1

Introducing AI & Law and Its Role in Future Legal Practice

1.1. INTRODUCTION

Artificial Intelligence and Law (AI & Law), a research field since the 1980s with roots in the previous decades, is about to experience a revolution. Teams of researchers in question answering (QA), information extraction (IE), and argument mining from text planted the seeds of this revolution with programs like IBM's Watson and Debater and the open-source information management architectures on which these programs are based. From these seeds, new applications for the legal domain are sure to grow. Indeed, they are growing now. This book explains how.

Programs like Watson and Debater will not perform legal reasoning. They may be able to answer legal questions in a superficial sense, but they cannot explain their answers or make legal arguments. The open-source text analysis tools on which they are based, however, will make a profound difference in the development of new legal applications. They will identify argument-related information in legal texts that can transform legal information retrieval into a new kind of conceptual information retrieval: argument retrieval (AR).

Computational models developed by AI & Law researchers will perform the legal reasoning. The newly extracted argument-related information will connect the computational models of legal reasoning (CMLRs) and argument directly with legal texts. The models can generate arguments for and against particular outcomes in problems input as texts, predict a problem's outcome, and explain their predictions with reasons that legal professionals will recognize and can evaluate for themselves. The result will be a new kind of legal app, one that enables cognitive computing, a kind of collaborative activity between humans and computers in which each performs the kinds of intelligent activities that they can do best.

This chapter introduces the subject of AI & Law and explains the role it will play in light of the new technologies for analyzing legal texts. It explains how these

technologies enable new tools for legal practice using computational models of legal reasoning and argumentation developed by AI & Law researchers.

Some questions addressed in this chapter include: What is the subject of Artificial Intelligence and Law? What is a CMLR? What are the new technologies for automated QA, IE, and argument mining from texts? What roles will AI & Law CMLRs and argument play given these new technologies? What are conceptual information retrieval and cognitive computing, and what kind of legal app will support them?

1.2. AI & LAW AND THE PROMISE OF TEXT ANALYTICS

The goal of much of the research in AI & Law has been to develop CMLRs that can make legal arguments and use them to predict outcomes of legal disputes. A *CMLR* is a computer program that implements a process evidencing attributes of human legal reasoning. The process may involve analyzing a situation and answering a legal question, predicting an outcome, or making a legal argument. A subset of CMLRs implements a process of legal argumentation as part of their reasoning. These are called computational models of legal argument (CMLAs).

CMLRs and CMLAs break down a complex human intellectual task, such as estimating the settlement value of a product liability suit or analyzing an offer and acceptance problem in a first-year contracts course, into a set of computational steps or *algorithm*. The models specify how a problem is input and the type of legal result to output. In between, the model builders have constructed a computational mechanism to apply domain knowledge to perform the steps and transform the inputs to outputs.

In developing these models, researchers address such questions as how to represent what a legal rule means so that a computer program can decide whether it applies to a situation, how to distinguish "hard" from "easy" legal issues, and the roles that cases and values play in interpreting legal rules. Their answers to these questions are not philosophical but scientific; their computer programs not only model legal reasoning tasks but also actually perform them; and the researchers conduct experiments to evaluate how well their programs perform.

While AI & Law researchers have made great strides, a knowledge representation bottleneck has impeded their progress toward contributing to legal practice. So far, the substantive legal knowledge employed by their computational models has had to be extracted *manually* from legal sources, that is, from the cases, statutes, regulations, contracts, and other texts that legal professionals actually use. That is, human experts have had to read the legal texts and represent relevant parts of their content in a form the computational models could use. An inability to automatically connect their CMLRs directly to legal texts has limited the researchers' ability to apply their programs in real-world legal information retrieval, prediction, and decision-making.

Recent developments in computerized QA, IE from text, and argument mining promise to change that. "A Question-answering system searches a large text

collection and finds a short phrase or sentence that precisely answers a user's question" (Prager et al., 2000). "Information extraction is the problem of summarizing the essential details particular to a given document" (Freitag, 2000). Argument mining involves automatically identifying argumentative structures within document texts, for instance, premises and conclusion, and relationships between pairs of arguments (ACL-AMW, 2016). All three technologies usually rely, at least in part, on applying machine learning (ML) to assist programs in processing semantic information in the texts.

A more general term for these techniques, *text analytics* or text mining, "refers to the discovery of knowledge that can be found in text archives . . . [It] describes a set of linguistic, statistical, and machine learning techniques that model and structure the information content of textual sources for business intelligence, exploratory data analysis, research, or investigation" (Hu and Liu, 2012, pp. 387–8). When the texts to be analyzed are legal, we may refer to "legal text analytics" or more simply "legal analytics," the "deriving of substantively meaningful insight from some sort of legal data," including legal textual data (Katz and Bommarito, 2014, p. 3).

The text analytic techniques may open the knowledge acquisition bottleneck that has long hampered progress in fielding intelligent legal applications. Instead of relying solely on manual techniques to represent what legal texts mean in ways that programs can use, researchers can automate the knowledge representation process.

As a result, some CMLRs and CMLAs may soon be linked with text analysis tools to enable the construction of a new generation of legal applications and some novel legal practice tools. Specifically, CMLRs and CMLAs developed in the AI & Law field will employ information extracted automatically from legal texts such as case decisions and statutes to assist humans in answering legal questions, predicting case outcomes, providing explanations, and making arguments for and against legal conclusions more effectively than existing technologies can.

In a complementary way, the AI & Law programs can provide answers to questions that are likely on the minds of technologists in commercial laboratories and start-ups: Now that we are able to extract semantic information automatically from legal texts, what can computer programs do with it? And, exactly what kind of information should be extracted from statutes, regulations, and cases? The CMLRs demonstrate how the new text processing tools can accommodate, adapt, and use the structures of legal knowledge to assist humans in performing practical legal tasks.

Some CMLRs and CMLAs could help advanced AI programs make intelligent use of legal sources. Certainly, the extracted information will be used to improve legal information retrieval, helping to point legal professionals more quickly to relevant information, but what more can be done? Can computers reason with the legal information extracted from texts? Can they help users to pose and test legal hypotheses, make legal arguments, or predict outcomes of legal disputes?

The answers appear to be "Yes!" but a considerable amount of research remains to be done before the new legal applications can demonstrate their full potential.

Indeed, that is what this book is about: how best to perform that research. This book will also assist practitioners and others in contributing to this research and in applying the resulting legal apps. This includes commercial firms interested in developing new products and services based on these models and public agencies wishing to modernize their workflows.

1.3. NEW PARADIGMS FOR INTELLIGENT TECHNOLOGY IN LEGAL PRACTICE

The technology of legal practice is changing rapidly. Predictive coding is transforming discovery in litigation. Start-ups like Ravel (Ravel Law, 2015a), Lex Machina (Surdeanu et al., 2011), and the Watson-based Ross (Ross Intelligence, 2015) (see Sections 4.7 and 12.2) are garnering attention and enlisting law firm subscribers. These and other developments in text analytics offer new process models and tools for delivering legal services, promising greater efficiency and, possibly, greater public accessibility.

These changes present challenges and opportunities for young attorneys and computer scientists, but it has not been easy to predict the future of legal practice. Declines in hiring by law firms have led to reductions in the number of law school applicants. Prospective applicants weigh the chances of gainful employment against the size of their student loans and look elsewhere. There is uncertainty about what law-related tasks the technology can perform. After citing press, academic, and commercial predictions of "the imminent and widespread displacement of lawyers by computers," Remus and Levy argue persuasively that the predictions "fail to engage with technical details . . . critical for understanding the kinds of lawyering tasks that computers can and cannot perform. For example, why document review in discovery practice is more amenable to automation than in corporate due diligence work, and why the automation of ... sports stories does not suggest the imminent automation of legal brief-writing" (Remus and Levy, 2015, p. 2).[1]

It is also unclear what law students need to learn about technology. Law firms have long called for law schools to graduate "practice-ready" students but even firms seem confused about the kinds of technology the firms will require, whether to develop the technology in house or rely on external suppliers, and the skills and knowledge that would best prepare law students for evaluating and using the new technologies.

William Henderson, a law professor at Indiana University's Maurer School of Law, has argued that legal *processing engineering* has changed law practice and will

[1] While I agree that these predictions of displacing attorneys are overblown, Remus and Levy have largely overlooked the AI & Law research reported in this book, research that will enable AR and cognitive computing to assist attorneys in legal practice.

continue to do so, necessitating that law schools teach students process engineering skills.

> Because of the emphasis on process and technology now taking hold within the legal industry, the practical technical skills and domain knowledge [now taught] may be inadequate for a large proportion of law students graduating in the year 2015 . . . [Students] . . . are unprepared to learn that law is becoming less about jury trials and courtroom advocacy and more about process engineering, predictive coding, and the collaborative and technical skills those processes entail. (Henderson, 2013, pp. 505f)

Process engineering (or "reengineering") has been defined in the business and information management literature as a "change process,"

> the aim of [which] is quick and substantial gains in organizational performance by redesigning the core business process, [addressing] a need to speed up the process, reduce needed resources, improve productivity and efficiency, and improve competitiveness. (Attaran, 2004, p. 585)

Information Technology (IT) has been called "the most effective enabling technology" for such business process reengineering, establishing "easy communication, improving the process performance," and helping "the reengineering effort by modeling, optimizing and assessing its consequences" (Attaran, 2004, p. 595).

Henderson emphasizes the role process engineering has played in the evolution of legal work, a concept he draws from Richard Susskind's *The End of Lawyers?*, according to which legal work is evolving from bespoke (or customized) to standardized, systematized, packaged, and, ultimately, to a commoditized format:

> These changes [from legal work that is bespoke to . . . commoditized] are made possible by identifying recursive patterns in legal forms and judicial opinions, which enables the use of process and technology to routinize and scale very cheap and very high quality solutions to the myriad of legal needs. [F]ormerly labor-intensive work that has traditionally been performed by entry-level United States law school graduates ... is now being done by Indian law graduates [working for Legal Process Outsourcers (LPOs)], who are learning how to design and operate processes that extract useful information from large masses of digital text. Not only are the Indian law graduates getting the employment, they are learning valuable skills that are entirely – entirely – absent from U.S. law schools. (Henderson, 2013, pp. 479, 487)

In focusing on the use of process and technology to design cost-efficient methods to deliver legal solutions, Henderson agrees with Susskind that *commoditization* is the culmination of this evolution of legal work.

> A legal commodity . . . is an electronic or online legal package or offering that is . . . made available for direct use by the end user, often on a DIY [Do It Yourself] basis. [T]he word "commodity" in a legal context [refers] to IT-based systems and services . . . [that are] undifferentiated in the marketplace (undifferentiated in

> the minds of the recipients and not the providers of the service). For any given commodity, there may be very similar competitor products. (Susskind, 2010, p. 31ff)

In other words, the result of legal commoditization is a software service or product that anyone can purchase, download, and use to solve legal problems without hiring an attorney, or, in current parlance, a kind of computerized legal application, a "legal app."

1.3.1. *Former Paradigm: Legal Expert Systems*

The two concepts, process engineering and commoditization, raise interesting questions. If process engineering of legal services is rethinking how to deliver "very cheap and very high quality" solutions, who or what will be responsible for tailoring those solutions to a client's particular problem? If, as Susskind mentions, commoditization means "Do It Yourself," does that mean the client is on its own? In other words, what kind of support does the legal app provide? In particular, can the legal app perform some level of customization?

Not so long ago, the paradigm computational model for designing a legal app would have been a legal expert system. As Susskind, the developer of a pioneering legal expert system, defined them,

> "expert systems" are computer applications that contain representations of knowledge and expertise . . . which they can apply – much as human beings do – in solving problems, offering advice, and undertaking a variety of other tasks. In law, the idea is to use computer technology to make scarce expertise and knowledge more widely available and easily accessible. (Susskind, 2010, p. 120f)

Typically, legal expert systems deal with narrow areas of law but have enough "knowledge and expertise" in the narrow domain to ask a client user pertinent questions about his/her problem, to customize its answer based on the user's responses, and to explain its reasons. Their "expertise" comprises *heuristics* that skilled practitioners use in applying legal rules to specific facts. These heuristics are "rules of thumb," frequently useful but not guaranteed to lead to a correct result (Waterman and Peterson, 1981).

The rules are represented in a declarative language specifying their conditions and conclusion. They are derived through a largely manual knowledge acquisition process: manually questioning human experts, presenting them with problem scenarios, inviting them to resolve the problems, and asking them what rules the experts applied in analyzing the problem and generating a solution (Waterman and Peterson, 1981).

Waterman's Product Liability Expert System

Don Waterman's legal expert system (let's call it W-LES) is a classic example from the 1980s of a CMLR that performed limited but automatic legal reasoning around a practical problem. It provided advice on settlement decisions of product liability

[RULE3.1: DEFINITION OF LOSS]
IF the type of the plaintiff's loss is "injury"
THEN assert the plaintiff is injured by the product.

[RULE3.2: DEFINITION OF LOSS]
IF the type of the plaintiff's loss is "decedent"
THEN assert the plaintiff does represent the decedent and the decedent is killed by the product.

[RULE3.3: DEFINITION OF LOSS]
IF the type of the plaintiff's loss is "property-damage"
THEN assert the plaintiff 's property is damaged by the product.

[RULE4: STRICT LIABILITY DEFINITION]
IF (the plaintiff is injured by the product
or (the plaintiff does represent the decedent
and the decedent is killed by the product)
or the plaintiff's property is damaged by the product)
and the incidental-sale defense is not applicable
and (the product is manufactured by the defendant
or the product is sold by the defendant
or the product is leased by the defendant)
and the defendant is responsible for the use of the product
and (California is the jurisdiction of the case
or the user of the product is the victim
or the purchaser of the product is the victim)
and the product is defective at the time of the sale
and (the product is unchanged from the manufacture to the sale
or (the defendant's expectation is "the product is unchanged
from the manufacture to the sale"
and the defendant's expectation is reasonable-and-proper))
THEN assert the theory of strict liability does apply to the plaintiff's loss

FIGURE 1.1. Heuristic rules defining loss and strict liability (Waterman and Peterson, 1981)

disputes (Waterman and Peterson, 1981). The inputs to W-LES were descriptions of disputes involving product liability. As outputs, W-LES recommended settlement values and explained its analyses.

The recommendations of W-LES whether to settle a legal dispute and for how much were based on heuristic rules, including claims adjusters' rules for calculating damages and "formalized statements of the California legal doctrine for product liability as stated in statutes, court opinions, and legal treatises" (Waterman and Peterson, 1981, p. 15). Figure 1.1 illustrates the program's heuristic rules defining three kinds of losses and the claim of strict liability.

W-LES mechanically processed a fact situation by applying these heuristic rules in a kind of *forward chaining*. Its inference engine cycled through the rules, testing if any could "fire," that is, if a rule's conditions were satisfied by the facts in the database representing the current problem. If so, the applicable rule did fire and its deduced consequences were added to the database. The inference engine repeatedly cycled through its rules until no more rules could apply.

Ideally, by the end of the process, the rules whose conclusions represented a solution to the problem have "fired" successfully, yielding a prediction and an assessment (or in other legal expert systems, a selection and completion of a relevant legal form). The explanation of the result consists of an "audit trail" or trace back through the rules that fired and the satisfied conditions that led to their firing (Waterman and Peterson, 1981).

Other expert systems applied rules through *backward chaining*. The inference engine begins with a set of desired goals, picks one, and cycles through its database of rules (and facts) in search of a rule whose conclusion is the desired goal. Then, it adds that rule's conditions to the set of desired goals and repeats the cycle until all of the goals are satisfied or there are no more rules (or facts) with which to satisfy remaining goals (Sowizral and Kipps, 1985, p. 3).

Waterman faced three design constraints in developing legal expert systems: legal rules vary across jurisdictions; legal rules employ ill-defined legal concepts; and inferences in the proof are uncertain.

First, different states' legal rules of product liability differ, for instance, in whether the rule of contributory or comparative negligence applies. If contributory negligence applies, the plaintiff's negligence eliminates liability. If comparative negligence, the plaintiff's negligence proportionately reduces the plaintiff's recovery. Waterman addressed this problem by representing multiple states' rules and allowing users to specify which rules to apply in order to demonstrate the differences in outcome.

Second, the legal rules employed some legal concepts without defining them (i.e., "imprecise terms" in Waterman's parlance), such as "reasonable and proper" or "foreseeable" (Waterman and Peterson, 1981, p. 18). Waterman considered a number of possible solutions. These included providing more "rules that describe how an imprecise term was used previously in particular contexts," displaying "brief descriptions of instances of prior use of the imprecise term" and letting the user decide, comparing "prior cases in which the term applied, and provid[ing] a numeric rating that indicates the certainty that the rule ... applies ... In the end, he settled on having the system ask the user if the term applied" (Waterman and Peterson, 1981, p. 26).

Third, litigators are uncertain about proving factual issues and applicable legal doctrine. Waterman's suggestions included incorporating the uncertainties as additional premises within each rule or treating uncertainties as a separate rule to be applied after other rules have been considered. Users would "consider a case independently of ... uncertainty, reach a tentative conclusion, and then adjust that conclusion by some probabilistic factor that represents their overall uncertainty about the case" (Waterman and Peterson, 1981, p. 26).

Modern Legal Expert Systems

Although no longer the paradigm, legal expert systems are still widespread in use in a number of contexts.

Neota Logic provides tools for law firms, law departments, and law school students to construct expert systems. Its website offers examples of computerized advisors concerning questions involving, for instance, the FCPA, bankruptcy risks in cross-border transactions, and the Family and Medical Leave Act (Neota Logic, 2016) (see Section 2.5.1).

CALI, the Center for Computer-Assisted Legal Instruction, and IIT Chicago-Kent College of Law's Center for Access to Justice & Technology, overseen by Professor Ron Staudt, provide a web-based tool to author expert systems. Using the tool, non-programmers with legal skills can create expert systems called A2J Guided Interviews® that lead self-represented litigants through a legal process resulting in a document to be filed in court (A2J, 2012).

As discussed in Section 2.5, firms employ management systems with expert-systems-style business rules to monitor whether their processes comply with relevant regulations.

While still widely used, legal expert systems may not be the paradigm "killer app" for the legal domain. There are at least three reasons for this. First, the techniques developed to enable expert systems to deal with uncertain and incomplete information tend to be *ad hoc* and unreliable. Second, the manual process of acquiring rules is cumbersome, time-consuming, and expensive, a knowledge acquisition bottleneck that has limited the utility of expert systems in law and many other fields (Hoekstra, 2010). Third, text analytics cannot solve this particular knowledge acquisition bottleneck. While the new text analytics can extract certain kinds of semantic legal information from text, they are not yet able to extract expert systems rules.

From time to time, we will return to expert systems, their promise, and their limitations in this book; suffice it to say here that if the legal app is to customize solutions to the particularities of the user's problem, it may be necessary to find some other paradigms.

1.3.2. *Alternative Paradigms: Argument Retrieval and Cognitive Computing*

Unlike expert systems, the two alternative paradigms, AR and cognitive computing, do not purport to solve users' legal problems on their own. Instead, computer programs extract semantic information from legal texts and use it to help humans solve their legal problems.

Conceptual information retrieval, of course, is not new. AI has long sought to identify and extract semantic elements from text such as concepts and their relationships. As defined by Sowa, "concepts represent any entity, action, or state that can be described in language, and conceptual relations show the roles that each entity plays" (Sowa, 1984, p. 8). Similarly, it has long been a goal of AI to make information retrieval smarter by using the extracted semantic information to draw inferences about the retrieved texts. Roger Schank employed the term, "conceptual information retrieval" in 1981 to describe:

> a system to deal with the organization and retrieval of facts in relatively unconstrained domains (for example, . . ., scientific abstracts). First, the system should be able to automatically understand natural-language text – both input to the database

> and queries to the system . . . in such a way that the conceptual content or meaning of an item can be used for retrieval rather than simply its key words . . . If categories are specified by concepts, and if the natural-language analyzer parses text into a conceptual representation, then inferences can be made from the conceptual representations (or meanings) of new items to decide which categories they belong in. (Schank et al., 1981, pp. 98, 102)

Nor is conceptual legal information retrieval new. Pioneering efforts to achieve conceptual retrieval in the legal domain were undertaken by Hafner (1978) and Bing (1987). As discussed in Sections 7.7 and 11.2, modern legal IR services take into account the substantive legal concepts and topics of interest that users intend to target. Other recent work has focused on extending conceptual information retrieval systems so that they return legal information conceptually related not just to the query but to the problem to which the user intends to apply the targeted information (see Winkels et al., 2000).

Today, *conceptual legal information retrieval* can be defined as automatically retrieving relevant textual legal information based on matching concepts and their roles in the documents with the concepts and roles required to solve the user's legal problem. As the definition makes clear, conceptual legal information retrieval is different from ordinary legal IR. It focuses on modeling human users' needs for the information they seek in order to solve a problem, for instance in the legal argument a user seeks to make, and on the concepts and their roles in that problem-solving process.

Even focusing conceptual legal IR on helping users construct viable arguments in support of a claim or counter an opponent's best arguments is not new. Dick and Hirst (1991) explored manually representing cases in terms of schematic argument structures to support lawyers' "information seeking . . . to build an argument to answer the problem at hand." At that time, however, the authors could only assume "that in due course, . . . both language analysis and language generation by machine will be possible."

Their assumption has finally come true. For years, robust means for extracting such conceptual, argument-related information from natural language texts for purposes of conceptual legal information retrieval were not available. Today, however, language analysis tools that can *automatically* identify argument-related information in case texts are finally available, and with them a new paradigm is born: robust conceptual legal IR based on argument-related information, or AR as it is referred to in Section 10.5.

Cognitive computing is a second new paradigm for system development. Despite its name, cognitive computing is *not* about developing AI systems that "think" or perform cognitive tasks the way humans do. The operative unit of cognitive computing is neither the computer nor the human but rather the collaborating team of computer *and* human problem-solver(s).

> [I]n the era of cognitive systems, humans and machines will collaborate to produce better results, each bringing their own superior skills to the partnership. The machines will be more rational and analytic – and, of course, possess encyclopedic memories and tremendous computational abilities. People will provide expertise, judgment, intuition, empathy, a moral compass, and human creativity. (Kelly and Hamm, 2013)

In a cognitive computing paradigm, human users are ultimately responsible for customizing their own solution using a legal app, but the commoditized legal service technology should apprise the humans of the need for customization and support them with customized access to relevant legal information to help them construct a solution. That is, the legal app will not only select, order, highlight, and summarize the information in a manner tailored to a human user's specific problem but also explore the information and interact with the data in new ways not previously possible.

In order for this approach to succeed, the technology does not need to solve the user's problem. It will not be a legal expert system. It will, however, need to have some "understanding" of the information at its disposal and of the information's relevance in the human's problem-solving process and to make the information conveniently available at the right times and in the right contexts. In this respect, AR is consistent with cognitive computing where responsibility for finding and applying resources to solve a user's problem is divided between intelligent tasks the computer can best perform and those addressed to human users' expertise.

Expert systems and cognitive computing paradigms differ in the sources of their respective "knowledge." In the former, expertise is embodied in rules that human experts apply in solving such problems, rules that usually have been constructed manually by engineers in the knowledge acquisition process.

In the cognitive computing paradigm, in contrast, the knowledge is embodied in the corpus of texts from which the program extracts candidate solutions or solution elements and ranks them in terms of their relevance to the problem. This assumes, of course, that an available corpus of texts contains information relevant to the type of problem. For instance, if the problem is a fact situation about which to make arguments concerning a legal claim, a corpus of legal cases involving that type of claim would be required.

The technology cannot read the texts in the sense that humans read, but it will have techniques for intelligently processing the texts, identifying those elements that are relevant to a problem, and bringing them to the user's attention in an appropriate way. Significantly, the program's knowledge for assessing relevance, that is, for identifying, ranking, and presenting candidate solutions or elements, is acquired not primarily manually but automatically by extracting patterns from some collection of domain-specific data using ML.

1.3.3. *Toward the New Legal Apps*

At least, that is the goal. Although researchers in university and commercial settings recognize its extraordinary potential, at the time of writing, probably no one really knows exactly how to implement cognitive computing in the legal domain. Clearly, it will not be easy, but it does seem feasible.

AI & Law researchers and technologists are actively engaged in applying the new QA, IE, and argument mining techniques to problem-solving processes in the legal domain. They see the potential for modeling legal reasoning, argumentation, and prediction of integrating computational techniques that have been developed over the years to represent statutory rules and case decisions. The AI & Law tools illustrate the elements in legal texts that the new text processing techniques should target and the legal tasks that can then be accomplished.

They recognize, too, that AI & Law research has identified design constraints that limit, or firmly guide, what CMLRs can accomplish. Sometimes the constraints can be finessed or ignored given the task a legal app addresses, but it is good to know about them in advance. The design constraints will help technologists avoid reinventing the wheel or charging down dead ends.

The next few years will be exciting times in the development of legal practice and the history of AI & Law! The aim of this book is to present the available tools, explore how they can be integrated with the new text processing tools, and equip readers to participate in this technological revolution.

1.4. WHAT WATSON CAN AND CANNOT DO

But wait a minute! Isn't the revolution already over? IBM's Watson performs remarkable feats of QA based on IE from text. Its cousin, the Debater program already mines arguments from text. Perhaps one can simply turn Watson and Debater loose on legal texts and watch them perform legal reasoning, no?

No, as already noted, programs like Watson and Debater will not perform legal reasoning. This section addresses why not. At the same time, Watson offers a conceptual framing and text analytic tools that can be instrumental in addressing the challenge of building programs that can perform legal reasoning from text.

Highlighting Watson and Debater here is *not* meant to suggest that the future development of intelligent tools for digital age legal practice depends on IBM's proprietary techniques. In fact, Watson is based on an open-source text processing and IE tool, the Unstructured Information Management Architecture (UIMA). An alternative to UIMA, the open-source GATE annotation environment, was used in topic labeling in connection with the Debater research.

In designing and explaining the Watson technology, however, IBM researchers have framed some of the component tasks of text analytics. It is convenient to take advantage of that framing in order to suggest the tasks' potential application in the legal domain.

