****/

```

```

do_dimension_query_menu(8, Phrase) :-
    name('(overwhelming_dimension_pair(Dimension, _, _);
overwhelming_dimension_pair(_, Dimension, _)', Phrase).

```

```

/***** WRITE OUT DIMENSIONS SATISFYING QUERY *****/
**
** Writes out dimensions satisfying the query.
**

```

*****/

```
write_out_dimensions_satisfying_query([]) :-
    write('No dimensions satisfy this query'),nl,nl.
write_out_dimensions_satisfying_query(Dimensions) :-
    do_write_out_dimensions_satisfying_query(Dimensions), !.

do_write_out_dimensions_satisfying_query([]).
do_write_out_dimensions_satisfying_query([Dimension|Tail]) :-
    describe(Dimension, Description),
    write(Dimension),nl,
    tab(3), write('Description : '), write(Description),nl,
    tab(3), write('Overwhelming : '),
    if(overwhelming(Dimension, _),
        write('Yes'),
        write('No')),nl,
    tab(3), write('Both-Sided : '),
    if(both_sided(Dimension, _),
        write('Yes'),
        write('No')),nl,
    write_range_of_dimension(Dimension),
    write_dominating(Dimension),
    write_dominated(Dimension),
    write_part_of_overwhelming_pair(Dimension),
    tab(3), write('Found in : '),
    write_found_in_cases(Dimension),nl,nl,
    do_write_out_dimensions_satisfying_query(Tail).

write_range_of_dimension(Dimension) :-
    range(Dimension, Range),
    tab(3), write('Range :'),nl,
    write_a_list(Range, 18, nl).
write_range_of_dimension(Dimension).

write_dominating(The_Dimension) :-
    setof(Dimension,
        X^dominated__dimension__pair(Dimension, The_Dimension, X),
        Dimensions),
    tab(3), write('Dominating : '),nl,
    write_a_list(Dimensions, 18, nl).
write_dominating(The_Dimension).

write_dominated(The_Dimension) :-
    setof(Dimension,
        X^dominated__dimension__pair(The_Dimension, Dimension, X),
        Dimensions),
    tab(3), write('Dominated by : '),nl,
    write_a_list(Dimensions, 18, nl).
write_dominated(The_Dimension).

write_part_of_overwhelming_pair(The_Dimension) :-
    (overwhelming_dimension_pair(The_Dimension, _, _);
    overwhelming_dimension_pair(_, The_Dimension, _)),
    tab(3), write('Part of Overwhelming Pair with :'),nl,
    do_write_part_of_overwhelming_pair(The_Dimension).
write_part_of_overwhelming_pair(The_Dimension).

do_write_part_of_overwhelming_pair(The_Dimension) :-
    setof(Dimension,
        X^overwhelming__dimension__pair(The_Dimension, Dimension, X),
```

```

        Dimensions1),
    setof(Dimension,
        X^overwhelming_dimension_pair(Dimension, The_Dimension, X),
        Dimensions2),!,
    write_a_list(Dimensions1, 18, nl),
    write_a_list(Dimensions2, 18, nl).
do_write_part_of_overwhelming_pair(The_Dimension) :-
    setof(Dimension,
        X^overwhelming_dimension_pair(The_Dimension, Dimension, X),
        Dimensions),
    write_a_list(Dimensions, 18, nl).
do_write_part_of_overwhelming_pair(The_Dimension) :-
    setof(Dimension,
        X^overwhelming_dimension_pair(Dimension, The_Dimension, X),
        Dimensions),
    write_a_list(Dimensions, 18, nl).

write_found_in_cases(The_Dimension) :-
    setof(Case, X^get_dimension(Case, The_Dimension, X), Cases),
    write_a_list(Cases, 0, sp).
write_found_in_cases(The_Dimension) :-
    write('No Cases.').

```

