Planning in an Open-Textured Domain: A Thesis Proposal

Kate Sanders

Department of Computer Science
Brown University
Providence, Rhode Island 02912

Technical Report No. CS-91-08
March 1, 1991

Planning in an open-textured domain: a thesis proposal

Kate Sanders

March 1, 1991

Abstract

Most work in artificial intelligence and law has concentrated on modelling the type of reasoning done by trial lawyers. In fact, most lawyers' work involves planning – for example, wills and trusts, real estate deals, and business mergers and acquisitions. Certain planning issues, such as the use of underspecified, or "open-textured" rules, are illustrated especially clearly in this domain.

In this proposal, I set forth the characteristic features of planning in law, place it in the context of past artificial intelligence work in both law and planning, outline my approach to solving this problem, and describe CHIRON, a system that I am developing to test my solution in the domain of personal income tax planning.

1 Introduction

In this thesis, I propose to investigate planning issues in law. The popular view of lawyers is the trial lawyer — Perry Mason, Clarence Darrow, or the lawyers on *LA Law*. Many lawyers, however, make their living planning transactions, such as the sale of a piece of property, the establishment of a trust, or the reorganization of a corporation.

Planning problems arise in law when an individual (or corporation) wants to perform a sequence of actions that raises legal issues. For example, suppose you want to sell your house. That transaction has both real estate and tax aspects. If you give a lawyer information about your goals — do you want to reinvest the proceeds in another house? — and facts relevant to the situation — how long have you owned the house? do you live in it? do you have any other residence? — the lawyer will suggest a plan or plans for achieving your goals, taking into account the legal issues raised by the transaction.

In constructing plans, lawyers generally have two types of information to work with: statutes and cases. In some domains, such as contract law, there are no statutes, and the lawyer has only case law to work with. In others, there are statutory provisions that are so recent that no cases have been decided interpreting them. In general, however, both statutes and cases must be taken into account.

In practice, lawyers often take advantage of a third source of information, plans based on past experience of similar transactions. Often, however, no such plan is available, perhaps because the lawyer has never performed this particular type of transaction before, perhaps because the law has changed so recently that no plans are available. And even where there is such a plan, if it is challenged in court, it must be justified in terms of the statutes and case law.

In this thesis, therefore, I propose to examine the way in which plans are developed from statutes and cases. This process is illustrated in Figure 1.[1] Specifically, I propose to focus on tax planning. Almost every transaction has some tax aspect, so tax planning forms part of almost all legal planning; and the way in which the statutes and cases are used in this area is typical of general legal planning.

Statutes are rules that have been created formally, by legislation. They are published by the government and often by private companies as well. For example,

[1] This figure is based on a cartoon that appeared in the *New Yorker*, September 10, 1990, page 46.

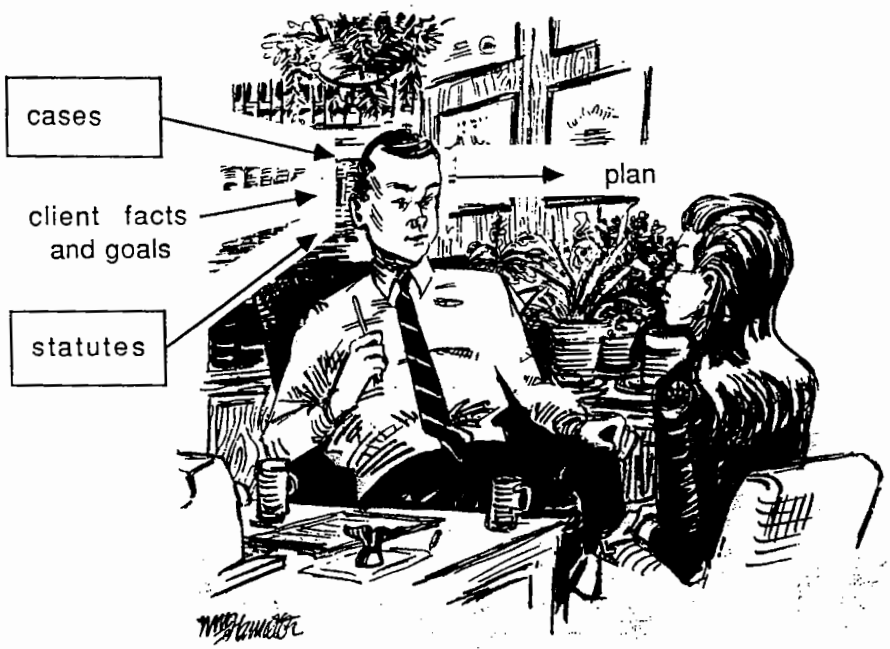

Figure 1: Legal planning.

§1034. Rollover of gain on sale of principal residence.
(a) Nonrecognition of gain.–If property (in this section called "old residence") used by the taxpayer as his principal residence is sold by him and, within a period beginning 2 years before the date of such sale and ending 2 years after such date, property (in this section called "new residence") is purchased and used by the taxpayer as his principal residence, gain (if any) from such sale shall be recognized only to the extent that the taxpayer's adjusted sales price (as defined in subsection (b)) of the old residence exceeds the taxpayer's cost of purchasing the new residence.

Figure 2: §1034(a) of the Internal Revenue Code.

consider §1034(a) of the Internal Revenue Code, governing the tax treatment of the income from the sale of a personal residence, given in Figure 2. The Internal Revenue Code is the most important statute for tax planning. It contains approximately 7000 sections.

Detailed as the Internal Revenue Code is, it still contains phrases that are not defined within the statute, for example, the phrase "principal residence" in §1034. To qualify for the benefit of §1034, a taxpayer must show, among other things, that he bought and sold properties which belong to this category. These phrases are defined partly by commonsense knowledge about the meaning of the words used and partly by example, not by statute. In determining whether a transaction satisfies a given statutory predicate, or planning a transaction that will satisfy it, a lawyer must use both rules and cases.

What makes reasoning about these statutory predicates difficult — and interesting — is that defining them is not just a matter of inferring defining characteristics from a set of examples. Generally speaking, there is no set of essential characteristics shared by all positive instances of the statutory predicate. Some examples are typical, and others are more or less similar to them along various dimensions. As a result, classifying a particular object as an instance or noninstance of one of these categories is not always a simple task.

This issue arises throughout legal reasoning, not just in tax. Indeed, it is part of a general natural language problem. Many ordinary categories, such as "tiger" or "cup," are surprisingly difficult to define. This indeterminacy has been studied in linguistics and philosophy, where it is labelled *open texture* [Waismann, 1965, Hart, 1961, Lakoff, 1987]. Any planning rule expressed in natural language, such as "be careful," "never get involved in a land war in Asia," or "buy low, sell high," suffers from the same problem.

In any domain, open-textured rules can be partially defined by examples. The

legal domain has the advantage that examples are recorded and published. Each court case is an example — an application of the law to a particular set of facts. Facts and results are recorded by the courts in "opinions" and published, both by the government and by private companies. Thousands of examples are readily available in any law library.

The facts of these examples can be used as the basis for new plans. Since the courts are bound by precedent, similar cases must be decided similarly. Thus, planners attempt to construct plans that are similar (or identical) to examples of previous successful plans.

In constructing a plan based on previous cases, a planner can make use of the court's reasoning. This must be included in the case report along with the facts and result. In a case interpreting a statute, for example, the court will suggest intermediate rules connecting the facts of the case to the open-textured statutory predicates in the statute.

These rules have some predictive value. The courts are likely to follow them in later cases. They are not required to do so, however. They are free to adopt new rules, as long as the new rules are consistent with the results in all previous cases. Thus, the courts' reasoning in previous cases is useful in constructing plans, but does not make the success of a plan certain.

For example, suppose you want to construct a plan for an academic who has just spent a year on sabbatical away from home and wants to sell his house, and you have §1034 and one case to work with. Suppose that the earlier case involved the following facts: John and Jane Smith bought a house in Providence, Rhode Island in June, 1980, and a second house in Chatham, Massachusetts in April, 1985. On September 3, 1988, they sold the house in Providence and bought a third house in Barrington, Rhode Island two days later. They are bankers at Hospital Trust. Before the sale, they and their children lived in the Providence home most of the year; now they live in the Barrington home most of the time. They stay in the Chatham home (on Cape Cod) for two weeks during the summer and occasional weekends during the rest of the year. Suppose the holding in that case was that the Providence home was their "principal residence" before the sale; the Barrington home became their principal residence after the sale; and so the income from the sale of the Providence house qualifies for deferral under §1034, and the court's reasoning was that if you have more than one residence, your "principal residence" is the one

where you spend most of your time.[2]

By the reasoning in the earlier case, your client's home is not his principal residence, since he has not spent any time there for the past year. On the other hand, if your case comes to court, you could argue that "principal residence" should be defined as, *e.g.*, the residence closest to your job, where you are registered to vote, and from which your tax returns are filed. Moreover, this rule would be consistent with the facts of the previous case, since the taxpayers there worked in Providence, not on the Cape. Therefore, a court would be free to adopt this rule and hold in favor of your client.

The use of open-textured rules and examples has a pervasive effect on legal reasoning. It makes the legal system flexible. The fact that terms like "principal residence" are underspecified means that the courts can respond to changing circumstances. For example, they can interpret "principal residence" to cover cooperatives and condominiums, even if those forms of ownership did not exist at the time the statute was passed. Similarly, the First Amendment protection of freedom of speech can be extended to cover television and radio, as well as newspapers.

On the other hand, because the system is flexible, it is also uncertain. In law, unlike domains such as chess, it is impossible to prove a plan correct. This uncertainty is not due to lack of factual information (we can assume complete knowledge of the facts); but to the underspecified nature of the rules. Given complete information about the client's situation and the relevant law, an experienced lawyer can give an opinion about the probability of success of a given plan, but even the most conservative plan is not certain to succeed.

This uncertainty may be an unavoidable part of the legal system. Our legal system is based in part on the deeply-held belief that it is not possible to specify in advance all the possible contingencies to which a statute should apply. Beginning in the early nineteenth century, the civil law countries, such as France, Italy, and the countries of South America, tried to design a different type of legal system. Motivated by distrust of the courts, they attempted to draft self-applying statutes: legislation so complete, detailed, and clear that its application would be a predictable, automatic process. To this day, these countries emphasize statutory law and give judges a less important role than they have here. Nevertheless, the attempt failed. It is still necessary for civil law courts to interpret statutes, to fill in gaps

[2]Like the other examples in this proposal, this one is strictly hypothetical, and not to be relied upon as tax advice.

and change the meaning of the text in response to changing conditions [Merryman, 1969].

Because there is uncertainty, there is room for argument. Lawyers are trained to find support for different conclusions in a given set of cases. A large part of a lawyer's training involves learning to make arguments for and against the application of some statutory predicate, and for and against the similarity of a previous case to the current one. Although one result may be more likely than another, it is generally possible to make these arguments in both directions. Contrast, for example, the situation of an engineer in San Francisco faced with the problem of predicting whether a given building will fall down during an earthquake. We can assume that, given perfect information about the building and the stresses imposed by an earthquake, plus perfect reasoning ability, all experts would reach the same conclusions. Lawyers, on the other hand, are not only permitted but expected to disagree.

Tax planning is no exception. Here, the adversaries are the taxpayers and the government. Taxpayers seek to exclude or defer items of income and deduct expenses, and the government seeks to include income and disallow deductions. Taxpayers must construct plans that are similar to favorable precedents and different from unfavorable ones in some relevant way. If the similarity is too distant, or the differences are too small, the plans will be vulnerable to challenge by the government.

The open-textured rules and examples interact in interesting ways. Being reminded of a similar case directs the lawyer's attention to the rules applied in that case; being reminded of a potentially applicable rule directs his or her attention to the cases interpreting that rule. In computational terms, rules limit the search through the case base. If you know what statutory rule you are interested in, it is easy to retrieve exactly those cases interpreting that particular rule. Similarly, cases limit the search through the statutory rules. If you know of a case similar to your own, the case report will indicate which rules were applied in that case.

In this thesis, I propose to test this theory of open-textured planning by designing and implementing CHIRON,[3] a system to solve simple problems in the domain of personal income tax planning. If successful, CHIRON will be able to construct plans that satisfy the tax laws for situations that are not identical to cases already in its case base. Moreover, it should use cases to guide its search through the rules and rules to guide its search through the cases.

[3] named for the centaur in Greek mythology, known for giving good advice.