1.4.1. *IBM's Watson*

In February 2011, "Jeopardy!," a TV game show popular with older retirees and younger nerds, captured the imagination of the American public. The game's setup and rules are straightforward: longtime host Alex Trebek presides as three contestants face a game board with six categories. Each category has five items of increasing value. Each item comprises a small window; when opened it displays an answer. The contestants race to hit the buzzer first for a chance to state the question that goes with the answer, win the value of the item, and choose the next category and item. The cardinal rule is that the contestant's response, his or her "answer," must be in the form of a question.

The game show had been an evening TV staple since 1984, but this evening was different: one of the three contestants was not human. A team at IBM Research led by David Ferrucci had designed a computer system named "Watson" especially to participate in the "Jeopardy!" game on prime time TV against the two top human champions: Brad Rutter, whose winnings from previous appearances on "Jeopardy!" topped $3.25 million, and Ken Jennings, who, with a winning streak of 74 games, was nearly a fixture of the show, himself.

By the end of three consecutive nights of play, Watson had beaten the human champions convincingly. It was a *tour de force* for IBM Research whose Deep Blue chess-playing program had beaten Gary Kasparov, the world's reigning human chess champion, 14 years before.

Of course, Watson was fallible. Famously it flubbed in "Final Jeopardy!," the last round of the evening when the host announces the category and the show jumps to a commercial break. In the meantime each, contestant wagers an amount up to his or her current total score. When the host finally reveals the "Final Jeopardy!" answer, the contestants have 30 seconds to write their responses on an electronic display, accompanied by a now familiar jingle that has come to epitomize the tension of thinking under time pressure (i.e., "Think," composed by Merv Griffin, the true genius of the "Jeopardy!" gameshow).

On this evening, the "Final Jeopardy!" category was "U.S. Cities for $400." The answer was "Its largest airport is named for a World War II hero; its second largest for a World War II battle." "Think" jingled to its inevitable conclusion, and the host asked each contestant to reveal his, or its, question.

The audience groaned when Watson's response appeared, "What is Toronto?????" Probably, it was not because the audience was amazed that Watson had gotten it wrong. The correct response was "What is Chicago?" Anyone could see that the question was tricky. One might know that Chicago's second largest airport, Midway Airport, was named for a famous World War II naval battle in the Pacific, but hardly anyone knows that Navy flying ace, Lieutenant Commander Edward Henry "Butch" *O'Hare*, was a hero of that war.

Instead, the audience probably was amazed that Watson did not know a common-sense bit of trivia: *Everyone* knows that Toronto is not a U.S. city!

Although Watson's blunder was not costly (Watson wagered a mere $947), it was revealing: Watson does not have knowledge of facts and information "hard wired" in some way such as expert rules. Rather, for each question/answer (Q/A) type, Watson *learns* how to extract candidate answers to the question (or questions to the answer in "Jeopardy!" speak) from millions of texts in its database. For each Q/A type, it also learns the kinds of evidence that enable it to recognize answers to that type of question, evidence in the form of syntactic features and semantic clues in the text, where the semantic clues include references to certain concepts and relations. For each Q/A type, Watson has also learned how much confidence to have in the various types of evidence associated with the texts. As indicated by the repeated question marks, Watson had little confidence in its response (Ferrucci et al., 2010).

Watson learns from a training set of documents, for which humans marked-up or "annotated" many instances of each type of Q/A pair. The annotated training texts serve as examples of how to extract information about that type of question and answer. Watson learns the how-to-extract information from the training examples and can apply it to extract information from other texts that have not been marked up, generalizing the how-to information in the process (Ferrucci et al., 2010).

In explaining Watson's response, two IBM Watson project researchers pointed out that Chicago was a very close second on Watson's list of possible answers, but that Watson had not found much evidence to connect either of the city's airports to World War II. In addition, Watson had learned that category phrases like "U.S. Cities" are not very dispositive. If "This U.S. city's . . ." had appeared in the answer, Watson would have given US cities more weight. Finally, there *are* cities named Toronto in the United States, for example, Toronto, IL, Toronto, IN, Toronto, IA, Toronto, MI, Toronto, OH, Toronto, KS, and Toronto, SD, and Toronto, Canada *does* have an American League baseball team (Schwartz, 2011).

Applying Watson in Law

It appears that IBM would like to apply Watson technology (also known as Deep QA) to the legal domain (see Beck, 2014).[2] According to IBM General Counsel, Robert C. Weber,

> Pose a question and, in milliseconds, Deep QA can analyze hundreds of millions of pages of content and mine them for facts and conclusions . . . Deep QA won't ever replace attorneys; after all, the essence of good lawyering is mature and sound reasoning . . . But the technology can unquestionably extend our capabilities and help us perform better . . . At IBM, we're just starting to explore about how Deep QA can be harnessed by lawyers. (We're pretty sure it would do quite well in a multi-state bar exam!) But already it's becoming clear that this technology will be useful in a couple of ways: for gathering facts and identifying ideas when building

[2] Ross Intelligence (2015), discussed at Section 12.2, applies Watson technology in the legal domain.

> legal arguments. The technology might even come in handy, near real-time, in the courtroom. If a witness says something that doesn't seem credible, you can have an associate check it for accuracy on the spot. (Weber, 2011)

Watson's mistake, however, suggests some of the challenges for applying Watson technology to the legal domain. One can but imagine the game of "LEGAL Jeopardy!" Host Alex reveals "The Category is: Sports law." Ken Jennings selects "Sports law for $1.2 Million"! The window slides open: The answer is: "American League Baseball teams that cannot legally hire replacement workers during an economic strike."

A buzzer sounds. "Watson?" Alex responds.

Watson replies, "What are the Toronto Blue Jays?"

Alex smiles. "Correct! The Toronto Blue Jays cannot hire replacement workers during an economic strike."

This time, knowing that Toronto is not a U.S. city is certainly a relevant jurisdictional consideration in legally analyzing the issue. Unlike the other American League teams, the Toronto Blue Jays are not subject to U.S. labor law, but to provincial labor law (Ontario) where the rules on hiring replacement workers differ, according to Lippner (1995), a law review article regarding the 1995 baseball strike.

Watson, however, would not necessarily need to know Toronto's location or nationality in order to answer the question correctly. Watson does not have a set of rules specifying the nation in which Toronto is located or the laws that apply to it, nor rules for reasoning about whether Canadian federal or Ontario provincial law would govern this labor law issue. But that is not how Watson would answer such questions, anyway.

As long as Watson's corpus contains the above law review article, an appropriately trained Watson could learn to identify it as relevant to this type of question, extract from it the relevant answer, and assess its confidence in the answer's responsiveness.

This is a *legal* question, however. When it comes to fielding legal questions, one expects more than just an answer. One expects an explanation of why the answer is well-founded. Presumably, Watson could not explain the answer it had extracted. Explaining the answer requires one to reason with the rules and concepts relevant to choice of law and legal subject matter, knowledge that Watson does not have and could not use.

An appropriately trained Watson could have learned types of evidence for recognizing relevant question and answer pairs, including semantic clues, for instance, concepts and relations like "legally hire," "replacement workers," "economic strike." It could also have learned how much weight to accord to this evidence in assessing its confidence that the question and answer are related.

Whether this kind of evidence is sufficient for Watson to explain the answer in a manner acceptable from a legal viewpoint is another matter. Watson's how-to-extract

knowledge does not appear to extend that far, yet (but see the discussion of IBM's Debater program in Section 1.4.3).

On the other hand, the author of the law review article *does* have that legal knowledge and has summarized in his article how application of that knowledge (i.e., of the rules and cases concerning jurisdiction, legal subject matter, and choice of law) justifies his conclusion. If Watson can be trained to recognize and extract those arguments explaining legal conclusions, it would be able to point human users to the author's explanation, even if Watson could not itself construct the explanation from first principles. Even then, of course, there is an issue about whether the article and its explanation are still current.

1.4.2. *Question Answering vs. Reasoning*

This raises a question: Can a program based on Watson's technology ever really reason? Could it, for example, analyze a first-year law school problem in contract law? In the above quote, IBM's Counsel, Robert Weber emphasized "sound reasoning" and declared parenthetically that "We're pretty sure it [Watson] would do quite well in a multistate bar exam!" (Weber, 2011).

But, could the Watson technology handle the *essay* part of a state bar exam? Or could it do so only if someone (Google?) has happened to store the contents of old exam blue books (assuming computerized analysis ever manages to "read" law students' handwriting, a superhuman task if ever there was one)? Will it only(!) be a highly sophisticated technique for retrieving past answers to similar questions, and, perhaps, for highlighting the evidence (syntactic features and semantic clues in the text concerning concepts and relations) that justifies its confidence in its answer? Will it be able to adapt past arguments to a new problem? Or will it be able to solve the new problem from first principles and explain its reasoning?

In order to gain some insight into the kind of legal reasoning involved in addressing a bar exam essay question, let's briefly examine a classic CMLR by Ann Gardner (her program was unnamed but let's call it AGP) which already in the 1980s had analyzed legal issues from typical first-year law school contracts course final exam problems (Gardner, 1987).

AGP is offered here as an example of a systematic approach to computationally modeling legal reasoning about exam questions involving contract law and as a contrast to the Watson approach.

Gardner's First-Year Contracts Problem Analyzer

Anyone who has attended law school will recognize (probably with a shudder) the type of problem AGP handled: A putative buyer and seller exchange two weeks' worth of chronologically overlapping and sometimes inconsistent telegrams and purchase orders concerning a possible purchase of a carload of salt. Having sent an

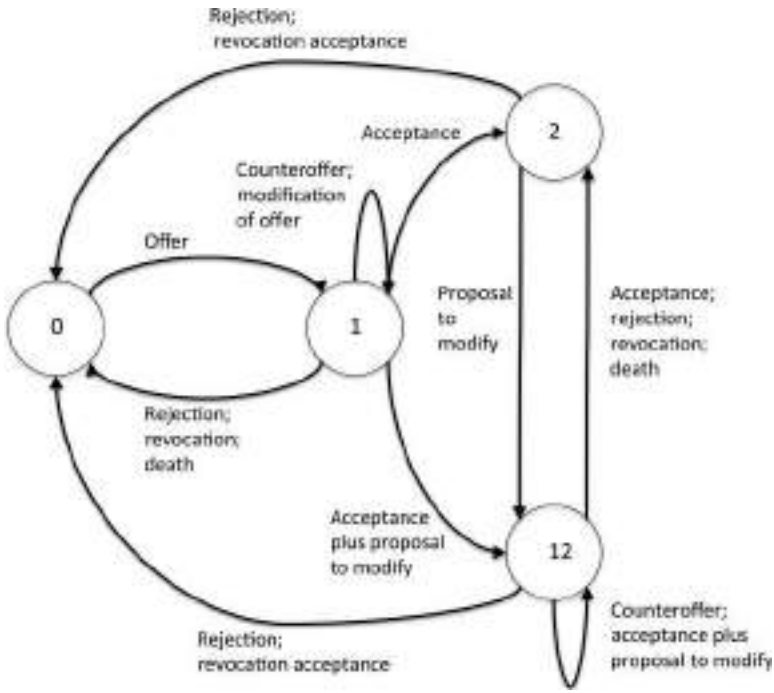


FIGURE 1.2. ATN for offer and acceptance problems with four states: (0) no relevant legal relations; (1) offer pending; (2) contract exists; (12) contract exists and proposal to modify is pending (Gardner, 1987, p. 124)

apparent acceptance of an apparent offer, the buyer finds a cheaper source and sends a telegram purporting to reject. The question is "Has a contract been concluded?"

The inputs to AGP were descriptions of this type of offer and acceptance problem represented by a human (Gardner) in a logic language (illustrated below). AGP used an *augmented transition network* (ATN) to analyze such problems and output an analysis of the contracts issues.

An ATN is a graph structure that analyzes problems involving sequences of events as a series of states and possible transitions from one state to the next. It is "augmented" with rules that define each such possible state transition.

AGP's ATN, shown in Figure 1.2, represented *legal* states in the analysis of an offer and acceptance problem in contract law (i.e., no relevant legal relations (0), offer pending (1), contract exists (2), contract exists and proposal to modify is pending (12)). The arcs represented events or transitions from one legal state to another: from no relevant legal relation (0) to an offer pending (1) via an offer, from an offer pending (1) to contract exists (2) via an acceptance, etc.

Each arc had associated with it the rules of contract law dealing with offer and acceptance. These rules set forth the legal requirements for moving from one state

to the next. For instance the offer arc from (0) to (1) has one associated rule, the definition of "offer," based on *Restatement of Contracts, Second*, section 24: "An offer is the manifestation of willingness to enter into a bargain, so made as to justify another person in understanding that his assent to that bargain is invited and will conclude it" (Gardner, 1987, p. 142).

AGP processed events in the problem in chronological order, storing its analysis in a detailed analysis tree and summarizing it in an output summary tree. The program repeats the following steps until it has processed each event in the problem:

1. Take the next event in the problem.
2. Find out the current state from the detailed analysis tree. Determine from the ATN the possible arcs out of that state.
3. For each possible arc, test if the event satisfies the rules associated with the arc and update the detailed analysis tree with the test results.
4. If the test involves a "hard" legal question (see below), that is, presents two legally defensible ways of evaluating the event, insert a branch for each interpretation into the detailed analysis tree.
5. Edit the detailed analysis tree to update an output summary tree of network states representing the different "interpretations" of the events.

For example, AGP starts with a first event:

> On July 1 Buyer sent the following telegram to Seller: "Have customers for salt and need carload immediately. Will you supply carload at $2.40 per cwt?" Seller received the telegram the same day.

The events were input not in English text, but in a logic-based representation language. AGP could not read text, so a human had to manually represent that information in the logic representation (i.e., predicate logic, defined in Section 2.3.2). For instance, some excerpts of the representation are:

> (send Send1) (agent Send1 Buyer) (ben Send1 Seller) (obj Send1 Telegram1)
> (telegram Telegram1) (sentence S13) (text S13 "Will you supply carload at $2.40 per cwt?")
> (prop-content S13 Prop13) (literal-force S13 Q13)(yes-no-question Q13)
> (effective-force S13 R13) (request R13).
> (Gardner, 1987, pp. 89, 105, 111)

In the above representation, Send1 is an instance of a Send with the Buyer as the Agent, the Seller as the Beneficiary, and Telegram1, an instance of a telegram, as the Object of the sending. S13 is a sentence whose text is quoted, whose propositional content is represented in Prop13 (defined elsewhere), whose literal force as a speech act is to pose a question but which also effectively presents a request (Gardner, 1987, pp. 89, 105, 111).

In step (3), testing if the event satisfies the arc, the program collects the rules associated with the arc. Like the events, all of the contracts rules were translated manually

into the logical language. For instance, the rule associated with the arc from (0) to (1) defining an "offer," *Restatement of Contracts, Second*, section 24 (above), includes rule antecedents and (italicized) predicates like the following:

1. There is a *manifestation* with some symbolic *content* about an *exchange* by some agent, the prospective *offeror*.
2. The terms of the exchange are specified with *reasonable certainty*.
3. By means of the content of the manifestation, the prospective offeror has performed some *speech act* that invites acceptance by a prospective *offeree* of a proposal for the exchange.
4. The offeree is invited to furnish consideration in the exchange and the prospective offeror is apparently ready to be bound to a contract for the exchange, without doing anything more (Gardner, 1987, pp. 142).

The program checks if the rule's antecedents are satisfied given the facts of all the events processed so far plus the new event. Basically, AGP attempts to bind the artifacts of the problem to the variables in the rule guided by very limited information about what the facts and the antecedents mean.

At any step, there are multiple possible ways to bind the facts and antecedents. The program needs to search through all the possible bindings, leading to a detailed analysis tree with multiple branches. As noted, "hard" questions also lead to branches representing alternative reasonable interpretations. In order to prevent an "exponential explosion" of alternative paths, an editing function prunes the branching analysis using heuristics to focus on the most promising branches.

Incidentally, recent work on so-called "smart" contracts employs finite state automata related to the ATN in Gardner's CMLR (Flood and Goodenough, 2015, p. 42). Researchers have also applied heuristic rules to model the United Nations Convention on the International Sale of Goods and to deduce the temporal legal states of affairs as events occur in the life of a contract (see Yoshino, 1995, 1998).

Gardner's heuristics are a typical example of an AI approach to enable a computer program to handle a task that is taxing even for humans. Law students need to decide on which of the multitude of cross-communications and their contents to focus at any point in their analyses of whether there is a contract.

Gardner's Algorithm for Distinguishing Hard and Easy Legal Questions

Law students (legal practitioners and judges) also need to learn how to distinguish hard and easy questions of law, a determination that takes into account an appreciation of the facts and the substantive legal issues, as well as procedural issues concerning who has the burden of raising the question.

This is a problem that has deep roots in legal philosophy (see, for example, Fuller's critique of Hart's assertion that legal terms have core and penumbral meanings (Hart, 1958; Fuller, 1958)). It also has very practical ramifications. A clinic intake advisor,

A. For every predicate in rule, apply 3 tests:
1. *CommonSenseKnowledge-Answer*: Does Commonsense knowledge rule provide an answer?
2. *Pos-Examples*: Does problem match positive examples of predicate?
3. *Neg-Examples*: Does problem match negative examples of predicate?

B.
If not *CommonSenseKnowledge-Answer*
If not (*Pos-Examples* or *Neg-Examples*) → ***Question hard***
If (only one of *Pos-Examples* / *Neg-Examples*) → ***Question easy***
If (*Pos-Examples* and *Neg-Examples*) → ***Question hard***
If *CommonSenseKnowledge-Answer*
If not (*Pos-Examples* or *Neg-Examples*) → ***Question easy***
If (only one of *Pos-Examples* / *Neg-Examples*)
If (agrees only-one with *CommonSenseKnowledge-Answer*) →
Question easy
else ***Question hard***
If (*Pos-Examples* and *Neg-Examples*) → ***Question hard***

FIGURE 1.3. Gardner's heuristic method for distinguishing hard and easy legal questions (Gardner, 1987; Rissland, 1990)

for instance, needs a way to distinguish clients' easy and hard legal questions in order to direct them appropriately.

From the viewpoint of computational modeling, however, distinguishing hard and easy questions of law presents a conundrum. As Gardner noted, for a computer program to apply a method for distinguishing hard and easy questions, the method must itself be "easy."

AGP employed a heuristic method for distinguishing hard and easy questions of law (Gardner, 1987, pp. 160–1). Figure 1.3 depicts Edwina Rissland's algorithmic recapitulation of AGP's method for distinguishing hard and easy questions (Rissland, 1990).

For every predicate in a rule, the method involves testing whether a commonsense knowledge (CSK) rule provides an answer, or whether the problem matches positive examples of the predicate, negative examples, or both. For instance, if no commonsense rule provides an answer, but there is a match to a positive instance, the question is easy. If, however, a negative instance also matches, the question is hard.

Consider the requirement of there being a *manifestation* of willingness by the prospective offeror to enter into a bargain. As operationalized for AGP, the offeror must have performed a speech act that invites acceptance by a prospective offeree of a proposal for an exchange. Whether there is such a manifestation does not, perhaps, usually present a hard question of law, but it is litigated from time to time. In principle, AGP has a way to decide if it presents a hard question in a particular case. If there is a commonsense rule-like definition of manifestation (or of an appropriate speech

act), and some instances of positive or negative examples of manifestations, AGP can apply them to the facts of the problem, and follow the heuristic in Figure 1.3.

By the time AGP completes its analysis of all of the events in an offer and acceptance problem, it has prepared a detailed analysis tree, traversals of which effectively provide a trace of its reasoning and an explanation of its answer. For instance, in AGP's analysis of event 1 above, it concludes that there is a pending offer, having found in the buyer/offeror's telegram a manifestation of an apparent and reasonably certain readiness to be bound to an exchange (see Gardner, 1987, Fig. 7.1, p. 165).

• • •

AGP illustrates some issues that a program like Watson would need to address if it were to be applied to tackle bar exam essay questions. Computationally modeling legal reasoning about contracts problems requires some model of reasoning with legal rules and concepts. It needs to distinguish between hard and easy questions of law. It also needs an ability to explain its reasoning, and that reasoning has to be intelligible to legal practitioners.

The Watson program that won the Jeopardy! game did not explain its answers. If it did explain its answers, it would probably do so in terms of the syntactic features and semantic clues in the text concerning concepts and relations that justified its confidence in its answer (Ferrucci et al., 2010, p. 73). That kind of an explanation, however, is not likely to correspond to what legal practitioners would expect.

Even if Watson could not perform the kind of reasoning AGP models, could it recognize the features of prior legal explanations and arguments, such as those in old exam blue books from past law school or bar review essay exams, and adapt them to a new problem? Would it be able to recognize when these arguments are relevant to users' queries? What level of detail could it recognize in prior explanations and arguments? Could it recognize not only the legal rules but also the application of the legal rules to the facts of a problem? Could it recognize arguments that particular rule antecedents are satisfied or not?

These are the kinds of questions discussed in detail in Part III of this book, but let us begin to explore them here.

1.4.3. *IBM's Debater Program*

Can Watson be trained to recognize and extract arguments from texts? It appears that the answer is "yes"! In Spring 2014, an IBM executive demonstrated a new program named "Debater," a descendant of Watson that employs some of the text processing technology of the Watson program to perform argument mining (see, e.g., Newman, 2014, demo at Dvorsky, 2014).

On any topic, the Debater's task is to "detect relevant claims" and return its "top predictions for pro claims and con claims." In the example of Debater's output, upon

FIGURE 1.4. Argument diagram of IBM Debater's output for violent video games topic (root node) (see Dvorsky, 2014)

inputting the topic, "The sale of violent video games to minors should be banned," Debater:

1. Scanned 4 million Wikipedia articles,
2. Returned the 10 most relevant articles,
3. Scanned the 3,000 sentences in those 10 articles,
4. Detected those sentences that contained "candidate claims,"
5. "[I]dentified borders of candidate claims,"
6. "[A]ssessed pro and con polarity of candidate claims,"
7. "Constructed a demo speech with top claim predictions,"
8. Was then "ready to deliver!" (Dvorsky, 2014)

While Debater's output in the video was aural, one can present the text of its output in visual terms. Figure 1.4 shows an argument diagram constructed manually from the video recording of Debater's aural output for the example topic (available at Dvorsky, 2014). The box at the top (i.e., the "root node") contains the topic proposition. Nodes linked to it with solid-lined arrows (i.e., "arcs") support that proposition; the dashed arcs attack it. The elapsed time from inputting a topic to outputting an argument reportedly is from three to five minutes. In subsequent presentations, Debater's output has been demonstrated for other diverse topics.

Debater's argument regarding banning violent video games in Figure 1.4 invites comparison to a *legal* argument involving a similar topic shown in Figure 1.5. It concerns the constitutionality of California (CA) Civil Code sections 1746–1746.5 (the "Act"), which restricted sale or rental of "violent video games" to minors. The Court in *Video Software Dealers Assoc. v. Schwarzenegger*, 556 F. 3d 950

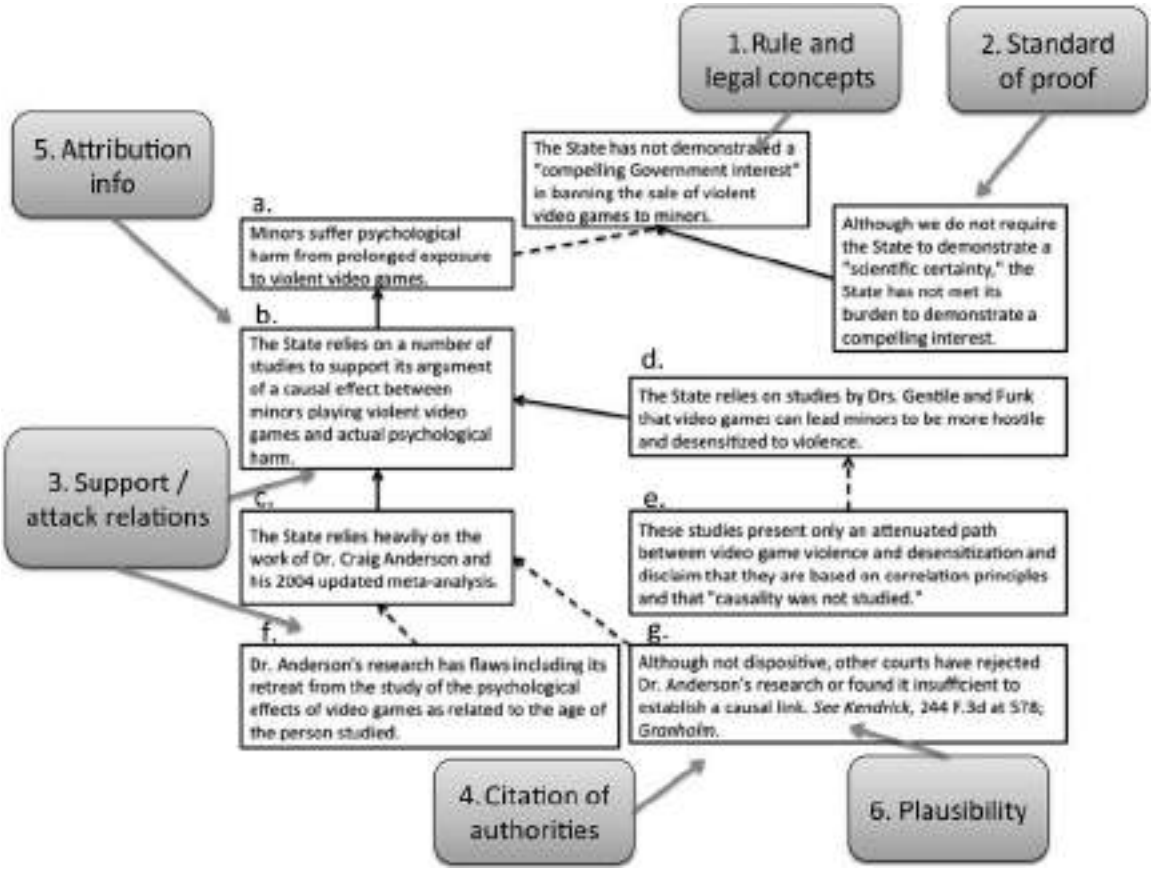


FIGURE 1.5. Diagram representing realistic legal argument involving violent video games topic

(9th Cir. 2009) addressed the issue of whether the Act was unconstitutional under the 1st and 14th Amendments of the U.S. Constitution. As a presumptively invalid content-based restriction on speech, the Court subjected the Act to the strict scrutiny standard.