```
/***** QUERY MENU CHOICE 4: RETRIEVE ALL DIMENSIONS *****/
```

```
**
```

```
** Creates a query which retrieves all dimensions in the domain.
```

```
**
```

```
*****/
```

```
do_query_menu(4):-
```

```
    get_output(Output),
```

```
    analyze_domain,
```

```
    name('query(Dimension) :- pro(Dimension, _).', Query_List),
```

```
    analyze_query_list(Query_List, Query_Atom),
```

```
    home,
```

```
    tell(Output),
```

```
    write('QUERY: '),nl,
```

```
    write(Query_Atom),nl,nl,
```

```
    write('RESULT:'),nl,
```

```
    get_results_of_query(Dimensions),
```

```
    write_out_dimensions_satisfying_query(Dimensions),
```

```
    tell(user),
```

```
    ready.
```

```
/***** QUERY MENU CHOICE 5: EXIT *****/
```

```
**
```

```
** By failing this goal, the entire query procedure will fail. Thus,
```

```
** any calling procedure must expect query to fail. This is done in
```

```
** order to prevent excessive demands on the global stack which has
```

```
** to be maintained for backtracking purposes.
```

```
**
```

```
*****/
```

```
do_query_menu(5):- !, fail.
```

```
/***** QUERY MENU, INAPPROPRIATE RESPONSE *****/
```

```
**
```

```
** Calls inappropriate response procedure found in the utilities
```

```
** file, and recalls the menu.
```

```
**
```

```
*****/
```

```
do_query_menu(_):-
```

```
    inappropriate_response(New_Choice),
```

```
    do_query_menu(New_Choice), !.
```

```

/***** GET CONJUNCTION *****/

```

```

**
** Gets a conjunction to put between segments of a case or
** dimension query. Either 'And' or 'Or' may be chosen, and
** they are implemented through the use of the prolog syntax
** for 'and' (';'), and the prolog syntax for 'or' (';').
**

```

```

** It is also at this point that the user may chose to end
** the construction of a query. Since get-conjunction
** would otherwise seek a further segment of the query, the
** end of a query is satisfied merely by returning a period
** (the syntax for the end of a procedure in prolog) from
** get-conjunction.
**

```

```

*****/

```

```

get_conjunction(Case_or_Dimension, Rest_of_Query) :-
    write_conjunction_menu,
    get_choice(Choice),
    do_conjunction_menu(Choice, Case_or_Dimension, Rest_of_Query).

```

```

write_conjunction_menu :-
    write_query_banner,
    write('Your choices:'),nl,nl,
    write('    1> AND (another clause),'),nl,
    write('    2> OR (another clause),'),nl,
    write('or   3> End construction of query and retrieve results. '),nl,
    nl.

```

```

/***** DO CONJUNCTION: AND *****/

```

```

**
**
**

```

```

*****/

```

```

do_conjunction_menu(1, Case_or_Dimension, Rest_of_Query) :-
    continue_query(Case_or_Dimension, Last_Part_of_Query),
    name(' ', And),
    append(And, Last_Part_of_Query, Rest_of_Query).

```

```

/***** DO CONJUNCTION: OR *****/

```

```

**
**
**

```

```

*****/

```

```

do_conjunction_menu(2, Case_or_Dimension, Rest_of_Query) :-
    continue_query(Case_or_Dimension, Last_Part_of_Query),
    name('; ', Or),
    append(Or, Last_Part_of_Query, Rest_of_Query).

```

```

/***** DO CONJUNCTION: END QUERY *****/

```

```

**
**
**

```

```

*****/

```

```
do_conjunction_menu(3, Case_or_Dimension, Period) :-  
    name('.', Period).
```

```
/***** DO CONJUNCTION: IMPROPER RESPONSE *****/  
**  
**  
**  
*****/
```

```
do_conjunction_menu(_, Case_or_Dimension, Rest_of_Query).
```

```
/***** CONJUNCTION UTILITIES *****/  
**  
** Procedures used by more than one part of conjunction menu.  
**  
*****/
```

```
continue_query(case, Last_Part_of_Query) :-  
    get_rest_of_case_query(Last_Part_of_Query).  
continue_query(dimension, Last_Part_of_Query) :-  
    get_rest_of_dimension_query(Last_Part_of_Query).
```