CHIRON will handle the following benchmarks. They have been chosen in part because they are simple; most tax planners would agree on at least the obvious solutions. In addition, they illustrate problems such as timing and satisfaction of open-textured predicates that are typical of the domain. Finally, many of the benchmarks involve buying and selling houses, so the commonsense knowledge that must be formalized to handle those problems will involve the same cluster of concepts.

1. Input: The client does not own a house.
   Plan: Buy a house in order to get the mortgage deductions.

2. Input: The client owns a house and wants to sell it.
   Plans:

   (a) Physically occupy the house until you sell it, buy another house within 2 years before or after the date of sale, and physically occupy the second house as soon as you sell the old house or buy the new one, whichever comes second.

   (b) If you are over 55 years old and have never taken this option before, you can elect to exclude up to $100,000 of gain on the sale of a house. Once you take advantage of this provision, you will not be able to use it again.

   (c) Combine the first two plans: exclude part of the gain and reinvest the remainder in another house.

   (d) If the house is a business property, exchange it for property of like kind.

   (e) If the house is insured and happens to burn down, collect the insurance and use it to buy another house.

   (f) Give the house to charity and take a charitable deduction.

   (g) Give the house away without generating a deduction (A low tax but also a low-profit strategy. Rather than generating this plan, show how we avoid generating it).

3. The client wants to sell a house that he lived in two years ago, but has not occupied since then.
   Plans: Each of the above plans is worth considering. If they don't work, show why. In addition, suggest moving back into the house.

4. Input: The client is planning to sell his house at a loss.
   Plan: Sell before January 1.

5. Input: The client is planning to sell his house at a gain.
   Plan: Sell after December 31.

6. Input: The client is going on an extended business trip.
   Plan: Stay away from home less than a year — then travel expenses will be deductible.

7. Input: The client wants to take some courses.
   Plan: Take courses that are related to your job; then you can deduct the tuition.

8. Input: The client is receiving money from his employer.
   Plans:

   (a) Show that it was a gift, not salary.

   (b) Show that it was a loan, not salary.

9. Input: The client is receiving money from his employer. His employer is a company of which he owns all the stock.
   Plan: Show that the money is salary, not a dividend.

10. Input: The client is embezzling money from his employer. Plans:

    (a) Show that it was a gift, not salary.

    (b) Show that it was a loan, not salary.

11. Input: The client is going to receive payments as part of a divorce settlement.
    Plan: Call those payments child support, not alimony.

12. Input: The client is going to make payments as part of a divorce settlement.
    Plan: Call those payments alimony, not child support.

In Section 2 of this proposal, I will place open-textured planning in the context of past artificial intelligence work in both law and planning. In Section 3, I will discuss CHIRON's design; in Section 4, I will discuss the knowledge representation language used and the representation of rules and cases; in Section 5, I will illustrate CHIRON's design and the knowledge representation with a specific example; and in Section 6, I will summarize what I have done, outline the work that remains to be done on this thesis, and discuss ways of validating the results.

```
(corporation Chrysler t1)
(issue Chrysler s1 t1)
(stock s1 t1)
(common s1 t1)
(piece-of p1 s1 t1)
(nshares p1 100 t1)
(own Iacocca p1 t1)
```

Figure 3: How TAXMAN would represent "Iacocca owns 100 shares of Chrysler stock."

2 Previous work

2.1 Artificial intelligence and law

In the past few years, there has been a growing amount of work in artificial intelligence and law. Here, I will focus on recent work and work that is closely related to this thesis; for more details on earlier work, see the thorough and readable survey in [Gardner, 1987].

2.1.1 McCarty

McCarty's work has been almost exclusively in the area of knowledge representation. His first project was TAXMAN, a program that analyzed cases in the domain of corporate tax law. For this project, he designed a representation language using an economical set of statutory predicates that was sufficiently expressive to state the facts of an input case in this domain in detail [McCarty, 1977]. For example, see how TAXMAN would represent "Iacocca owns 100 shares of Chrysler stock" (Figure 3).

TAXMAN took as input the description of a transaction in a specialized area of law, corporate reorganizations, and, upon request from the user, determined whether the transaction qualified for tax-free treatment under certain provisions of the Internal Revenue Code. First, it processed the input facts in order, using forward chaining rules to expand them to a greater level of detail; then it used backwards chaining rules to determine whether the expanded facts satisfied the relevant provisions of the Internal Revenue Code. The system moved flexibly back and forth between concrete descriptions and abstractions.

TAXMAN was limited by the fact that all of its abstractions were defined by rules. This works fairly well for some simple inferences, such as determining that someone

is a stockholder. For an example of such a rule, see Figure 4. Like other areas of law, however, corporate tax involves open-textured concepts. Important concepts that are not defined in the statute include "business purpose" and "step transaction." McCarty concluded that TAXMAN's rules were insufficient for representing these concepts.

In his next project, TAXMAN II, McCarty began investigating ways of modifying TAXMAN to handle open-textured concepts [McCarty, 1980, McCarty and Sridharan, 1982]. One approach might have been to add cases to the knowledge base. Examples of "business purpose" and "step transaction" are given in various cases (see, *e.g.*, *Gregory v. Helvering,* 293 U.S. 465 (1935), and *Helvering v. Elkhorn Coal Co.*, 95 F. 2d 732 (4th Cir. 1938), *cert. denied,* 305 U.S. 605 (1938)). If TAXMAN had some facility for representing or reasoning with cases, it might be able to make use of these examples.

But McCarty did not add cases, at least not directly. Instead, he proposed to represent open-textured concepts using a *prototype*, a concrete description expressed in the lower-level representation language, and a sequence of *deformations*, or transformations of one concrete description into another. For example, a stockholding relationship could be represented by a pointer to the prototype of a pure equity interest, represented by a particular set of rights and obligations between the owner and the issuing corporation; plus an incremental set of transformations in the direction of a debt interest. The detail and precision of his low-level representation language make it effective for representing such small incremental changes.

This prototype-and-deformation approach predates case-based reasoning, to which it bears a striking resemblance (see discussion below). McCarty's contributions are first, the idea that a set of related cases could be used to represent an open-textured concept; and second, the suggestion that the set of possible transformations of a case could be limited to those which preserve *conceptual coherence* in the corresponding concept [McCarty, 1980, McCarty, 1989a]. This term is not defined precisely, but it suggests an interesting direction for future work.

Unfortunately, TAXMAN II was never implemented. As a result, McCarty never faced the significant algorithmic issues addressed by the case-based reasoning community: how to choose the prototypes, how to index them, how to search the space of prototypes, how to search the space of transformations, and the relationship of the prototypes to actual cases.

In the course of this project, McCarty developed a deontic logic for representing

```
(theorem abstract (o c s p)
 (stockholder ?o ?c)
    (goal (issue ?c ?s))
    (goal (stock ?s))
    (goal (piece-of ?p ?s))
    (goal (own ?o ?p)))
```

Figure 4: A backwards chaining rule from TAXMAN: an individual is a stockholder if some company has issued stock and he owns a piece of that stock.

the concepts of permissions and obligations that occur in the legal domain [McCarty and Sridharan, 1982, McCarty, 1986, McCarty, 1983]. More recently, he has elaborated a knowledge representation language with a formal intuitionistic semantics that incorporates time, events, and actions, as well as permissions and obligations [McCarty, 1989b, McCarty, 1989a].

In joint work with Dean Schlobohm, an estate planning attorney, McCarty has sketched out a design for a legal planning system [Schlobohm and McCarty, 1989]. They argue that lawyers construct plans by retrieving *prototype plans* and transforming them to meet the clients' goals. They discuss how trusts and the Internal Revenue Code can be represented using McCarty's representation language. However, as with TAXMAN II, algorithmic issues such as how the prototypes are chosen, indexing, and search are ignored. And again, there is no explicit facility for representing or reasoning with legal cases.

2.1.2 Gardner

Gardner has also been concerned with the open-textured statutory predicate problem. Her system, GP[4] [Gardner, 1987], like TAXMAN, takes a sequence of events as input and determines whether that sequence satisfies certain legal requirements. GP's domain is contract law, but the problem is essentially the same.

Gardner's solution is quite different from McCarty's, however. Like McCarty, she starts with a rule-based system. Unlike McCarty, she uses cases. When processing an input case, GP fires its rules until they "run out," *i.e.*, until it reaches a rule that contains a term that is not expanded by some further rule. Then it looks at the commonsense meaning of the term and past cases whose facts match the

[4] "Gardner's program." So called for convenience, since Gardner did not name her system.

current case. Where the commonsense meaning and past cases are all consistent, GP classifies the case accordingly, either in or outside of the term in question. Where no commonsense meaning is given and there are no past cases on point, or where the past cases disagree with each other, the input is considered a "hard case" and GP leaves its classification to the user.

This combination of rules and cases is clearly a step in the right direction. The system is limited by its case representation, however. For each past case, Gardner determined which open-textured statutory predicates were involved in the case and which of the facts set forth in the case report were relevant to the satisfaction or nonsatisfaction of each open-textured statutory predicate. An open-textured statutory predicate and the facts relevant to it form a "case pattern," and cases are represented as a list of case patterns.

Each of these case patterns involves a judgment call — the system designer's decision as to which of the facts set forth in the case report are relevant to the application of a given statutory predicate. When analyzing an input case with regard to a particular open-textured statutory predicate, GP retrieves the cases involving that statutory predicate whose facts match the facts of the statutory predicate's case pattern. If too few facts are included, it will be too easy for a new case to match the prior case; if too many, the system will predict failure where it should have found a match. Moreover, there might be facts included in the official case report that at the time of representation do not appear to be relevant to any particular statutory predicate. Apparently, those would not be included at all.

The strength of this representation is that it starts with the facts. The facts of the current situation are compared with the facts of past cases. On the down side, some facts are not represented; no attempt is made to represent levels of abstraction between the facts and the statutory predicates; and there is no representation of the court's argument. Moreover, GP has no information about case transformations, such as those suggested by McCarty. As a result, it is hard for GP to compare and contrast past cases, and it has no mechanism for handling novel situations. It cannot modify an earlier case to fit a new situation or resolve conflict between cases. Cases are used to determine that a problem exists, but not as a basis for a solution.

2.1.3 HYPO, TAX-HYPO, AND CABARET

Edwina Rissland and her group at the University of Massachusetts have worked on a series of projects in the area of artificial intelligence and law, emphasizing

particularly algorithmic and control issues.

HYPO illustrates a use of cases that is different from Gardner's [Ashley, 1988]. Like TAXMAN and GP, HYPO takes a sequence of facts as input. As in the earlier systems, the problem is to determine whether the facts satisfy certain legal requirements (in this case, whether they constitute a trade secrets violation). Unlike the earlier systems, however, HYPO does not output a yes or no answer. Instead, it generates arguments based on previous cases.

User input and cases are both represented in simplified form, using a standard "legal case-frame" to hold the important facts of the case. Legal case frames also include such information as the date of the decision, the court deciding the case, and the official citation. Each of the cases in HYPO's case base is stored using a fixed set of indices (termed "dimensions"). When the user inputs a legal situation, expressed in the legal case-frame language, HYPO uses this representation to calculate the dimension values for the situation, and then uses these values to index into the most similar cases. The system then organizes the retrieved cases into a subset lattice based on the dimensions each retrieved case shares with the current situation and uses the lattice in constructing arguments for both plaintiff and defendant. Arguments are constructed by reciting the dimensions shared with favorable cases and the differences from unfavorable cases (nonshared dimensions or differences in the values of shared dimensions).

Suppose our situation is the following:

> Company D develops a line of home-style cookies. Company N places one of its employees as a "spy" (gets him hired) at Company D. He learns the secret method, then returns to Company N and sets up the technology for making home-style cookies, which Company N markets.

We distill this down to the salient dimensional information: Company D develops secret method, Company N gains competitive advantage by having lower development costs (due to stealing method), Company N and Company D are competitors. Using this information, we determine the case includes zero *secrets-voluntarily-disclosed* and positive *competitive-advantage-gained* and use this to index to *Telex v. IBM*, a case where IBM was successfully sued for hiring a Telex employee for big money and using his development notes to produce a competing product for less money, and any other cases sharing one of these dimensions. For simplicity, suppose *Telex v. IBM* is the only case retrieved. Then we construct an argument for the plaintiff D by reciting the dimensions D's situation shares with *Telex v. IBM*, and for the

defendant N by focusing on nonshared dimensions or differences in the values of shared dimensions (e.g., the number of people to whom information was disclosed) that make N's case stronger than IBM's.