The Court held the Act unconstitutional because the State had not demonstrated a compelling interest that "the sale of violent video games to minors should be banned." Figure 1.5 shows excerpts from the portion of the opinion in which the Court justified this conclusion. The nodes contain propositions from that portion and the arcs reflect the explicit or implied relations among those propositions based on a fair reading of the text. As above, the solid arrows signify that the proposition in the node at the base of the arrow supports the proposition in the node to which the arrow points; dashed arrows signify an attack relation. Thus, nodes a, b, c, and d contain propositions on which the State of CA relied to support its compelling government interest. Nodes e, f, and g contain propositions the Court employs to attack the State's propositions.

The argument diagrams in Figures 1.4 and 1.5 address nearly the same topic and share similar propositions, reflecting the fact that the Court's argument addresses some of the very same kinds of reasons and evidence as Debater's argument.

The callout boxes in Figure 1.5, however, illustrate some key features of legal argument evidenced by the Court's argument. In particular, (1) legal rules and concepts govern the Court's decision of the issue. (2) Standards of proof govern its assessment of evidence. (3) The argument has an internal structure; support and attack relations connect the various claims. (4) The Court explicitly cites authorities (e.g., cases, statutes). (5) *Attribution* information signals or affects the Court's judgments about belief in an argument (e.g., "the State relies"). (6) Candidate claims in a legal document have different degrees of plausibility.

This is not to criticize Debater's argument, which is *not* and does not purport to be a legal argument.

On the other hand, given the intention of applying Watson and, presumably, Debater to legal applications and argumentation, the comparison emphasizes the importance of addressing these features of legal argument if and when Debater *is* applied in a legal domain. It would be essential that Debater can identify the types of concepts, relationships, and argument-related information enumerated above and illustrated in Figure 1.5 in order for the system to be able to recognize and interpret legal arguments. A program so endowed could improve legal information retrieval, focusing users on cases involving concepts, concept relations, and arguments similar to the one the human user is aiming to construct. It could also highlight and summarize the relevant arguments for the user's benefit (see Section 11.3).

Finally, if the system were to perform any automated reasoning based on the retrieved texts in order to assist the user in solving his/her problem, such as by comparing arguments, predicting outcomes, or suggesting counterarguments, it would need an ability to identify concepts, concept relations, and arguments in the texts.

It is in this connection that the kinds of legal reasoning argument models and argument schemes described in Part I will likely be essential. This is the focus of Section 12.3.

1.4.4. *Text Analytic Tools for Legal Question Answering*

Watson's fundamental task was to answer questions. In the context of the "Jeopardy!" game that was enough to beat the reigning human champions.

Legal QA could be a great boon to making legal knowledge more accessible. Imagine the utility of a service that answers questions about landlord tenant law in a large metropolitan area. Of course, lawyers know that legal QA can be quite complex. An answer needs to be tailored to the questioner's circumstances. It matters, for example, if the apartment building is in Toronto, Canada, or Toronto, Kansas. Explanations and arguments need to be provided. Assumptions need to be clarified on which the answer is based and which often limit its applicability.

Many practical legal questions, however, do not require explanation and argument. At a November 2014 workshop in IBM's Chicago offices, Paul Lippe of Legal OnRamp (LOR) demonstrated an application with a large corpus of contracts involving two corporations engaged in a high volume of repeat transactions over time (Legal OnRamp, 2015). Corporate legal staffs involved in contract monitoring and maintenance would like to be able easily to answer such questions as: Which contracts include certain terms or term language such as a disclaimer of liability for consequential losses? For which contracts is a particular type of term embedded in the body of the contract as opposed to in an appendix? Such queries may be quite useful. For instance, certain terms may need to be updated frequently, and it may be easier or cheaper to do so if the terms are located in a contract's appendix. Finding the contracts in a large corpus with such a term in the body can assist the legal staff to target contracts that should be restructured.

Such queries cannot be easily and reliably answered with ordinary information retrieval tools. Using Boolean searches and keywords, one cannot easily specify locations within a contract structure or deal with the wide variety of language with which certain kinds of terms may be expressed. For instance, consider the variety of ways in which a disclaimer of liability for consequential losses can be expressed.

In answering questions, Watson analyzes the question, searches for candidate responses from a text corpus, and ranks the candidates according to its confidence that each candidate addresses the question.

Question analysis means analyzing the question text for clues "to determine what [the question] is asking about and the kind of thing it is asking for." This includes parsing the question text, which "produces a grammatical parse of a sentence[,] identifies parts of speech and syntactic roles such as subject, predicate, and object, [and identifies how some sentence segments relate to other] sentence segments." This also includes decomposing suitable questions into "useful and relevant

subparts." The query analysis process does not result in one certain interpretation of what the query means. The "parsing and question analysis result in multiple interpretations of the question and . . . a variety of different queries" (Ferrucci, 2012, pp. 6, 9).

Retrieval and ranking involves searching for candidate answers for each of the query interpretations. "These queries are run against different sources to first generate a broad set of candidate answers." This leads to generating multiple hypotheses about what the query means and how to answer it. "Each candidate answer combined with the question represents an independent hypothesis." "Each [hypothesis] becomes the root of an independent process that attempts to discover and evaluate supporting evidence in its candidate answer" (Ferrucci, 2012, p. 6).

The system uses a set of evidence scoring programs to rank the candidate answers by the likelihood that the answer addresses the question and to assess its level of confidence in the answer's correctness. "Each evidence–answer pair may be scored by 100 independent scorers. Each scoring algorithm produces a confidence. For any one candidate, there may be on the order of 10,000 confidence scores – on the order of one million in total for a single question" (Ferrucci, 2012, p. 9).

Judging the likelihood that each candidate answer is correct is a matter of combining weights associated with the different evidence scores. Watson learns the weights associated with the evidence scores "using a statistical machine learning framework" (Ferrucci, 2012, p. 9).

Thus, in constructing a contracts QA facility, it is likely that the LOR team developed a set of concepts and relations for distinguishing among different types of contractual terms or provisions and for identifying structural features of the contracts. Such concepts probably included *InContractBody*, *InAppendix*, *LiabilityDisclaimer*, *ConsequentialDamages*. The team probably manually annotated a subset of contracts (a training set) for these features. The Watson system then learned statistically to associate various syntactic and semantic information with these features and applied them to annotate the remaining contract texts (the test set).

Figure 1.6 shows a high-level architecture for analyzing texts of legal documents including contracts. Given a query, the program analyzes the question, translates it into a set of structural and conceptual feature constraints on the type of answer sought, identifies candidate documents responsive to the question, and then ranks the candidates. In the contracts application, there may be only a few evidence scorers, some more useful in answering structure-type questions, others better for answering questions regarding provision type. The weighted utilities between evidence scores and types of questions would not be hardwired but learned from positive and negative instances of question/answer pairs.

For semantic text analysis and conceptual information retrieval, two additional tools, shown in dashed boxes in Figure 1.6, are helpful. Relation extraction and concept expansion help to analyze questions and retrieve candidate answers from a corpus.

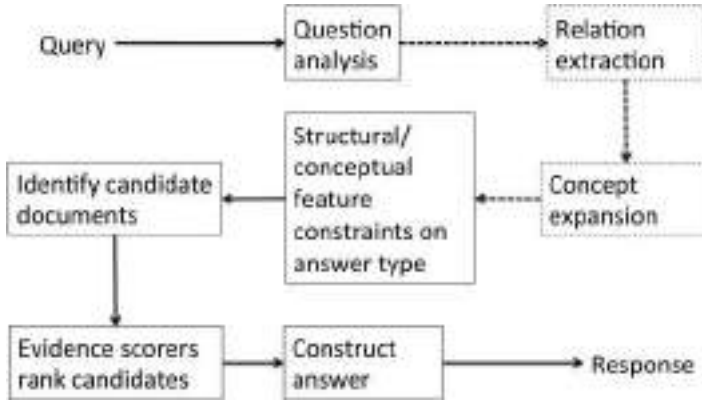


FIGURE 1.6. Architecture of text analyzer for legal documents including contracts. Dashed boxes show components for semantic analysis and conceptual information retrieval

Relation extraction attempts "to find semantic relationships (e.g., [a person may have] starred in, visited, painted, invented, [or have been] naturalized in) between concepts, although they may have been expressed with different words or with different grammatical structures" (Ferrucci, 2012, p. 7).

A system's ability to identify a conceptual relationship, for instance, a particular kind of party signing a particular kind of agreement, is essential for specifying the constraint for purposes of conceptual information retrieval and prediction (see Section 4.5.2). In the above contracts example, the concept of *LiabilityDisclaimer* may be expressed in a variety of ways, for instance, "disclaims liability for incidental or consequential damages," "assumes no responsibility for any loss," or "undertakes no liability for any loss or damage suffered as a result of the misuse of the product," all of which the program must learn are instances of *LiabilityDisclaimer*.

Another example involves claims under a federal statute for injuries caused by vaccines (see Section 10.5). One might seek to retrieve all cases involving assertions that:

<specific-vaccine> <can cause> <generic-injury>

For instance, a court may have held that "DPT vaccine can cause acute encephalopathy and death," a case that would be a useful point of reference to an attorney representing a decedent who had suffered a similar circumstance. A system's ability to identify a causal relationship between a specific vaccine and a type of injury would be essential if the system is to preferentially rank such a case and highlight its finding for the benefit of the user.

In a different legal context, one may wish to retrieve all trade secret misappropriation cases where the:

<defendant> <signed> <nondisclosure-agreement>

Some examples drawn from real trade secret cases include:

1. Newlin and Vafa had signed nondisclosure agreements prohibiting them from using ICM software and tools upon leaving ICM.
2. Ungar signed a nondisclosure agreement.
3. Defendant Hirsch was employed by plaintiff, where he executed a nondisclosure agreement (Ashley and Brüninghaus, 2009, p. 141).

Concept expansion identifies "concepts that are closely related to those given in the question," which may be key to "identifying hidden associations and implicit relationships" (Chu-Carroll et al., 2012, p. 1).

For instance, in the above legal examples of conceptual legal information retrieval, various concepts would need to be expanded:

<**nondisclosure-agreement**> includes: 'nondisclosure agreement', 'agreement not to disclose', 'employment contract with a nondisclosure clause'

<**noncompete-agreement**> includes: 'noncompete agreement', 'noncompetition agreement', 'covenant not to compete'

<**varicella-vaccine**> includes: 'varicella vaccine', 'Chickenpox vaccine', 'VARIVAX'

It is apparent in these examples of relation extraction that relevant concepts like "vaccine," "to sign," or "nondisclosure agreement" can be expressed in multiple ways. Concept expansion identifies semantically related concepts in a corpus, in effect, deriving a dictionary or thesaurus rather than starting with one.

1.4.5. *Sources for Text Analytic Tools*

Tools like those in Watson have become available commercially as web-based services. As noted, IBM is attempting to capitalize on its investment in the Watson system by making a selected set of Watson's functionalities available for developers in a commercially convenient form. IBM offers a variety of services under the IBM Watson Developer Cloud (also referred to as Watson Services and BlueMix) for building cognitive apps (IBM Watson Developer Cloud Watson Services, 2015). These are commercial services subject to license and to license fees. Versions are also available for academic research, such as AlchemyLanguage, a set of text analysis/natural language processing (NLP) tools (IBM Watson Developer Cloud Watson Services, 2016).

Whether or not one wishes to avoid using IBM's proprietary tools, the Watson services are an instructive example for anyone interested in the future of legal practice. Even absent an ability to directly access the services, the framing of the tools on the website is instructive. It represents a creative effort by IBM to demonstrate how the new text analytic technologies can be packaged in an accessible form. IBM's efforts provide at least one example of the kinds of IE services that are needed, how to group them, and how to present them to noncomputer programmers (IBM Watson Developer Cloud Watson Services, 2015).

Presumably, IBM will not be the only source of such tools in the future. Open-source alternatives are currently available in a rougher form that requires developers to adapt them. As noted, IBM Watson is built on the *UIMA* platform, an open-source Apache framework that has been deployed in several large-scale government-sponsored and commercial text processing applications (Epstein et al., 2012). Academic researchers in the UIMA community are developing alternative open-source versions of tools like the above. For instance, Grabmair et al. (2015), discussed in Sections 6.8, 10.5, 11.3, and 11.4, demonstrate the utility of open-source tools for extracting argument-related information from legal texts (involving the corpus of federal vaccine compensation cases mentioned above) and using it to improve a full-text legal information system's ranking of retrieved documents.

Those who wish to create legal applications based on either the Watson Developer Cloud services or UIMA tools still have to solve some challenging problems. We illustrated a few of these problems above in contrasting Watson's and Debater's outputs with what legal problem-solving demands. It is the goal of this book to frame these problems so that students and other developers can tackle them with the techniques and tools that the AI & Law field offers.

1.5. A GUIDE TO THIS BOOK

It is intriguing to imagine how a QA text-analysis program could both answer legal questions *and* provide explanations and arguments that a legal practitioner could credit. Will there be a software service for:

> *Generation of explanations and arguments in law*: assists in structuring explanations of answers and supportive legal arguments?

That has not happened yet, however, and before it does, researchers will need to answer two questions: How can text analytic tools and techniques extract the semantic information necessary for AR and how can that information be applied to achieve cognitive computing?

Readers will find answers to those questions in the three parts of this book.

Part I introduces more CMLRs developed in the AI & Law field. It illustrates research programs that model various legal processes: reasoning with legal statutes and with legal cases, predicting outcomes of legal disputes, integrating reasoning with legal rules, cases, and underlying values, and making legal arguments. These CMLRs did not deal directly with legal texts, but text analytics could change that in the near future.

Part II examines recently developed techniques for extracting conceptual information automatically from legal texts. It explains selected tools for processing some aspects of the semantics or meanings of legal texts, including: representing legal concepts in ontologies and type systems, helping legal information retrieval systems

take meanings into account, applying ML to legal texts, and extracting semantic information automatically from statutes and legal decisions.

Part III explores how the new text processing tools can connect the CMLRs, and their techniques for representing legal knowledge, directly to legal texts and create a new generation of legal applications. It presents means for achieving more robust conceptual legal information retrieval that takes into account argument-related information extracted from legal texts. These techniques will enable some of the CMLRs of Part I to deal directly with legal digital document technologies and to reason directly from legal texts in order to assist humans in predicting and justifying legal outcomes.

Taken together, the three parts of this book are effectively a handbook on the science of integrating the AI & Law domain's top-down focus on representing and using semantic legal knowledge and the bottom-up, data-driven and often domain-agnostic evolution of computer technology and IT.

The recentness of the legal tech boom belies the fact that AI & Law researchers have already invested a great deal of thought in how to model legal reasoning. This book does not aim to provide a complete history of that research. Instead, it highlights selected trends in the development of CMLRs and CMLAs and explains their implications for the future given the opportunities for integrating text analytics.

Nor does this book cover all of the ways in which legal tech start-ups are harnessing data to predict legal outcomes. Instead, the focus is on how to employ and integrate semantic legal knowledge into predicting outcomes and explaining predictions. Over years of pursuing a methodology that is both empirical and scientific, AI & Law researchers have discovered what works in computationally modeling legal reasoning and what does not. By carefully attending to these lessons, constraints, and limitations, developers in the current legal tech boom interested in incorporating semantic legal knowledge may achieve AR and create a new kind of software service, a cognitive computing legal app (CCLA).

The remainder of this section summarizes the book's narrative in more detail and serves as a chapter outline.

1.5.1. *Part I: Computational Models of Legal Reasoning*

The examples in Part I of rule-based and case-based programs that can perform intelligent tasks such as legal reasoning and explanation, argumentation, and prediction, all share something in common: giving reasons.

Reasoning means "the drawing of inferences or conclusions through the use of reason." *Explanation* is "the act or process of explaining," that is, giving "the reason for or cause of" or showing "the logical development or relationships of." *Argument* involves "a reason given in proof or rebuttal" or "discourse intended to persuade." *Prediction* means "an act of predicting"; to predict means "to declare or indicate in advance; esp: foretell on the basis of observation, experience, or scientific reason"

(Merriam-Webster's Collegiate Dictionary, 2015). In law, a reason in support of an inference or conclusion usually involves asserting that a legal rule warrants the conclusion, citing an authoritative source for the rule, for example, a statute or an applicable case, and explaining or arguing that the rule applies. (Circularity can hardly be avoided in defining these fundamental inferential tasks!)

The models employ knowledge structures for representing information in the statutory or court-made rules or in the facts of the cases and schemes of inference and argument to process reasons. Heretofore, the knowledge representation structures had to be filled in manually, the source of the previously mentioned knowledge acquisition bottleneck.

Much work, discussed in Chapter 2, has addressed constructing formal logical models of statutory reasoning, a kind of model that probably is *not* yet ready to automatically connect directly to legal texts. The chapter contrasts logical models of reasoning with statutory rules and realistic statutory interpretation. It considers some computational approaches to assisting humans to find and interpret statutory rules that are alternatives to logical models and that may be able to connect with legal texts.

Models of case-based legal reasoning, discussed in Chapter 3, address analogical reasoning with legal cases or precedents, an important phenomenon in common law jurisdictions that is more likely to result in successful applications of text analytics. The chapter compares a number of case-based models in terms of: how the CMLR represents legal information in cases, the aspects of legal reasoning with cases and precedents the CMLR captures or misses, the extent to which the CMLR integrates rules, cases, and underlying values, and the compatibility of the CMLR's representational techniques with the new techniques for extracting information from texts.

Some computational models for predicting legal outcomes, described in Chapter 4, are also ripe for applying text analytics. The chapter surveys case-based and ML techniques for predicting outcomes of legal cases and assesses their compatibility with text analytics.

The culmination of all of this work in AI & Law has been the development of computational models of legal argument and legal argument schemes, described in Chapter 5, completing Part I. The chapter focuses on CMLAs that unify reasoning logically with legal rules and analogically with legal precedents. The models generate legal arguments, sometimes represented diagrammatically, for purposes of planning written arguments, instruction, or public discussion of legal issues. Some aspects of these models are also ready for applying text analytics.

1.5.2. *Part II: Legal Text Analytics*

Meanwhile, other fields of research and development, such as information retrieval, QA, IE, and argument mining, have been perfecting techniques for representing

legal concepts and relations. Programs can then process the concepts and relations semantically and computational models of legal reasoning can use them intelligently. As explained in Chapter 6, this includes the development of legal ontologies and, more recently in UIMA type systems of the sort employed in Watson and Debater, and in LUIMA, an extended type system for legal domains. This part addresses how to adapt these text analytic tools to achieve conceptual legal information retrieval.

Some of the new text analytic techniques are already being integrated with commercial legal information retrieval (CLIR) tools. Chapter 7 introduces current technology for legal IR, explains these initial applications, and offers some new ones. Chapter 8 addresses how to apply ML to textual data in the contexts of e-discovery (litigation-related discovery of evidence from electronic information including texts) and legal information retrieval.

The text analytic techniques are extracting functional information from statutes and regulations and argument-related information from legal cases. As explained in Chapters 9 and 10, the techniques include rule-based extraction guided by LUIMA types and ML adapted to corpora of legal decisions.

The statutory conceptual information of interest includes not only the topics and types of statutes (e.g., regulatory domain and whether a provision is a definition or prescription) but also functional information such as the agents that a statute directs to communicate with each other. Conceptual information in cases includes argument-related information such as whether a sentence states a legal rule for deciding an issue, whether it is an evidentiary statement of fact about the case, or whether it indicates an application of the rule or elements of the rule to the facts of a case.

1.5.3. *Part III: Connecting Computational Reasoning Models and Legal Texts*

By integrating the models and tools of Parts I and II, programs can use the conceptual information extracted directly from legal texts to perform legal reasoning, explanation, argumentation, and prediction. Basically, the goal is for text analytics automatically to fill in the computational models' knowledge representation structures. In this way, the Watson services and UIMA tools can reduce the knowledge acquisition bottleneck, accomplish conceptual legal information retrieval, and address the challenges mentioned above of legal QA including the need to explain its answers. Part III explains how to make these connections and achieve CCLAs.

Chapter 11 addresses how to integrate the QA, IE, and argument mining techniques with certain CMLRs to yield new tools for conceptual legal information retrieval, including AR. Fortunately, these tools do not depend on processing *all* of a repository's documents. In designing a proposed legal app, it is not necessary that a corpus be available wholesale for text processing to identify concepts, concept roles and relations, and other argument-related information. Instead, the new text processing techniques can be applied as a kind of filter between a full-text retrieval

system and human users. The filter is applied just to the documents retrieved as relevant through traditional full-text retrieval searches, promoting the documents that should be ranked higher in terms of the extent to which concepts, conceptual relationships, and other argument-related information match the user's need. Chapter 11 demonstrates this filtering approach; argument-related information extracted from legal texts improves a full-text legal information system's ranking of retrieved documents.

As explained in Chapter 12, these tools, in turn, can be integrated even more fully with some of Part I's computational models of legal reasoning and argument to create a new breed of legal apps in which computer and human user collaborate, each performing the intelligent tasks it performs best. In a complementary way, the computational models of reasoning, explanation, argument, and prediction will play significant roles in customizing commoditized legal services. They provide examples of the processes and tasks that may be adapted for the new apps and the concepts, roles, and relations that should be implemented. The new legal practice tools, based on information extracted with UIMA or other text analytic technology, can reason with legal texts, enabling practice systems to tailor their outputs to a human user's particular problem. In effect, they are the means by which a commoditized legal service, in Susskind's terms, can be customized.

Chapter 12 presents the idea that legal information queries and QA should be thought of as means for testing hypotheses about the law and how it applies. It introduces the possibility that a legal app could engage users in collaboratively posing, testing, and revising hypotheses about how an issue should be decided. Section 12.7 illustrates some practical use cases and the different kinds of legal hypotheses they involve. Readers interested in a high-level view of how legal apps could address these use cases might begin with the last chapter and then circle back to the beginning of this book.

The new apps will be subject to some limitations. While current text analytic techniques can extract much conceptual information, they cannot extract it all. Many conceptual inferences are simply too indirect and require too much background CSK to identify. Thus, it is an important empirical question how much can be accomplished with current text analytic techniques. Before concluding, Chapter 12 explores these remaining challenges.

1.6. IMPLICATIONS OF TEXT ANALYTICS FOR STUDENTS

Legal instructors, law schools, and authors have been urging legal educators to focus more on the developing technologies of legal practice. For example,

- Granat and Lauritsen (2014) identified 10 law school programs that focus students on the technology of law practice. These programs cover such topics as practice systems automating data gathering, decision-making, and document

drafting, developing legal expert systems for public interest legal services and legal clinics, redesigning legal processes, applying ML to legal data and legal informatics.

- Georgetown University Law Center sponsors the Iron Tech Lawyer Competition. Law students are building legal expert systems and entering them in competition (Iron Tech Lawyer, 2015). Additional information about some of these activities may be found in Staudt and Lauritsen (2013).
- Two far-sighted authors, Lippe and Katz (2014), have urged the legal field to reckon specifically with the impact of Watson technology on the future of legal practice.

This book is intended to help law students, computer science graduate students, legal practitioners, and technologists to take up that challenge and to design and implement legal applications of a kind that has not previously been technically possible. As argued, the combination of new text analytic tools and computational models of legal reasoning provides an opportunity for those who see potential in implementing processes of legal practice computationally.

Law students and practitioners may not have computer programming expertise, but they will not necessarily need it. What they will need is an ability to think about legal practice in terms of engineering a cognitive computing process.

This book assumes readers do *not* have familiarity with computer programming. The focus is not on computer code but, more generally, on systematic descriptions of legal and computational processes. For instance, in each of the examples of CMLRs of Part I, we examine: the legal process, the program models, and the assumptions made, the inputs to and outputs from the program and how they are represented, the computational processes (at a high level of description such as via architecture diagrams, flow charts, and algorithms) with which the program transforms inputs to outputs, concrete examples of the algorithmic steps transforming specific inputs to specific outputs, how the researchers evaluated the programs, the strengths and weaknesses of the approach, and its relevance given recent developments in legal text processing.

Actually writing computer code is the last step in designing successful computer applications. Key steps inevitably precede coding. They involve specifying requirements for the ultimate program and designing a high-level software architecture to realize it. Only then do programmers attempt to implement the software. Recent models of software development may focus on a modularized process involving multiple, nested instances of these steps, but even then, specifying requirements and a high-level design of a module always precedes the coding to implement it (Gordon, 2014).

The pedagogical goal, therefore, is not to teach the reader computer programming but how to propose and design apps that assist users in performing legal processes.

If the high-level designs are sound, there will always be computer programmers to implement them.

Law students are ideally suited to engage in identifying legal processes to model, specifying requirements, and designing high-level architectures. Law students are continually introduced to legal processes that are new to them and instructed how to perform the processes step-by-step. This occurs repeatedly in the first-year curriculum, in moot court competitions, in legal clinics, in legal internships and part-time jobs with law firms, corporate legal departments, and university tech transfer departments, and in pro bono activities. Today, law students are also likely to have used computer apps from a tender age. They are intimately familiar with the new modes of communication, with the current interface conventions, and with accessing web-based resources.

Along the way, the descriptions of computational models expose readers to a variety of assumptions and uncertainties inherent in legal reasoning that affect human legal reasoning. Indeed, law students study the sources of these uncertainties throughout the law school curriculum. These assumptions and uncertainties present some design constraints that AI & Law researchers have learned to avoid, finesse, or accommodate in their CMLRs, and that will necessarily affect efforts to apply text processing tools like those in Watson and Debater. Students will also learn how, and the extent to which, the performance of these technologies can be measured experimentally, and what the measures signify.

With respect to developing cognitive computing tools for legal practice, it is a time of exploration, even for IBM. The Watson Developer Cloud is indicative of a trend to make text analytic tools convenient to use even without computer programming expertise. A well-formed proposal from a law school student, legal academic, or practicing attorney might well engage IBM's material interest. This is not as far-fetched as it may appear. Indeed, it has happened already. Law students at the University of Toronto (that's in Canada, Watson, in case you are reading this) have already engaged in building legal apps in collaboration with IBM using Watson services (Gray, 2014). They created the Silicon Valley start-up called Ross, discussed in Chapter 12. As an extra incentive, IBM has announced a "$5 million competition . . . to develop and demonstrate how humans can collaborate with powerful cognitive technologies to tackle some of the world's grand challenges" (Desatnik, 2016).

Why couldn't a law student win with a CCLA for cross-jurisdictional issues in cybercrime and security? Tutoring students' imaginations about what is possible may be all that is necessary to enable them to design and propose such an app. This book aims for that.

