Developing Legal Knowledge Based Systems Through Theory Construction

Alison Chorley and Trevor Bench-Capon
Department of Computer Science
The University of Liverpool
Liverpool
UK
040151 794 2697
{alison,tbc}@csc.liv.ac.uk

ABSTRACT
Bench-Capon and Sartor have proposed that reasoning with legal cases be seen as a process of theory construction, evaluation and application. They have proposed a set of theory constructors to specify the process of theory construction. In this paper we describe an implementation of these constructors as part of a system intended to support the development of a legal Knowledge Based System (LKBS) from a set of cases. The constructors provide a means of building a theory from a background analysis. Once a theory has been constructed, the system generates Prolog code conforming to the theory, including the priorities demanded by the theory. This code can then be incorporated into a shell to provide a simple LKBS, which can be used for testing and evaluation, or upgraded into a usable application. The process is illustrated by showing how the tool could be used to develop a LKBS for US Trade Secret Law, drawing on the analysis used in Aleven's CATO system. This case study raises a number of issues which merit discussion and further exploration.

Categories and Subject Descriptors
I.2.1 [**Computing Methodologies**] : Applications and Expert Systems – *law*.

General Terms
Design, Experimentation

Keywords
Legal Knowledge Based Systems, Theory Construction, Development

1. INTRODUCTION
In [2] Bench-Capon and Sartor proposed that reasoning with legal cases be seen as a process of theory construction, evaluation and application. That paper also provides definitions of a set of theory constructors that could provide the means by which theories are constructed given a background describing the domain in terms of factors and cases. In this paper we describe a program which implements these definitions so as to provide a practical theory construction tool. This tool is of use first as a means of rapidly constructing theories of the domain. Its main purpose, however, is to form the centrepiece of a system to support the development of Legal Knowledge Based Systems (LKBS) from cases. So far, development methodologies (e.g [3]), have taken as their starting point written sources, whether statutes or commentaries, so that taking a body of case law as the starting point is novel. To act as a development tool the implementation augments the theory constructors with the ability to generate executable Prolog code conforming to the constructed theory which can be used as the knowledge base of a LKBS. This code enforces all the preferences explicitly made in the theory, and also applies additional rule preferences consequent on established value preferences.

The paper is organised as follows. Section 2 outlines the proposed methodology. Section 3 describes the implementation. Section 4 offers a case study, using the analysis of US Trade Secrets cases used in Aleven's CATO [1]. Section 5 discusses some questions prompted by the case study. Section 6 provides some concluding discussion and proposals for future work.

2. A METHODOLOGY FOR DEVELOPING LKBS FROM CASES

The methodology comprises the following stages:

- domain analysis;
- theory construction;
- code generation;
- evaluation;
- refinement;
- embedding in a system

Each of these stages will be briefly elaborated.

Domain analysis. Inevitably there is no substitute for a thorough reading of the cases and some systematic representation of them. The aim of the domain analysis is to establish what [2] refers to as the background. Two things are required. First we must establish a set of factors which can be used to describe the cases, determine the side which they favour, and associate them with some social value. At this stage one can afford to be fairly profligate with factors: as there is no commitment to use them in the theory or the resulting LKBS it is preferable that any potential factors are made available. This information is recorded in a file *FactorList* with the format:

<factorName,sideFavoured,valuePromoted>

Having established the set of factors, the next step is to use them to describe the set of cases which will be used to build the theory. These case descriptions are recorded in a second text file, *CaseList*, which has the form:

<caseName,setOfFactors,Outcome>

Here again we do not have to be selective: we may represent as many cases as are available. Some will be used in constructing the theory, and the remainder will be available to test the resulting LKBS.

Theory Construction. In this stage the theory constructors described in [2] are used to build a theory, using the theory construction tool described in section 3. Possible approaches to constructing the theory are described in conjunction with the case study in Section 4.

Code Generation. Once the theory has been built, it can be used to generate a set of Prolog clauses. The Prolog code contains each case included as a set of facts of the form

factor(caseName, factorName),

and the rules developed in the theory as a set of clauses of the form

outcome(X,O):-factor(X,f1),factor(X,f2),… , factor(X,fN).

where X is a variable standing for the case, O is one of p or d depending on the outcome suggested by the rule, and f1… fN are the factors in the antecedent of the corresponding rule, and factor(fn,X) is satisfied if factor n is present in case X. Priorities are represented by the order of the clauses for the rules: they are written so as to ensure that the preferred rule always appears before the rule to which it is preferred. This order may not, of course, be unique, but this is because the theory itself does not determine every possible conflict: the order generated is one of the family of theories consistent with the partial theory constructed. This point is discussed further in the next section.