HYPO's case representation does not include all of the facts in the official case report. It is more detailed than GP's, but much simpler than TAXMAN's representation language would permit. HYPO's representation abstracts from the facts as stated in the judge's opinion (the official description of the case) to the legal case-frames, and from the legal case-frames to the dimensions. The legal case-frames have a fixed number of slots, and there are only thirteen dimensions. By the time cases have been boiled down to dimensions, they have been substantially simplified.

The advantage of this approach is that simplifying cases makes it easy to compare and contrast them and propose hypotheticals. The dimensions are carefully chosen, and their significance is firmly grounded in domain knowledge. The cases that are most similar on these dimensions are likely to be most useful in arguing the current case. For example, in the trade secrets area, we have dimensions for *secrets-voluntarily-disclosed, disclosures-subject-to-restriction,* and *competitive-advantage-gained* — the values of these tend to predict how strong a trade secrets case is for the plaintiff and defendant and can be used as a point of comparison with other similar cases.

At the same time, however, simplifying cases limits HYPO's ability to compare and contrast them. The system can only reason at one level of abstraction. It cannot access more specific information to distinguish unfavorable cases, or formulate new abstractions to show that favorable cases are similar. In comparing and contrasting cases, HYPO looks only at the dimensions.

For example, suppose we have a case where the defendant has obtained the plaintiff's customer list. A piece of information is not a trade secret if it's a matter of general knowledge in an industry. Suppose the plaintiff is selling spaceships, and there is a previous case, otherwise similar, where the plaintiff sold magazine subscriptions. The defendant's lawyer might want to distinguish the earlier case on the grounds that the plaintiff there was involved in a different line of business. The list of spaceship buyers is probably short and obvious to anyone in the business; a list of magazine subscribers can be a valuable (and marketable) item. But HYPO would be unable to make this distinction. There is no dimension or legal case-frame slot for "type of business." There is a legal case-frame slot for the product manufactured by the parties, but that is not reflected in the dimensions.

HYPO does not incorporate rules explicitly, but it is addressing the same open-textured statutory predicate problem as TAXMAN II and GP. In effect, the whole system can be seen as using cases to interpret a single open-textured statutory predicate, "trade secret violation." GP addressed the question of how to distinguish between easy and hard questions of interpretation; HYPO provides an algorithm for dealing with hard questions. In borderline cases, this work suggests (correctly) that the appropriate response is not to look for a yes or no answer, but to generate arguments by comparing and contrasting past examples of a particular statutory predicate.

In their next project, Rissland's group translated HYPO into a new domain. TAX-HYPO takes as input a fact situation expressed in legal case-frames and applies a HYPO-style dimensional analysis repeatedly to determine whether the situation satisfies each of the statutory requirements for a home office deduction under §280 of the Internal Revenue Code [Rissland and Skalak, 1989a].

TAX-HYPO demonstrated that HYPO's techniques could be used to compare and contrast cases and generate arguments with respect to individual statutory predicates in a legal domain other than trade secrets law. However, this experiment revealed that additional knowledge is necessary to reason in a statutory domain: TAX-HYPO was unable to combine its arguments concerning individual statutory predicates into an argument for or against the application of an entire statutory provision. It could only argue for or against the application of specific phrases within the provision.

As a result, Rissland's group has been exploring ways of combining case-based with other forms of reasoning. Their most recent project, CABARET [Rissland and Skalak, 1989a, Skalak, 1989a, Skalak, 1989b, Rissland and Skalak, 1989c, Rissland and Skalak, 1989b], includes three modules: a case-based reasoner, a "co-reasoner," and a separate control module. The case-based reasoner is modelled on HYPO. The "co-reasoner" is currently a simple rule-based module including both forward and backwards-chaining. The control module uses heuristics to add, delete, or order tasks on a common agenda.

The main contribution of this project is in the area of controlling mixed case-based and rule-based systems. Rissland's group considered various possible structures:

- rule-based reasoning dominant: the rule-based module can call the case-based module, but not vice versa;

- case-based reasoning dominant;
- equal modules, each of which can call the other; and
- equal modules, with control knowledge isolated in a separate module.

They adopted the last of these, which allows them to simulate each of the other three and in addition, facilitates experimentation with a variety of control heuristics [Skalak, 1989a].

Heuristics are production rules that add, delete, or order tasks on the system agenda. The tasks on the agenda are calls to a procedure exported by one of the other modules, such as "backchain on subgoal *foo*." Heuristics they have considered include, among others:

1. begin analyzing a problem by using the case-based reasoner to find a similar case;
2. begin analyzing a problem by backchaining through the rules until they run out;
3. if one module fails, switch to the other; and
4. once a conclusion is reached, double-check it by using the other form of reasoning.

TAXMAN and GP use the second of these heuristics: they are both systems where rule-based reasoning is dominant (TAXMAN of course does not use cases at all). GREBE (below) always tries both case-based and rule-based reasoning, generates all the solutions it can find, and uses heuristics afterwards to choose among them [Branting, 1990a]. To date, only CABARET has examined a more complex form of control.

So far, CABARET's experiment with control heuristics is only preliminary; no detailed results have been obtained. The system is designed to be module-independent, so the control heuristics are generalized. They do not attempt to take advantage of the specific features of particular types of modules. In addition, the system is limited, like HYPO, by its simplified case representation. Finally, in natural language terms, it is a parser, not a generator: it takes facts and determines whether they satisfy a statutory predicate, rather than taking the predicate and constructing a set of facts to satisfy it.

2.1.4 GREBE

GREBE [Branting, 1988, Branting, 1989a, Branting, 1989b, Branting, 1989c, Branting, 1990a], like HYPO, generates arguments for the satisfaction or nonsatisfaction of a given legal statutory predicate. Unlike HYPO, GREBE has a detailed case representation and uses rules as well as cases. In addition, its arguments are based on the reasoning used in previous cases, rather than a comparison of their facts.

GREBE's approach is similar to GP's, but it significantly extends Gardner's work. GREBE uses a more complex case representation: a semantic net in which individual facts correspond to relation/unit/value triples and the facts of an entire case correspond to a labelled graph. Labelled subgraphs of this graph correspond to open-textured statutory predicates involved in the case: the subgraphs correspond to the facts used by the court to explain its result with respect to that statutory predicate (the *criterial facts* for that statutory predicate). The subgraph-statutory predicate pairs are similar to the rules GP uses to represent cases, but GP's case representation is a bundle of separate rules; here, all the facts of a particular case are joined into a larger graph. Moreover, it is possible to represent facts that are mentioned in the case report, but not cited as relevant to any open-textured statutory predicates, as part of the case graph not covered by any of the labelled subgraphs.

GREBE takes as input a case and a desired statutory predicate in the domain of workers' compensation and outputs an argument supporting the application of that statutory predicate in the given case. First, it tries to find the desired conclusion in the input facts. If that fails, it backwards-chains through its rules like TAXMAN or GP, attempting to show that the desired conclusion is the consequent of some rule, all the antecedents of which are themselves satisfied. Like GP, GREBE turns to cases if its rules run out. Unlike GP, GREBE can handle partial matches. Instead of looking for cases which exactly match the current situation, it looks for the best match (determined by mapping the criterial facts of previous cases involving the desired statutory predicate onto the current situation, and choosing the case with the fewest unmatched facts). If both positive and negative cases exceed a certain (arbitrary) threshold, GREBE constructs arguments in both directions. Finally, if the previous case is a partial match, GREBE calls itself recursively to infer any missing facts.[5]

[5]For clarity, the flow of control has been simplified here: in fact, GREBE always tries to solve its goals using *both* rules and cases, generates all the solutions it can find, and uses heuristics afterwards to choose among them.

GREBE uses the richest knowledge representation of all the systems discussed (other than TAXMAN II, which was never implemented). In addition, it can construct new arguments, by piecing together elements of different past cases and domain rules. Moreover, this was the first system to attempt to represent and use the reasoning from past cases. However, nothing comes for free. Using a richer knowledge representation has several disadvantages: first, representing cases is time-consuming and difficult. A feature vector representation (which is essentially what HYPO uses) makes it easier to translate from case report to representation. Second, a richer representation language makes it harder to enforce consistency in the case representations. In addition, the use of the court's reasoning involves judgment calls. The exact scope of the criterial facts and the precise logical connection between various reasoning steps are seldom completely explicit in legal opinions. Perhaps most critically, the process of matching cases becomes equivalent to graph isomorphism, rather than simple vector comparisons.

2.2 Planning

There is a long tradition of work on planning in artificial intelligence. In this section, I will give a brief outline of that work in order to put CHIRON in context.

For purposes of this proposal, a *planner* is a program that takes a goal or goals as input, along with an initial state description, and outputs a sequence of actions, or *plan*. The plan is correct if the program user (human or robot) can execute it, starting in the initial state and after execution, the goals have been achieved. The actions should be *primitive actions*, *i.e.*, actions that can be executed by the program user.

Planning can be viewed as a search through the space of possible plans. Since this search space is exponential in the number of primitive actions known to the system, much of the work in planning has been aimed at controlling search. Another key problem is *projection*: reasoning about the order in which actions can be performed and their effects, in order to determine whether a sequence of actions can be performed in the given order and if performed, whether the goals will be true in the resulting state.

Early work in planning was primarily linear: that is, an intermediate plan consisted of a sequence of primitive actions, and plans were constructed by adding steps to this prefix [Fikes and Nilsson, 1971]. Thus, for example, suppose you normally have a cup of coffee for breakfast. A linear planner with the goal, "have breakfast,"

would start with a primitive action, perhaps, "get filter," then select another one that can be performed after the first one, perhaps "put filter in basket," until it found a sequence that resulted in a cup of fresh coffee. This approach is receptive to formal treatment [Lifschitz, 1987]. However, it fails to take advantage of the structure that can be provided by abstractions.

In hierarchical planning, by contrast, intermediate plans are complete but abstract. Instead of simply adding primitive actions to an intermediate plan, the planner repeatedly refines the plan, gradually substituting more and more specific plans until it has a sequence of primitive actions. (See, *e.g.*, [Sacerdoti, 1974]). Working on the above example, a hierarchical planner might start with the plan, "Make breakfast," then substitute "Make coffee," then "Get filter, get coffee, put coffee in filter, put filter in basket, ..." and so forth. The planner can delay ordering the subtasks, as well as adding details, until sufficient information is available.

Neither of these approaches is sufficient for our task. If we start with primitive actions, as in linear planning, we have no way of testing whether they satisfy the open-textured rules; and if we start with abstractions, we will be unable to refine the plan completely. Without cases, we have no way of making the connection between abstractions, based on open-textured rules, and primitive actions.

Recently, some work has been done on case-based planning — using previous plans as the basis for constructing new ones. Case-based planning is part of a more general research effort known as case-based reasoning (CBR). CBR has attracted a good deal of attention in recent years; it has been incorporated into a number of systems that perform a variety of tasks in a variety of domains. It is characterized by its use of memory in problem-solving. Where possible, CBR systems rely upon "memories" of past experiences to solve problems, rather than deducing the answer from first principles. The "memories" stored in the system's knowledge base are referred to as *cases*.[6]

By making use of past cases, CBR systems can gain in both efficiency and reasoning power. Even in a simple blocks world domain, if you've solved the same problem before, it's easier to look up the solution than to figure it out again; and in a more complex domain, where your knowledge is incomplete, you can create plans

[6]The use of the term *case* in both CBR and law has caused some confusion. It is important to distinguish between the two. Law cases, as described above, consist of a statement of facts, in which all the relevant facts are included (by definition); a result; and a suggested chain of reasoning connecting the facts and the result. CBR cases include facts and result. Generally speaking, there is no reasoning connecting the two, and there is no guarantee that all the relevant facts are present.

based on experience when it would be impossible to deduce them logically. For example, a child can plan to turn on the light by flipping the light switch without any understanding of the electrical wiring in the building or the physics of electricity, simply because it knows that this plan has usually worked in the past. By extension, when the child wants to turn on another device, it may adapt its previous plan and look for an on-off switch, again without any understanding of how the device works.

CBR is an attractive candidate for use in a legal reasoning system. In law, there is no strong theory connecting facts to open-textured statutory predicates. Traditional deductive approaches are not effective. CBR in general, and case-based planning in particular, are especially well-suited to domains where the reasoner has a weak causal theory.

Certain issues must be addressed by any CBR system:

1. What do you know? What cases are contained in memory, how are they represented, and what kind of information do they include?
2. How can you find it? How are cases indexed and retrieved?
3. How can you make it useful? How are cases adapted and repaired for use in the current situation?
4. How can you evaluate the output of the system?