Cognitive computing in law will be happening soon!

2

Modeling Statutory Reasoning

2.1. INTRODUCTION

The Law is a domain of rules, and many of those legal rules are embodied in statutes and regulations. Since rules can be expressed logically and computers can reason deductively, computationally modeling statutory reasoning *should* be easy. One simply inputs a fact situation to the computer program; the program identifies the relevant rules, determines whether or not the rules' conditions are satisfied, and explains the answer in terms of the rules that applied or did not apply.

Building a computational model of statutory reasoning, however, presents challenges. As explained below, statutes routinely are vague, syntactically ambiguous as well as semantically ambiguous, and subject to structural indeterminacy. If a computer program is to apply a statutory rule, which logical interpretation should it apply, how can it deal with the vagueness and open-texture of the statute's terms, or determine if there is an exception?

The chapter draws a contrast between deductively applying a statute and the complex process of statutory interpretation, which frequently involves conflicting reasonable arguments. Classical logical models may break down in dealing with legal indeterminacy, a common feature of legal reasoning: even when advocates agree on the facts in issue and the rules for deciding a matter, they can still make legally reasonable arguments for and against a proposition.

Reasoning with statutes, however, remains a pressing necessity. The chapter examines various AI & Law approaches that address or finesse these issues: a normalization process for systematically elaborating a statute's multiple logical versions, a logical implementation for applying a statute deductively, and more recent models of business process compliance and network-based statutory modeling, both potentially useful for cognitive computing.

Questions addressed in this chapter include: How can statutory rules be ambiguous, both semantically *and* syntactically? How do lawyers deal with these

ambiguities, and how can computer programs cope? What are normalized legal drafting, Prolog, and a Prolog program? What is depth-first search and how does it differ from breadth-first search? What is legal indeterminacy and why is it a problem for logical models of legal reasoning? How can logic programs assess business process compliance with regulations? What problems do isomorphic knowledge representations of statutes address? What are citation networks and statutory network diagrams, and how can they support cognitive computing?

2.2. COMPLEXITIES OF MODELING STATUTORY REASONING

Statutes and regulations are complex legal texts. An often intricate maze of provisions written in legal technical jargon define what is legal and not. With their networks of cross-references and exceptions, statutes and regulations are often too complicated for the untutored citizen to understand. Even legal experts may have difficulty simply identifying all and only the provisions that are relevant to analyzing a given question, problem, or topic.

The field of AI & Law has long studied how to design computer programs that can reason logically with legal rules from statutes and regulations. It has made strides, and demonstrated some successes, but it has also developed an appreciation of just how difficult the problem is. In the process, the field has identified a number of constraints that need to be addressed or finessed in attempting to design a computer program that can apply statutory rules. As noted, these constraints include vagueness and two kinds of ambiguity in statutory rules, the complexity of statutory interpretation, the need to support conflicting but reasonable arguments about what a legal rule means, and practical problems in maintaining logical representations of statutes alongside textual ones.

Of the two kinds of ambiguity that complicate computationally modeling statutory reasoning, semantic ambiguity, and its cousin, vagueness, are familiar. The regulatory concepts and terms the legislature selects may not be sufficiently well defined to determine if or how they apply. The second kind, syntactic ambiguity, may be less familiar: the *logical* terms legislatures use, such as "if," "and," "or," and "unless," introduce multiple interpretations of even simple statutes.

2.2.1. *Semantic Ambiguity and Vagueness*

Semantic ambiguity "is uncertainty between relatively few . . . distinct alternatives" concerning a term's meaning (Allen and Engholm, 1978, p. 383). "Vagueness is a semantic uncertainty about precisely where the boundary is with respect to what a term does and does not refer to" (Allen and Engholm, 1978, p. 382).

Both are due to the fact that legislatures may employ terms that are vague or otherwise not well-defined. Waterman confronted the problem that ill-defined legal terms present for constructing legal expert system rules (Section 1.3.1), and Gardner

attempted to address it with her algorithm for distinguishing hard and easy legal questions (Section 1.4.2).

Semantic ambiguity and vagueness are concessions to human, social, and political reality. The legislature cannot fashion language sufficiently detailed to anticipate all of the situations it may wish to regulate. Instead, it employs more general terminology in statutory rules and relies on the courts to interpret and apply the abstract terms and concepts in new fact situations. Intentionally rendering key provisions in a semantically ambiguous way can also facilitate legislative compromise. If the legislature attempted to use specific, detailed language, it might compound the difficulty of obtaining political consensus (Allen and Engholm, 1978, p. 384).

Semantic ambiguity and vagueness, however, are also a source of *legal indeterminacy*: opponents can agree on what legal rule applies and what the facts are and *still* generate reasonable legal arguments for opposing results (Berman and Hafner, 1988).

Even when the legislative intent is clear and the statute's language straightforward, legal adversaries routinely make reasonable but conflicting arguments about what the rule's terms mean. In their example, the case of *Johnson v. Southern Pacific Co.*,117 Fed. 462 (8th Cir. 1902) *rev'd* 196 U.S. 1 (1904), a federal statute made it "illegal for railroads to use in interstate traffic 'any car not equipped with couplers coupling automatically by impact.' " According to the statute's preamble, the act's purpose was "to promote the safety of employees . . . by compelling carriers . . . to equip their cars with automatic couplers . . . and their locomotives with drive wheel brakes" (Berman and Hafner, 1988, p. 196).

There was no disagreement about the facts. "The plaintiff, a railroad brakeman, was injured when he attempted to couple a locomotive to a dining car, in order to move the dining car off the track." Causation was not an issue: the plaintiff's injury was caused by the fact that although the locomotive was equipped with such a coupler, it was not one that could couple automatically with this particular dining car.

Nevertheless, courts disagreed about whether the statutory rule's conditions were satisfied, and, in particular "on the meaning of all three of the predicates in the condition part of this rule: the meaning of 'car,' the meaning of 'used-in-interstate-commerce,' and the meaning of 'equipped' " (Berman and Hafner, 1988, p. 198). Are locomotives included in the "cars" required to have automatic couplers or not? Does "interstate commerce" include the time when a car is awaiting its next load or not? Were the dining car and locomotive "equipped" with automatic couplers or not? The trial and appellate courts disagreed on the answers to these questions and they certainly did not treat those answers as determined by the terms' literal meanings.

2.2.2. *Syntactic Ambiguity*

The other kind of ambiguity, syntactic ambiguity, arises from a different reality: statutory language does not always follow a single, coherent logical structure. This results

in part from the properties of natural language text. Unlike mathematical and logical formalisms and computer code, text does not allow one explicitly to specify the scopes of the logical connectors, such as "if," "and," "or," and "unless." The syntax of a statute can also be unclear due to the language used in implementing exceptions and cross-references. Exceptions to a provision may be expressed explicitly but even implicitly and may appear not only within a provision but also in other provisions or even in other statutes (Allen and Engholm, 1978).

Layman Allen demonstrated that syntactic ambiguity leads to multiple possible logical interpretations of even relatively simple statutory provisions, with potentially profound consequences for those subject to regulation. He provided an example from a Louisiana statute defining a crime:

> No person shall engage in or institute a local telephone call, conversation or conference of an anonymous nature and therein use obscene, profane, vulgar, lewd, lascivious or indecent language, suggestions or proposals of an obscene nature and threats of any kind whatsoever. (Allen and Engholm, 1978)

Presumably, the legislature intentionally selected vague terms like "obscene" and "indecent" with full knowledge of their open texture.

It is much less likely that they intentionally promulgated a criminal standard with an inherent syntactic ambiguity: To be in violation of the statute, is it sufficient that a call include either obscene language OR threats, or, as the defendant in *State v. Hill*, 245 La 119 (1963) argued successfully at the District court, must it include obscene language AND threats? The Louisiana Supreme Court disagreed; it interpreted "and" as meaning "or," seemingly violating a common law maxim that criminal statutes should be strictly construed. Surely, it would have been better legislative policy to issue a syntactically unambiguous standard (Allen and Engholm, 1978).

Allen described a systematic *normalization* process for identifying such ambiguities. Given a statute, one:

1. Identifies the statute's "atomic" substantive propositions and replaces them with labels (S1, S2, . . .).
2. Uses propositional logic to clarify the syntax of the statute.
3. Restores the text of the substantive propositions.

In *propositional logic*, symbols stand for whole propositions. Using logical operators and connectives, propositions can be assembled into complex statements whose truth values depend solely on whether the component propositions are true or false. Unlike, predicate logic, defined below, propositional logic does not consider the components or structure of individual propositions (see Clement, 2016).

Applying the normalization process to the Louisiana statute yields a number of versions including the two shown in Figure 2.1. Each version is an expression in propositional logic, which renders its logical structure more clearly.

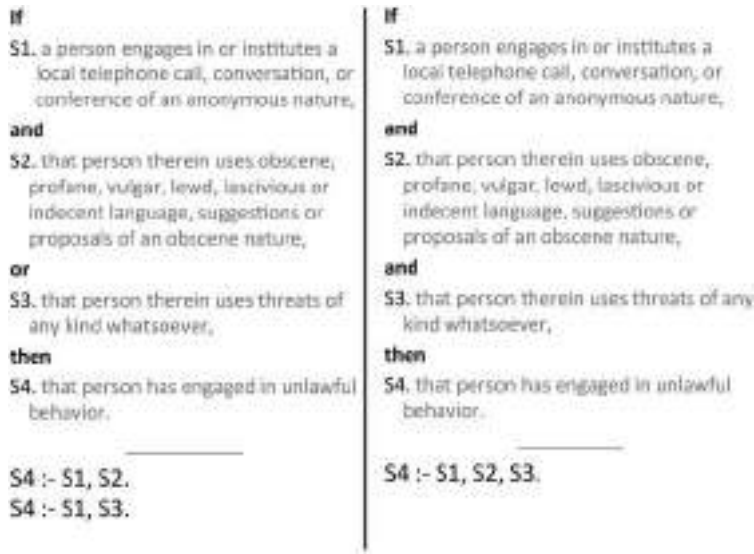


FIGURE 2.1. Normalized versions of two alternative interpretations of the Louisiana statute and corresponding Prolog rules (bottom) (Allen and Engholm, 1978)

2.3. APPLYING STATUTORY LEGAL RULES DEDUCTIVELY

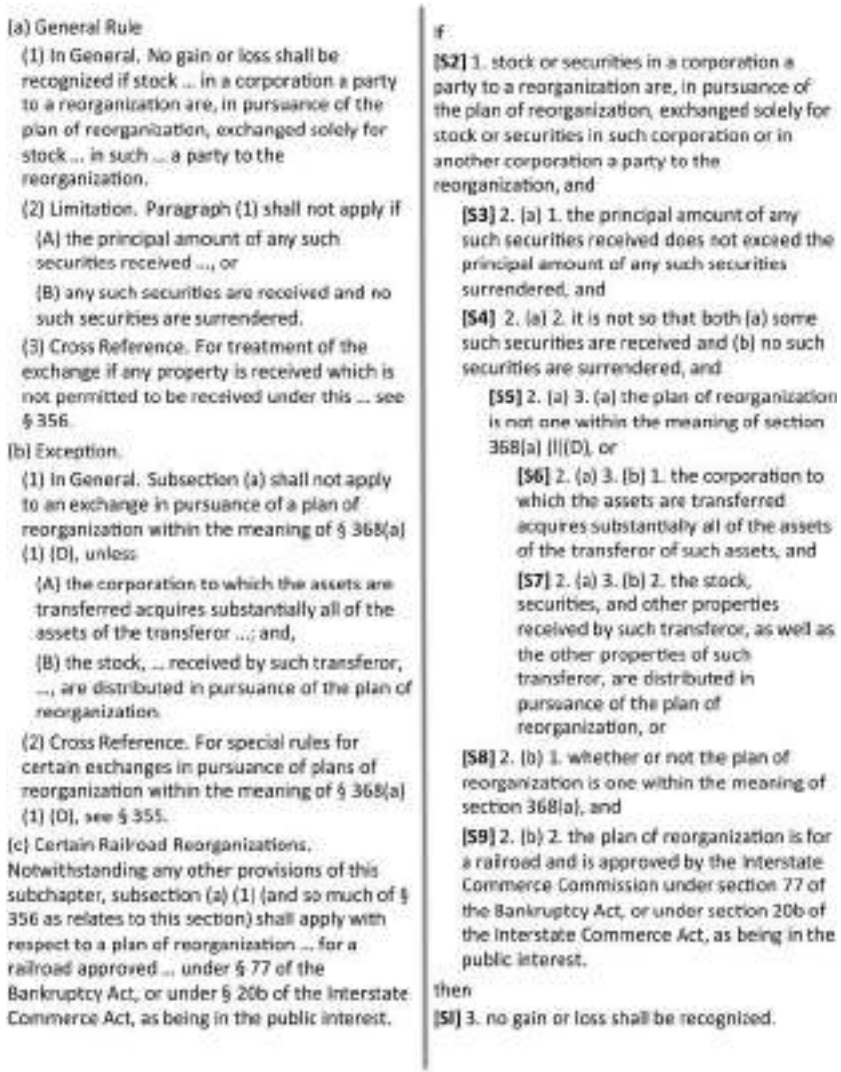
A normalized statute in propositional logical form offers several advantages.

First, using propositional logic to clarify the syntax of the statute can make a complex statute much easier to understand. For instance, Allen contrasts a complex provision of the Internal Revenue Code (IRC section 354), which deals with the tax treatment of exchanges of securities in certain corporate reorganizations, with a normalized version as shown in Figure 2.2 on the right. The normalized version identifies the "atomic" substantive propositions and employs indentation to convey the simplified logical structure.

Allen also provided a kind of flow chart through the logic of the "propositionalized" version of the statute where every node in the graph is one of the requirements of the statute (see Figure 2.3). The labeled nodes, S1 through S9, refer to the labeled propositions in the normalized version (right) in Figure 2.2.

The flow chart can be much easier to understand than the textual or even normalized versions of the statute. It demonstrates three alternative paths through the statute from a starting point of an exchange of securities in the corporate reorganization (S2) to the desired conclusion of "no gain or loss" recognition (S1). These paths remain more or less implicit in the textual and normalized versions (although Allen combined the flow chart and the normalized version to make the paths explicit).

In the context of corporate compliance, for instance, such flow charts can help to clarify obligations of the various corporate constituents. *Corporate compliance* involves detecting and preventing violations of law by the agents, employees, officers



(a) General Rule

(1) In General. No gain or loss shall be recognized if stock ... in a corporation a party to a reorganization are, in pursuance of the plan of reorganization, exchanged solely for stock ... in such ... a party to the reorganization.

(2) Limitation. Paragraph (1) shall not apply if

(A) the principal amount of any such securities received ..., or

(B) any such securities are received and no such securities are surrendered.

(3) Cross Reference. For treatment of the exchange if any property is received which is not permitted to be received under this ... see § 356.

(b) Exception.

(1) In General. Subsection (a) shall not apply to an exchange in pursuance of a plan of reorganization within the meaning of § 368(a)(1)(D), unless

(A) the corporation to which the assets are transferred acquires substantially all of the assets of the transferor ...; and,

(B) the stock, ... received by such transferor, ..., are distributed in pursuance of the plan of reorganization.

(2) Cross Reference. For special rules for certain exchanges in pursuance of plans of reorganization within the meaning of § 368(a)(1)(D), see § 355.

(c) Certain Railroad Reorganizations. Notwithstanding any other provisions of this subchapter, subsection (a)(1) (and so much of § 356 as relates to this section) shall apply with respect to a plan of reorganization ... for a railroad approved ... under § 77 of the Bankruptcy Act, or under § 20b of the Interstate Commerce Act, as being in the public interest.

if

[S2] 1. stock or securities in a corporation a party to a reorganization are, in pursuance of the plan of reorganization, exchanged solely for stock or securities in such corporation or in another corporation a party to the reorganization, and

[S3] 2. (a) 1. the principal amount of any such securities received does not exceed the principal amount of any such securities surrendered, and

[S4] 2. (a) 2. it is not so that both (a) some such securities are received and (b) no such securities are surrendered, and

[S5] 2. (a) 3. (a) the plan of reorganization is not one within the meaning of section 368(a)(1)(D), or

[S6] 2. (a) 3. (b) 1. the corporation to which the assets are transferred acquires substantially all of the assets of the transferor of such assets, and

[S7] 2. (a) 3. (b) 2. the stock, securities, and other properties received by such transferor, as well as the other properties of such transferor, are distributed in pursuance of the plan of reorganization, or

[S8] 2. (b) 1. whether or not the plan of reorganization is one within the meaning of section 368(a), and

[S9] 2. (b) 2. the plan of reorganization is for a railroad and is approved by the Interstate Commerce Commission under section 77 of the Bankruptcy Act, or under section 20b of the Interstate Commerce Act, as being in the public interest.

then

[S1] 3. no gain or loss shall be recognized.

FIGURE 2.2. IRC section 354 and a normalized version (right) (see Allen and Engholm, 1978)

and directors of a corporation, firm, or other business. Presenting an employee's regulatory obligations in the form of a flow chart could help the employee understand what is legal and what is not legal.

2.3.1. *Running a Normalized Version on a Computer*

Second, a statutory provision in propositional logic can be run on a computer! At the bottom of Figure 2.1 are the two normalized versions of the Louisiana statute

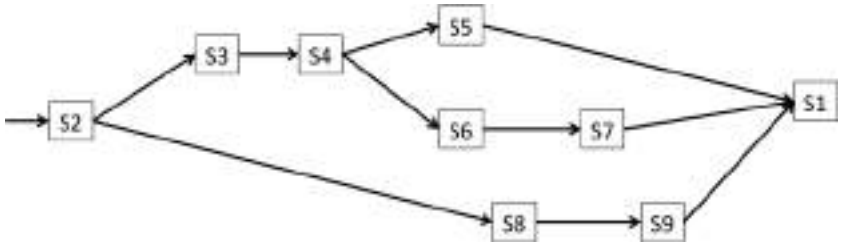


FIGURE 2.3. Flow chart for propositionalized IRC section 354 (see Allen and Engholm, 1978)

expressed in Prolog, a programming language based on so-called "Horn clause logic" associated with artificial intelligence and computational linguistics.

The normalized versions of this statute are examples of simple propositional logic. The version on the right, S4 :- S1, S2, S3, means "*If* $S1 \wedge S2 \wedge S3 \Rightarrow S4$," where $\wedge$ means "and." The version on the left uses two formulas, S4 :- S1, S2 and S4 :- S1, S3 to implement the disjunction (i.e., the "or") in the version.

Prolog interprets the Horn clauses, treating them as a program. For example, it treats the horn clause on the right as saying, in effect, "To show S4, show S1, show S2, and show S3."

By asking the user whether each substantive proposition S1, S2, and S3 is true or false, a computer can prove the truth or falsity of S4. In this way, the logic of the statute is automated. The logic of these normalized versions of the statute is simple enough to handle manually with truth tables. Nevertheless, a computer can also process the propositional logic of more complex statutes like that in Figure 2.2.

2.3.2. *Predicate Logic*

One would also like to express the content of a statutory provision's substantive propositions, not just its overall logical syntax. This can be done with *classical logic* (also known as predicate logic, predicate calculus, or first-order logic).

> Classical logic is the formal logic known to introductory logic students as "predicate logic" in which, among other things, (i) all sentences of the formal language have exactly one of two possible truth values (TRUE, FALSE), (ii) the rules of inference allow one to deduce any sentence from an inconsistent set of assumptions, (iii) all predicates are totally defined on the range of the variables, and (iv) the formal semantics is the one invented by Tarski that provided the first precise definition of truth for a formal language in its metalanguage. (Dowden, 2016)

Classical logic employs symbols for predicates, subjects, and quantifiers. In propositional logic, the proposition 'All men are mortal' is represented with just one symbol and has no internal structure. In contrast, in classical logic one can define a predicate $M(x)$ to express that x is mortal or employ the universal quantifier ("For all"): All x. $M(x)$ to express that all x are mortal.

Horn clause logic, the basis of Prolog, implements most (but not all) of predicate logic and allows one to express both the content of substantive propositions of a normalized version of a statutory provision as well as its logical syntax. For instance, HLA Hart's famous sample statutory provision, "Vehicles are not permitted in the park,"[1] could be expressed in Prolog (that is, Horn clause predicate logic) as:

violation(X, S) :- vehicle(X), park(S), in(X, S).

That is, "*If X is vehicle* $\wedge$ *S is park* $\wedge$ *X in S* $\Rightarrow$ *X in S is violation*." In Prolog, commas between predicates indicate "and"; the universal quantifier is implied.

If one inputs to the Prolog program information such as:

vehicle (X) :- motorcycle (X)
vehicle (X) :- automobile (X)

the program can prove that one may not take into the park a motorcycle, an automobile, or indeed, anything else we input as qualifying as a vehicle, by chaining the conclusion of one rule to the premise of the other.

2.3.3. *Syntactic Ambiguity as Design Constraint*

Before turning to an example of a large-scale legal logic program, let's summarize the implications of syntactic ambiguity.

Syntactic ambiguity makes the task of translating statutory texts into computationally formalized logical rules problematic (Allen and Saxon, 1987). In computationally expressed rules, syntactic ambiguity can be eliminated. The problem is that the version of a statutory rule selected for formalization into a logic programming language like Prolog is not necessarily the one that the legislature intended.

As a result of syntactic ambiguity, a knowledge engineer cannot be certain what the legislature intended. The number of syntactically possible interpretations that result from applying the normalization process to even fairly simple statutory provisions can be disconcertingly large. Section 3505, a proposed limitation of the fourth amendment exclusionary rule, stated:

> Except as specifically provided by statute, evidence which is obtained as a result of a search or seizure and which is otherwise admissible shall not be excluded in a proceeding in a court of the United States if the search or seizure was undertaken in a reasonable, good faith belief that it was in conformity with the fourth amendment to the Constitution of the United States. A showing that evidence was obtained pursuant to and within the scope of a warrant constitutes prima facie evidence of such a reasonable good faith belief, unless the warrant was obtained through intentional and material misrepresentation.

[1] Actually, Hart's example was "A legal rule forbids you to take a vehicle into the public park" (Hart, 1958, p. 607).

Although just two sentences long, when normalized, section 3505 yielded 48 interpretations of varying strength, as measured by their restrictiveness or inclusiveness, based simply on the syntactic ambiguities in the provision (Allen and Saxon, 1987).

As a practical matter, only a few versions may seem clearly reasonable and other versions may be clearly unreasonable. The question is who selects? If a knowledge engineer decides which normalized version to implement, it is not an authoritative choice. Legal academics or other experts in a field may express opinions about which version the legislature intended or should have intended. The only body able to make the selection authoritatively, however, is the legislature.

Unfortunately, the legislators were probably unaware of the ambiguity (Allen and Saxon, 1987). While semantic ambiguity can facilitate legislative compromise and is usually intended, syntactic ambiguity serves no legitimate function in the political process and does not facilitate political comprise. Indeed, in laying out a systematic procedure for generating normalized versions of statutory provisions, one of Layman Allen's goals was to sensitize legislators and law students to the phenomenon (Allen and Engholm, 1978; Allen and Saxon, 1987).

A law professor in Tennessee, Grayfred Gray, achieved some success in convincing a state legislative drafting committee to adopt normalization as a means of eliminating unintended syntactic ambiguity in provisions of the State's mental health law provisions concerning commitment and discharge of mental patients. The Committee was concerned "that the law be clear to the people who would have to work with it, most of whom were not lawyers" (Gray, 1985, pp. 479–80). The legislature did not seem to have a problem with normalization. The publisher of the state's statutory code, on the other hand, worried that normalization's liberal use of indentation to convey statutes' simplified logical structure would take up too much space, increasing the cost of the printed publications. Ultimately, only a few statutes were published in normalized form.

Today, on the World Wide Web, space is not an issue. Normalized versions of statutes and accompanying flowcharts could be published economically via the web, making it much easier for nonlawyers to read and understand the legal requirements. In a web-based publication, help links and dropdown menus could assist the uninitiated in using and interpreting the normalized provisions.

In the meantime, of course, the multiplicity of logical interpretations of statutes has not brought the legal profession to its knees. On the contrary, it generates employment. Attorneys and legal experts representing taxpayers or insurance companies are retained to generate and exploit alternative syntactic interpretations of complex provisions. In an adversarial context, identifying alternative logical interpretations of a statute or of a complex insurance policy provision opens the opportunity for arguing for an interpretation favorable to one's client, as in the criminal trial in *State v. Hill* above.

In a different context, such as corporate compliance, risk-averse attorneys might recommend adopting a more expansive logical interpretation of a statute to

formalize in a business rules system. Selecting a safely expansive interpretation would help to reduce subsequent infractions of the legal rules.

From the viewpoint of cognitive computing, a system that could detect latent syntactic ambiguities would be a nice tool for legislative drafters. Given the input of a statutory provision in natural language, could a system automatically generate a comprehensive listing of normalized versions or partially order them in terms of their strength? In other words, can the normalization process of (Allen and Engholm, 1978) be automated? I am not aware of any research attempts to do so, but it seems worth exploring.

2.3.4. *The BNA Program*

Marek Sergot and his colleagues successfully implemented a large portion of the British Nationality Act (BNA) as a logic program written in Prolog (Sergot et al., 1986). The system ran approximately 150 rules dealing with the acquisition of British citizenship. The rules were implemented as Horn clauses in Prolog; Figure 2.4 shows a translation of three of the rules into pseudo English.

Inputs to the BNA program were descriptions of problems involving a question of citizenship. The program output an answer and an explanation. Asking a question was equivalent to stating a proposition and asking Prolog to prove it. For instance, one such proposition is:

A: Peter is a British citizen on date (16 Jan 1984) by sect. *z*.

Here, *z* is a variable standing for the number of some section of the statute that would warrant the conclusion.

1-(1) A person born in the United Kingdom after commencement shall be a British Citizen if at the time of birth his father or mother is:
(a) a British Citizen, or
(b) settled in the United Kingdom.

This is represented in the computer as:

Rule1: X acquires British citizenship on date Y
under sec. 1.1
IF X was born in the U.K.
AND X was born on date Y
AND Y is after or on commencement of the act
AND X has a parent who qualified under 1.1 on date

Rule2: X has a parent who qualifies under 1.1 on date Y
IF X has a parent Z
AND Z was a British citizen on date Y

Rule3: X has a parent who qualifies under 1.1 on date Y
IF X has a parent Z
AND Z was settled in the U.K. on date Y.