Evaluation. We may now evaluate the theory by executing queries of the form *outcome(Case, Result)*, for those cases which represent our test data. If the answer *Result* corresponds to the actual outcome, then the program handles the case correctly; if not, the program is incorrect with respect to the case, and needs to be refined. Tracing the execution of the query determines which rule produced the incorrect response.

Refinement. We now have a list of rules which led to incorrect results in test cases. For each of these we need to find another rule applicable to the case with the opposite outcome, and assert that this is to be preferred to the problem rule. We can repeat this cycle of evaluation and refinement until we are satisfied with the behaviour of the program. Refinement can be performed either on the theory, or on the program itself.

Embedding in a system: We now have a knowledge base expressed in Prolog which is correct with respect to the test cases. The interface provided by Prolog is, however, likely to be considered unacceptable for normal use as an LKBS. It is therefore necessary to embed the rule base in a more suitable interface to produce the final LKBS.

3. IMPLEMENTATION

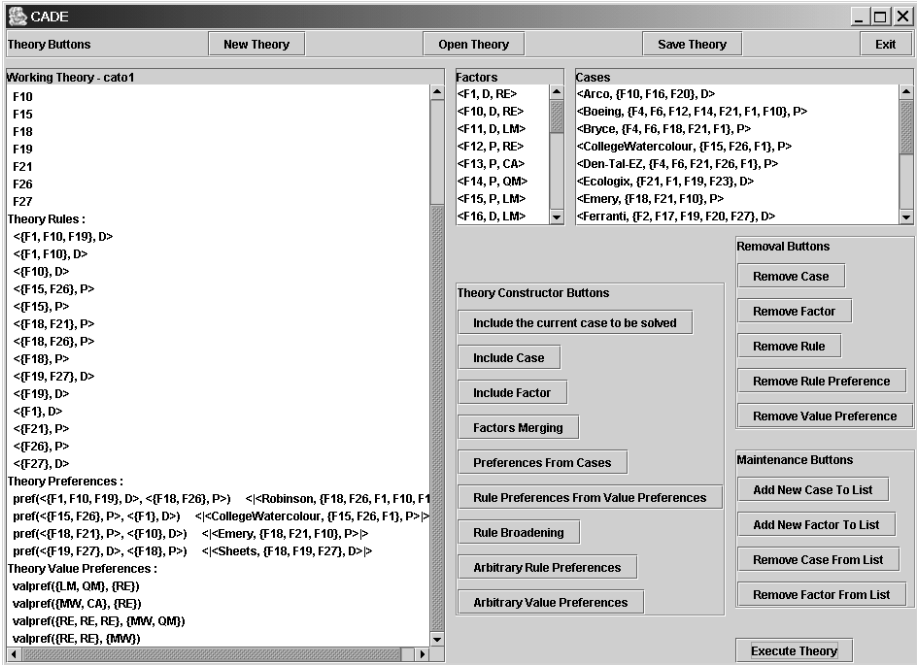

Figure 1 Screen Shot of Theory Construction Tool

3.1 Description of Implementation

Figure 1 gives a screen shot of the theory construction tool in action. The large theory window on the left contains the theory being constructed. It separates the different sections of the theory and shows any changes made to the theory as they are made. The smaller case window at the top right holds the background cases that can be included in the theories. The small factor window shows the background factors available for use in the theories.

The *theory button panel* across the top contains the buttons to start a new theory, open an existing theory, save a currently open theory and exit the tool. The *theory constructor panel* contains the buttons that implement the theory constructors in [2] and also includes an extra button to allow the entry of the case to be solved so that a case already contained in the case background can be given a different outcome. The *removal button panel* contains buttons to remove items from the theory. This enables mistakes to be corrected, and modifications to the theory to be made. The *maintenance button panel* allows changes to be made to the background case and factor lists without amending the underlying text files. Finally the *execute* button generates Prolog code conforming to the theory.

The theory is created by selecting buttons to include items in the theory. The tool checks that the theory is consistent whenever a preference is added. If adding the preference would make the theory inconsistent then a warning is issued and the preference is not added. If the user of the tool still wishes to include the preference then the conflicting preference must be removed first. The tool also tracks where the rule preferences came from by labelling each rule preference depending on which theory constructor was used. When the *Preference From Cases* constructor is used the preference is labelled with the case supporting it and is checked to make sure that the case does indeed support it. The preference is labelled with <| *From Value Preference* |> if the *Rule Preference From Value Preference* constructor is used and the preference is labelled with <| *Arbitrary Rule Preference* |> when the *Arbitrary Rule Preference* constructor is used.