```
/****** QUERY UTILITIES *****/
```

```
**
```

```
** Subprocedures used in more than one location in query procedures.
```

```
**
```

```
*****/
```

```
analyze_query_list(Query_List, Query_Atom) :-
```

```
    name(Query_Atom, Query_List),
```

```
    tell('query.tmp'),
```

```
    write(Query_Atom),nl,
```

```
    close('query.tmp'),
```

```
    tell(user),
```

```
    reconsult('query.tmp'),
```

```
    system("del query.tmp;*").
```

```
get_a_dimension(Dimension) :-
```

```
    home,
```

```
    write('Do you want to '),nl,nl,
```

```
    write('    1> name a particular dimension,')nl,
```

```
    write('    2> have the search look for any possible dimension.').nl,nl,
```

```
    get_choice(Choice),
```

```
    do_get_a_dimension(Choice, Dimension).
```

```
do_get_a_dimension(1, Dimension) :-
```

```
    get_a_valid_dimension(Dimension),
```

```
    write_query_banner,!
```

```
do_get_a_dimension(2, '_').
```

```
do_get_a_dimension(_, Dimension) :-
```

```
    inappropriate_response(New_Choice),
```

```
    do_get_a_dimension(New_Choice, Dimension).
```

```
insert(Head_Atom, Middle_Atom, Tail_Atom, Phrase) :-
```

```
    name(Head_Atom, Head_List),
```

```
    name(Middle_Atom, Middle_List),
```

```
    name(Tail_Atom, Tail_List),
```

```
    append(Middle_List, Tail_List, Ending_List),
```

```
    append(Head_List, Ending_List, Phrase).
```

```
get_results_of_query(Items) :-
```

```
    setof(Item, query(Item), Items).
```

```
get_results_of_query([]).
```

```
/***** UTILITIES *****/
```

```
**
```

```
** These procedures are used throughout DAn.
```

```
**
```

```
*****/
```

```
/***** IF *****/
```

```
**
```

```
** Used to implement the conditional format. The provided  
** conditional format (supposedly defined identically), did  
** not function as expected.
```

```
**
```

```
*****/
```

```
if(Condition, Then, Else) :-
```

```
    Condition, !, Then.
```

```
if(Condition, Then, Else) :-
```

```
    Else.
```

```
/***** LIST CONTROL UTILITIES *****/
```

```
**
```

```
**
```

```
**
```

```
*****/
```

```
member(O, [O|_]) :- !.
```

```
member(O, [_]Tail) :- member(O, Tail).
```

```
append([], List2, List2) :- !.
```

```
append([Head|Tail], List2, [Head|New_Tail]) :-
```

```
    append(Tail, List2, New_Tail).
```

```
intersection(List1, [], []) :- !.
```

```
intersection(List1, [Element|Tail], [Element|NewTail]) :-
```

```
    member(Element, List1),
```

```
    intersection(List1, Tail, NewTail), !.
```

```
intersection(List1, [Element|Tail], NewTail) :-
```

```
    intersection(List1, Tail, NewTail).
```

```
sublist([], List) :- !.
```

```
sublist([Element|Tail], List) :-
```

```
    member(Element, List),
```

```
    sublist(Tail, List).
```

```
/***** VT100 Terminal Commands *****/
```

```
**
```

```
** Used to improve interface. Are compatible with VT 100 terminals  
** and compatibles.
```