For example, consider the case-based planner CHEF, which constructs plans, or recipes, in the cooking domain [Hammond, 1986b]. It takes as input the user's goals, *i.e.*, the desired tastes, textures, and types of dishes, and uses its case base of previous cooking experiences to construct a recipe that satisfies those goals.

CHEF started with an initial library of ten recipes and created twenty-one others in response to user requests. Each plan includes a list of ingredients, a list of steps, and a type (*e.g.*, stir-fry or souffle.) A sample recipe is given in Figure 5.

The issue of what to include in a case is not addressed explicitly in CHEF. CHEF's recipes, like the one given in Figure 5, include the kind of information that would be found in a cookbook, *i.e.*, the ingredients and the steps to be performed. They do not include, for example, the name of the store where the ingredients were bought, their price, the utensils used, or the height above sea-level of the kitchen.

In theory, a case should include all the information that will be useful in constructing future plans. Arguably, therefore, the case descriptions in CHEF should

```
(def:rec broccoli-with-tofu
  (ingredients
     ingr1 (tofu lb .5)
     ingr2 (soy-sauce tablespoon 2)
     ingr3 (rice-wine spoon 1)
     ingr4 (corn-starch tablespoon .5)
     ingr5 (sugar spoon 1)
     ingr6 (broccoli lb 1)
     ingr7 (r-pepper piece 6))
  (actions
     act1 (chop object (ingr1) size (chunk))
     act2 (marinade object (result act1)
                   in (& (ingr2) (ingr3) (ingr4) (ingr5))
                   time (20))
     act3 (chop object (ingr6) size (chunk))
     act4 (stir-fry object (& (result act2) (ingr7)) time (1))
     act5 (add object (result act3) to (result act4))
     act6 (stir-fry object (result act5) time (2)))
  (style style-stir-fry))
```

Figure 5: CHEF's recipe for broccoli with tofu.

have been much more complete. On the other hand, it's inefficient to include information that will never be used. In cooking, as in most domains, it is impossible to be certain which facts will be useful; the system designer can only make an informed guess at the time each case is represented.

In this respect, law differs from most other domains. In law, previous plans are described in the statement of facts in the official case report. Any of these facts, and only these facts, can be used in future arguments. Thus, the system designer can be sure that these facts should be included in the case representation.

Suppose you have decided what to include in your cases. The next question is, how do you find what you need? Or, in planning, how can you find a plan that can be used (possibly with modifications) to help you achieve your current goals?

The simplest solution is a linear search through the case base. But this is too inefficient. In systems such as CHEF, with a case base of ten to thirty cases, an exhaustive search would be possible. But in a system that is even slightly larger, this becomes impractical. A cook is unlikely to read through every page of a cookbook looking for a particular recipe. And in a case base the size of a law library, it would

be impossible. Thousands of cases are decided every day by the state and federal courts in this country.

To cut down on search time, the case base must be indexed in some way. Lawyers have faced this issue in practice. In response, they have developed more and more sophisticated ways of indexing case law. In addition to subject indexes and treatises on particular fields of law, for example, there are services that list every case that cites a given case, and every case that cites those cases, and so on. Now there are computer databases that contain the text of reported decisions, handed down as recently as the past couple of days. Lawyers search the databases by designing regular expressions, making it possible to find, for example, all Massachusetts decisions between 1963 and 1965 involving basketball players from Indiana, or any other combination of facts from almost any jurisdiction.

CBR has addressed the indexing problem in theory, as well as in practice. The main questions that must be answered by anyone indexing a case base are:

- What features should a case be indexed under?
- Should they be features found in the case description, or abstractions from those features, or both?
- Should they be ordered? If so, how?
- Should the feature set be static or dynamic?

The features used for indexing should be the ones that will allow you to retrieve a case in the future when you need it. In planning, you start with a goal. So one reasonable solution is to index plans under the goals that they achieve. CHEF, for example, would index the above recipe under the goals, "make stir-fry" and "include broccoli." In law, this might correspond to indexing plans under "sell house" or "satisfy the statutory predicate *principal residence.*"

One planner's side effects are another planner's goals, however. Any action performed or any state achieved during the execution of a plan might be considered a goal. For example, CHEF considers including a certain ingredient or making a certain type of dish to be goals. Performing a certain type of action is not considered a goal, so recipes are not indexed under the actions they include. But suppose that you have entered a cooking contest that requires you to demonstrate certain skills. You might be looking for a recipe that involves, say, separating eggs. If you asked CHEF for such a recipe, it would be unable to retrieve one, even though it has one

(a souffle recipe) in its case base. Or suppose you're looking for a dinner that you can cook in less than an hour. Again, you could not retrieve one, since recipes are not indexed under the time they take.

Thus, the choice of feature set involves a tradeoff between efficiency and reasoning power. The ideal feature set would limit search through the case base, but not so much that the system fails to retrieve cases it could use. The same tradeoff must be faced by the designer of a legal reasoning system.

The question of whether or not to include abstractions has caused a certain amount of controversy in CBR [Hammond, 1989, Waltz, 1989]. CHEF indexes plans under both the ingredients included and abstractions of those ingredients, in order to allow partial matches. Thus, the above recipe is indexed under both "include broccoli" and "include vegetable," and it can be retrieved in response to a request for "a stir-fry recipe with green beans" as well as "a stir-fry recipe with broccoli," since green beans and broccoli are both vegetables.

In law, it seems clear that you want to include both surface features and abstractions. This corresponds to the difference between indexing under the actual words of the court's opinion, as the current computer databases do, and under abstractions, as HYPO does [Ashley, 1988]. Ideally, a system would use both.

A more difficult problem, not addressed in CHEF, is how to choose the abstractions. Note that the broccoli and tofu recipe is indexed under "stir-fry with tofu and vegetable," not "stir-fry with tofu and something green." Presumably the intuition is that type of food is relevant to a cooking plan, and color is not (except on Saint Patrick's Day). But this raises a difficult theoretical issue, the idea of relevance, which has not been discussed in the CBR literature.

Because legal plans are more complex, it will be hard to avoid facing this question explicitly in a legal reasoning system. On the other hand, the law of evidence and legal notions of relevance might shed some light on this problem that would be of use to CBR generally.

CHEF orders its indices. This simplifies search — plans are indexed in a discrimination net — but the ordering is fairly arbitrary. Type of dish is considered to be more important than ingredients, for example. Thus, the above recipe can only be retrieved if you know you are looking for a stir-fry dish. If you have broccoli and tofu in the refrigerator and want to know what to make for dinner, you will not be able to access this recipe. In law, as in cooking, there is no obvious ordering. As with the choice of features, there will be a tradeoff between the efficiency provided

by ordering and the corresponding loss of reasoning power.

Finally, CHEF uses a static set of indices. Static indexing is easier to implement, and as a result, has been used in most CBR systems. It makes the initial choice of features, abstractions, and ordering more important, however. Dynamic indexing gives the system the flexibility to solve problems involving a demand for information that is not originally accessible. In CHEF, for example, information about ingredients is included in the case base. Thus, if CHEF had dynamic indexing, and it was presented with a request for a recipe using broccoli and tofu, it could identify the request as a request for information that might be in its case base. Finding that the information is not accessible at the top level, it could modify its indexing and retrieve an appropriate case. With static indexing, this is not possible. There is a tradeoff between ease of implementation and flexibility; again, this is a problem that legal reasoning systems will share with systems in other domains.

Once you have decided what to include in the case base and how to find it, the next question that must be answered by anyone designing a CBR system is, how can I use this information? Or, in planning, how can I use the retrieved plan (with modifications, if necessary) to achieve my current goal? At one extreme, if no adaptation is involved at all, CBR reduces to table lookup; the system can only solve problems that are exactly the same as those it has seen before. At the other extreme, it has been suggested that in some domains unlimited adaptation might be allowed [Riesbeck and Schank, 1989, page 42]. In other words, the system could generate a solution from any given case for any problem. But if a solution can be generated from any case, the system does not benefit from having a case base. In fact, if you can generate a solution from any given case, you can generate one from scratch, so a system with unlimited adaptations would share the computational problems of hierarchical planning.

In CHEF, old plans are modified in relatively simple ways, such as the substitution of an ingredient of the same type. Thus, CHEF could generate a plan for a raspberry souffle from its strawberry souffle recipe, simply by substituting raspberries for strawberries. In law, this might correspond to taking the facts of an old tax case and substituting in the name and address of your client.

Often, more complex modifications will be necessary. In addition to simple object substitutions, these might include adding, deleting, or reordering plan steps. Relative to indexing, little attention has been given to adaptation in CBR. What is needed is some principled way of characterizing the set of allowable transforma-

tions and the order in which they are to be applied. In law, we can examine the transformations applied by the courts in their reasoning. Because this reasoning is recorded, it provides a source of material relating to the adaptation problem that might be of general use.

In case-based planning, since the planner doesn't have a detailed domain theory, it is impossible to prove that the planner's output is correct. So how do you evaluate it? One alternative is to try the plan out and see how it works. And this is what CHEF does. After it generates a recipe, it executes it (in a simulator), repairs it if necessary, and stores the final recipe in its case base.

For example, CHEF's recipe for beef and green beans allows the meat and vegetables to be stir-fried simultaneously. A simple substitution of broccoli into this recipe yields a recipe for beef and broccoli stir-fry. However, when the recipe is executed in the simulator, CHEF discovers that the broccoli is soggy. It deduces (correctly) that this error is caused by cooking the broccoli with the beef, repairs the recipe by separating the cook-broccoli and cook-beef steps, and stores the repaired recipe under "soggy broccoli," the problem it is designed to avoid, as well as under "stir-fry with beef and broccoli."

But in law, the planner can neither prove a plan correct nor try it out. The equivalent of executing a plan would be having a court pass judgment on it. The courts do not have the time to rule on every plan, and even if they did, most planners would be unwilling to accept the cost and uncertainty of a trial. Legal reasoning systems will have to develop new approaches to validation. I will discuss this issue further in Section 6.

In summary, CBR provides many insights for the designer of a legal reasoning system. It clarifies many of the issues involved in case representation, indexing, and adaptation. On the other hand, CBR has several limitations. It provides no role for legal rules. It makes no direct use of the reasoning given in legal opinions. Most CBR systems have based their output on a single "best" case selected by the retriever, rather than using a set of relevant past cases, as a lawyer would be likely to do. Moreover, legal cases are temporally ordered, and it is sometimes possible to project trends and use those projections to construct a plan. While this is not outside the scope of case-based reasoning, no CBR system has yet attempted it. Thus, CBR provides at most a partial solution to our problem.

3 CHIRON: design

In this section, I will outline the design of CHIRON. The hypothetical user of this system is a lawyer, who will input a statement of the client's goals and facts about the client's current situation. CHIRON will coordinate rules and cases to produce simple tax plans for achieving the client's goals. These plans should seem reasonable to the lawyer; ideally, they would include plans the lawyer had not already thought of, as well as the obvious solutions.

The previous work in planning discussed in Section 2 suggests two possible tools: a hierarchical planner and a case-based planner. As discussed above, neither of these would be sufficient on its own to solve our problem. In CHIRON, I propose to experiment with a hybrid system, incorporating both a hierarchical and a case-based module. These modules will be tightly coupled: the hierarchical planner will call the case-based module for guidance in choosing transformations and refining the open-textured predicates that remain in plans when the transformations run out, and the case-based planner will use the hierarchical planner to help with indexing and controlling adaptations.

Contrast GP and CABARET, both of which use loosely-coupled modules. GP fires its rules until they run out, and only then looks at cases. In the context of planning, this approach would mean having the hierarchical planner refine the plan as far as possible and then hand it over to the case-based module. The two modules would not interact except at that point. CABARET was designed to be module-independent, so that a case-based reasoner could be combined with, say, a rule-based or a model-based reasoner. As a result, most of its control heuristics are independent of the types of modules involved: for example, "if one mode of reasoning fails, switch to the other," or "once a conclusion is reached, try the other mode of reasoning to confirm the answer." In CHIRON, the hierarchical and case-based planners will be more interdependent than GP's modules, and the control heuristics will be more specific than CABARET's.

After reading in the client goals and background facts input by the lawyer and determining that there may be tax issues involved, CHIRON's hierarchical planner will construct an abstract plan and consult the case-based planner for guidance in choosing a transformation. The case-based module will retrieve cases with similar facts. If CHIRON has solved the same problem before, the case-based module will return that solution; otherwise it will compile a list of the transformation sequences that the hierarchical planner would have used to reach the similar cases in the case

base. If there is only one such sequence, it will construct the plan. Otherwise, it will return the list to the hierarchical planner. The hierarchical planner will choose a transformation, refine the plan, and repeat the process until no further transformations are applicable. Then it will call the case-based planner to refine any open-textured predicates remaining in the plan. The case-based planner will retrieve all the cases in which taxpayers attempted to execute that plan and construct a plan based on all of those cases, if possible.