3.2 Description of Code Generation

First the value preferences are converted into the set of corresponding rule preferences. This is done by converting each value in the preference into its associated factors. Only the rules that are contained in the rule section can be used in the rule preferences so only these are created when converting the value preferences. These rules, however, will include both those introduced when a factor is included, and those explicitly constructed by factor merging. The tool checks for any inconsistencies and only if the theory is consistent will the code generation continue.

To illustrate this process consider the following example using the factors and values of the case study in Section 4, which will explain what the various abbreviations stand for. Suppose that our theory contains factors F1, F6, F8, F10, F12, F18 and F27, all of which relate to the values RE and MW, Suppose we have included the specific rule preference to express that F1 is preferred to F8, and hence RE is preferred to MW. This value

```
valpref({RE}, {MW})
    => pref(<{F1}, D>, <{F8}, P>)
    => pref(<{F1}, D>, <{F18}, P>)
    => pref(<{F6}, D>, <{F8}, P>)
    => pref(<{F6}, D>, <{F18}, P>)
    => pref(<{F10}, D>, <{F8}, P>)
    => pref(<{F10}, D>, <{F18}, P>)
    => pref(<{F12}, D>, <{F8}, P>)
    => pref(<{F12}, D>, <{F18}, P>)
    => pref(<{F19}, D>, <{F8}, P>)
    => pref(<{F19}, D>, <{F18}, P>)
    => pref(<{F27}, D>, <{F1}, P>)
    => pref(<{F27}, D>, <{F18}, P>)
```

Figure 2: Automatically generated rule preferences

preference will give rise to the rule preferences shown in Figure 2.. All of these could have been explicitly included using *Rule Preference From Value Preference*, but this is unnecessarily tedious.

Now the rule preferences are used to order the rules in the theory. The rules are held in a list and this list is compared to the rule preferences. If the preferred rule is below the unpreferred rule then the unpreferred rule is moved below the preferred rule. The list is thus guaranteed to be ordered according to the rule preferences but since the theory is not complete this is does not determine a unique ordering. Conflicts are resolved using the alphabetical order of the rules contained in the theory.

Finally the code is generated and output to a file that can be executed in a standard Prolog interpreter.

4. A CASE STUDY

We now illustrate the methodology with a case study. For our analysis we draw on the CATO system [1]. CATO uses 26 base level factors, each associated with either the plaintiff (p) or the defendant (d), and we will take these as the starting point for our background. CATO, however, does not make use of values, and so we need to identify a set of values and associate them with the factors.

4.1 Domain Analysis

So what values seem to underlie the factors? First a number of factors relate to confidentiality agreements. Clearly if all trade secret disputes were governed by a specific agreement, the task of deciding them would be much eased. We would therefore expect the law to encourage such agreements to be made. Our first value then is *Confidentiality Agreement*: the side favoured will depend on the nature of the agreement. This value secures five factors:

- F4 Agreed not to disclose (p)
- F5 Agreement not specific (d)
- F13 Noncompetition Agreement (p)
- F21Knew Info Confidential (p)
- F23 Waiver of Confidentiality (d).

Next it seems that the law does not wish to condone lax behaviour, so that it wishes people with secrets to take

reasonable measures to protect them. This gives the second value *Reasonable Efforts*. Making such efforts are encouraged if having made them favours the plaintiff, and having failed to make them favours the defendant. Six factors share this value.

- F1 Disclosure in Negotiations (d)
- F6 Security Measures (p)
- F10 Secrets Disclosed Outsiders (d)
- F12 Outsider Disclosures Restricted (p)
- F19 No Security Measures (d)
- F27 Disclosure in Public Forum (d).

Third the law wishes to encourage competition by legitimate means. Therefore if a person can develop the product using *Legitimate Means*, this should tell their favour. This covers eight factors. Note that one of them is pro-plaintiff; the uniqueness of a product creates a presupposition that it cannot be developed by legitimate means, and so places an extra burden of proof on the defendant.