```
**
```

```
*****/
```

```
escape :- put(27).
```

```
home :- telling(user), !, escape, write('[H)'), erase__end_of__screen.
```

```
home.
```

```
go_home :- escape, write('[H)').
```

```
scrolling_region_home :- escape, write('[?6h').
```

```
absolute_home :- escape, write('[?6l').
```

```

erase_screen :- escape, write('[2J').
erase_end_of_screen :- escape, write('[J').
erase_end_of_line :- escape, write('[K').
big_top :- escape, write('#3').
big_bottom :- escape, write('#4').
wide :- escape, write('#6').
normal :- escape, write('#5').
move_cursor(Line, Column) :-      escape, write('['), write(Line),
                                write(';'), write(Column), write('f').
up_line :- escape, write('[A').
set_window(Top, Bottom) :-        escape, write('['), write(Top),
                                write(';'), write(Bottom), write('r').
auto_wrap :- escape, write('[?7h').
no_auto_wrap :- escape, write('[?7l').
beep :- put(7).
save_cursor :- escape, write('7').
restore_cursor :- escape, write('8').
wide_screen :- screen_type(dimension_list).
wide_screen :- system("set term/width=132").
narrow_screen :-
    screen_type(dimension_list),
    system("set term/width=80"), !.
narrow_screen.

```

```

/***** SET SCREEN TYPE *****/

```

```

**
** Used to keep track of what the current display screen is.
** For instance, whether the dimension list is currently
** displayed and therefore the terminal width is 132 characters.
**
*****/

```

```

set_screen_type(Screen_Type) :-
    retract(screen_type(_)),
    asserta(screen_type(Screen_Type)).

```

```

/***** GENERIC RESPONSES TO MENU SELECTIONS *****/

```

```

**
** Sorry is used when a menu selection has not been implemented.
** Inappropriate response is used when choice is not listed on menu.
**
*****/

```

```

sorry(Choice) :-
    write_message('SORRY. Choice not implemented'),
    up_line, erase_end_of_line,
    get_choice(Choice).

```

```

inappropriate_response(Choice) :-
    write_message('Inappropriate response.'),
    up_line, erase_end_of_line,
    get_choice(Choice).

```

```

/***** WRITE MESSAGE *****/

```

```

**
** Outputs a message on the bottom of the screen in wide print
** with a beep.

```

```

**
***** /

write_message(Message) :-
    save_cursor,
    move_cursor(24,1),
    erase_end_of_line,
    wide, beep,
    write(Message),
    restore_cursor,!.

/***** READY *****/
**
** Used when output must wait for user to indicate readiness.
**
*****/

ready :-
    write('HIT RETURN WHEN READY :'),
    get0(Return),!.

/***** GET UTILITIES *****/
**
** These utilities symplify the retrieval of information from
** the user.
**
*****/

get_choice(Choice) :-
    write('Your choice (please complete entry with a period) : '),
    read(Choice),
    nonvar(Choice), !.
get_choice(Choice) :-
    up_line, erase_end_of_line,
    write_message('Don"t start with a capital letter.'),
    get_choice(Choice),!.

get_a_number(New_Level) :-
    get_choice(New_Level),
    number(New_Level),!.
get_a_number(New_Level) :-
    up_line, erase_end_of_line,
    write_message('Please enter a number.'),
    get_a_number(New_Level),!.

get_yes_or_no(Choice) :-
    get_choice(Temp_Choice),
    do_get_yes_or_no(Temp_Choice, Choice), !.

do_get_yes_or_no(yes, yes) :- !.
do_get_yes_or_no(y, yes) :- !.
do_get_yes_or_no(no, no) :- !.
do_get_yes_or_no(n, no) :- !.
do_get_yes_or_no(_, Choice) :-
    up_line, erase_end_of_line,
    write_message('Enter yes (y) or no (n).'),
    get_yes_or_no(Choice).

```

```

get_case(Case) :-
    get_choice(Case),
    case_exists(Case),!.
get_case(Case) :-
    up_line, erase_end_of_line,
    write_message('Error in Case Name.'),
    get_case(Case),!.

case_exists(Case) :- holding(Case, Side).

get_a_valid_case(Case) :-
    write('If you make an error in the name of the case, the case list'),nl,
    write('will be displayed.'),nl,nl,
    get_choice(Entered_Case), !,
    verify_validity_of_case(Entered_Case, Case).

verify_validity_of_case(Case, Case) :-
    case_exists(Case), !.
verify_validity_of_case(_, Case) :-
    get_case_through_menu(Case).

get_case_through_menu(Case) :-
    write_out_cases, nl,
    write('Select the case you desire.'),nl,
    get_case(Case).