FIGURE 2.4. BNA provisions as represented in rules (Sergot et al., 1986)

Prolog is both a programming language and theorem prover. Given A, Prolog attempts to construct a proof that A by reasoning backward from conclusion A to identify conditions that need to be satisfied. (Backward chaining was introduced in Section 1.3.1.) In the process, it finds all of the rules that conclude with a proposition of the form A. There may be a number of different rules with which to establish conclusion A. Prolog will try them all in the order in which the rules are written.

Let's say there is a list of n rules whose conclusions are A. If Prolog is considering a rule r_i from that list, let's call the next rule in the list r_{i+1}. Finally, when Prolog is considering rule r_j, if it finds a new rule whose conclusion is the antecedent of rule r_j, let's refer to the newly found rule as r_{j_1} and call it a "descendant" of rule r_j on a path to proving A.

Prolog will try the n rules on the list in a depth-first search order. In *depth-first search*, the program follows a path from one of the r_i on the list to its end, either success or failure, before starting a new path from r_{i+1}. In other words, if Prolog is considering rule r_i, the program always chooses to try to find a descendant r_{i_1} of rule r_i, before moving on to the next rule on the list, r_{i+1}. If and only if the path from r_i runs out without proving A will it move on to r_{i+1}.

In contrast, in a *breadth-first search*, the program tries to open a path for each of the n rules before searching for any descendants of a descendant. That is, even if a search-based program has found a descendant for rule r_i (i.e., r_{i_1}) it chooses to try to find a descendant for each of the next rules on the list $r_{i+1}, r_{i+2}, \ldots, r_n$, before trying to find a descendant of r_{i_1}.

Each rule will have conditions $B1 \ldots Bn$ that need to be satisfied in order to conclude C. Remember that Prolog treats such a rule as a program: To show C, show B1, show B2, . . ., and show Bn. For each of these Bn subproblems, Prolog solves it in one of four ways:

1. There may be another rule whose conclusion is of the form of Bn.
2. There may be a fact satisfying Bn.
3. The system can ask the user if Bn is true.
4. The system can ask an expert if Bn is true.

Figure 2.5 shows excerpts of the BNA program's output for the question regarding proposition A. The boldface indicates inputs from the user. Notice that the program could answer "why" and "how" questions and explain its answers. Basically, having constructed a proof of A or of an intermediate conclusion, it could fashion an explanation by reiterating the inference steps in its proof.

2.3.5. *Some Problems of Translating Statutes into Programs*

In designing and implementing the computational model of the BNA, the designers encountered four problems or constraints: Sometimes a previously formulated rule

IS Peter a British citizen on date (16 Jan 1984) by sect. 2?
Which X: Peter was born on date X ? ***(3 May 1983)***
...
Is it true that Peter was born in the U.K. ? **yes**
Which X : X is father of Peter ? **why**

if X is father of Peter then X is a parent of Peter
If X is a parent of Peter and X is a British citizen on date (3 May 1983)
then Peter has a parent who qualifies under 1.1 on date (3 May 1983)
Peter was born in the U.K.
Peter was born on date (3 May 1983)
(3 May 1983) is after or on commencement, so
if Peter has a parent who qualifies under 1.1 on date (3 May 1983)
then Peter acquires British citizenship on date (3 May 1983) by sect. 1.1
Peter is alive on (16 Jan 1984), so
if Peter acquires British citizenship on date (3 May 1983) by sect. 1.1
and (16 Jan 1984) is after or on (3 May 1983)
and not[Peter ceases to be a British citizen on date Y
and Y is between (3 May 1983) and (16 Jan 1984)]
then Peter is a British citizen on date (16 Jan 1984) by sect 1.1

Which X : X is father of Peter ? ***William***
Which X : William was born on date X ? ***(1 March 1952)***
...
Is it true that William was born in the U.K. ? **yes**
...
Is it true that William was a citizen of the U.K. and Colonies on date (31 Dec 1982) ? **yes**
IS it true that on date (31 Dec 1982) William had the right of abode in the U.K. under the Immigration Act 1971 as then in force ? **yes**
...

Yes, I can confirm that
Peter is a British citizen on date (16 Jan 1984) by sect. 1.1 **how**
I deduced Peter is a British citizen on date (16 Jan 1984) by sect. 1.1
from the rule

FIGURE 2.5. BNA program output (excerpts) (Sergot et al., 1986, p. 376f)

or predicate needed to be *reformulated*. In addition, the statute's use of *negation*, *default reasoning*, *counterfactual conditions*, and *open-textured terms* needed to be addressed.

Reformulation

Formalizing an extensive statute is a process of trial and error. One frequently encounters a new context in which a previous formulation of a rule concept is inadequate and has to be reformulated to accommodate additional constraints imposed by subsequent rules in the act. For instance, the researchers discovered that it is "insufficient to conclude only that an individual is a British citizen; it is also necessary to determine the section under which citizenship is acquired." Also, a newly

encountered section made evident the need for "a more explicit treatment of time" to compute constraints that enabled one not a citizen subsequently to be registered as one under certain circumstances (Sergot et al., 1986, p. 374). The researchers had to change some existing rules, conditions, or parameters or add new ones to address the new constraints.

Negation

To implement some rules in the BNA and other statutes, it would be desirable to employ rules that state a negative conclusion (i.e., not *A*), such as "*x* was not a British citizen at the time of *y*'s birth" or "*x* was not settled in the U.K. at the time of *y*'s birth."

Such negative conclusions require an ability to deal with ordinary or classical negation, something that Prolog does not support. Prolog can employ only "negation by failure." The theorem prover uses a rule, "infer not *P* if fail to show *P*." In other words, if there is a finite list of ways to show *A*, the theorem prover will check them all. If all of them fail, then it concludes not *A*. Negation by failure is adequate when one can make the "closed world assumption" (that is, that anything which is not known is assumed to be false).

Often, however, the closed world assumption is not reasonable. "It is notoriously difficult in law to determine all the legal provisions that might be relevant to deciding a particular case" (Sergot et al., 1986, p. 379). The researchers demonstrated some formulations in the BNA where using negation by failure would be prohibitively complex or lead the program to draw conclusions opposite from what the legislature intended. For instance, consider the difficulty of listing all of the ways that *x* can be shown to be a British citizen at the time of *y*'s birth.

A theorem prover that can handle classical negation could deal with this problem automatically, but Prolog's theorem prover would require an extended logic, which introduces other difficulties. As a result, the researchers simply resorted to having the BNA program ask the user to confirm certain negative information such as that "*x* was not a British citizen at the time of *y*'s birth" (Sergot et al., 1986, p. 381).

Default Reasoning

The authors point out that the BNA employs reasoning by default. "Conclusions made by default in the absence of information to the contrary may have to be withdrawn if new information is made available later" (Sergot et al., 1986, p. 381).

One example is section l-(2) of the BNA, the provision that deals with abandoned infants. What would happen, the authors ask, if the abandoned infant's parents, to whom citizenship had been conferred by default, suddenly turned up but were *not* British citizens? (Sergot et al., 1986, p. 381). The BNA does not seem to have a provision for that eventuality, but even if it did, there would be a problem.

Default reasoning is *non-monotonic*: propositions once proven may need to be withdrawn in light of new facts. Predicate logic (i.e., classical first-order logic as

implemented in Prolog) is monotonic; it does not support withdrawing propositions that have been proven.[2]

As discussed below, a more expressive logic is required.

Counterfactual Conditionals

Statutes also commonly make use of counterfactual conditionals such as "would have [become a British Citizen] but for his having died or ceased to be a citizen . . . [by] renunciation." The legislature may employ such a formulation as a shortcut means of reference. "The drafters avoid listing a complicated set of conditions explicitly by [referring] to some other part of the legislation" from which the conditions may be inferred (Sergot et al., 1986, p. 382).

The researchers created special rules to deal with such counterfactual conditionals. They wrote "additional alternative rules; one set describing, for example, the conditions for acquisition of citizenship at commencement for individuals who were alive on that date, and another set for individuals who had died before that date, but otherwise met all the other requisite conditions before death" (Sergot et al., 1986, p. 382).

The researchers carefully analyzed the statute to hypothesize which requirements might reasonably apply in the counterfactual condition. This increased the number of rules that needed to be formalized. Presumably, the legislative drafters employed the counterfactual condition to avoid the tedious task of spelling out these conditions. On the other hand, it is always possible that the drafters meant to leave the issue open-ended.

In any event, it is another example where knowledge engineers are required to make difficult interpretive decisions without legislative authority.

Open-Textured Terms

Finally, the legislature employed open-textured predicates in the statute that they did not define. The act contains such vague phrases as "being a good character," "having reasonable excuse," and "having sufficient knowledge of English" (Sergot et al., 1986, p. 371).

The researchers adopted a straightforward approach to dealing with vague terms. The system simply asks the user whether the term is true or not in the current inquiry. Alternatively, they might have programmed it to assume that a particular vague concept always applied (or always did not apply) and to qualify its answer based on this assumption, for instance: "Peter is a citizen, if he is of good character" (Sergot et al., 1986, p. 371). The researchers note that one might also apply rules of thumb, derived from analysis of past cases where courts applied the terms, in order to reduce the terms' vagueness. Such heuristic rules, however, would not be guaranteed to cover all cases, nor would they be authoritative.

[2] In fact, Prolog is non-monotonic, but the implementation used for the BNA program assumed that a user had perfect information and could always answer the questions it posed (Gordon, 1987, p. 58).

The problems of resolving syntactic ambiguity, reformulation, negation, counterfactual conditions, and semantic ambiguity are problems of interpreting natural language text. Potentially, they affect any attempts to translate legislation into runnable computer code regardless of whether humans are performing the translation manually, as in the BNA program research, or programs are extracting the rules automatically from statutory texts as discussed in Chapter 9.

2.4. THE COMPLEXITY OF STATUTORY INTERPRETATION AND THE NEED FOR ARGUMENTS

The BNA project focused on "the limited objective of implementing rules and regulations with the purpose of applying them mechanically to individual cases" (Sergot et al., 1986, p. 372). The BNA program was never intended to simulate the output of a court's reasoning about a statute, but it is interesting to compare the way it generates an answer through logical deduction as illustrated in Figure 2.5 with what a court might do.

In a landmark piece, the legal philosopher Lon Fuller demonstrated the limitations of a mechanical approach to applying legal rules. Does "a truck used in World War II" to be "mount[ed] on a pedestal in the park" and "in perfect working order" fall afoul of the no-vehicles-in-the-park regulation? (Fuller, 1958, p. 663). Or suppose that a municipal regulation states, "It shall be a misdemeanor, punishable by a fine of five dollars, to sleep in any railway station." A policeman encounters two people in the station:

> The first is a passenger who was waiting at 3 A.M. for a delayed train. When he was arrested he was sitting upright in an orderly fashion, but was heard by the arresting officer to be gently snoring. The second is a man who had brought a blanket and pillow to the station and had obviously settled himself down for the night. He was arrested, however, before he had a chance to go to sleep. (Fuller, 1958, p. 664)

According to a mechanical application of the rule, the first person violates the rule but the second does not; the former is asleep in the railway station but the latter is not. Given the likely purpose of the municipal regulation, however, this seems to be exactly the wrong result. As Fuller asks, "[I]s it really ever possible to interpret a word in a statute without knowing the aim of the statute?" (Fuller, 1958, p. 664).

The process of establishing the meaning of a statutory provision and applying it in a concrete fact situation is commonly referred to as statutory interpretation. A law court engages in *statutory interpretation* when it applies "statutes to particular cases with a view to giving authoritative and binding decisions upon the matters in dispute or under trial," "forms a view as to the proper meaning of the statutes which seem to them applicable in the case," and articulates a "view as to the way in which the statute should be understood" (MacCormick and Summers, 1991, p. 11f).

The process of statutory interpretation involves logical deduction but is quite a bit more complex. MacCormick and Summers identify a hierarchy of types of statutory interpretive *arguments*, including:

- *Linguistic*: arguments from the statute's ordinary meaning or technical meaning (i.e., legal or domain-specific technical meaning).
- *Systemic*: arguments from contextual harmonization, from precedent, and by analogy, logical-conceptual arguments, and arguments from general principles of law and from history.
- *Teleological/Evaluative*: arguments from purpose and from substantive reasons.
- *Transcategorical*: including arguments from intention (MacCormick and Summers, 1991, pp. 512–15).

An argument from the purpose of the municipal regulation banning sleeping in railway stations would be an example of a teological/evaluative argument. The list of acceptable techniques and their labels are relative to a legal system or tradition and may be subject to debate.

2.4.1. *A Stepwise Process of Statutory Interpretation*

The authors organize these argument types into a simplified, nearly algorithmic model for statutory interpretation (MacCormick and Summers, 1991, p. 531). According to the process, in interpreting a statutory provision, one considers three levels of argument in the following order: (1) linguistic arguments, (2) systemic arguments, and (3) teleological-evaluative arguments.

More specifically, the process specifies steps for making decisions based on the arguments:

Level 1: Accept as *prima facie* justified a clear interpretation at level 1 unless there is some reason to proceed to level 2;

Level 2: Where level 2 has been invoked for sufficient reason, accept as *prima facie* justified a clear interpretation at level 2 unless there is reason to move to level 3.

Level 3: If at level 3, accept as justified only the interpretation best supported by the whole range of applicable arguments (MacCormick and Summers, 1991, p. 531).

Generally, in the above series of steps, the authors recommend that arguments from intention and other transcategorical arguments (if any) be taken as grounds which may be relevant for departing from the above *prima facie* ordering.

Given this complex description of statutory interpretation, one can appreciate Ann Gardner's observation that law is a "rule-guided rather than a rule-governed activity: 'The experts can do more with the rules than just follow them . . . (they) can argue about the rules themselves' " (Gardner, 1985 quoted in Berman and Hafner, 1988, p. 208).

In order to apply the jurisprudential model of statutory interpretation in MacCormick and Summers (1991) to a concrete scenario, one might have to integrate reasoning with rules, cases, and the underlying social values and legislative purposes. Crucially, a reasoner would need to make or consider arguments for and against an interpretation. Every step of the interpretive process involves making and evaluating arguments of various types. A reasoner would need to draw analogies between a current case and past cases where courts applied the legal rule or statute and reason with the values and purposes underlying the legal rules articulated in the statutes and precedents. Even if one would apply the statutory rule deductively, he/she would need to consider whether the proposed result is consistent with the purposes and policies underlying the statute.

Although the BNA program and other programs described in Part I of this book implement computational models of legal reasoning, none of them implements a process of statutory interpretation as comprehensive as that described in MacCormick and Summers (1991).

Instead, the AI & Law field has invented components that could implement parts of the process. For instance, the BNA program constructed a proof from the plain meaning of the statute as represented by Prolog rules. Chapter 3 describes computational models of case-based legal reasoning and considers how to take underlying policies and values into account. Chapter 5 describes computational models of legal argument that provide a framework into which one could imagine implementing a computational process of statutory interpretation using the MacCormick/Summers model. See, for example, a preliminary formal framework to capture such interpretive arguments in Sartor et al. (2014).

2.4.2. *Other Sources of Legal Indeterminacy*

If the goal is to model arguments for purposes of statutory interpretation, however, there is a theoretical reason why classical logical deductive methods like those in the BNA program will not suffice. Legal adversaries frequently start with different premises. They disagree as to the facts of the case at hand or the rules of law that apply. In law, however, it is common to encounter reasonable arguments for inconsistent results where the adversaries appear to agree about the facts and about which legal rules apply. As noted, this is the phenomenon of "legal indeterminacy" (Berman and Hafner, 1988).

One source of legal indeterminacy has already been illustrated in the *Johnson* case in Section 2.2.1. Legal rules employ open-textured legal concepts about which reasonable but contradictory arguments are made.

Another source involves unstated conditions on the rule's application, such as that its result not be inconsistent with certain countervailing principles.

This is illustrated in the case of *Riggs v. Palmer*, 115 N.Y. 506 (1889) involving an heir who killed his grandfather under whose will he was to inherit. The Court stated,

"It is quite true that statutes regulating the making, proof and effect of wills, and the devolution of property, if literally construed, . . . give this property to the murderer." (Berman and Hafner, 1988).

The Court, however, refused to enforce the statutes where it would contradict "fundamental maxims of the common law," "dictated by public policy," that "[n]o one shall be permitted to profit by his own fraud, or to take advantage of his own wrong, or to found any claim upon his own iniquity, or to acquire property by his own crime" (Berman and Hafner, 1988).

Legal rules may have other unstated conditions such as: Does the rule satisfy choice of law requirements? Is the rule constitutional? Conceivably, some of these conditions can be represented as additional conditions of legal rules. Berman and Hafner point out, however, that abstract conditions like the frequently violated "fundamental maxim" above would be very difficult to formalize (Berman and Hafner, 1988).

Given the reality of legal indeterminacy, Berman and Hafner argued that classical logical models are inappropriate for modeling how lawyers reason.

> Legal indeterminacy presents a direct challenge to the concept of logical validity, by the fact that a lawyer must be able to argue for either a conclusion or its opposite.
>
> Suppose there is a theory T which has a consequence C (i.e., there is a valid logical argument whose premises are the axioms of T and whose conclusion is C). We then know that C is true in every model of T; that is, C is true in every universe where T's axioms are all true. We also know, by the law of contradiction, that if C is true, then NOT C must be false: so, NOT C is false in every model of T. . . [w]e can [also] show that no valid argument (no matter what additional assumptions we make) that begins with the axioms of T can ever conclude NOT C.
>
> [I]t is logically impossible to begin with a set of premises, and create a valid argument for both a conclusion and its opposite. This restriction certainly makes sense – but in the law, such a "logical impossibility" seems to be precisely what happens! (Berman and Hafner, 1988, p. 191).

Contradictory propositions are also problematic for classical logic models because if both propositions are true, one can prove anything (see Carnielli and Marcos, 2001). An instructive example of this "explosive" feature of classical deduction, drawn from the history of philosophy, is discussed in Ashworth *et al.* (1968, p. 184). A sixteenth-century Italian demonstrated that "anything follows from an impossible proposition, by proving that 'Socrates is and Socrates is not' entails 'Man is a horse' ":

1. "Socrates is and Socrates is not implies Socrates is not."
2. "Socrates is and Socrates is not implies Socrates is."
3. "Socrates is implies Socrates is or Man is a horse."
4. "(Socrates is or Man is a horse) and Socrates is not implies Man is a horse." Hence
5. "Socrates is and Socrates is not implies Man is a horse."

If one would like a computer to interpret statutes as a court does, by considering arguments pro and con and selecting the stronger arguments, ordinary classical logical deduction is problematic. One needs to use something else. Logicians have developed some alternative logics that can deal with inconsistency subject to various constraints. In the field of AI & Law, however, the current answer to "what else is there?" is a computational model of argument with appropriate argument schemes as explained in Chapter 5.

2.5. MANAGEMENT SYSTEMS FOR BUSINESS RULES AND PROCESSES

Not all problem-solving with statutes involves litigation to determine if a statute has been violated. Not all reasoning with legal statutes involves complex issues of interpretation and arguments pro and con. In many situations, businesses and institutions simply want to design their business processes and conduct their day-to-day operations in such a way as to avoid violating the law. Surely, one can computationally model legal rules for solving practical problems in a way that does not require modeling full-scale statutory interpretation.

Indeed, most programs modeling statutes are probably designed to assist administration of institutional compliance to avoid litigation. The BNA program, for example, was probably not designed to deal with litigation between adversaries seeking to convince a judge about the meaning of a disputed term. Instead, it would be more likely used as an administrative aid to address the run-of-the mill scenarios involving questions of citizenship. An agency charged with administering the complex BNA could use the tool to handle the large percentage of cases that are complex enough as to befuddle civil servants but that ordinarily do not give rise to litigated disputes about the meanings of the statutes or regulations.

Descendants of logical models of statutes like the BNA program and of legal expert systems like Waterman's in Section 1.3.1 still play a role in helping institutions comply with relevant regulations.

2.5.1. *Business Process Expert Systems*

Companies are obligated to ensure compliance with complex legal requirements and regulations. There is always a risk that an existing business process violates a regulatory requirement or that a proposed modification will introduce a violation at some point. Businesses also need an ability to document compliance to auditors (Scheer et al., 2006, p. 143). This requires firms to identify the applicable legal rules and regulations, to "define requirements resulting from these laws for the individual company," to identify the particular business processes that are affected and the "concrete risks which result from these requirements within [those] processes," to define measures and controls to minimize those risks, and to test whether they are being applied (Scheer et al., 2006, p. 146).

One way to implement the legal-risk-reducing measures and controls is to translate the legal requirements and regulations into *business rules*, which, if followed, reduce the risks in the affected business processes. "In general, business rules are guidelines or business practices which design or lead the conduct of an enterprise" (Wagner and Klueckmann, 2006, p. 126). Once business rules have been formulated, human managers can enforce them as policies via the company's ordinary managerial hierarchy.

The business process rules can also be implemented in software systems that assist human managers to ensure compliance (see Scheer et al., 2006, p. v). For example, the expert system can warn managers about the need to conform company policies to general regulatory requirements or warn managers of specific instances of noncompliant behavior. The rules can be represented in a logical formalism as in the BNA approach or, more likely, as heuristic rules as in Waterman's program, Section 1.3.1, and incorporated into an expert system designed to test whether a business process is compliant.

Such business compliance expert systems are being applied in the commercial sector. Today, companies like Neota Logic provide technology with which law firms and companies can easily author their own expert systems for business compliance. For instance, the Neota website reports that the law firm, Foley & Lardner LLP, has authored a number of web-based expert systems modules under the name, Global Risk Solutions, to guide clients in their efforts to ensure compliance with the Foreign Corrupt Practices Act (FCPA), a federal anti-corruption/anti-bribery statute (Neota Logic, 2016, Case Studies).

The modules collect information from a client concerning its marketing methods, location business volume, and customers and outputs visual and quantitative assessments of a client's business risks under the FCPA.

Another module provides more specific counseling based on automated information gathering. "For example, if a GRS user clicks through a variety of intake questions related to meals and entertainment, they are asked questions such as whether they are going to entertain a foreign official." Depending on the answers, a Foley attorney can follow-up with specific counseling.

Where the business rules are formulated in propositional form, they can also be organized graphically in ways that are more intelligible to business personnel. For example, propositionalized business rules can be organized in a kind of work flowchart not unlike that shown in Figure 2.3. Since humans can readily understand the flowcharts, they are an effective way to communicate the legal requirements to employees and for purposes of audits. Norm graphs are another visual tool that can assist with business compliance.

A *norm graph* embodies "an abstract model of the legal norms" (Dietrich et al., 2007, p. 187). For each legal compliance result of interest, a graph is constructed, which "enables [one] to decide whether [an] intended legal result can be reached or not . . . [It] consists of legal concepts (represented by nodes) and links between

them (represented by arrows)" (Oberle et al., 2012, p. 281). The "norms determine a legal consequence (LC), given one or more states of facts (SF)" (Dietrich et al., 2007, p. 187).

Norm graphs are conceptually organized to support a process of *subsumption*, a kind of taxonomic reasoning with an ontology, a lexicon of concepts organized hierarchically (Oberle et al., 2012).

> [T]he norms in the positive law do not address singular cases but rather cover general classes of real-world situations. [A decision maker] faces a specific real-world situation ... [and] must try to find the norms whose general domain covers the situation (subsumes the situation) ... [T]o mechanize subsumption the semantics must be considered ... beyond thesauri. Ontologies ... reflect semantic relationships between terms, and these relationships can particularly be defined [to] directly support the subsumption process." (Dietrich et al., 2007, p. 188)

The norm graphs in Figure 2.6 illustrate subsumption with legal norms. The figure shows norm graphs for two legal conclusions involving compliance with data protection regulations, here the German Federal Data Protection Act (FDPA) concerning the *legality* of data collection and *effective consent*:

Section 4 (1) FDPA Legality of data collection, processing, and use: The collection, processing, and use of personal data shall be lawful only if permitted or ordered by this Act or other law, or if the data subject provided consent. (Oberle et al., 2012, p. 285)

Section 4a (1) FDPA Effective Consent: Consent shall be given in writing unless special circumstances warrant any other form ... Consent shall be effective only when based on the data subject's free decision. Data subjects shall be informed of the purpose of collection, processing or use and, as necessary in the individual case, or on request, of the results of withholding consent. (Oberle et al., 2012, p. 287)

The norm graphs have associated rules or tests represented in predicate logic, which determine if the legal conclusions apply. For instance, the following formula abstracts the norms in section 4a (1) FDPA Effective Consent associated with the left side of Figure 2.6 (see Oberle et al., 2012, p. 293).

$$\text{Effectiveness(E) AND givenFor(E,C)} \leftarrow (\text{Consent(C) AND givenIn(C,F) AND WrittenForm(F)) OR Exception(F) AND } \ldots$$

This formula means that the result E of Effectiveness is assigned to Consent C if the result F assigned to Consent C is WrittenForm or Exception and some other conditions, not shown, are satisfied. Another formula specifies when the result Exception is assigned to F.