- F3 Employee Sole Developer (d)
- F11 Vertical Knowledge (d)
- F15 Unique Product (p)
- F16 Info Reverse Engineerable (d)
- F17 Info Independently Generated (d)
- F20 Info Known to Competitors (d)
- F24 Info Obtainable Elsewhere (d)
- F25 Info Reverse Engineered (d)

The reverse of this is that illegal or immoral means should be discouraged. Five factors relate to this value, *Questionable Means*, which always favours the plaintiff:

- F2 Bribe Employee (p)
- F7 Brought Tools (p)
- F14 Restricted Material Used (p)
- F22 Invasive Techniques (p)
- F26 Deception (p)

The final two factors are intended to show that the secret had *Material Worth*. The law would naturally attempt to discourage litigation about secrets of no worth, and so will favour the plaintiff if his secret had demonstrable value. Two factors, both of which favour the plaintiff, are used here:

- F8 Competitive Advantage (p)
- F18 Identical Products (p).

We have now assigned the factors to five values. Conveniently the distribution is reasonably equal, with only *Material Worth* represented by substantially fewer factors.

We must now select a set of cases. A number are described in [1], and here we present a fairly arbitrary selection, although excluding some of the cases flagged as problematic by Aleven. Our selection includes seven found for the plaintiff, seven found for the defendant, and two which are discussed as "undecided" cases. In Table 1 we show the case, the factors present, split according to whether they favour the plaintiff or the defendant, the values that would be promoted by deciding in favour of the plaintiff and the values that would be promoted by finding for the defendant, as well, as the outcome where given.

Table 1: Cases Used

	Pro-P Factors	Pro-D Factors	Pro-P Values	Pro-D Values	Outcome
Bryce	4 6 18 21	1	CA RE MW	RE	P
Television	6 12 15 18 21	10 16	CA RE LM MW	RE LM	P
Space Aero	8 15 18	1 19	MW LM	RE	P
Den-Tal-Ez	4 6 21 26	1	CA RE QM	RE	P
College Watercolour	15 26	1	LM QM	RE	P
Boeing	4 6 12 14 21	1 10	CA RE QM	RE	P
Emery	18 21	10	CA MW	RE	P
Yokana	7	10 16 27	QM	RE LM	D
Robinson	18 26	1 10 19	QM MW	RE	D
Ferranti	2	17 19 20 27	QM	RE LM	D
Arco		10 16 20		RE LM	D
Sheets	18	19 27	MW	RE	D
Ecologix	21	1 19 23	CA	CA RE	D
Sandlin		1 10 16 19 27		RE LM	D
Mason	6 15 21	1 16	CA RE LM	RE LM	
National Rejectors	7 15 18	10 16 19 27	QM LM MW	RE LM	

4.2 Theory Construction

Having established our background, we can proceed to construct a theory. In previous work, theory construction has typically been directed towards a particular, as yet undecided, case. Here, however, we are free to choose our cases. If we are to have a methodology, we must have some principles for how we will construct our theory. Here we will consider three approaches.

First we may aim at a safe theory. Here we are willing to include as many factors as possible, and to produce rules which do not go beyond the minimum that we are entitled to infer. This latter effect is given by using the method of Prakken and Sartor for producing rules from cases given in [4], whereby the conjunction of pro-plaintiff factors gives one rule, the conjunction of pro-defendant factors another, and the priority is determined by the decision. For choosing cases we want to get the most powerful rules, and this will mean those with the

fewest factors favouring the winner and the most factors favouring the loser.

The second method will aim at the simplest theory. Here we will want to restrict the number of factors as far as possible, and will be willing to make assumptions which enable us to produce rules not strictly justified by the cases. Hence we will select a small set of factors which covers all the cases, and choose cases to establish priorities between them.

Our third approach will be value driven. Here we will first reflect on the values and produce a ranking. We will then choose factors to represent these values, and cases to establish the desired value order.

Let us see what results from these three approaches. In each case we will restrict ourselves to four cases to construct the theory, two won by the plaintiff and two won by the defendant. The other cases will then be used in the evaluation and refinement stages.

For the first method we select Emery and College Watercolour as plaintiff cases and Robinson and Sheets as our defendant cases. Representing the rules from these four cases in the manner of [4] yields the following rule and value preferences:

```
Theory Preferences :
  pref(<{F18, F21}, P>,
       <{F10}, D>)
  pref(<{F1,  F10, F19}, D>,
       <{F18, F26}, P>)
  pref(<{F15, F26}, P>,
       <{F1}, D>)
  pref(<{F19, F27}, D>,
       <{F18}, P>)
Theory Value Preferences :
  valpref({MW,CA}, {RE})
  valpref({RE},{MW,QM}
  valpref({LM, QM}, {RE})
  valpref({RE}, {MW})
```