Using multiple cases in planning will be one of CHIRON's contributions. Most case-based reasoning systems have based their output — plans, diagnoses, explanations, or whatever — on a single "best" case. The use of a group of cases is characteristic of the legal domain, however. Arguments or plans are made after considering the current situation in relation to all of the past relevant cases. As a result, legal reasoning systems have generally used multiple cases. HYPO and CABARET construct arguments by comparing the current situation to a group of past cases; GREBE selects a single case to solve each part of its problem and pieces together the answers into a single argument.

How can we construct a plan from multiple cases? McCarty's idea of prototypes and deformations suggests an answer. As discussed in Section 2, he proposed representing each open-textured concept or plan using a *prototype*, a concrete description of some particular instance, and a sequence of *deformations*, or transformations of one concrete description into another. Unfortunately, he never addressed the questions of how to choose the prototypes, how to index them, how to choose the transformations, how to limit the degree to which the prototype can be transformed, and how prototypes are related to actual cases.

I propose to base the prototypes on the commonsense meaning of the statutory predicates. Terms such as "principal residence," although they are underspecified, do have some commonsense meaning. Additional information can be obtained from the cases. For example, if the issue in a case is whether a house can qualify as a principal residence if the owner is not living there, we can infer that actually living in the house is part of the prototype. Prototypes will be indexed in the same way as cases, under their facts and the statutory predicates that they interpret.

Transformations are suggested by the cases. For example, the cases interpreting principal residence consider whether a house is a principal residence if you have one or more other residences, if you have not lived in the house for years and cannot move back because the rent control law prohibits it, if you have not lived in the house

for years and do not choose to move back, etc. The corresponding transformations are: increase the number of residences you have; increase the amount of time since you occupied the house; remove the reason for not returning to the house.

It follows that prototypes are related to actual cases by the transformations. If you start with the prototype and apply the appropriate transformations, you will obtain an actual case. Cases indicate the possible transformations; they also limit their extent. Negative cases — for example, one that holds that if you've been away from it for five years, a house ceases to be your principal residence, or one that holds that if you spend an equal amount of time in each of ten houses, none of them is your principal residence – limit the degree to which a plan can vary from the prototype.

The closer a plan is to the prototype, the more likely it is to succeed. The cases in the case reports are usually peripheral — cases where a statutory predicate is not satisfied mark the boundaries of a concept, cases where it is satisfied extend its boundaries. Central cases are easy, so they are usually not taken to court.

Thus, the court cases can be seen as partially defining a space of possible plans. CHIRON's goal is to construct a plan that is within that space of possibilities. Specifically, it will construct a plan as similar to the prototype as possible. This strategy gives plans a conservative bias, which is consistent with much of tax planning. If the current situation is weaker than a negative precedent along some dimension, CHIRON will reject the plan.

If the case-based planner succeeds in constructing a plan, it will output the plan with supporting citations; otherwise it will return control to the hierarchical planner, which will backtrack to the most recent choice point and try to construct another plan.

Finally, after each plan is output, the user will have the option of requesting alternative plans. CHIRON will continue constructing plans until it runs out of alternatives or the user tells it to halt. This feature makes the system more attractive as a lawyer's assistant, since it leaves the final choice among the possible plans to the lawyer.

To perform these tasks, the hierarchical planner will need the following submodules:

- the rule base. This stores the system's knowledge of the Internal Revenue Code and commonsense knowledge (*e.g.*, that selling a house is one way of transferring it). Each plan transformation will be based on some Internal Revenue Code or commonsense rule.

- retriever. This module takes a task and returns a list of subtask decompositions that apply in the current situation.
- choose-transformations. This module takes a list of transformations and returns one of the transformations from the list. It can call the case-based planner for help in making this choice.
- order-tasks. This module takes a transformation and a task network and adds ordering constraints as necessary.
- task network. Stores information about the current state of the plan.

The case-based planner will contain the following submodules:

- the case base. This module contains representations of real cases, prototypes, and plans developed by the system, all represented in enough detail to support compute-paths and compute-plan.
- dimensions. This module contains domain knowledge about which features of a case are likely to be significant and which changes to those features strengthen or weaken the taxpayer's case.
- retriever. This module takes a partial plan and returns a set of cases in which the taxpayer attempted to execute that plan.
- compute-paths. This module takes an abstract plan and a list of cases and returns a list of the transformation-sequences followed to reach those cases from the abstract plan.
- compute-plan. takes an abstract plan and a list of cases and returns a plan.
- storer. takes a plan constructed by the system and adds it to the case base.

4 Knowledge representation

CHIRON's statutes, cases, and input will all be hand-coded into an internal representation.[7] The representation language will be a variant of predicate calculus, augmented with temporal information and the three modal operators "want," "know,"

[7]The issues involved in processing the natural language of legal texts are outside the scope of this thesis. For interesting work in this area, see [Cohen, 1987].

and "believe." Timing is critical in tax problems, so it will be necessary to represent the time of actions and events. In addition, certain other concepts, such as ownership and property, are clearly required. Some of these I have already formalized [Sanders, 1989a, Sanders, 1989b]. I will add others as needed. Finally, in representing the courts' reasoning, it will be necessary to express information such as "statement p supports conclusion q." To represent this information it will be necessary to use a second-order logic. [Gallin, 1975]. This language is influenced by McDermott's temporal logic and also by the work of McCarty in representing legal concepts, especially [McCarty, 1989b, McCarty, 1977].

The plan decomposition rules will correspond to statutory and commonsense knowledge rules. Statutes often contain syntactic ambiguities [Allen, 1980]. The Internal Revenue Code is better drafted and clearer than most, but my representation will necessarily involve some judgment calls. For purposes of comparison, I will include the text of the statutory provisions on which the representation is based.

Cases can be seen as the evaluation of a taxpayer's plan by a court. The case representation will include a detailed representation of the facts of the case (the plan executed by the taxpayer), a list of the decomposition rules that the hierarchical planner would have used to reach that plan, a list of the open-textured predicates that were interpreted in the case and whether or not they were satisfied, information about which facts support and which facts weaken each of the court's conclusions, the name of the court deciding each case, and the date of the decision.

This information is required for the tasks the case representation supports. First, the case-based planner must be able to retrieve cases that are similar to the current situation. To do this, it will use the facts of the case and the open-textured predicates interpreted in the case. Second, the case-based planner must be able to provide a list of the transformation sequences leading to each case. In general, the transformation rules will be based on the Internal Revenue Code provisions interpreted in the case, which are found in the case report.

Finally, the case-based module will use the cases in constructing plans. To do this, it must reason about the relation between facts and open-textured statutory predicates. It must be able to:

- distinguish between two cases;
- show similarities between two cases;
- indicate which cases are central and which are peripheral; and

- determine trends in a series of cases illustrating a given statutory predicate.

The case-based reasoner will be able to distinguish cases by showing that they contain different facts or interpret different predicates, or interpret the same predicate with different results. It will be able to show that cases are similar by showing that they contain some or all of the same facts, or that they interpret the same predicate with the same result. It will be able to examine trends using the dates of the cases and the names of the courts deciding the cases.

The case-based module needs to reason about which cases are central and which are peripheral because the more central a plan is, the more likely it is to succeed.

Prototypes will be represented in the same way as plans, but included in the case base. Information about transformations will be included in the dimensions module of the case-based reasoner.

5 Extended example

In this section, we will illustrate CHIRON's design with a specific example. Suppose you are a tax lawyer and a client asks you for advice about selling a house. You consult CHIRON for suggestions. In a standard fact situation, your input might look something like this:

```
(plan (do Client (sell house23)))
(facts ((own Client house23) now)
       (occurs (do Client (physically-occupy house23)) now)
       (occurs (do Client (physically-occupy (only house23))) now)
       ((isa house23 house) always))
```

In other words, your client owns a house, is living in the house (and only that house), and wants to sell. Generally, your input will include both a plan — the action or actions your client intends to perform — and some background information. Either of these elements can be missing. The plan will be missing if the action has already been performed, for example if your client has already sold his house. The facts will be missing if you only know the intended actions and have no background information.

The term "Client" will be used to indicate the individual whose taxes CHIRON should consider. Our temporal logic is interval-based. Occasionally, as in the input given above, we will only specify a single time point. This is syntactic sugar for an

interval both of whose endpoints are the same, that is, an interval of zero duration. The token "now" will be used to indicate the time at which the plan is being constructed, some point in time later than all the cases in the case base and earlier than the performance of the client's intended actions. The token "always" means "at all times about which the program will need to reason." "(P (only x) t)" is syntactic sugar for the proposition "for all y such that (P y t), y = x."

After reading in this input, the system will construct a partial plan containing the actions the client intends to perform. In our example, this partial plan would be:

```
(plan
   :name nil
   :parameters (Client house23 t1)
   :preconditions nil
   :steps
      ((action
          :agent (Client)
          :descr (sell house23)
          :start t1
          :end t1))
   :postconditions nil
   :citations nil)
```

This plan representation has five parts which it shares with plans in other domains: name, parameters, preconditions, steps, and postconditions. In addition, there is a slot for citations, that is, statutory provisions and past cases that support the plan. Citations are included because of the requirements of the domain: you, the lawyer using the system, would expect the plans it generates to be supported by citations. This information is not domain-specific, however. In a business context, it corresponds to the case histories of business successes and failures; in military strategy, to past battles; and in medicine, to the case histories of patients. In general, these examples serve as a partial justification of the plan itself.

The background information is stored in a separate database, *background*, as a list of propositions and events.

```
((p1
   :descr (own Client house23)
```

```
   :start *now*
   :end *now*)
 (ev2
   :agent (Client)
   :status *occurs*
   :descr (physically-occupy house23)
   :start *now*
   :end *now*)
 (ev3
   :agent (Client)
   :status *occurs*
   :descr (physically-occupy (only house23))
   :start *now*
   :end *now*)
 (p4
   :descr (isa house23 house)
   :start *begin*
   :end *end*))
```

These structures are a fairly straightforward translation from the input language. Note the status of these events, "*occurs*." Other possibilities are "*prohibited*," "*permitted*," and "*obligated*."

After translating the input into its internal representation, the system will filter out cases with no tax consequences. If you input, for example, "George is wearing a hat," the system should print "There are no tax issues involved in this situation," and halt. Tax issues will arise if the situation described could affect the client's income, deductions, or credits. The heuristic used will be: is there a transfer of income or property? A transfer to the taxpayer might be income; a transfer by the taxpayer to someone else might generate a deduction or tax credit.

To implement this, we will need to formalize some commonsense knowledge. In this case, for example, the system will assert the facts it knows about the situation and ask a deductive retriever to prove "(do Client (transfer ?x ?y))" or "(do ?x (transfer Client ?y))." For the proof to succeed, the system must know that sales by definition mean that the seller transfers property to and receives cash from the buyer. I have already formalized some of the necessary knowledge about buying and selling [Sanders, 1989b]. The remaining knowledge should involve a relatively

constrained "cluster" of concepts.

If there are potential tax issues, CHIRON will print "There may be tax issues here," and invoke the hierarchical planner. Its top-level plan is (decrease-tax (do Client ?a ?t1 ?t2)), or here, (decrease-tax (do Client (sell house23) ?t1 ?t2)). This reflects the system goal of lowering the client's tax wherever possible.

There are a number of ways the top-level plan can be refined: the system can attempt to show that no taxable income results from the transaction, or if that fails, exclude income, defer it, attribute it to someone other than the client, or find deductions or credits. The transformation rules look like this:

```
(to-do ?tsk (decrease-tax (do Client ?a ?t1 ?t2))
   (establish (= (income-of (do Client ?a ?t1 ?t2)) 0)))
(to-do ?tsk (decrease-tax (do Client ?a ?t1 ?t2))
   (establish (excluded-income (do Client ?a ?t1 ?t2))))
(to-do ?tsk (decrease-tax (do Client ?a ?t1 ?t2))
   (establish (deferred-income (do Client ?a ?t1 ?t2))))
(to-do ?tsk (decrease-tax (do Client ?a ?t1 ?t2))
   (establish (and (attributed (income-of (do Client ?a ?t1 ?t2)) ?x)
                   (not (= ?x Client)))))
(to-do ?tsk (decrease-tax (do Client ?a ?t1 ?t2))
   (establish (deductions (do Client ?a ?t1 ?t2))))
(to-do ?tsk (decrease-tax (do Client ?a ?t1 ?t2))
   (establish (credits (do Client ?a ?t1 ?t2))))
```

"Establish" is like "achieve," but with a specific legal meaning: perform actions such that if the case were heard in court, the court would hold that the predicate was satisfied.