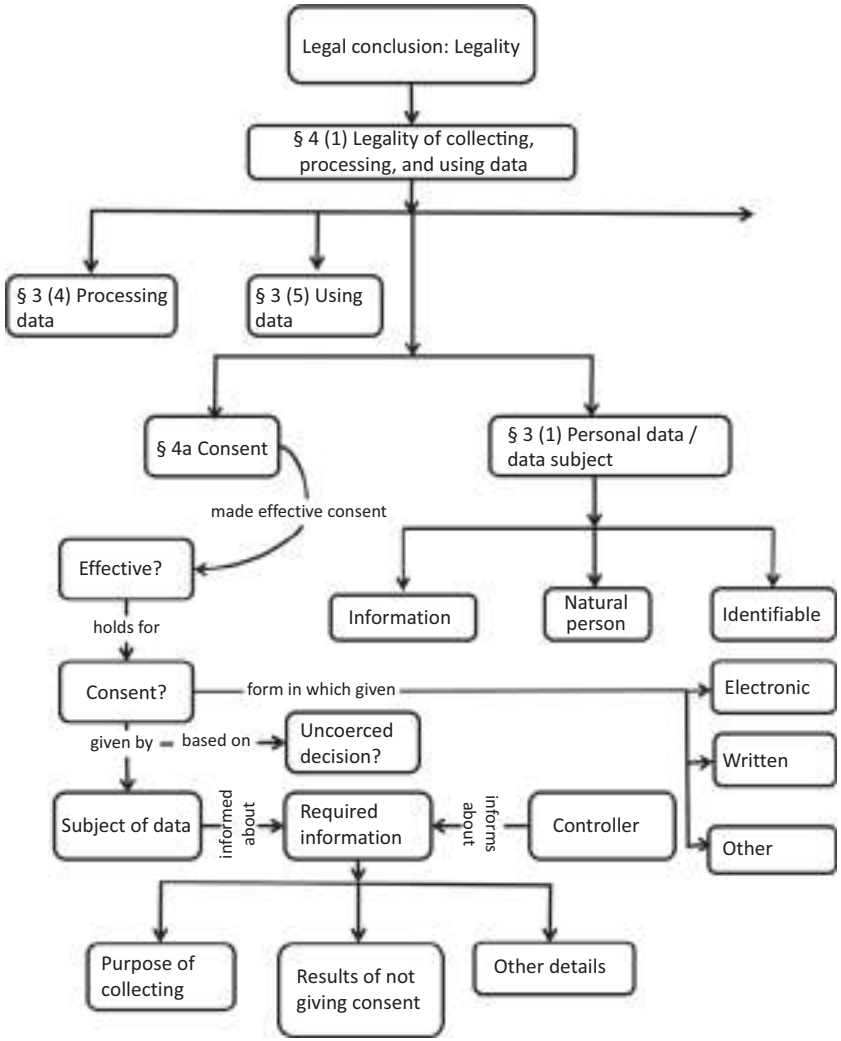


FIGURE 2.6. Norm graphs for concluding "Legality" in section 4 (1) FDPA and "Effective Consent" (see Oberle et al., 2012, pp. 305–6, Figs. 13 and 14)

With rules like these, an expert system could warn managers of the requirements that need to be satisfied in order to conclude that a business process is in compliance. The program could apply the tests to descriptions of real-world situations to determine if they are instances of the top-level norm classes representing the legal conclusions of interest, that is, whether the top-level concepts *subsume* the fact descriptions. For the subsumption to work, however, the factual scenarios must be represented in particular terms provided by a taxonomy of concepts associated with

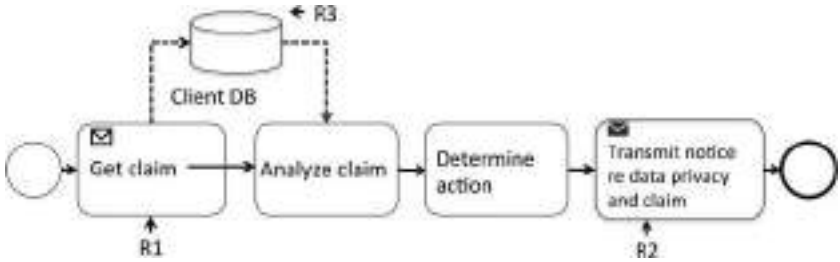


FIGURE 2.7. Sample BPMN diagram of simple insurance claim process with business rule annotations (see Table 2.1) (Koetter et al., 2014, Fig. 2, p. 220)

the regulated subject matter. In other words a subject matter ontology must also be constructed, as discussed in Section 6.5.

2.5.2. *Automating Business Process Compliance*

A goal of some research is to streamline the compliance process by enabling an expert system to analyze a model of the business process directly. The inputs to such an expert system are *business process models*, formal descriptions of proposed or operating business processes. These processes can be represented graphically in terms of schematic descriptions using the Business Process Model and Notation (BPMN), a standardized, modularized visual iconography for this purpose. The models can also be represented in a formal rule-modeling language so that expert systems can reason with them.

As noted, if an appropriate language for formalizing business rules is available and if it is compatible with the language for formalizing the descriptions of the business processes, then the business rules can be applied directly to the process model descriptions. In effect, the business rules are used to "annotate" the process models (and their graphical representations) in order to assess compliance.

For example, a BPMN diagram of a simplified insurance claim management process is shown in Figure 2.7. This insurance company happens to be subject to various requirements of the German Insurance Association (GDV) and to data protection laws in the German Federal Data Protection Act (FDPA). A human expert has translated those requirements manually into a set of three business rules, R1 through R3, shown in Table 2.1. The figure shows where each rule applies to the modeled process.

First, let's examine more closely what these three business rules are and where they came from. Human experts, knowledgeable about the business processes, need to know which regulations apply to such an insurance claim process in the relevant jurisdictions; here it happens to be a insurance company operating in Germany. For instance, according to Koetter et al. (2014), at least two regulatory provisions

TABLE 2.1. *From regulatory texts to business rules to annotations of business process (see Figure 2.7) to predicate logic forms (Koetter et al., 2014, p. 220)*

Paraphrased regulations	Business rules	As applied to business process	Predicate logic form
GDV Code of Conduct §§5–8: customer who provides personal data must be asked for agreement if this data is to be used for marketing purposes. This agreement has to be solicited within a short time span.	R1: After activity Receive claim an activity asking the claimant for agreement has to follow.	R1: Follow this by sending a data privacy notification	followedBy("Receive claim," "Send claim and data privacy notification") AND unknown
	R2: Activity asking for agreement has to be performed at most 14 days after activity Receive claim.	R2: Send at most 14 days after claim is received	followedBy("Receive claim," "Send claim and data privacy notification") AND maxTime BetweenActivities ("Receive claim," "Send claim and data privacy notification," "14 days")
German company outsourcing its data processing must ensure service providers comply with German FDPA §4b II sentence 1 BDSG re processing, storage, and exposure of personal data.	R3: Customer DB shall not be hosted outside of Germany.	R3: Do not store outside of Germany	hostingRegion ("CustomerDB," "Germany")

apply to the claims process in Figure 2.7. They are shown in the first column in Table 2.1.[3]

The human expert would need to read the actual provisions (i.e., the GDV code of conduct and the German FDPA provisions) and manually translate them into paraphrases and propositions summarizing the requirements like the three business rules shown in the second column of Table 2.1 (see Koetter et al., 2014).

[3] In column 1, the paraphrases of provisions in the code of conduct of the German Insurance Association (GDV) §§5–8, and in the German FDPA §4b II sentence 1 BDSG, are adapted from the authors' paraphrases in Koetter et al. (2014).

The business rules then need to be operationalized so that they can be applied to the specific business process model in question, perhaps with the assistance of a business process expert. The third column presents a simplified version as applied to the business process in Figure 2.7 (see Koetter et al., 2014).

A final step is to translate the operationalized rules, perhaps with the help of a knowledge representation specialist, into the predicate logic form shown in column 4 so that they can be applied by an expert system.

Translating the legal requirements and regulations into business rules is a complex interpretive task involving text understanding, commonsense reasoning, and business experience. For instance, German FDPA §4b II sentence 1 BDSG states:

1. The transfer of personal data to bodies
    1. in other Member States of the European Union, . . .
    shall be subject to Section 15 (1), Section 16 (1) and Sections 28 to 30a in accordance with the laws and agreements applicable to such transfer, in so far as transfer is effected in connection with activities which fall in part or in their entirety within the scope of the law of the European Communities.
2. Sub-Section 1 shall apply *mutatis mutandis* to the transfer of personal data . . . to other foreign, supranational or international bodies. Transfer shall not be effected in so far as the data subject has a legitimate interest in excluding transfer, in particular if an adequate level of data protection is not guaranteed at the bodies stated in the first sentence of this sub-section.
3. The adequacy of the afforded level of protection shall be assessed in the light of all circumstances.
4. Responsibility for the admissibility of the transfer shall rest with the body transferring the data.[4]

Formalizing this provision in its entirety would be very difficult, but a human expert would know that it is perhaps unnecessary. The expert might know from experience that the easiest way to finesse this requirement concerning the Act's protections of personal data would be to avoid transferring the personal data out of Germany. Thus, the expert would prepare a business rule (R3 in Table 2.1) as a kind of heuristic rule of thumb to ensure that the data is processed only in Germany (Koetter et al., 2014).

2.5.3. *Requirements for a Process Compliance Language*

Predicate logic alone is not adequate for the task of modeling the application of regulatory rules to business process models. A suitable language needs to support:

1. Reasoning with defeasible legal rules.
2. Isomorphic linking from the logical rules to regulatory sources.

[4] English translation from www.gesetze-im-internet.de/englisch_bdsg/englisch_bdsg.html, last accessed August 6, 2016.

3. Expressing the kinds of obligations that statutes and regulations employ.
4. Temporal reasoning (Gordon et al., 2009).

Each of these requirements is briefly described below.

Defeasible Legal Rules

First, the language has to support defeasible legal rules. *Defeasible* rules have the property that:

> When the antecedent of a rule is satisfied by the facts of a case, the conclusion of the rule presumably holds, but is not necessarily true. (Gordon et al., 2009)

The need for defeasible legal rules arises because, "Legal rules can conflict, namely, they lead to incompatible effects" (Gordon et al., 2009). One legal rule may be an exception to the other or exclude it as inapplicable or otherwise undermine it. We have encountered this above. As Berman and Hafner observed, "the logic-based formalism breaks down when applied to cases involving the existence of conflicting rules and precedents" (Berman and Hafner, 1988, p. 1). In addition, as discussed above, reasoning with legal rules often involves reasoning by default, and such reasoning is non-monotonic. Proven propositions may have to be withdrawn, that is, reasoning with legal rules is defeasible.

When designing business processes to ensure compliance with legal regulations, we have assumed that modeling litigation-style arguments about conflicting rules could be avoided. Nevertheless, according to Guido Governatori, a veteran modeler of business process compliance, the language still needs to support "the efficient and natural treatment of exceptions, which are a common feature in normative reasoning" (Governatori and Shek, 2012).

For example, in Figure 2.2, compare the complex textual version of the IRC provision (IRC section 354) and the propositional form with its simplified logical structure on the right.

Linking to Regulatory Sources for Explanation and Maintenance

Since business management systems monitor compliance, the system's rules must be updated, maintained, and validated and its results must be explainable with reference to the regulatory texts. These functions are simplified to the extent that the linkages between the logical versions of the rules and their sources in the regulatory texts are straightforward. More specifically, the legal rule modeling language needs to support *isomorphism*:

> There should be a one-to-one correspondence between the rules in the formal model and the units of natural language text which express the rules in the original legal sources, such as sections of legislation. (Gordon et al., 2009)

Ideally, the language maintains a one-to-one correspondence between the rules in the formal model and the sections of the regulatory texts. "This entails, for

example, that a general rule and separately stated exceptions, in different sections of a statute, should not be converged into a single rule in the formal model." (Gordon et al., 2009).

Maintaining isomorphism makes explanation more effective. Business rule systems can explain their analyses by recapitulating the rules that "fired," as illustrated above in the output of the BNA system. In the context of an audit, however, explaining the compliance analysis in terms of the business rules is not sufficient. An explanation must justify it in terms of the textual statutory provisions, not simply the business rules that a human expert has constructed to interpret and operationalize those provisions. For purposes of citing statutory texts and interweaving textual excerpts, an isomorphic mapping is essential.

Isomorphic mappings between statutory text and implementing rules, however, are difficult to maintain. Frequently, the mapping is complex especially where multiple, cross-referenced provisions are involved. The versions of statutes and regulations that computers can reason with logically are different from the authoritative textual versions. Statutes may be so convoluted that even a "faithful representation" remains unhelpful. As Layman Allen noted, statutes may include complex and sometimes implicit exceptions and cross-references both within and across provisions.

The fact that statutes and regulations are dynamic complicates maintaining the correspondence. The legislature may modify statutes or enact new ones, agencies may revise and update regulations, and court decisions announce new interpretations of the provisions' requirements. Even when the set of statutory provisions to be implemented is taken as static, as discussed in Section 2.3.4, the development of the BNA program was a process of trial-and-error requiring frequent revision to accommodate newly encountered rules.

When regulatory texts are amended, both the textual and corresponding logical versions need to be updated. Rule-based legal expert systems (e.g., like Waterman's program in Section 1.3.1) that use heuristic rules to summarize statutes avoid some aspects of the maintenance problem, but they still need to be updated when the statutes and regulations change. Since updating introduces modifications and additions to the rule-set, it is also important to revalidate the business rules whenever they are introduced or modified, in part by comparing them to their sources.

Some automated techniques have been developed to maintain isomorphic representations of regulations. The development environment in Bench-Capon (1991), for example, maintained a complex set of linkages between textual, logical, and intermediary representations of statutes. In such an environment, changes to the rules can be undertaken in a localized fashion. The links between the textual and logical rules can assist validation. In addition, decision aids such as textual excerpts from the statutory rules and links to commentary and cases can be linked into the program's logical explanations of a conclusion. Techniques for maintaining this faithful representation, however, require maintaining multiple representations (this development environment had three) and require complex software to keep track of them all.

Able to Express Different Types of Obligations and Reason Temporally

A language for business process compliance modeling needs to have the right "semantics" for expressing the kinds of normative concepts that statutes and regulations employ and the kinds of obligations they impose. The obligations may differ in terms of whether it:

- needs to be obeyed at all time instances in the interval in which it is in force,
- needs only to be achieved at least once while it is in force,
- could be fulfilled even before the obligation is actually in force,
- needs to be achieved immediately or else a violation is triggered,
- is such that a violation can be compensated for, or
- persists after being violated. (Hashmi et al., 2014)

A language that supports expressing these different types of obligations also needs to be able to reason temporally. Beside how long an obligation holds, legal rules have other temporal properties including "the time when the norm is in force and/or has been enacted" and "the time when the norm can produce legal effects." For a discussion of techniques for maintaining and reasoning temporally with multiple versions of statutory provisions (see Palmirani, 2011).

2.5.4. *Connecting Legal Rules and Business Processes*

The Process Compliance Language (PCL) was designed to satisfy all of the above requirements (Hashmi et al., 2014). It can represent legal rules as defeasible and avoids the problems of reasoning with contradictory rules. It also can define obligations with the above semantics and reason temporally.

A complex business process may have a lot of moving parts, however, and compliance needs to be assessed when the process is in operation. How does the model represent a process without oversimplification so that the business rules can be applied realistically?

For this to work, the model must account for the artifacts the business process produces and the changes that it makes in its environment (Hashmi et al., 2014, p. 104). The authors model a business process as a workflow-net, a kind of Petri net. *Petri nets* (introduced by the German mathematician and computer scientist C.A. Petri in 1962) are used to represent processes abstractly. Petri nets are not unlike the ATNs in Section 1.4.2, another kind of process representation. There, the process modeled was a legal one, the process of offer and acceptance in contract law. The nodes in Gardner's ATN represented states in the legal analysis; the arcs represented the possible transitions from one state to another governed by legal rules associated with each arc.

Petri nets are different from ATNs, however, in that they use two types of nodes, places and transitions, with arcs connecting one type of node to the other. In addition, the production and consumption of "tokens" are used to represent the events

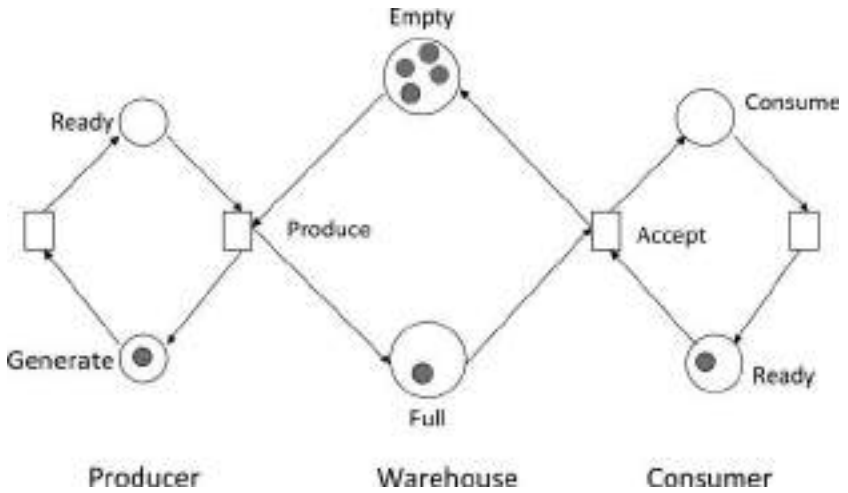


FIGURE 2.8. Petri net representing simple producer–consumer resource allocation problem (see Kafura, 2011, p. 8)

that occur in the process and the changes in the system's state that an event causes. In a Petri net, each event is modeled as a transition that consumes and produces tokens, and "the state of a system is modeled at any moment by a distribution of tokens in the net's places" (Palanque and Bastide, 1995, p. 388).

> Places and transitions are connected by directed arcs, which define when each transition is allowed to occur, and what the effect of its occurrence will be. A transition is allowed to occur when each of its input places holds at least one token; The occurrence of the transition consumes a token in each input place and sets a token in each output place. (Palanque and Bastide, 1995, p. 388)

The Petri net in Figure 2.8 represents a simple producer–consumer scenario involving resource allocation synchronization. Rectangular nodes represent transitions; circular nodes represent places. In this example, the resources are "items" that are produced and consumed. The items may be widgets, but they may also comprise information. A producer creates new items but may not generate a new item unless the number of available items is less than some maximum number. The consumer accepts one produced item at a time, but cannot accept an item unless at least one is available.

The capacity of the warehouse in the middle of the figure constrains the maximum number of available items. In this example, the maximum number is five, represented by the total number of tokens in the Full and Empty places: four in the Empty place plus one in the Full place. Thus, the figure represents a situation after the initial state where the producer has generated one item for the consumer to accept as represented by the token in the Full place. The producer then generated

a new item in the Generate place and is ready to ship it to the warehouse when the item transitions to the Ready place. The production transition, however, may only occur (i.e., "fire") when there is a token in the Empty place. Similarly, the accept transition can only fire when the consumer is ready, that is, there is a token in the consumer's Ready place at the lower right, and at least one token in the Full place. Once accepted, the item may be consumed.

The Petri net model of the process can be implemented in software with rules defining the conditions for the transitions. According to the transition rules, a transition is enabled (that is, the transition can fire) "when there is at least one token on each of the transition's input places; when a transition fires it removes one token from each of its input places and produces a single token on each of its output places" (Kafura, 2011, p. 2). One could imagine variants of the transition rules that change the maximum possible number of available items or that specify the (likely different) numbers of items that can be produced or consumed at a time or the rates of production and consumption.

While admittedly very simple, the Petri net in Figure 2.8 conveys an intuition about how a complex business process can be modeled in software. Petri nets can model nondeterministic system behavior; "If there is more than one enabled transition any one of enabled transitions may be the next one to fire" (Kafura, 2011, p. 2). A Petri net extension, *labeled workflow net*, makes traces of a process's possible execution sequences; it requires each node of process model to lie on direct path between the sole source and end places, and labels some transitions as "visible" (Hashmi et al., 2014, pp. 104, 111).

The traces of the business process operation generated by the labeled workflow net can be the inputs for an expert human, or for an expert system with formalized business rules, to analyze for compliance. The business rule obligations are associated with "each task in a trace . . . [and] represent the obligations in force for that combination of task and trace. These are among the obligations that the process has to fulfill to comply with a given normative framework" (Hashmi et al., 2014, p. 108). A program evaluates whether those facts, associated with the tasks and recorded in the traces, that should be true according to the business rules really are true.

The compliance analysis can be performed at various points in the life cycle of a business process:

Design-time: When the process is being designed, by analyzing the developing process model in a computerized design environment that enforces compliance with regulatory constraints *a priori*.

Run-time: While the process is running, by governing how the process unfolds to ensure execution is in compliance.

Post-execution: After execution, by analyzing a trace or history of the operations of a process to identify instances of noncompliance.

In order to make the determinations at design-time, run-time, or post-execution, a workflow net representing the business process at that point needs to be constructed and traces of its operation need to be generated for analysis (see Hashmi et al., 2014, p. 112).

2.5.5. *Example of Business Process Compliance Modeling*

Hashmi et al. (2014) applied the PCL to model regulation of a business process for complaint handling under the Australian Telecommunication Consumers Protection Code (TCPC) 2012. The code specifically mandates that every Australian entity operating in the telecommunication sector must certify that their day-to-day operations comply with the code.

Specifically, they modeled TCPC § 8, which governs the management and handling of consumer complaints (Hashmi et al., 2014, p. 113f). TCPC §8 was manually mapped into "176 PCL rules, containing 223 PCL (atomic) propositions (literals)" using all of the obligation types listed above. The authors secured the regulator's informal approval of the business rules for purposes of the exercise.

With the assistance of domain experts from an industry partner, they drew process models to capture the company's existing procedures for handling complaints and related matters under TCPC §8. This process resulted in six business process models, annotated in terms of the relevant business rules, five of which were small enough to be "checked for compliance in seconds." Evaluating compliance in the largest business process, with 41 tasks and 12 decision points, took about 40 seconds of computational time (Hashmi et al., 2014, p. 114).

The system outputs a report of traces, rules, and tasks responsible for noncompliance like that in Figure 2.9. Although the figure deals with a different business process for opening credit card accounts, it illustrates the kind of information the system can generate based on its analysis of a business process's compliance. It identifies noncompliant execution paths and cites the regulatory rule that is the source of a noncompliance issue.

In the compliance evaluation of the complaint handling process, the team identified various points at which the business processes failed to comply with TCPC §8. "Some of the compliance issues discovered by the tools were novel to the business analysts and were identified as genuine non-compliance issues that need to be resolved" (Governatori and Shek, 2012). The noncompliance issues involved ensuring that "some type of information was recorded in the databases associated [with] the processes," that customers were made "aware of documents detailing the escalation procedure," and that "a particular activity does not happen in a part of the process." Two of these noncompliance issues resulted from "new requirements in the 2012 version of the code" (Hashmi et al., 2014, p. 114).

The team employed the compliance software environment to rectify some of the noncompliance issues. The repairs included modifying the existing processes to

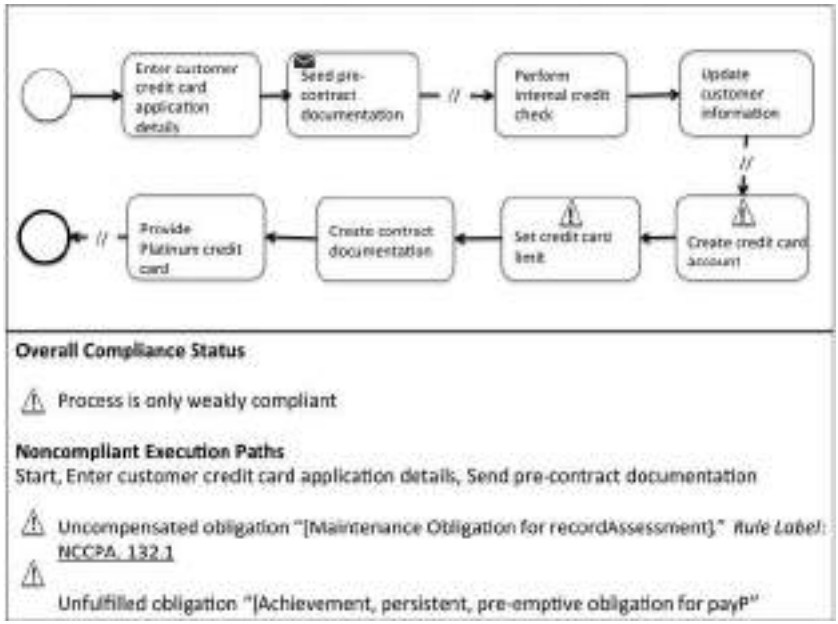


FIGURE 2.9. Compliance system report of traces, rules, and tasks responsible for noncompliance (excerpts) (see Governatori and Shek, 2012)

comply with the code or designing and adding some new business process models, such as a novel way to handle in person or by phone complaints (Governatori and Shek, 2012; Hashmi et al., 2014, p. 114).

Governatori's system performed real legal work in a realistic setting. It required extensive manual effort, however, both in developing the business rules and in representing the business process for analysis with the business rules. The formulation of the business rules from the regulatory sources was entirely a manual effort. The construction of the model of the business process as inputs for the business rules to annotate appears to have been the result of an intricate manual task as well.

Governmental agencies might benefit from automating administrative process compliance with regulations. van der Pol (2011) described a business process compliance model to be fielded by the Dutch Immigration and Naturalization Service (IND). An information system called INDiGO was to contain an expert system of business rules based on relevant laws, regulations, and policies governing the processing of IND clients' applications. The rule engine would contain a model of the process workflow including the order in which business services are to be executed, and could be consulted to provide those services relevant to a client's specific case and circumstances. The system was to analyze trace histories of the operations of the business process to identify instances of noncompliance, unsatisfactory results, or inefficiencies and provide feedback to regulators on modifying relevant statutes

and regulations. The goal is to create a flexible system in which changes in law, regulation, or procedures could be implemented quickly through modifications of relevant business rules in this rule engine (van der Pol, 2011). After auspicious beginnings, however, it is not clear whether the INDiGO project has succeeded due to a lack of published information.

2.6. REPRESENTING STATUTORY NETWORKS

Expert systems and logic programming are not the only paradigms that can support computational reasoning with statutes and regulations. Regulatory systems contained in statutes can be represented as networks or graphs of the relations between objects. The connected objects can be other statutes and provisions, a *citation network*, or a set of reference concepts referred to by, and subject to, regulation across multiple statutes, a *statutory network diagram*.

For instance, in a recent project, states' systems of regulations for dealing with public health emergencies are represented as networks of nodes. The nodes represent the agents that a statute directs to communicate with other agents under specified conditions (Sweeney et al., 2014). Using expert handcrafted queries, a team of researchers retrieved candidate statutes concerning public health emergency preparedness from the LexisNexis legal databases for 11 US states. Each provision was manually coded according to a standardized codebook to identify if the provision was relevant and, if so, the provision's citation, the public health agents that are the objects of the provision, the action the provision directs and whether it is permitted or obligatory, the goal or product of the action, the purpose of the statute, the type of emergency in which the direction applies, and under what time frame and conditions (Sweeney et al., 2014).

Once different states' regulatory systems are represented as networks, the networks can be compared visually and quantitatively using network analytical measures, and tentative inferences can be drawn about a state's regulatory scheme as compared to another state's scheme. For example, Figure 2.10 compares statutorily mandated institutional interactions relating to emergency surveillance between Florida and Pennsylvania.