Figure 3: Rule and Value Preferences from "Safe" Method

For the second method we must first select our factors. What we need is a set of factors such that at least one pro-plaintiff factor occurs in every case decided for the plaintiff, and at least on pro-defendant factor occurs in every case decided for the defendant. F21 occurs in 6 of the pro-plaintiff cases, so we choose this together with F15 to handle Space Aero. For our pro-defendant factors F19, F20 and F27 will cover all defendant cases. Now we need express preferences only where we have both a pro-plaintiff and a pro-defendant factor. In only two cases do we have a conflict to resolve: Space Aero and Ecologix, so we express preferences according to the outcomes of these two cases. For this approach, nothing is to be gained by including additional cases, so only these two are used in this theory. We thus get the following rule and value preferences:

```
Theory Preferences :
  pref(<{F15}, P>, <{F19}, D>)
  pref(<{F19}, D>, <{F21}, P>)
Theory Value Preferences :
  valpref({LM}, {RE})
  valpref({RE}, {CA})
```

Figure 4: Preferences for the "Simple" method

For our final approach we must first decide on a value order. We do not need to distinguish between *Questionable Means* and *Material Worth*, since these both always favour the same side. Let us suppose that the most highly rated value is *Confidentiality Agreement*, since if all the dealings were regulated by properly drafted agreements, there would be no problems to decide. Let us rate *Legitimate Means* next: in the absence of a specific agreement, the right to enterprise must be protected. We rate *Reasonable Efforts* third, since people must take some steps to protect themselves. This leaves *Questionable Means* and *Material Value* at the bottom. Is *Material Worth* so unimportant, when surely it a *sine qua non* for an action? Well, it is of little importance here, since while if it is not present the action seems pointless, it does not really cast much light on whether the defendant behaved incorrectly. It does not, in fact, appear in every case. We assume that this is because it was accepted by both sides, and so is made explicit only if the matter is raised in an effort to discredit the action.

In order to establish this order on values we need four cases. In choosing representative factors we should have an eye mainly to coverage. First we choose a case where CA > LM. Televation, and F21 and F16 can play this role. For LM > RE we choose Space Aero and F15 and F19. We now need RE > QM, for which we can have Robinson with F19 and F26. Finally for RE > MW we choose Sheets with F19 and F18. This yields the third theory.

```
Theory Preferences :
  pref(<{F21}, P>, <{F16}, D>)
  pref(<{F15}, P>, <{F19}, D>)
  pref(<{F19}, D>, <{F26}, P>)
  pref(<{F19}, D>, <{F18}, P>)
Theory Value Preferences :
  valpref({CA}, {LM})
  valpref({LM}, {RE})
  valpref({RE}, {LM})
  valpref({RE}, {MW})
```

Figure 5: Preferences from the "Value Driven" Method

Having obtained our first cut theories we can proceed to evaluate and refine them.

4.3 Evaluation

We now generate Prolog code for each of the three theories. One choice we need to make is whether we should augment the theory by including factors not explicitly used in the construction of the theory. On the one hand, unused factors might have been held to be irrelevant by the person constructing

the theory. On the other, they were considered relevant by the original analyst and users might feel uncomfortable if the system were giving them no consideration. Accordingly we produced two tests for each theory, one using only factors explicitly included in the theory and the other using all 26 factors.

We evaluate our theories by running them against the cases not used in their construction. The results are tabulated in Table 2.

Table 2: Results from Executing Programs

Case	T1		T2		T3	
	sel	all	sel	all	sel	all
Bryce	d	d	p	p	p	p
Televation	p	p	p	p	p	p
Space Aero	d	d	p	p	p	p
Den-Tal-Ez	d	d	p	p	p	p
College Wat	p	p	p	p	p	p
Boeing	d	p	p	p	p	p
Emery	p	p	p	p	p	p
Yokana	d	d	d	d	d	d
Robinson	d	d	d	d	d	d
Ferranti	d	d	d	d	d	d
Arco	d	d	d	d	d	d
Sheets	d	d	d	d	d	d
Ecologix	d	d	d	d	d	p
Sandlin	d	d	d	d	d	d
Mason	d	d	p	p	p	p
National Rej	d	d	p	p	p	d

First we can note that using some or all the factors makes little difference, although it does get Boeing right in Theory 1 and changes the decision in National Rejectors in Theory 3. Since the performance in the test cases is better in one instance, and never worse, we will do our refinement on the code generated from the augmented theories.