Conceivably several of these strategies could apply simultaneously to the same transaction. For example, you could exclude part of the income on the sale of a house, attribute part of it to a co-owner, defer part of it, include part of it in income, and take deductions for the costs of sale. Depending on the transaction, the number of possible transformations may be closer to 2^6 than 6.

For guidance in choosing a transformation, the hierarchical planner consults the case-based planner. It gives the case-based planner the client's intended actions — here, (sell house23) — and the case-based planner returns a list of plan decompositions that have been used in previous cases with similar facts.

The cases will suggest positive and negative interactions among the rules. For example, if you characterize a payment related to business property as "rent" you get a current deduction; if you characterize it as a mortgage payment, you can't deduct it but you do get depreciation; and the rules conflict, because you can't characterize a given payment both ways. Cases interpreting these provisions will show that a given payment was rent and not a mortgage payment or vice versa, and give the tax results of each conclusion. An example of a positive interaction is the combination of §121 and §1034, which can be used together to exclude part of the gain and defer part of the gain on the same transaction. If there is an example of the combination of the two rules in the case base, CHIRON will consider the possibility of combining them in its plans.

Which cases are similar? CHIRON will use two definitions. For this task, "similar" means "contains some of the same predicates." In other words, here the case-based planner would look for other cases where a house was sold. This is implemented by indexing each case under the predicates that appear in the statement of facts. For the purpose of interpreting statutory provisions, once we have determined which statutory provisions to consider, "similar" cases are the cases that interpret the same statutory provision or statutory predicate. This is implemented by indexing each case under the statutory provisions and statutory predicates that it interprets. For example, a case under §1034 would be indexed under §1034, "principal residence," "(sell house)," "(buy house)," and so on.

This scheme gives us a combination of concrete and abstract indexing. Abstract indices other than those found in the statutes might also be useful. For example, if you are selling your house, you might want to retrieve cases about other types of transfers, such as gifts or leases, or other types of property, such as commercial real estate, airplanes, or stock. Once you extend the search in this way, however, it is difficult to know where to stop; and generally speaking, you only want to look at these cases (also known as "cross-context remindings"), if you cannot find a closer match. At first, therefore, I would like to see how far the system can go using only this simple indexing scheme. For work on cross-context remindings, see, e.g. [Cuthill, 1990].

The cases retrieved have the following structure:

```
(defstruct case name court date facts links holdings path)
```

This structure reflects the way that lawyers analyze cases. Law students are taught to "brief" cases, that is, to prepare short abstracts of cases including their important

elements: the name, court, and date, the facts, the holdings, and the *ratio decidendi*, or reasoning in the case. This representation has the same structure, with two exceptions: first, the "path," a list of the plan refinements that the hierarchical planner would apply to reach this plan, is included; and second, the "links," a list indicating which facts are relevant to each of the holdings, is included instead of the reasoning in the case.

For example, one of the cases relevant to our house sale looks like this:

```
(case
   :name ''Trisko v. Commissioner, 29 T.C. 515 (1957)''
   :court (Tax Court)
   :date (1957)
   :facts
      ((p1 :descr (near house7 house18)
           :start (*begin*)
           :end (*end*))
       (ev2 :agent (TP)
            :status *occurs*
            :descr (physically-occupy house7)
            :start (June 1941)
            :end (June 1943))
       (ev3 :agent (TP)
            :status *occurs*
            :descr (physically-occupy house7)
            :start (October 1944)
            :end (February 1948))
       (p4  :descr (not (in TP USA))
            :start (February 1948)
            :end (June 1951))
       (ev5 :agent (TP)
            :status *occurs*
            :descr (not (physically-occupy house7))
            :start (February 1948)
            :end (June 1951))
       (ev6 :agent (TP)
            :status *occurs*
```

```
     :descr (lease house7 (tenant house7))
     :start (February 1948)
     :end (*end*))
(p7 :descr (employed TP Foreign-Service)
    :start (February 1948)
    :end (*end*))
(ev8 :agent (TP)
     :status *intention*
     :descr (physically-occupy house7)
     :start (February 1948)
     :end (October 9 1951))
(ev9 :agent ((tenant house7))
      :status *occurs*
      :descr (maintain house7)
      :start (February 1948)
      :end (October 9 1951))
(ev10 :agent (TP)
      :status *occurs*
      :descr (not (physically-occupy house7))
      :start (June 1951)
      :end (*end*))
(ev11 :agent (TP)
      :status *prohibited*
      :descr (physically-occupy house7)
      :start (June 1951)
      :end (*end*))
(ev12 :agent (TP)
      :status *occurs*
      :descr (sell house7)
      :start (October 9 1951)
      :end (October 9 1951))
(ev13 :agent (TP)
      :status *occurs*
      :descr (buy house18)
      :start (October 10 1951)
```

```
          :end (October 10 1951)))
   :links
      ((supports ev2 holding1)
       (supports ev3 holding1)
       (supports ev8 holding1)
       (supports p1 ev8)
       (weakens ev5 holding1)
       (excuses p4 ev5)
       (excuses p7 p4)
       (weakens ev6 holding1)
       (excuses ev9 ev6)
       (weakens ev10 holding1)
       (excuses ev11 ev10))
   :holdings
      ((holding1 (principal-residence TP house7 (October 9 1951)))
       (holding2 (principal-residence TP house18 (October 10 1951)))
       (holding3 (satisfied Section-61 (do TP (sell house7)
          (October 9 1951))))
       (holding4 (satisfied Section-1034 (do TP (sell house7)
          (October 9 1951)))))
   :path
      (((decrease-tax (do TP (sell house7) (October 9 1951))))
       ((establish (deferred-income (do TP (sell house7)
          (October 9 1951))))
       ((establish (satisfied Section-1034 (do TP (sell house7)
          (October 9 1951)))))
       ((do TP (sell house7) (October 9 1951))
        (do TP (buy house18) (October 10 1951))
        (do TP (establish (principal-residence TP house7
          (October 9 1951))))
        (do TP (establish (principal-residence TP house18
          (October 10 1951))))))
```

The name field of the case gives its official legal citation, as it would appear in a legal brief or memorandum. This will be used for output. The court field gives the name of the court that decided the case. This is useful for comparing cases.

A case from the Supreme Court is stronger support for your position than a Tax Court case, for example, which in turn is stronger support than a decision from a lower-level federal court that does not have jurisdiction over the taxpayer. Similarly with the date field: more recent cases are stronger support for your position than older ones.

The facts of the case are represented as a list of states and events. This representation corresponds closely to the logical representation used for the input language. States have a description, a start time, and an end time; events have an agent, description, start time, end time, and status. Thus, for example, the proposition, "(holds (in TP USA) (October 9 1951))," would become:

```
(p14 :descr (in TP USA)
     :start (October 9 1951)
     :end (October 9 1951))
```

Similarly, the proposition, "(occurs (do TP (sell house7)) (October 9 1951))," becomes ev12. Because the correspondence with the logical representation is so close, it should be a relatively simple exercise to define a rigorous translation from one notation to the other.

Some of the details of the representation are: "TP" stands for "taxpayer"; it will be used to represent the individual whose taxes are at issue in a case. The description of the action is singled out, since we will need to use it to index cases, and the starting and ending times are singled out, since we will need to reason about the amount of time that passes between two events. Events are listed in chronological order by start date. The tokens "*start*" and "*end*" will be used to represent the first and last dates about which the system can reason; in other words, any other date is by definition later than "*start*" and earlier than "*end*." Dates can be expressed using the year only, month and year, or month, day, and year.

Events can have several different kinds of status. Actions that are actually performed have status "*occurs*"; actions that are prohibited or obligated by law have status "*prohibited*" or "*obligated*," respectively; and actions that the agent intends to perform have status "*intention*." In the case of intended actions, the time period specified by the start and end dates is the time during which the agent has the intention, not the time of the intended action.

A negated action or state is not performed (or not true) at any time during the given interval. Thus p4 means that at no time during the period from February,

1948, to June, 1951, was the taxpayer in the United States; and ev5 means that at no time during that interval did he physically occupy his house.

When indexing the cases, negated actions or states will be treated as if they were the corresponding positive, and duplicates will be disregarded. Thus, *Trisko* will be indexed under "(physically-occupy house)," "(lease house (tenant house))," "(employed TP Foreign-Service)," "(maintain house)," "(sell house)," and "(buy house)." To make matching easier, instances, such as house7, are replaced by the name of the class to which they belong. Conversely, propositions such as "(in TP USA)," which will hold for almost all cases, are not used as indices because they would make matching too easy.

In this case, the house was rented to a series of tenants, so for simplicity, the function "(tenant house7)" is used to represent the individual who was living in the house at a given time.

There are three types of links between propositions and events: "supports," "weakens," and "excuses." A proposition or event p is said to support another proposition or event q if (1) p is relevant to q and (2) p makes q more probable. Similarly, p is said to weaken q if p is relevant to q and p makes q less probable. Finally, p excuses q if, without p, q would weaken some proposition or event r, but p cancels out the weakens link between q and r.

The holdings list the statutory provisions and statutory predicates that were or were not satisfied in a particular case. These will also be used as indices. Thus, *Trisko* will be indexed under "(principal-residence house)," "(satisfied Section-61)," and "(satisfied Section-1034)." If the taxpayer had lost, the holding would be "(not (satisfied Section-1034 (do TP (sell house7) (October 9 1951))))," but the case would still be indexed with the cases in which the statute was satisfied.

In addition to representations of actual legal cases, the case-base includes prototype plans. An example of a prototype plan that is related to our house-sale example is:

```
(prototype
   :name ''Section 121 exclusion''
   :parameters (TP ?house ?t)
   :preconditions
      ((ev1 :agent (TP)
            :descr (not (elect section-121))
            :start (*begin*)
```

```
        :end (tbef ?t))
    (p2  :descr (>= (age TP) years 55)
         :start (?t)
         :end (?t)))
:steps
   ((ev3 :agent (TP)
         :descr (physically-occupy ?house)
         :start (?t)
         :end (?t))
    (ev4 :agent (TP)
         :descr (physically-occupy (only ?house))
         :start (?t)
         :end (?t))
    (ev5 :agent (TP)
         :descr (sell ?house)
         :start (?t)
         :end (?t))
    (ev6 :agent (TP)
         :descr (elect Section-121)
         :start (?t)
         :end (?t)))
:postconditions
   ((p7
      :descr (principal-residence ?house TP)
      :start (?t)
      :end (?t))
    (p8
      :descr (excluded TP (lesser-of 125000
                (income-of (do TP (sell ?house) ?t))))
      :start (?t)
      :end (*end*))
    (p9
      :descr (satisfied Section-121 (do TP (sell ?house) ?t))
      :start (?t)
      :end (*end*)))
```

```
:links ((supports ev3 p7)
        (supports ev4 p7))
:path  (((decrease-tax (do TP (sell ?house) ?t)))
        ((establish (excluded-income (do TP (sell ?house) ?t))))
        ((establish (satisfied Section-121
                               (do TP (sell ?house) ?t))))
        ((do TP (sell ?house) ?t)
         (do TP (establish (principal-residence TP ?house ?t)))
:citations (61 121))
```

In other words, if the taxpayer is over 55 years of age, is living in a house and only that house, sells the house, and has never elected §121 before, he or she can elect to exclude a portion of the gain on the sale of the house.

Prototypes have the same fields as plans, with two additions, the "links" field and the "path" field, borrowed from the case representation. As in cases, the "path" field indicates the sequence of transformations the hierarchical planner would perform to reach this plan, and the "links" field gives the relationship between the steps of the prototype and open-textured predicates. In case you need to adapt the prototype plan, these links indicate which predicates may be affected. "(tbef ?t)" matches any time point before ?t. ">=" takes a parameter indicating the unit of measurement; this can be years, months, or days. (There will also be an arithmetic package for adding and subtracting sums of money.) "Elect" is a legal term, meaning the taxpayer formally chooses (generally by filing a form with the Internal Revenue Service) to have a particular provision apply in his or her case.

This prototype will be indexed under (elect Section-121), (physically-occupy house), (physically-occupy (only house)), (sell house), (principal-residence house TP), (excluded TP (lesser-of 125000 (income-of (do TP (sell house) ?t)))), and (satisfied Section-121).