get_a_valid_dimension(Dimension) :-
    write_all_dimensions,
    home,
    write('Enter the number of the dimension you desire.'),nl,
    get_choice(Number),
    dimension_number(Dimension, Number), !.
get_a_valid_dimension(Dimension) :-
    home,
    write_message('Invalid choice.'),
    get_a_valid_dimension(Dimension).

get_existing_file(File_Name) :-
    get_file_name(File_Name),
    exists(File_Name),!.
get_existing_file(File_Name) :-
    home,
    write_message('File does not exist.'),
    write('Current Directory:'),nl,nl,
    system("dir"),nl,nl,
    get_existing_file(File_Name),!.

get_unused_file_name(File_Name) :-
    get_file_name(File_Name),
    unused_file_name(File_Name), !.
get_unused_file_name(File_Name) :-
    home,
    write_message('File already exists.'),
    get_unused_file_name(File_Name),!.

unused_file_name(File_Name) :-
    not(exists(File_Name)), !.
unused_file_name(File_Name) :-
    File_Name == user.

```

```

get_file_name(File_Name) :-
    write('A file name chosen should be either a single word name without'), nl,
    write('punctuation or should be entered inside of single quotation marks.'),
    nl,nl,get_choice(File_Name), !.

get_ok_side(Side) :-
    get_opposite_side(Side1, Side2),
    write('Enter either '), write(Side1), write(', '),
    write(Side2), write(' or unknown.'),nl,
    get_choice(Side),
    verify_side(Side), !.
get_ok_side(Side) :-
    up_line, erase_end_of_line,
    up_line, erase_end_of_line,
    write_message('Not an appropriate side.'),
    get_ok_(Side),!.

verify_side(Side) :-
    get_opposite_side(Side, _), !.
verify_side(Side) :-
    Side == unknown.

/***** GET DIMENSION *****/
**
** Used throughout program to retrieve a dimension that is
** used in a case, or to get a case that uses a dimension.
** The procedure checks whether no value dimensions are
** considered relevant.
**
*****/

get_dimension(Case_name, Dimension, Size) :-
    has_dimension(Case_name, Dimension, Size),
    if(include_no_dimension_values_as_relevant,
        true,
        Size > 0).

/***** GET SUPPORTING AND OPPOSING CASES *****/
**
** Retrieve related cases given a specific fact situation.
**
*****/

get_opposing_case(Case_name, Opposing_case):-
    get_dimension(Case_name, Dimension, X),
    holding(Case_name, Side),
    get_dimension(Opposing_case, Dimension, Y),
    get_opposite_side(Side, Opposing_side),
    holding(Opposing_case, Opposing_side).

get_supporting_case(Case_name, Supporting_case):-
    get_dimension(Case_name, Dimension, X),
    holding(Case_name, Side),
    get_dimension(Supporting_case, Dimension, Y),
    holding(Supporting_case, Side).

share_a_dimension(Case1, Case2, Shared_Dimension) :-
    get_dimension(Case1, Shared_Dimension, Size1),

```

```
get_dimension(Case2, Shared_Dimension, Size2).
```

```
shared_dimensions(Case1, Case2, Shared_Dimensions) :-  
    setof(Shared_Dimension,  
        share_a_dimension(Case1, Case2, Shared_Dimension),  
        Shared_Dimensions).
```