Comparative diagrams like these can suggest hypotheses to public health system analysts about the differences across states, which can then be studied in light of the legislative texts. For instance, based on the white links in Figure 2.10, one might ask why Community Health Centers and Home Health Agencies are linked to other public health agents in Pennsylvania but not in Florida? Investigating possible answers would involve researching the legislative texts in Pennsylvania and Florida.

The statutory network diagrams can help. They are a kind of visual interface into a state's statutes. They could enable researchers or field personnel to retrieve the provisions that direct institutional agents' interactions simply by clicking the network links representing those interactions (Sweeney et al., 2014). Thus, a researcher could,

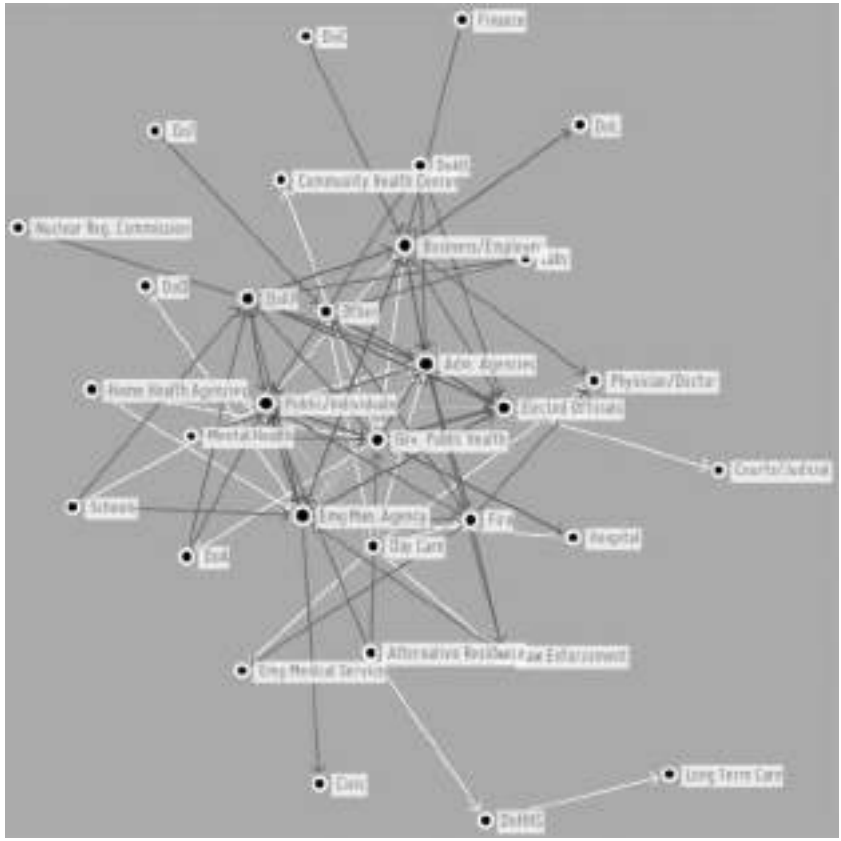


FIGURE 2.10. Statutory network diagram comparing Pennsylvania (PA) and Florida (FL) statutory schemes re public health emergency surveillance: Circles indicate public health system actors and partners in FL and PA. Grey links indicate relationships present in both states; white links indicate legal relationships present in PA but not in FL (Sweeney et al., 2014)

at least, retrieve the relevant statutes directing the linkages to Community Health Centers and Home Health Agencies in Pennsylvania. Based on those texts, one could frame queries for similar statutes in Florida using conventional legal IR tools. The queries will reveal either that Florida law contains similar directives that have been missed in constructing the statutory network, or more interestingly, that there is a gap in Florida's laws that policy-makers might conclude should be filled.

Tools like statutory network diagrams and citation network diagrams can help humans solve problems involving statutory reasoning where the computer and human share responsibility for performing the tasks most within each's capabilities. Chapter 11 on conceptual legal information retrieval examines more closely the use of citation networks and statutory network diagrams in cognitive computing. Citation

information can be extracted automatically from statutory texts or retrieved from a repository of statutes and used to create citation networks. Creating a statutory network diagram is more complex, requiring extensive manual encoding of the statutes. Chapter 9 on extracting information from statutes addresses techniques to apply ML to automate or semiautomate the encoding task for constructing statutory network diagrams.

...

In the remainder of this book, we revisit the subject matter of representing business rules, statutes, and regulations. Section 6.5 addresses the construction of ontologies for statutes and regulations. Standardized schemes have been developed for annotating or tagging statutes and regulations with procedural and substantive semantic information that can then be used to search for relevant provisions. Chapter 9 explains how the automated approaches for extracting information from statutory and regulatory texts can support conceptual information retrieval. Other projects, discussed in Section 9.5, tackle the task of automatically extracting logical rules and constraints from regulatory texts, focusing on a small set of regulations in repetitive stereotypical forms.

3

Modeling Case-based Legal Reasoning

3.1. INTRODUCTION

Since legal rules employ terms and concepts that can be vague and open-textured, a computational model of reasoning with cases would help. Courts often interpret the meaning of legal terms and concepts by drawing analogies across cases illustrating how a term or concept has been applied in the past.

This chapter presents computational models of analogical reasoning with legal cases. The models are based on three basic approaches. The first, prototypes and deformations, focuses on how to decide a case by constructing a theory based on past cases. The second, dimensions and legal factors, employs stereotypical patterns of fact that strengthen or weaken a side's argument concerning a legal claim or concept. The third, exemplar-based explanations (EBEs), represents legal concepts in terms of prior courts' explanations of why a concept did or did not apply.

The models illustrate how to represent legal cases so that a computer program can reason about whether they are analogous to a case to be decided. In particular, they illustrate ways in which a program can compare a problem and cases, select the most relevant cases, and generate legal arguments by analogy for and against a conclusion in a new case.

Legal rules and concepts are promulgated for normative purposes. Teleological arguments (i.e., arguments from the purposes or values served by a rule) play an important role in drawing legal analogies. Computational models that integrate legal rules, intermediate legal concepts (ILCs) from those rules, and cases applying the rules need to take underlying values into account. This chapter introduces techniques for computationally modeling teleological reasoning by integrating values into the measures of case relevance and models of legal analogy.

None of these systems deals directly with legal texts. Instead, they work on the basis of formal representations of case facts and legal concepts that have been manually constructed. The assumption, however, has been that one day, these case

representations will be extracted automatically from natural language texts of case opinions or fact summaries. With text analytics, that day is fast approaching. The chapter contrasts how amenable the different case representations are to text analytic approaches and their implications for cognitive computing.

This chapter answers the following questions: How can legal concepts be represented computationally in a way that reflects their dialectical relationship with cases? How can cases' facts and courts' reasoning be represented computationally? What are prototypes and deformations, dimensions or factors, and EBEs? What aspects of a court's decision do they capture, and what aspects do they miss? What is a trumping counterexample? What are semantic networks and "criterial" facts? How can the legal relevance of a case to a problem be measured computationally? How can a program select relevant cases, compare them in terms of similarity, analogize them to, and distinguish them from fact situations and other cases? How can such programs be evaluated empirically? What is teleological reasoning? What roles does teleological reasoning play in drawing analogies across legal cases? What roles do hypotheticals play in teleological reasoning? How can values underlying legal rules be represented computationally, and how can a computer program integrate values into its methods for selecting relevant cases, drawing analogies, and distinguishing cases?

3.2. RELATIONSHIP OF LEGAL CONCEPTS AND CASES

Computational models of case-based legal reasoning model the interactions between legal concepts and cases. The legal concepts correspond to the open-textured terms in constitutional, statutory, or court-made legal rules. In common law and, to some extent, in civil law jurisdictions, cases play a role in elucidating the meanings of the open-textured legal concepts and in mediating the way in which those rules and meanings change.

3.2.1. *The Legal Process*

Edward Levi famously contrasted the process of legal reasoning by example with the pretense of law that it "is a system of known rules applied by a judge" (Levi, 2013, p. 1). For Levi, law involves a "moving classification scheme," where the legal concepts are the classifiers. "The kind of reasoning involved in the legal process is one in which the classification changes as the classification is made. The rules change as the rules are applied" (Levi, 2013, pp. 3–4).

In this process, courts decide whether the result of a precedent's rule should apply in a new case, in part by comparing the facts of the new case with those of the precedent. In determining whether the new case is similar to or different from a precedent, courts may elucidate but often muddy the meaning of the rule's legal concepts. When a concept's meaning becomes too incoherent, a court may introduce an exception to the rule by introducing a new legal concept, the rule

is modified, and the process continues. Eventually, even the rule with exceptions becomes incoherent, and a court jettisons it in favor of a new rule (Levi, 2013).

3.2.2. *The Legal Process Illustrated*

Levi illustrated the legal process in his recounting of the development of modern product liability law. Strict product liability law, as modeled in Waterman's legal expert system (Section 1.3.1), originated in a process of case-based reasoning. Exceptions eroded the "privity" requirement limiting manufacturers' liability in a series of cases including *Thomas v. Winchester*, 6 N.Y. 397 (1852). The rule was replaced in *MacPherson v. Buick*, 217 N.Y. 382, 111 N.E. 1050 (1916) and in the later formulation of modern strict product liability law, for example, in the Restatement (Second) of Torts.

As most American law students are taught, the longstanding common law rule had been that "[a] manufacturer or supplier is never liable for negligence to a remote purchaser" (Levi, 2013, p. 25). That is, "no privity, no liability." There were some exceptional fact situations where a manufacture was held liable even to a third party despite the rule. In *Thomas v. Winchester*, a court announced a concept to name the exceptions: if an item were *imminently dangerous*, there could be liability without privity of contract. In subsequent decisions, courts classified various products as imminently dangerous, and others not, and introduced some variations on the concept, such as "inherently dangerous" or even "eminently dangerous." After all, one "concept sounds like another, and the jump to the second is made" (Levi, 2013, p. 8).

The process of classification continued with the courts seemingly enlarging the class of inherently dangerous articles but refusing to allow recovery for articles that were merely dangerous if defective:

> One who manufactures articles inherently dangerous, e.g., poisons, dynamite, gunpowder, torpedoes, bottles of aerated water under pressure, is liable in tort to third parties... On the other hand, one who manufactures articles dangerous only if defectively made, or installed, e.g., tables, chairs, pictures or mirrors hung on the walls, carriages, automobiles, and so on is not liable to third parties for injuries caused by them, except in case of willful injury or fraud. *Cadillac v. Johnson*, 221 Fed. 801, 803. (Levi, 2013, pp. 19–20)

Eventually, these example-based classifications may come to look silly and irrational, and a court throws out the rule altogether. In *MacPherson v. Buick*, the New York Court of Appeals allowed plaintiff MacPherson, a third party, to recover for injuries caused by a Buick, a type of article the Court in the previous year had classified as dangerous only if defective, denying liability in the *Cadillac* case. In Judge Cardozo's landmark opinion, the Court ruled,

> If the nature of a thing is such that it is reasonably certain to place life and limb in peril when negligently made, it is then a thing of danger... If to the element

> of danger there is added knowledge that the thing will be used by persons other than the purchaser, and used without new tests, then, irrespective of contract, the manufacture of this thing of danger is under a duty to make it carefully. 217 N.Y. 389.

In 1964, the rule regarding a seller's liability for physical harm caused by defective products to third-party users or consumers was transformed into the modern product liability law that Waterman modeled. See Restatement, Second, Torts §402A. Special Liability of Seller of Products for Physical Harm to User or Consumer.

3.2.3. *Role of Legal Concepts*

To summarize, according to Levi, legal concepts play a number of roles. They are components of the rules of law. They have meanings and, to some extent at least, support deductive reasoning about whether the concept applies to a new case. The legal process is rule-guided to some extent, but it is far from just a matter of applying the rules deductively to new situations (Levi, 2013).

A primary role of concepts is to focus on particular similarities that, at any given time, society deems important in making this determination of justice. A legal concept is thus a "label" that reifies these similarities across a collection of cases. Courts reason with the similarities when they decide the new case. As Levi puts it,

> The problem for the law is: When will it be just to treat different cases as though they were the same? A working legal system must . . . be willing to pick out key similarities and to reason from them to the justice of applying a common classification. (Levi, 2013, p. 3)

In the process of deciding that certain cases are similar or different, legal rules and their concepts change. A concept expands or contracts as courts decide that it applies or not in new cases. In addition, the assessments of particular similarities as relevant or irrelevant may change as social circumstances and values change. Thus, previous analogies become suspect and lead to decisions now deemed unjust. When the facts of cases stretch the concept's meanings beyond credulity, a court may (subject to various constraints such as its place in the judicial hierarchy) replace it with a new concept in a reformulated rule. Existing legal rules and the arguments in previous cases suggest new concepts for restricting, extending, or replacing existing rules to deal with changed factual circumstances and social values (Levi, 2013) (see also Ashley and Rissland, 2003).

A closer examination of the history of product liability law in Levi's account identifies some features or factors courts applied in their example based on reasoning and argument. Courts compared cases in terms of:

- Whether the manufacturer knew about the hidden defect.
- How difficult it would be to discover the defect.
- Whether the manufacturer had fraudulently hidden the defect.

- The likelihood that the article would be used by one such as the victim.
- Who had control of the article.
- How dangerous the article was.
- Whether the danger was because of some additional act.
- The nature of the injury resulting from the defect.
- Social expectations regarding reliance on the manufacturer (e.g., pharmacist, auto manufacturer).

In the context of product liability, these are sensible criteria in terms of which to compare cases and to assess the justness of a proposed outcome based on a rule or interpretation of a legal concept (Ashley and Rissland, 2003). Changes in what society wants and what technology affords affect which criteria are deemed important. Focusing on different collections of these criteria as important leads to different orderings of cases. The legal reasoning process (and the legal forum of which it is a part) supports this dynamism with its use of rules, concepts, and case examples (Levi, 2013).

3.3. THREE COMPUTATIONAL MODELS OF LEGAL CONCEPTS AND CASES

Modeling case-based legal reasoning requires techniques to represent knowledge about case facts and to assess legally relevant similarities. Since the models must decide whether to treat cases the same way from a legal viewpoint, the similarities and differences must be represented in a form that a program can process, analyze, and manipulate.

Three types of computational models have been developed to represent case facts, define relevant similarities and differences, and relate them to legal concepts and to compare cases: prototypes and deformations, dimensions and legal factors, and EBEs. The three models vary the mix of intensional and extensional elements they employ to represent legal concepts. An *intensional* definition specifies the necessary and sufficient conditions for being an instance of the concept. For example, a "vehicle" is any instrument of conveyance used, or capable of being used, as a means of transportation. An *extensional* definition simply provides examples of what is/is not an instance of a concept. For instance, automobiles, bicycles, and a 103.1 cc Harley-Davidson Low Rider motorcycle are examples of a "vehicle" but an inoperable World War II Sherman tank is not. As explained in Chapter 5, computational models of legal argument now incorporate aspects of these case-based models in their schemes for analogical argumentation.

Computational models of legal reasoning *approximate* the process of legal reasoning with cases and concepts. Given the complex interaction of concepts and cases illustrated in Levi's examples, AI & Law researchers necessarily must simplify the process. Specifically, the models focus on a comparatively small number of cases, for example, 40 cases and hypotheticals involving workmen's compensation, fewer than

200 cases in trade secret law, or a half dozen property law cases involving hunters' rights in quarry. In addition, the models all focus on an area of the law in a less dynamic period, when the relevant concepts are more or less fixed and reasoning by example is used to classify items as in or out of the concept (Levi, 2013, p. 9). This is before the concept breaks down or is rejected because it obstructs a reclassification of the cases. For instance, the trade secrets and workmen's compensation cases involve fairly static legal concepts. (Some interesting AI & Law work by Edwina Rissland, however, does model and monitor conceptual change, Section 7.9.4).

Despite the constraints and simplifications, the developers try to ensure that the resulting models are still complex enough to perform some useful tasks. The focus has been on modeling the role of cases as exemplars of concepts and on normative values as informing the determinations of similarity and difference.

3.3.1. *Prototypes and Deformations*

Thorne McCarty's Taxman II program modeled arguments by analogy to past cases. Legal concepts in Taxman II were represented intensionally and supplemented extensionally using a technique called "prototypes and deformations."

McCarty represented three components of legal concepts: "(1) an (optional) invariant component providing necessary conditions; (2) a set of exemplars providing sufficient conditions; and (3) a set of transformations that express various relationships among the exemplars." He referred to the exemplars as *prototypes*: precedent cases and hypotheticals that were positive and negative examples of the legal concept whose meaning was being argued about. The transformations were *deformations*, mappings that allowed prototypes to be compared in terms of their constituent concepts (McCarty, 1995, p. 277, see also McCarty and Sridharan, 1981). In terms of Levi's domain, for instance, "imminently dangerous" might be thought of as a prototype concept. Groups of cases deform it into "inherently" or "eminently" dangerous, thereby preserving a quality of danger but partially altering it given the circumstances of particular cases.

The *Eisner v. Macomber* Example

The program focused on one scenario at the heart of a U.S. Supreme Court case, *Eisner v. Macomber*, 252 U.S. 189 (1920) concerning the issue of whether a pro rata stock dividend in connection with a stock split was taxable income to its shareholders under the Sixteenth Amendment to the U.S. Constitution. If not, it would fall outside the Congress's power to levy an income tax (McCarty, 1995). In the *Eisner* scenario, Mrs. Macomber owned 2,200 shares of Standard Oil. When Standard Oil declared a 50% stock dividend, she received 1,100 additional shares, part of which represented accumulated earnings by the company.

This and related cases involved subsidiary concepts such as "distribution," "shares," "bonds," "common stock," and "preferred stock." For each of these, rule-like templates represented the rights and obligations associated with the concept. For instance, the rights of corporate interest holders specified that bondholders received a fixed amount. Preferred stock holders received a fixed amount per share after bondholders. Common stock holders received a portion only of whatever is left after the bondholders and preferred stock holders were paid.

As input, Taxman II received a description of the fact situation, expressed not in natural language text but in terms of logic propositions employing the subsidiary concepts. The program output "arguments" (also in propositional form) that the dividend was or was not income, based on analogies to two prior cases and a hypothetical example.

At the time of the *Macomber* decision, these real and hypothetical cases were three available prototypes, positive and negative exemplars of *taxable income*, the main legal concept whose meaning was subject to dispute:

1. *The Lynch case*: Distribution of a corporation's cash was held to be taxable income to the shareholder.
2. *The Peabody case*: Distribution by a corporation to shareholders of the stock of another corporation was held to be taxable income.
3. *The Appreciation Hypothetical*: Appreciation in the value of a corporation's stock, held by the shareholder, without transfer of the shares was universally assumed *not* to be taxable income.

The deformations included some built-in mappings like ConstantStockRatio, which compared shareholder ownership ratios before and after a distribution.

Argument as Theory Construction

McCarty characterized legal argumentation about the meaning of a legal concept as a kind of theory construction, which he justified as follows. An arguer constructs a theory of how to decide an issue based on aligning the current facts with prototypical exemplars. McCarty focused on the arguments of the taxpayer and the Internal Revenue Service, as reflected in those of the majority and the dissent in the *Macomber* case, and designed the program to reconstruct the arguments pursuant to a template (or scheme). According to the argument template:

Taxpayer: defines taxable income so the *Eisner* facts and any negative prototypes of taxable income (the Appreciation Hypothetical) are excluded but any positive prototypes (*Lynch* and *Peabody*) are included.

Internal Revenue Service: defines taxable income so *Eisner* and any positive prototypes (*Lynch* and *Peabody*) are included but any negative prototypes of taxable income (the Appreciation Hypothetical) are excluded.

In doing so, the program, in effect, searched for a theory that links the current case with the favorable prototypes for a side (taxpayer or IRS) and excludes the unfavorable ones. Deformations or mappings across cases provided the raw material for these links. If a mapping preserves some constituent concept across the positive instances and the current case, then the *invariant property* becomes the basis of a theory that they should be decided alike.

The program employed argument strategies like looking for some continuum that could serve as an invariant property across a problem and a favorable prototype case. In linking the *Eisner* facts and the nontaxable Appreciation Hypothetical, the program found such an invariant via the built-in ConstantStockRatio mapping: before and after the "distribution," the taxpayer retained the same proportionate share of ownership of the corporation. After the dividend, Mrs. Macomber owned 3,300/750,000 of the corporation, the same ratio as before (2,200/500,000). It is as if there were no transfer.

If the program could not find an invariant property, it would search a space of options in trying to construct one, for example, by selecting and applying elementary mappings to build more complex ones. In trying to find conceptual links between prototypes, the program reasons about the meaning of their constituent components.

This appears to be similar to human argumentative reasoning. Justice Brandeis (in dissent) proposed a continuum linking distributions of equity, debt, and cash in support of his argument that the distribution was taxable income: Distributions of cash, bonds, preferred stock, and common shares all confer upon the recipient an expected return of corporate earnings. They differ only in how much return and at what risk. If one such distribution yields taxable income, so should all.

The program examined the prototypes' constituent concepts and, apparently, discovered or constructed the same continuum from the *Lynch* prototype's taxable distribution of a corporation's cash to distribution of a corporation's bonds, distribution of its preferred stock, and distribution of its common stock (i.e., the *Eisner*) scenario. Each confers on the recipient some trade-off between expected return of corporate earnings and risk.

Utility of Prototypes and Deformations for Cognitive Computing

From a legal viewpoint, Taxman II's model of arguing with concepts and cases is both sophisticated and realistic. The model focused on legal argumentation as constructing a theory by aligning selected cases in terms of a concept. Many attorneys, judges, and law clerks employ legal information retrieval systems to construct arguments like these. The challenge for cognitive computing is how to design computer programs that can assist users in constructing such arguments by formulating theories, linking them to analogous positive case examples, and distinguishing them from negative instances.

On the other hand, as a source of computational tools for achieving this goal, McCarty's approach in Taxman II may be too complex to be helpful. Searching

through the intensionally defined subsidiary concepts and mappings in order to discover invariants is an intricate affair whose robustness still needs to be demonstrated in domains involving issues other than the meaning of "taxable income." Indeed, the model was implemented for only one Supreme Court argument involving one argued about legal concept and four cases.

3.3.2. *Dimensions and Legal Factors*

Dimensions and legal factors are knowledge representation techniques designed to enable comparing the similarity of cases, drawing analogies to positive case instances, and distinguishing negative ones. They provide a simpler, more extensional scheme for representing legal concepts and cases than Taxman II that may be easier to connect to case texts for purposes of cognitive computing.

Hypo's Dimensions

As introduced in the Hypo program, legal "factors are a kind of expert knowledge of the commonly observed collections of facts that tend to strengthen or weaken a plaintiff's argument in favor of a legal claim " (Ashley, 1990, p. 27). "In Hypo, [legal] factors are represented with Dimensions. A Dimension is a general framework for recording information for the program to manipulate" (Ashley, 1990, p. 28, see also Ashley, 1991).

As a note on terminology, "factor" has two meanings: (1) The term "factor" (lower case) means a legal factor, the phenomenon that a dimension represents, namely a stereotypical pattern of facts that tends to strengthen or weaken a plaintiff's argument in favor of a legal claim. (2) As we will see, the CATO program introduced Factors (initial caps), a knowledge representation technique that simplified dimensions. Like dimensions, Factors represent legal factors.

Hypo dealt with the claim of trade secret misappropriation, that is, where the plaintiff claims defendant gained an unfair competitive advantage by using plaintiff's confidential product information. It dealt with one legal concept, whether a fact situation was an instance of trade secret misappropriation. For modeling this concept, it employed 13 legal factors, represented by 13 dimensions, and used them to index 30 trade secret cases.

The legal factors underlying the 13 dimensions were identified in a number of sources including the Restatement (First) of Torts, section 757, Liability for Disclosure or Use of Another's Trade Secret, which many jurisdictions adopted as an authoritative statement of the law of trade secrets. Comment (b) identifies six factors that courts should take into account in determining if information is a trade secret. Other legal factors came from the opinions of trade secret cases, where courts identify particular factual strengths and weaknesses, and from treatises and law review articles. These secondary sources tend to group cases in footnotes that illustrate the effect on outcomes of particular factual strengths and weaknesses. They may also list

FIGURE 3.1. Secrets-Disclosed-Outsiders dimension in Ashley (1990)

counterexamples where a court reaches a conclusion in spite of a particular strength or weakness.

As illustrated in Figure 3.1, each dimension instantiated a structured template of information that defined prerequisites for the represented legal factor's application to a fact scenario. Since a case may be a more or less extreme example of a legal factor, each dimension specified a focal slot whose value in a case could vary along a range representing a stronger or weaker magnitude for the plaintiff. For instance, the focal slot value for the Secrets-Disclosed-Outsiders dimension represented the number of disclosures to outsiders in a case. The focal slot value for the Competitive-Advantage dimension captured the amount of development time and cost saved by accessing the plaintiff's information. Cases can be compared in terms of their magnitudes along a dimension, that is, in terms of their focal slot values. In the figure, the *Data-General* case is rather remarkable with disclosures to 6,000 outsiders.

A legal factor's magnitude, as represented by a dimension's focal slot value, should be distinguished from its weight. "A [legal] factor's weight is some kind of measure of the support it lends to a conclusion that the plaintiff should win a claim." Hypo did not represent a legal factor's weight quantitatively. Instead, Hypo was intended to express legal factors' weights via arguments about specific scenarios.

One reason for not representing a legal factor's weight numerically is that such weights are context-sensitive. Three cases indexed along the *Secrets-Disclosed-Outsiders* dimension in Figure 3.1 illustrate this. The *Crown* and *Midland-Ross* cases, both won by defendants, lie at the left end of the dimension; even a few disclosures to outsiders can weaken a plaintiff's claim. On the other hand, the plaintiff won in

the *Data-General* case despite thousands of disclosures. Clearly, that case is inconsistent with the tenor of the dimension. The dimension indicates that this pro-plaintiff case is an exception or a counterexample by its position far to the pro-defendant end of the range. Other legal factors may counteract or "outweigh" the effect of the disclosures. In *Data-General* the disclosures were subject to confidentiality restrictions, represented in the *Outsider-Disclosures-Restricted* dimension.

Beside the fact that legal factor weights are sensitive to the particular context, Hypo did not represent weights for two other reasons. First, judges and attorneys do not argue about the weight of legal factors in quantitative terms. Second, legal domain experts do not agree what the weights are, and combining positive and negative weights numerically obscures the need for arguing about the resolution of competing legal factors. Chapter 4 presents ways to deal with legal factor weights for purposes of prediction.