Suppose for simplicity that these are the only two cases retrieved, and there is no case combining the two possibilities. Then the case-based planner would return the following list of paths:

```
((((decrease-tax (do Client (sell house23) (?t))))
  ((establish (deferred-income (do Client (sell house23)
         (?t)))))
  ((establish (satisfied Section-1034 (do Client (sell house23)
         (?t)))))
```

```
((do Client (sell house23) (?t))
 (do Client (buy ?house) (?t2))
 (<= (- ?t2 ?t) years 2)
 (do Client (establish (principal-residence Client house23
       (?t))))
 (do Client (establish (principal-residence Client ?house (?t2)))))
(((decrease-tax (do Client (sell house23) ?t)))
 ((establish (excluded-income (do Client (sell house23) ?t))))
 ((establish (satisfied Section-121 (do Client (sell house23) ?t))))
 ((do Client (sell house23) ?t)
  (do Client (establish (principal-residence Client house23 ?t))))))
```

The first is the sequence of plan refinements that would be used to reach *Trisko*; the second, the sequence of refinements to reach the §121 prototype.

Next, CHIRON must choose which plan to consider first. The control heuristics are all aimed at maintaining or increasing the client's net income. They include:

1. prefer selling property to giving it away (and more generally, prefer selling at a higher price);
2. prefer charitable gifts to gifts that do not generate a deduction;
3. prefer buying at a lower price, or receiving a gift, to buying property at a higher price.
4. prefer buying property to spending money without receiving property in exchange, for example paying rent or taxes.
5. prefer lower taxes to higher taxes, that is, prefer plans that
   (a) do not generate taxable income;
   (b) generate excluded income;
   (c) generate deferred income;
   (d) get or accelerate deductions and credits; and/or
   (e) if there are income and deductions that are tied together, generate a smaller amount of income after deductions.

In other words, CHIRON incorporates a naive theory of financial planning. According to this theory, buying and selling property involve the exchange of money

for another asset of equal value, so they have no effect on net worth. Paying rent lowers your net worth. So does making gifts. In theory, taxes could be considered as payment for your share of roads, state universities, and so forth, but since you are not free to re-sell those assets and realize their value, taxes are also considered to lower your net worth. The system goal is the selfish one of increasing the client's individual net worth.

In our example, the options are excluding income or deferring it; since exclusion is generally better than deferral, CHIRON will apply that transformation first. As a result, the abstract plan is transformed from (decrease-tax (do Client (sell house23) ?t1 ?t2)) to (establish (excluded-income (do Client (sell house23) ?t1 ?t2))).

At this point, the hierarchical planner projects the effects of the new subtask, checks for protection violations, and adds ordering constraints or backtracks as necessary. Here the effect is that income from the sale of the house will be excluded, and there are no protection violations, so no further action is necessary.

This plan has various possible refinements. Income can be excluded in a variety of ways, for example, by showing that it was received as a gift, by showing that it was interest on a tax-free municipal bond, and so forth. In our simplified example, only one of these refinements was suggested by the case-based planner, so we disregard the others. In general, the hierarchical planner has knowledge about the legal effects of the abstract plans; knowledge about the effects of primitive actions, such as (physically-occupy house23), is obtained from the commonsense knowledge base and from cases.

The system will repeat the process of choosing a task and refining it until no more tasks can be transformed. At this point, the plan in our example will look like this:

```
(plan
   :name ''Section 121 exclusion''
   :parameters (Client house23 ?t)
   :preconditions
      ((ev1 :agent (Client)
            :descr (not (elect section-121))
            :start (*begin*)
            :end (tbef ?t))
       (p2  :descr (>= (age Client) years 55)
            :start (?t)
```

```
          :end (?t)))
   :steps
      ((ev3 :agent (Client)
            :descr
               (establish (principal-residence house23 Client ?t))
            :start (?t)
            :end (?t))
       (ev4 :agent (Client)
            :descr (sell house23 ?t)
            :start (?t)
            :end (?t))
       (ev5 :agent (Client)
            :descr (elect Section-121)
            :start (?t)
            :end (?t)))
   :postconditions
      ((p6
         :descr (principal-residence house23 Client)
         :start (?t)
         :end (?t))
       (p7
         :descr (excluded Client (lesser-of 125000
                    (income-of (do Client (sell house23) ?t))))
         :start (?t)
         :end (*end*))
       (p8
         :descr (satisfied Section-121 (do Client (sell house23) ?t))
         :start (?t)
         :end (*end*)))
   :citations (61 121))
```

This plan is still too abstract. One of the actions is an "establish," which is by definition not a primitive action. No more transformation rules can be applied, but the plan has still not been completely refined.

The next step is to call the case-based planner. First, it retrieves cases interpreting the current abstract plan. In particular, it will look for cases interpreting the

predicate or predicates to be established, preferably under the same code section involved in this plan. Note that, unlike most case-based systems, CHIRON will not attempt to choose a single "best" case; instead, it will use a group of similar cases in constructing our plan. Here, we look at the cases under §121 and possibly some additional cases interpreting the statutory predicate "principal residence."

Suppose for simplicity that the two cases given above, *Trisko* and the §121 prototype, are the only two retrieved in our example. The cases retrieved will be organized into a "space" of possible plans, with the prototype at the center and cases indicating (or extending) the boundaries in various directions. In comparing these two cases, the relevant dimension is time, the amount of time that an owner can spend away from home without it ceasing to be his "principal residence." In the prototype, this amount of time is zero; in *Trisko*, roughly 3 years. The cases are ordered along that dimension.

Next, the system will compare the current situation with the prototype, using the background information from the input. Where possible, it will add facts to make the plan similar to the prototype. In this example, our plan will become:

```
(plan
   :name ''residence-exclusion''
   :parameters (Client house23 t1)
   :preconditions
      ((p1
         :descr (>= (age Client) years 55)
         :start (t1)
         :end (t1))
       (p2
         :descr (not (elect Client Section-121))
         :start (*begin*)
         :end (tbef t1)))
   :steps
      ((action
          :agent (Client)
          :descr (sell house23)
          :start (t1)
          :end (t1))
       (action
```

```
        :agent (Client)
        :descr (physically-occupy house23)
        :start (t1)
        :end (t1))
  :postconditions
     ((p3
        :descr (elect Client Section-121)
        :start (t1)
        :end (t1))
      (p4
        :descr (excluded Client (lesser-of 125000
                  (income-of (do Client (sell house23) t1))))
        :start (t1)
        :end (t1)))
  :citations (61 121))
```

This plan is now consistent with the prototype (or "safe harbor"), so we're done. The system will print out the plan with citations and a continuation, so the user can request another plan if alternatives are desired.

Suppose the plan had not been consistent with the prototype. Suppose, for example, that the client is an 80-year-old woman who moved into a nursing home two years ago and rented out her home. Now she has finally given up hope of moving back and wishes to sell the house. The case is stronger than *Trisko* along the time dimension, so the system should construct the §121 plan and cite *Trisko* as support.

In general, if a plan is inconsistent with the prototype, the changes either strengthen or weaken the plan. If they strengthen the plan, fine. At most, you might want to note the possibility of weakening the plan if there would be some financial advantage in it. If the changes weaken the plan, then there are four possible solutions:

1. find a successful case that is at least as weak along this dimension. Cite it in support of your position. If that case strengthened its plan in other ways to compensate, try that. (This will be implemented using the "excuses" links in the case representation.)

2. Order the cases that are weak along this dimension. Are they getting weaker? If so, suggest a trend. Cite the cases to support your position.

3. Change the facts. The system will need information about which facts are mutable. In the above example, if your client is not living in her house, she might be able to move in; but if she is 30 and the statutory provision only applies to taxpayers over 55, that can't be helped.

4. Find a level of abstraction at which your case is relevantly the same as the prototype. This will require an action hierarchy and a definition of relevance.

If the case is weak, you also need to ensure that it differs from the unfavorable past cases (if any). This can be done by finding a level of detail at which the cases differ (using the same action hierarchy and notion of relevance as above) or by changing the facts, if possible.

Finally, if a plan's weakness can't be fixed and there is no support for it in past cases, CHIRON will reject it. It will not attempt to find new creative arguments. For example, if the case base did not include any cases like *Trisko*, this system would not generate a plan for establishing personal residence that involves living away from the house.

CHIRON will generate a stream of plans. It will attempt to construct the plans suggested by the case-based planner one after another and output the ones that are not rejected, until there are no more such plans or the user tells it to halt.

In our example, CHIRON's next step would be to construct a plan for a rollover under §1034. This plan will be quite similar to the §121 plan, with the added step of buying another house and establishing that the second house becomes a principal residence. As it constructs each plan, CHIRON will save information about which subplans have succeeded or failed, to avoid duplicating work. Here, for example, it will generate a subplan for establishing that house23 is the client's principal residence for purposes of §121, and reuse that plan when it is necessary to show the house is a principal residence for §1034.

6 Conclusions

In this thesis, I propose to investigate planning issues in law. Planning problems arise in law when an individual (or corporation) wants to perform a sequence of actions that raises legal issues. Using statutes and cases, lawyers construct plans to achieve the desired legal treatment for their clients.

Statutes are rules that have been created formally, by legislation. No matter how detailed, they always contain some important phrases that are not defined in

the statute itself. Instead, these phrases (referred to as "open-textured predicates") are defined partly by commonsense knowledge, partly by example. Each court case is an example — an application of the law to a particular set of facts.

The cases only partially define the statutes. In general, defining an open-textured predicate is not just a matter of inferring defining characteristics from a set of examples. There may not be any set of essential characteristics shared by all positive instances of the statutory predicate. Some examples will be typical, and others more or less similar to them along various dimensions. Some cases clearly qualify, some clearly do not qualify, and there will always be some borderline cases that could go either way.

For example, "principal residence" is a statutory predicate that is used in the Internal Revenue Code (see, *e.g.*, §1034). The typical case of a principal residence is a house that the taxpayer lives in year round. A condominium, a cooperative, or a houseboat could also be your principal residence. In some cases, a principal residence can even be a place you haven't lived in for several years. On the other hand, a house that you sell without ever living in it is clearly not your principal residence.

The challenge for lawyers — and for a legal planning system — is to construct plans that are similar enough to past cases that they satisfy (or avoid satisfying) the open-textured predicates necessary to receive the desired legal treatment. This is the problem I propose to address in my thesis.

Previous work in artificial intelligence and law has generally addressed the open-textured predicate problem in the context of trial law, rather than planning. This corresponds to recognition, as opposed to generation, of fact situations satisfying statutory predicates. Systems have been designed to distinguish easy cases from hard borderline cases [Gardner, 1987]; and to construct arguments for and against the application of a given statutory predicate to a particular situation. See, *e.g.*, [Ashley, 1988, Branting, 1990b, Rissland and Skalak, 1989c].

One recent paper sketched out a design for a legal planning system based on the theory that lawyers construct plans by retrieving prototype plans and transforming them to meet the client's goals [Schlobohm and McCarty, 1989]. The paper is primarily concerned with representation, however, and algorithmic issues such as how the prototypes are chosen, indexing, how the transformations are chosen, and the relationship of the prototypes to actual cases and rules are not addressed.

Previous work in planning is insufficient to handle this problem. If we start with

primitive actions, as in linear planning, we have no way of determining whether they satisfy the open-textured rules; if we start with abstract plans, as in hierarchical planning, we will be unable to refine the plan as far as primitive actions. Without cases, we have no way of making the connection between abstractions, based on open-textured rules, and primitive actions. Case-based planning suggests ways to use legal cases, but provides no role for statutory rules. In addition, most case-based reasoning systems have based their output on a single "best" case selected by the retriever, rather than using a set of relevant past cases, as a lawyer would be likely to do.

In this proposal, I have described the design of CHIRON, a system that I am developing to construct plans in the domain of personal income tax planning. In order to reason with both rules and cases, I will use a hybrid system, combining hierarchical and case-based modules in a tightly-coupled architecture. The hierarchical planner will call the case-based planner for guidance in choosing transformations, and when no more transformations can be applied, to refine the open-textured predicates in the plan; and the case-based planner will use the hierarchical planner for help with indexing and controlling adaptation. The case-based planner will use a formal knowledge representation, and it will base its plans on a group of past cases, rather than a single case. The plans output by the hierarchical planner will be open-textured, *i.e.*, they will contain open-textured predicates derived from the open-textured rules on which the hierarchical planner's transformation rules are based. Each of these plans will correspond to a prototype and a set of cases in the case-base. The prototype represents a typical version of the plan, based on commonsense knowledge and information in the cases. The cases indicate variations from the prototype in different directions, defining a space of possibilities. CHIRON will attempt to construct a plan within that space of possibilities that is consistent with the client's current situation.