```
/****** WRITE UTILITIES *****/
```

```
**
```

```
** These utilities handle output to the user and to files.
```

```
**
```

```
*****/
```

```
/****** PP (PRETTY PRINT) *****/
```

```
**
```

```
** Used only occasionally to simplify the output
```

```
** of a mixture of direct text and variables.
```

```
**
```

```
*****/
```

```
pp([]) :- !.
```

```
pp([Element|Tail]) :-
```

```
    write(Element),
```

```
    pp(Tail).
```

```
/****** WRITE UTILITIES FOR LISTS *****/
```

```
**
```

```
**
```

```
**
```

```
*****/
```

```
write_list_of_cases([], _) :- !.
```

```
write_list_of_cases([Case_Name|_], _) :-
```

```
    write_short_name(Case_Name), !.
```

```
write_list_of_cases([Case_Name|Tail], sp) :-
```

```
    write_short_name(Case_Name), write(' '),
```

```
    write_list_of_cases(Tail, sp), !.
```

```
write_list_of_cases([], Tab_Amount, nl) :- !.
```

```
write_list_of_cases([Case|Tail], Tab_Amount, nl) :-
```

```
    tab(Tab_Amount),
```

```
    write_long_name(Case), nl,
```

```
    write_list_of_cases(Tail, Tab_Amount, nl).
```

```
write_a_list([], Tab_amount, Space) :- !.
```

```
write_a_list([Head|Tail], Tab_amount, sp) :-
```

```
    tab(Tab_amount), write(Head), write(' '),
```

```
    write_a_list(Tail, Tab_amount, sp), !.
```

```
write_a_list([Head|Tail], Tab_amount, nl) :-
```

```
    tab(Tab_amount), write(Head), nl,
```

```
    write_a_list(Tail, Tab_amount, nl).
```

```
write_element_number([Element|Tail], Number) :-
```

```
    Number < 1,
```

```
    write('Error in Element Number'), !.
```

```
write_element_number([Element|Tail], 1) :-
```

```
    write(Element), !.
```

```

write_element_number([Element|Tail], Number) :-
    New_Number is Number - 1,
    write_element_number(Tail, New_Number).

```

```

/***** WRITE OUT CASES *****/

```

```

**
** Outputs a list of cases in column format, and defines
** the scrollable window so as to not disturb the file list.
**
****/

```

```

write_out_cases :-
    screen_type(case_list), !.
write_out_cases :-
    narrow_screen,
    set_screen_type(case_list),
    set_window(1,24),
    home,
    holding(A_Case, Side1),
    get_opposite_side(Side1, Side2),
    move_cursor(1,0), wide,
    write('Cases in CaseBase, pro: '), write(Side1),
    setof(Side1_Case, holding(Side1_Case, Side1), Side1_Cases),
    do_write_cases(Side1_Cases, 1, Lines_Used_1),
    New_Line_1 is Lines_Used_1 + 1,
    move_cursor(New_Line_1,0),
    wide, write('Cases in CaseBase, pro: '), write(Side2),
    setof(Side2_Case, holding(Side2_Case, Side2), Side2_Cases),
    do_write_cases(Side2_Cases, New_Line_1, Lines_Used_2),
    New_Line_2 is Lines_Used_2 + 1,
    if(holding(Some_Case, unknown),
        write_undecided_cases(New_Line_2, New_Line_3),
        New_Line_3 = New_Line_2),
    move_cursor(New_Line_3,0),
    write('-----'),
    Top_of_Window is New_Line_3 + 1,
    set_window(Top_of_Window,24).

```

```

write_undecided_cases(New_Line_2, New_Line_3) :-
    move_cursor(New_Line_2,0),
    wide, write('Cases in CaseBase, undecided'),
    setof(Unknown_Case, holding(Unknown_Case, unknown), Unknown_Cases),
    do_write_cases(Unknown_Cases, New_Line_2, Lines_Used_3),
    New_Line_3 is Lines_Used_3 + 1.

```

```

do_write_cases([], Number_of_Lines, Number_of_Lines) :- !.
do_write_cases([Case1, Case2, Case3|Tail], Old_Line, Number_of_Lines) :-
    New_Line is Old_Line + 1,
    move_cursor(New_Line,0), write(Case1),
    move_cursor(New_Line,27), write(Case2),
    move_cursor(New_Line,54), write(Case3),
    do_write_cases(Tail, New_Line, Number_of_Lines), !.
do_write_cases([Case1, Case2|[]], Old_Line, New_Line) :-
    New_Line is Old_Line + 1,
    move_cursor(New_Line,0), write(Case1),
    move_cursor(New_Line,27), write(Case2), !.
do_write_cases([Case1|[]], Old_Line, New_Line) :-
    New_Line is Old_Line + 1,

```