Analogizing and Distinguishing Cases in Hypo's 3-Ply Arguments

The inputs to Hypo consisted of problem scenarios inputted in terms of an instantiated frame for representing facts of trade secret cases. The input problem is referred to as the current fact situation (cfs). Hypo's outputs were a three-ply argument in English that a plaintiff's trade secret misappropriation claim should [not] be successful. The three-play argument comprised:

1. An argument analogizing the cfs to a pro-plaintiff case.
2. An argument distinguishing the cited case from the cfs on behalf of defendant and citing pro-defendant counterexamples.
3. A rebuttal distinguishing the counterexample cases from the cfs and, where possible, a hypothetical suggesting facts to strengthen the plaintiff's argument in the cfs.

Hypo also made similar three-ply arguments on behalf of the defendant.

Analogizing a cfs and a cited case means stating *legally relevant* similarities that give rise to reasons why they should be decided the same way. In Hypo, such similarities are represented as shared dimensions. These dimensions represent legal factors common to the cfs and cited case. If at least one of these shared dimensions favors the side making the argument, Hypo considers the fact that the cited case was decided for that side as potential grounds for an argument for assigning the same outcome to the cfs.

Distinguishing a cited case is stating legally relevant *differences* between the cfs and the cited case, that is, reasons why they should be decided differently. In Hypo, such differences were represented as certain unshared dimensions: in an argument for the plaintiff, dimensions in the cfs, but not in the cited case, that favored plaintiff, and dimensions in the cited case, but not in the cfs, that favored the defendant. These particular unshared dimensions give rise to reasons for deciding the cases differently.

Counterexamples are cases that evidence the same or similar reasons as the cited case for deciding in favor of the side making the argument but where the opposite outcome was reached. Counterexamples make good cases for the opponent to cite in response.

Let's illustrate Hypo's arguments with the facts of a case called *Mason v. Jack Daniels Distillery*, 518 So. 2d 130 (Ala. Civ. App. 1987). In 1980 Tony Mason, a restaurant owner, developed a recipe to ease a sore throat: Jack Daniel's whiskey, Triple Sec, sweet and sour mix, and 7-Up. He promoted the drink, dubbed "Lynchburg Lemonade" for his restaurant, "Tony Mason's, Huntsville," served it in Mason jars, and sold T-shirts. Mason told the recipe only to his bartenders and instructed them not to reveal the recipe to others. The drink was only mixed out of the customers' view. The drink comprised about one-third of the sales of alcoholic drinks. Despite its extreme popularity, no other establishments had duplicated the drink, but experts claimed it could easily be duplicated. In 1982, Randle, a sales representative of the Jack Daniel's Distillery, visited Mason's restaurant and drank Lynchburg Lemonade. Mason disclosed part of the recipe to Randle in exchange, Mason claimed, for a promise that Mason and his band would be used in a sales promotion. Randle recalled having been under the impression that Mason's recipe was a "secret formula." Randle informed his superiors of the recipe and the drink's popularity. A year later, the Distillery began using the recipe to promote the drink in a national sales campaign. Mason was not invited to participate in the promotion nor did he receive any other compensation, so he sued the distillery for misappropriating his secret recipe.

An attorney with some knowledge of trade secret law would be able to identify in the *Mason* facts some legal factors that favor the plaintiff and others that favor the defendant. Plaintiff Mason adopted some security measures, F6 Security-Measures (P).[1] Mason was the only restaurant preparing the Lynchburg Lemonade drink, F15 Unique-Product (P). The defendant distillery's sales representative knew that the information Mason provided was confidential, F21 Knew-Info-Confidential (P). On the other hand, Mason disclosed the information about mixing Lynchburg Lemonade in negotiations with the distillery's agent, F1 Disclosure-in-Negotiations (D), and the recipe could be learned by reverse engineering the drink, F16 Info-Reverse-Engineerable (D).

Figure 3.2 shows an example of a 3-Ply Argument that Hypo could generate for the plaintiff in the *Mason* case. Hypo would analogize *Mason* to the pro-defendant *Yokana* case, then respond by distinguishing *Yokana* for the plaintiff and by citing a pro-plaintiff (trumping) counterexample, the *American Precision* case, and finally,

[1] The *Mason* case was introduced in Aleven (1997) as an example for the CATO program, discussed below, to analyze. CATO employed 26 Factors, numbered F1 through F27. (There is no F9.) For convenience, we will refer to Factors (and the corresponding legal factors) by number. See Table 3.1 for a complete list.

rebut by distinguishing the counterexample on behalf of the defendant. A look at Hypo's model of argument will explain how such arguments would be generated.

Hypo's Argument Model

Figure 3.3 shows the cfs (i.e., the *Mason* case), the *Yokana* case decided for the defendant (D), the *American Precision* case won by the plaintiff (P), the legal factors that apply in each case, and the overlap of legal factors across the cases. The intuition underlying Hypo's model of argument is conveyed in the Venn diagram.

As illustrated in Figure 3.3, the cfs shares pro-defendant F16 with the pro-defendant *Yokana* case. In the Hypo model, this leads to an argument that the cfs is relevantly similar to (i.e., shares a citable legal factor with) *Yokana* and should be decided the same way for defendant (see Figure 3.2, top).

The plaintiff Tony Mason could respond, however, in a number of ways. First, he could distinguish the *Yokana* case. It has a pro-defendant legal factor, F10, not shared in the cfs. In other words, there is a reason to decide *Yokana* for defendant that does not apply to the cfs. Similarly, the cfs has pro-plaintiff legal factors, F6, F15, and F21 that are not in the *Yokana* case. Those are reasons to decide the cfs for plaintiff that do not apply in the cited case (see Figure 3.2, middle).

Second, Tony Mason could cite a favorable precedent: In the *American Precision* case, the plaintiff won where pro-plaintiff F21 applied just as in the cfs.

Third, Mason could use the *American Precision* case to *trump* the defendant's argument based on *Yokana*. In *American Precision*, the plaintiff won *despite* the application of pro-defendant F16. The cfs is even more analogous to *American Precision*

FIGURE 3.2. Hypo-style three-ply argument for the *Mason* case (see Ashley, 1990)

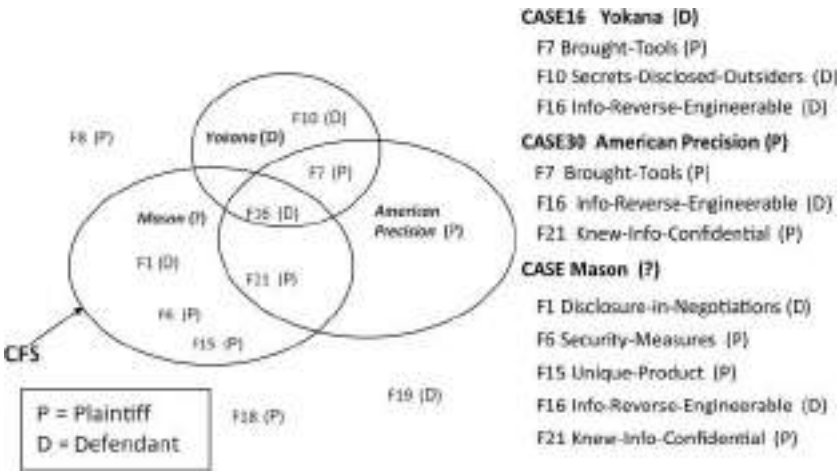


FIGURE 3.3. Hypo argument model with Venn diagram (Ashley, 1990)

because they share a set of factors, F16 and F21; *Yokana* and the cfs share only a subset of that set, namely F16. In other words, *American Precision* is a trumping counterexample to *Yokana* (see Figure 3.2, middle).

More specifically, a counterexample is a case whose outcome is the opposite of the cited case and which satisfies an additional constraint as follows. If the set of legal factors a cited case shares with the cfs is a subset of the set that the counterexample shares with the cfs, the counterexample is more on point than the cited case and is called a *trumping* counterexample. If the counterexample shares the same set of legal factors with the cfs as the cited case, it is an *as-on-point* counterexample. If the counterexample shares a legal factor with the cited case and the cfs, but the magnitude of the legal factor (the corresponding dimension's magnitude) is stronger for the side favored in the cited case, it is a *boundary* counterexample. It tends to undermine a conclusion that the dimension favors that side.

Hypo could make all of these arguments. This example does not illustrate a boundary counterexample, but if the *Mason* case had involved disclosures to outsiders and defendant had relied on *Secrets-Disclosed-Outsiders*, plaintiff could cite the pro-plaintiff *Data General* case with 6,000 disclosees as a boundary counterexample. (Of course, in the rebuttal, Hypo would distinguish the *Data General* case for the defendant by pointing out that there the disclosures to outsiders were restricted.)

Case Retrieval and Ordering in Hypo

Given an input fact situation, Hypo retrieved all cases in its database that shared a dimension with the cfs. It then ordered the cases in terms of the overlaps of the sets of legal factors (as represented by dimensions) the cases shared with the cfs. Hypo organized the cases in a graph structure called a *claim lattice* by the inclusiveness

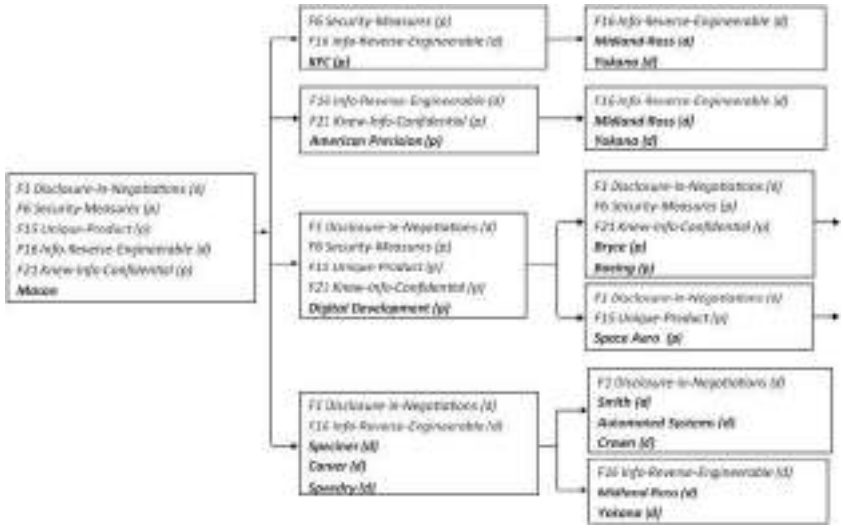


FIGURE 3.4. Hypo claim lattice (Ashley, 1990)

of the sets of dimensions they shared with the problem. Figure 3.4 shows the claim lattice Hypo could construct for the *Mason* cfs. The cfs is at the root. Each of the cfs's immediate descendants shares some subset of its applicable dimensions. Each of their descendants shares some subset of their set of dimensions shared with the cfs and so forth. Notice in the claim lattice that *American Precision* is closer to the cfs than *Yokana* reflecting the trumping counterexample relationship illustrated in Figures 3.2 and 3.3.

The Hypo model illustrates one way to computationally compare cases' similarity and relevance. Hypo does *not* compare cases in terms of the *numbers* of dimensions shared with the cfs. Rather, it compares them in terms of the inclusiveness of the sets of dimensions each case shares with the cfs. In other words, Hypo compares the *sets* of legal factors each case shares with the cfs and determines if one case's set is a subset of another case's set. If it is a subset, the former case is less on point than the latter. In Figure 3.4, for instance, the *Digital Development* case shares four dimensions with the cfs compared with *American Precision*'s two, but that does *not* make it more on point. Also, since *American Precision*'s set of dimensions shared with the cfs is not a subset of *Digital Development*'s the two cases are not comparable according to the Hypo model.

Comparing sets of dimensions makes legal sense. It approximates comparing how well a case covers the legal strengths and weaknesses in a cfs. Comparing cases in terms of the number of dimensions shared ignores the semantic differences among the legal factors.

Two programs extended the Hypo model. CABARET applied dimensions to reasoning with statutory rules and CATO implemented new argument templates for downplaying or emphasizing distinctions.

Dimensions of Legal Rule Predicates in CABARET

CABARET, a first successor to Hypo, dealt with a statutory domain, in particular, a provision of the U.S. IRC dealing with the income tax home office deduction (Rissland and Skalak, 1991). It employed dimensions to represent stereotypical fact patterns that strengthened or weakened a claim that a legal rule's predicate (e.g., "principal place of business" in the tax code provision) was satisfied.

CABARET integrated two models, one rule-based and the other case-based. The rule-based model represented legal rules from the relevant IRS provisions and their ILCs. Given a problem scenario, the rule-based model forward-chained from facts to confirm goals and backward-chained from desired goals to facts needing to be shown.

The rules were similar to those in the Waterman program (Section 1.3.1), but with one major difference. Where the rules "ran out" (i.e., no further rules defined a statutory term), the program could resort to Hypo-style case-based reasoning. Dimensions in CABARET were associated with legal factors strengthening or weakening an argument that a statutory term was satisfied. These dimensions indexed cases in which courts held that the statutory terms were satisfied or not.

Given a problem scenario and a statutory term, the case-based reasoning model determined which dimensions applied, retrieved cases indexed by those dimensions, and generated claim lattices like that illustrated in Figure 3.4 for the statutory term that was subject to argument. The claim lattice organized past cases relevant to that statutory term according to relevance as measured in the Hypo model.

CABARET integrated both computational models via an agenda mechanism: an algorithm that could reason about the current state of the analysis and call either the rule-based reasoning (RBR) or case-based reasoning (CBR) model as appropriate. The agenda mechanism employed a set of heuristic rules to reason about the current state of analysis. Examples of the control heuristics included:

- *Try other*: If CBR fails, then switch to RBR (and vice versa).
- *Sanity check*: Test conclusion of RBR with CBR (and vice versa).
- *RBR Near-miss*: If all a rule's antecedents are established but one, use CBR to broaden application of the rule with respect to the missing antecedent. For example, use CBR to show that there are cases where the conclusion was true but the rule did not fire because of the missing antecedent.
- *Match statutory concepts*: Find cases that failed or succeeded on the same statutory concepts.

Figure 3.5 shows excerpts of CABARET's analysis of a real case, *Weissman v. IRS*, involving whether a CCNY Philosophy professor's home office (two rooms and bath)

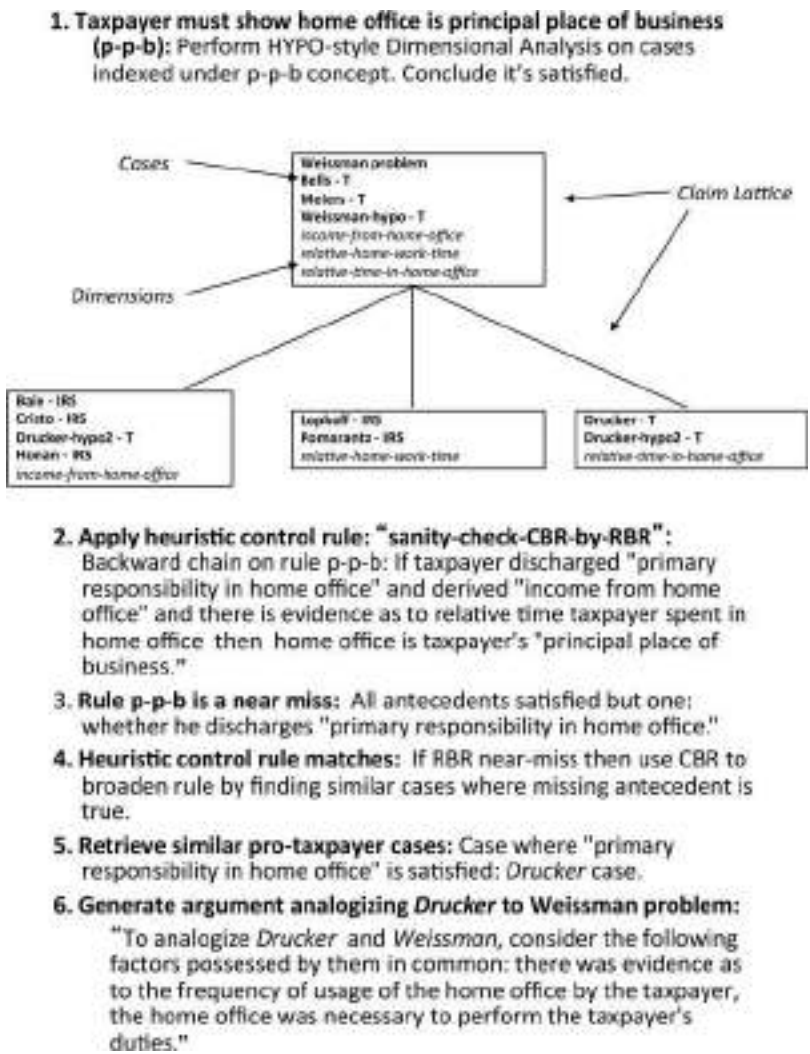


FIGURE 3.5. Example of CABARET's process for analyzing *Weissman v. IRS*, 751 F. 2d 512 (2d Cir. 1984) (Rissland and Skalak, 1991)

in his 10-room apartment qualified for a home office tax deduction under section 280A of the IRC. Professor Weissman spent only 20% of his time at the CCNY office where it was not safe to leave equipment and materials. The IRS challenged his home office deduction of $1,540 rent and expenses because, among other things, it was not his "principal place of business" (p-p-b).

Directed by the control heuristics, CABARET's analysis begins with a case-based dimensional analysis that turns up a number of most-on-point cases citable for the taxpayer. Then, a control heuristic leads to a "sanity check" with a rule-based

analysis, in which the program identifies that the rule for concluding that the home office was his "principal place of business" (p-p-b) *nearly* applied: all of its antecedents were satisfied but one: whether the taxpayer discharges "primary responsibility in home office." Again, a control rule switches to CBR, finding a case where the missing antecedent is satisfied, the *Drucker* case, which it analogizes to the cfs.

CABARET demonstrated that dimensions representing legal factors were useful techniques for modeling a domain beside trade secret law, showed how to use dimensions representing legal factors in a statutory domain, and applied the dimension and legal factor-based approach to reasoning about concepts in legal rules.

Factors in CATO

CATO, Hypo's second successor, simplified the dimensional representation with Factors (Aleven, 2003). Like Hypo, CATO dealt with trade secret misappropriation in terms of legal factors, but it did so *without* using dimensions to represent them.

Instead, it replaced each dimension with a corresponding binary Factor. A Factor either applies to a scenario or it does not. It does not make use of magnitudes or ranges, nor does it have associated prerequisites to test if a Factor applied. CATO employed a more complete list of Factors, shown in Table 3.1, and modeled how to downplay or emphasize a distinguishing Factor (Aleven, 1997). CATO employed its enhanced Factors in a computerized instructional environment to help students learn skills of case-based argument such as distinguishing. As explained in Chapter 4, it also used Factors to predict case outcomes (Aleven, 2003).

CATO added a factor hierarchy, excerpts of which are illustrated in Figure 3.6, a knowledge scheme for representing reasons why the presence of a Factor mattered from a legal viewpoint (Aleven, 2003, Fig. 3, p. 192). The factor hierarchy's reasons explained why a Factor strengthened (or weakened) a trade secret claim.

Using these reasons, CATO could generate new kinds of legal arguments downplaying or emphasizing distinctions, arguments that Hypo could not. It could organize an argument citing multiple cases by issues, grouping together cases that shared common issues with the cfs even if they did not share the same Factors. In this way, CATO could draw analogies at a higher level of abstraction.

CATO could also downplay or emphasize distinctions. As illustrated in Figure 3.7, if a side's argument cites a particular distinguishing Factor in the cfs, the program could downplay it by pointing out another Factor in the cited case that mattered for the same reason. Alternatively, the program could emphasize the distinction by characterizing the difference between the cases more abstractly based on other Factors with common roots in the factor hierarchy.

Aleven's algorithms for downplaying and emphasizing interacted with the information about Factors represented in the factor hierarchy. Given a Factor-based distinction between the cfs and a cited case, it traversed the nodes of the factor hierarchy upward from the distinguishing Factor to identify a focal abstraction that could be used to draw an abstract parallel across the cases and could lead to identifying

TABLE 3.1. *Trade secret Factors (Aleven, 1997)*

Factor	Meaning	Rationale
F1 Disclosure-in-negotiations (D)	P disclosed its product information in negotiations with D.	P gave his property away.
F2 Bribe-employee (P)	D paid P's former employee to switch employment, apparently in an attempt to induce the employee to bring P's information.	D obtained P's property through improper means.
F3 Employee-sole-developer (D)	Employee D was the sole developer of P's product.	D should have property rights in his invention.
F4 Agreed-not-to-disclose (P)	D entered into a nondisclosure agreement with P.	P takes reasonable steps to protect his property.
F5 Agreement-not-specific (D)	The nondisclosure agreement did not specify which information was to be treated as confidential.	P did not specify in what he claims a property interest.
F6 Security-measures (P)	P adopted security measures.	P takes reasonable steps to protect his property.
F7 Brought-tools (P)	P's former employee brought product development information to D.	D steals P's property.
F8 Competitive-advantage (P)	D's access to P's product information saved it time or expense.	P's trade secret is valuable property.
F10 Secrets-disclosed-outsiders (D)	P disclosed its product information to outsiders.	P gave his property away.
F11 Vertical-knowledge (D)	P's information is about customers and suppliers (which means that it may be available independently from customers or even in directories).	P cannot have a property interest in its customer's business info.
F12 Outsider-disclosures- restricted (P)	P's disclosures to outsiders were subject to confidentiality restrictions.	P protects his property.
F13 Noncompetition- agreement (P)	P and D entered into a noncompetition agreement.	P protected against former employee's use of confidential information.
F14 Restricted-materials-used (P)	D used materials that were subject to confidentiality restrictions.	D used P's property despite P's protections.
F15 Unique-product (P)	P was the only manufacturer making the product.	P's trade secret is valuable property.
F16 Info-reverse-engineerable (D)	P's product information could be learned by reverse-engineering.	P's property interest is limited in time.
F17 Info-independently-generated (D)	D developed its product by independent research.	P has no property interest in information D generated independently.
F18 Identical-products (P)	D's product was identical to P's.	D copied P's trade secret property.
F19 No-security-measures (D)	P did not adopt any security measures.	P did not protect his property.
F20 Info-known-to-competitors (D)	P's information was known to competitors.	P cannot have property interest in something known.
F21 Knew-info-confidential (P)	D knew that P's information was confidential.	D knew p claimed property interest.
F22 Invasive-techniques (P)	D used invasive techniques to gain access to P's information.	D used invasive techniques to steal P's property.
F23 Waiver-of-confidentiality (D)	P entered into an agreement waiving confidentiality.	P claimed no property interest in trade secret.
F24 Info-obtainable-elsewhere (D)	The information could be obtained from publicly available sources.	P cannot have property interest in something available from public sources.
F25 Info-reverse-engineered (D)	D discovered P's information through reverse engineering.	P's property interest is limited by time.
F26 Deception (P)	D obtained P's information through deception.	P was cheated of his property
F27 Disclosure-in-public-forum (D)	P disclosed its information in a public forum.	P gave his property interest in the trade secret away.

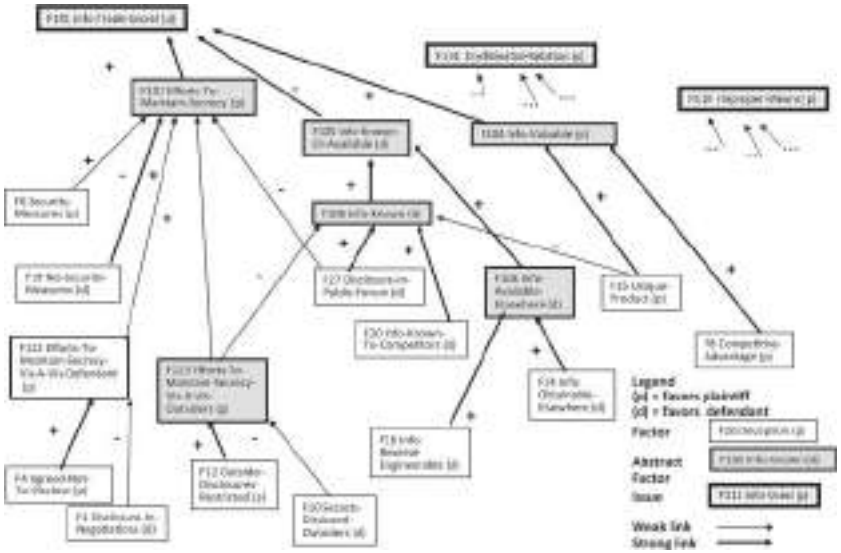


FIGURE 3.6. CATO Factor hierarchy (Aleven, 1997, 2003)

Arguments about the significance of distinction F16

⇒ **Plaintiff's argument downplaying distinction F16 in *Mason*.** In *Mason*, plaintiff's product information could be learned by reverse-engineering [F16]. This was not so in *Bryce*. However, this is not a significant distinction. In *Bryce*, plaintiff disclosed its product information in negotiations with defendant [F1], yet plaintiff won. In both cases, therefore, defendant obtained or could have obtained its information by legitimate means [F120]. But plaintiff may still win.

⇐ **Defendant's argument emphasizing distinction F16 in *Mason*.** In *Mason*, plaintiff's product information could be learned by reverse-engineering [F16]. This was not so in *Bryce*. This distinction is highly significant. It shows that in *Mason*, plaintiff's information was available from sources outside plaintiff's business [F108]. This was not so in *Bryce*.

FIGURE 3.7. CATO argument downplaying/emphasizing distinction (Aleven, 2003)

Factors in the other case that undercut the significance of the distinction. Another algorithm could emphasize a distinction by finding a focal abstraction in the factor hierarchy for abstractly contrasting the two cases. It could lead to identifying further corroborating Factors in one case and contrasting Factors in the other case, with which to support the distinction's importance (Aleven, 2003, pp. 202–8).

Aleven evaluated CATO in two ways. First, he assessed its efficacy in teaching students basic skills of case-based legal argument, as compared to being taught the same skills by an experienced human instructor. Second, he evaluated the argument model in terms of how successfully it predicted outcomes of cases, as discussed in Chapter 4.