One final issue remains to be considered. Once this system has been designed and implemented, how can its performance be evaluated? Validation is an issue throughout artificial intelligence (See, *e.g.*, [Cohen and Howe, 1988].) The solution depends on the domain and perhaps also on the style of research. If a theory is being implemented, then the implementation is a test of the theory. If the theory claims to be a cognitive model, then it should be tested against the way people think, perhaps by comparing the operation of the system with the results of psychological experiments. If not, then the system may be evaluated in terms of its computational

efficiency or by examining the output (the "ends justify the means" test).

I am not attempting to create a cognitive model of a lawyer. Like Gardner, I propose to implement a system that will function within the "legal task environment," rather than modelling the psychology of a lawyer [Gardner, 1987, page 2]; or in other words, to design a program that uses the same kind of materials that a lawyer uses, cases and open-textured rules, and produces similar output, but not necessarily in the same way that a human lawyer would.

If you focus on the output of a system, rather than the process by which it is reached, it becomes even more important to evaluate that output. In some domains, such as tic-tac-toe, you can prove that a plan output by your system is optimal. In others, such as cooking or chess, you can experiment, by trying out the recipe produced by your cooking program or testing your chess program against better and better human players. In law, neither of these strategies is available. Because of the nature of the domain, you cannot prove your plans correct. Because of the realities of tax audits, you do not want to experiment and invite a verdict from the Internal Revenue Service. A lawyer might simply try out a plan and wait until the statute of limitations has run to see if it is audited (failure to be audited is in itself a kind of success). But that's a long time to wait to validate a program.

So how can we evaluate the output of a legal planner? There are a number of possibilities. First, we can input the facts of an actual case, not in the case-base, and compare the system's output with the result of the case. Second, the development of a set of benchmarks can itself be a useful contribution. It will allow the law and artificial intelligence community to compare systems with each other. Third, we can ask lawyers to judge the system's output. In spite of the fact that they can neither prove their plans correct nor test them, experienced lawyers can generally give an opinion about whether a particular plan is likely to succeed. Finally, we can attempt to formalize the intuitions by which lawyers evaluate plans and incorporate that knowledge into the system itself.

For CHIRON, I will use several of these strategies. CHIRON is intended as a lawyer's assistant, not as a substitute for a lawyer. It will produce a number of suggested plans for the user to evaluate, rather than a single best plan. On that basis, its output will be valid if it seems reasonable to a lawyer. Ideally, the output would also be interesting, that is, it would suggest plausible alternatives that had not already occurred to the lawyer.

I will not attempt to formalize the way in which lawyers evaluate plans — that

is a formidable knowledge engineering task, and would be a large project in itself. Instead, I will start by designing a set of benchmarks, including those given in Section 1. They will be simple examples, for which most tax planners would agree on at least the obvious solutions. I will test the output against my intuitions and those of other lawyers. And I will also compare the system's output with the results in actual cases.

The design and implementation of this system will make several contributions. They include:

- implementing a legal planner using both rules and cases;
- combining hierarchical and case-based planners in a hybrid system;
- combining a case-based reasoner with another module in a tightly-coupled architecture;
- basing plans on a group of past cases, rather than a single "best" case;
- implementing a case-based system with a formal knowledge representation;
- developing prototype definitions and a set of transformations for each of the open-textured predicates used by the system, such that the prototype and transformations give the dimensions of the space of possible cases satisfying the open-textured predicates. The utility of this idea was pointed out by McCarty [McCarty, 1980, McCarty and Sridharan, 1982, Schlobohm and McCarty, 1989], but he proposed no method for determining the prototypes and transformations or relating them to actual cases;
- using cases to indicate the boundary of the space of possible cases satisfying the open-textured predicates; and
- developing a validation scheme for the program.

Acknowledgements

This research has been supported by IBM contracts 17290066, 17291066, 17292066, and 17293066. The author gratefully acknowledges the criticisms and encouragement of Mark Boddy, Karl Branting, Tom Dean, Anne Gardner, Mary Harper, Richard Hughey, Robert McCartney, Thorne McCarty, Leora Morgenstern, Edwina Rissland, and David Skalak.

References

[Allen, 1980] Layman E. Allen. Language, law, and logic: plain drafting for the electronic age. In Bryan Niblett, editor, *Computer Science and Law*, pages 75–100. Cambridge University Press, Cambridge, England, 1980.

[Ashley, 1988] Kevin D. Ashley. Modelling legal argument: reasoning with cases and hypotheticals. Technical Report 88-01, University of Massachusetts, Amherst, Department of Computer and Information Science, 1988. (PhD Thesis).

[Branting, 1988] L. Karl Branting. The role of explanations in reasoning from legal precedents. In *Proceedings of a Workshop on Case-Based Reasoning*, pages 94–103, 1988.

[Branting, 1989a] L. Karl Branting. Integrating generalizations with exemplar-based reasoning. In *Proceedings of a Workshop on Case-Based Reasoning*, pages 224–229, 1989.

[Branting, 1989b] L. Karl Branting. Integrating generalizations with exemplar-based reasoning. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society,* Ann Arbor, Michigan, pages 139–146, 1989.

[Branting, 1989c] L. Karl Branting. Representing and reusing explanations of legal precedents. In *Proceedings of the Second International Conference on Artificial Intelligence and Law,* Vancouver, British Columbia, pages 103–110, 1989.

[Branting, 1990a] L. Karl Branting. (personal communication), March 1990.

[Branting, 1990b] L. Karl Branting. Techniques for retrieval of structured cases. In *Proceedings of the AAAI Spring Symposium on Case-Based Reasoning,* Palo Alto, California, pages 22–26, 1990.

[Charniak and McDermott, 1985] Eugene Charniak and Drew V. McDermott. *Artificial intelligence.* Addison-Wesley, Reading, Massachusetts, 1985.

[Cohen and Howe, 1988] Paul R. Cohen and Adele E. Howe. How evaluation guides AI research. *Artificial Intelligence Magazine*, 9:35–43, 1988.

[Cohen, 1987] Robin Cohen. Interpreting clues in conjunction with processing restrictions in arguments and discourse. In *Proceedings of the Sixth National Conference on Artificial Intelligence,* Seattle, Washington, pages 528–533, 1987.

[Cuthill, 1990] Barbara B. Cuthill. A dynamic memory approach to case-based legal reasoning. In *Proceedings of the AAAI Workshop on Artificial Intelligence and Legal Reasoning,* Boston, Massachusetts, 1990.

[Fikes and Nilsson, 1971] Richard Fikes and Nils J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence,* 2:189–208, 1971.

[Gallin, 1975] Daniel Gallin. *Intensional and higher order modal logics.* American Elsevier, New York, 1975.

[Gardner, 1987] Anne v.d.L. Gardner. *An artificial intelligence approach to legal reasoning.* MIT Press, Cambridge, Massachusetts, 1987.

[Hammond, 1986a] Kristian Hammond. Chef: a model of case-based planning. In *Proceedings of the Fifth National Conference on Artificial Intelligence,* 1986.

[Hammond, 1986b] Kristian J. Hammond. Case-based planning: an integrated theory of planning, learning, and memory. Technical Report YALEU/CSD/RR 488, Yale University Department of Computer Science, 1986. (PhD Thesis).

[Hammond, 1989] Kristian J. Hammond. On functionally motivated vocabularies: an apologia. In *Proceedings of a Workshop on Case-based Reasoning,* pages 52–56, 1989.

[Hart, 1961] H. L. A. Hart. *The concept of law.* Clarendon Press, Oxford, 1961.

[Lakoff, 1987] George Lakoff. *Women, fire, and dangerous things: what categories reveal about the mind.* University of Chicago Press, Chicago, Illinois, 1987.

[Lifschitz, 1987] Vladimir Lifschitz. Formal theories of action. In *Proceedings of the 1987 Workshop on the Frame Problem in Artificial Intelligence,* 1987.

[McCarty and Sridharan, 1982] L. Thorne McCarty and N.S. Sridharan. A computational theory of legal argument. Technical Report LRP-TR-13, Laboratory for Computer Science Research, Rutgers University, 1982.

[McCarty, 1977] L. Thorne McCarty. Reflections on TAXMAN: an experiment in artificial intelligence and legal reasoning. *Harvard Law Review,* 90:837–893, 1977.

[McCarty, 1980] L. Thorne McCarty. The TAXMAN project: towards a cognitive theory of legal argument. In Bryan Niblett, editor, *Computer Science and Law*, pages 23–43. Cambridge University Press, Cambridge, England, 1980.

[McCarty, 1983] L. Thorne McCarty. Permissions and obligations. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* Karlsruhe, West Germany, pages 287–294, 1983.

[McCarty, 1986] L. Thorne McCarty. Permissions and obligations: an informal introduction. Technical Report LRP-TR-19, Laboratory for Computer Science Research, Rutgers University, 1986.

[McCarty, 1989a] L. Thorne McCarty. Computing with prototypes (preliminary report). In *Proceedings of the Bar-Ilan Symposium on the Foundations of Artificial Intelligence*, 1989.

[McCarty, 1989b] L. Thorne McCarty. A language for legal discourse: I. basic features. In *Proceedings of the Second International Conference on Artificial Intelligence and Law,* Vancouver, British Columbia, pages 180–189, 1989.

[Merryman, 1969] John Henry Merryman. *The civil law tradition.* Stanford University Press, Palo Alto, CA, 1969.

[Moore, 1981] Michael S. Moore. The semantics of judging. *Southern California Law Review*, 54:151–294, 1981.

[Riesbeck and Schank, 1989] Christopher K. Riesbeck and Roger C. Schank. *Inside Case-based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

[Rissland and Skalak, 1989a] Edwina L. Rissland and David B. Skalak. Case-based reasoning in a rule-governed domain. In *Proceedings of the Fifth IEEE Conference on Artificial Intelligence Applications,* Miami, Florida, pages 45–53, 1989.

[Rissland and Skalak, 1989b] Edwina L. Rissland and David B. Skalak. Combining case-based and rule-based reasoning: a heuristic approach. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* Detroit, Michigan, pages 524–530, 1989.

[Rissland and Skalak, 1989c] Edwina L. Rissland and David B. Skalak. Interpreting statutory predicates. In *Proceedings of the Second International Conference on Artificial Intelligence and Law,* Vancouver, British Columbia, pages 46–53, 1989.

[Sacerdoti, 1974] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 7:231–272, 1974.

[Sanders, 1989a] Kathryn E. Sanders. A logic for emotions: a basis for reasoning about commonsense psychological knowledge. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society,* Ann Arbor, Michigan, 1989.

[Sanders, 1989b] Kathryn E. Sanders. A logic for emotions: a basis for reasoning about commonsense psychological knowledge. Technical Report 89-23, Brown University Computer Science Department, 1989.

[Sanders, 1990] Kathryn E. Sanders. The truth, the whole truth, and nothing but the truth: case representations for legal reasoning systems. In *Proceedings of the AAAI Workshop on Legal Reasoning,* Boston, Massachusetts, 1990.

[Schlobohm and McCarty, 1989] Dean Schlobohm and L. Thorne McCarty. Eps ii: Estate planning with prototypes. In *Proceedings of the Second International Conference on Artificial Intelligence and Law,* Vancouver, British Columbia, pages 1–10, 1989.

[Skalak, 1989a] David B. Skalak. Options for controlling mixed paradigm systems. In *Proceedings of a Workshop on Case-Based Reasoning*, pages 318–323, 1989.

[Skalak, 1989b] David B. Skalak. Taking advantage of models for legal classification. In *Proceedings of the Second International Conference on Artificial Intelligence and Law,* Vancouver, British Columbia, pages 234–241, 1989.

[Smith and Deedman, 1987] J.C. Smith and Cal Deedman. The application of expert systems technology to case-based law. In *Proceedings of the First International Conference on Artificial Intelligence and Law,* Boston, Massachusetts, pages 84–93, 1987.

[Twining and Miers, 1976] William L. Twining and David Miers. *How to do things with rules: a primer of interpretation.* Weidenfeld and Nicolson, London, 1976.

[Waismann, 1965] Friedrich Waismann. Verifiability. In Antony Flew, editor, *Logic and language: first and second series*, pages 122–151. Anchor Books, Garden City, New Jersey, 1965. (first published in *Proceedings of the Aristotelian Society,* 119-150 (1945).).

[Waltz, 1989] David L. Waltz. Is indexing used for retrieval? In *Proceedings of a Workshop on Case-based Reasoning*, pages 41–44, 1989.