```
move_cursor(New_Line,0), write(Case1).
```

```
/****** WRITE DIMENSION LIST *****/
```

```
**
```

```
** Outputs a list of dimension with corresponding numbers  
** (see assign dimension numbers below) on a 132 character  
** wide screen. The scrolling window is defined to protect  
** the created list.  
**
```

```
*****/
```

```
write_all_dimensions :-
```

```
    screen_type(dimension_list), !.
```

```
write_all_dimensions :-
```

```
    assign_dimension_numbers,  
    wide_screen,  
    set_screen_type(dimension_list),  
    set_window(1,24),  
    home,  
    setof(Dimension, X^pro(Dimension, X), Dimension_List),  
    wide, write('Possible Dimensions:'),nl,  
    do_write_all_dimensions(Dimension_List, 1, Lines_Used),  
    Next_Line is Lines_Used + 2,  
    set_window(Next_Line, 24).
```

```
do_write_all_dimensions([], Lines_Used, Lines_Used) :- !.
```

```
do_write_all_dimensions([Dimension|[]], Old_Line, Lines_Used) :-
```

```
    Lines_Used is Old_Line + 1,  
    move_cursor(Lines_Used,1),  
    dimension_number(Dimension, Number),  
    write(Number), write('> '), write(Dimension),nl, !.
```

```
do_write_all_dimensions([Dimension1,Dimension2|Tail], Old_Line, Total_Lines_Used) :-
```

```
    Lines_Used is Old_Line + 1,  
    move_cursor(Lines_Used,1),  
    dimension_number(Dimension1, Number1),  
    write(Number1), write('> '), write(Dimension1),  
    move_cursor(Lines_Used, 65),  
    dimension_number(Dimension2, Number2),  
    write(Number2), write('> '), write(Dimension2),  
    do_write_all_dimensions(Tail, Lines_Used, Total_Lines_Used), !.
```

```
/*** ASSIGN DIMENSION NUMBERS ***/
```

```
**
```

```
** Used to assign numbers to dimensions so that dimension  
** can be entered by user according to number, rather than  
** by name.  
**
```

```
** Fail in assign_dimension_numbers used to save stack space.  
**
```

```
*****/
```

```
assign_dimension_numbers :-
```

```
    not(dimension_number(A_Dimension, A_Number)),  
    setof(Dimension, X^pro(Dimension, X), Dimension_List),  
    do_assign_dimension_numbers(Dimension_List, 1), fail.  
assign_dimension_numbers.
```

```
do_assign_dimension_numbers([], Number) :- !.
```

```

do_assign_dimension_numbers([Dimension|Tail], Number) :-
    assert(dimension_number(Dimension, Number)),
    New_Number is Number + 1,
    do_assign_dimension_numbers(Tail, New_Number), !.

```

```

/** WRITE UTILITIES FOR SPECIFIC FEATURES **

```

```

**

```

```

** Handle the output for specific features in the program.

```

```

**

```

```

*****/

```

```

write_range(Dimension, Size) :-
    range(Dimension, Range_List),
    Number is Size + 1,
    write_element_number(Range_List, Number).

```

```

write_dimensions(Case_name) :-
    write(' Dimensions : '),
    get_dimension(Case_name, Dimension, Size),
    pro(Dimension, Side),
    write(Dimension),
    write(' (pro: '), write(Side), write(')'), nl,
    write('          '),
    fail.

```

```

write_dimensions(Case_name).

```

```

write_dimension_description(Dimension) :-
    describe(Dimension, Description),
    write(Description).

```

```

write_short_name(Case) :-
    shortname(Case, Name), !,
    write(Name).

```

```

write_short_name(statute) :-
    write('a state statute').

```

```

write_short_name(Case) :-
    write(Case).

```

```

write_long_name(Case) :-
    longname(Case, Name),
    write(Name), !.

```

```

write_long_name(Case) :-
    write_short_name(Case).

```

```

write_citation(Case) :-
    citation(Case, Citation),
    year(Case, Year),
    write(Citation), write(' '), write(Year), write(')').

```