4.4 Refinement

In order to refine the programs, we first execute the program for the incorrect case to identify the rule causing the problem. We then reorder the clauses to fix the problem. We then re-run the test to ensure that no new problems have been created, and to see what cases remain incorrect.

To refine Theory 1 we begin with Bryce, the first case showing a problem. Here we find that F1, associated with the value *Reasonable Efforts* was the decisive factor in reaching the wrong decision. So let us examine the code for outcome, shown if Figure 6.

```
outcome(X, d) :- factor(X, f1), factor(X,
f10), factor(X, f19).
outcome(X, p) :- factor(X, f18), factor(X,
f26).
outcome(X, p) :- factor(X, f12).
outcome(X, p) :- factor(X, f15), factor(X,
f26).
* outcome(X, d) :- factor(X, f1).
outcome(X, p) :- factor(X, f18), factor(X,
f21).
outcome(X, d) :- factor(X, f19), factor(X,
f27).
outcome(X, d) :- factor(X, f10).
outcome(X, d) :- factor(X, f19).
outcome(X, d) :- factor(X, f27).
outcome(X, p) :- factor(X, f6).
***
outcome(X, p) :- factor(X, f8).
outcome(X, p) :- factor(X, f18).
outcome(X, d) :- factor(X, f1), factor(X,
f10).
outcome(X, d) :- factor(X, f11).
outcome(X, p) :- factor(X, f13).
outcome(X, p) :- factor(X, f14).
outcome(X, p) :- factor(X, f15).
outcome(X, d) :- factor(X, f16).
outcome(X, d) :- factor(X, f17).
outcome(X, d) :- factor(X, f20).
outcome(X, p) :- factor(X, f21).
outcome(X, p) :- factor(X, f22).
outcome(X, d) :- factor(X, f23).
outcome(X, d) :- factor(X, f24).
outcome(X, d) :- factor(X, f25).
outcome(X, p) :- factor(X, f26).
outcome(X, p) :- factor(X, f2).
outcome(X, d) :- factor(X, f3).
outcome(X, p) :- factor(X, f4).
outcome(X, d) :- factor(X, f5).
outcome(X, p) :- factor(X, f7).
```

Figure 6: Code for Theory 1

The clauses relating to the value *Reasonable Efforts* are emboldened, and those explicitly preferred in the theory are also italicised. Recall that the order of clauses with the same value which do not feature in explicit rule preferences is determined by the program, not the theory. We may therefore move the problem clause (marked by "*") to be the last clause for *Reasonable Efforts*, the position marked by "***". Rerunning the program shows that this has dealt with Bryce, and also fixed Den-Tal-Ez. It has also, incidentally, changed the outcome for Mason. Only Space Aero now has the wrong outcome. The key factor here is F19. But moving this below the other factors for *Reasonable Efforts* will not succeed here, since in Space Aero, unlike Bryce, no security measures (F6) were taken, and so no clause for a factor associated with *Reasonable Efforts* will give a decision for the plaintiff. Our choice is therefore either to move the clause for F19 somewhere below that for F18 (the highest factor satisfied in Space Aero), or to amend the theory by explicitly recording the decision in Space Aero. The latter seems

most thematic for the approach of Theory 1. The resulting theory, with the extra value preference *valpref({MW,LM},{RE})*, correctly decides the test cases.

When we wish to refine Theory 3, we need to handle Ecologix. In order to get the correct outcome for this case, we need to ensure that F21 is given less weight that one of F1, F19, and F23. In order to be consistent with the value order that we used to produce this theory, this would require that F23 was preferred to F21. Accordingly – and given that F23 represents an explicit waiver of confidentiality this seems reasonable – we make the clause for F23 the highest relating to confidentiality. Re-running the resulting program gives the correct answer for all cases.

Note that with the exception of Space Aero which required special treatment, problems occur when we have both pro-plaintiff and pro-defendant factors relating to the same value. Because we do not explicitly determine the order of factors *within* a value, we have to hope the program gets it right by chance, since otherwise we will have to tinker with the program. Of course, a useful addition to the method might be a declaration of the order of, at least some, factors within a value, which would then be used to govern the order of clauses in the resulting program. Space Aero is a different case: it demonstrates the need to include an additional preference between combinations of values in order to explain the cases. Note that in this case we modified *the theory*, rather than choosing a different program conforming to the original theory.

4.5 Embedding in a system

We will not say much about this. The very simplest system would be something like:

```
go:-nl, write([input,case,name]),
    nl,read(C),
     write([input,factor,list]),
     nl,read(F),
     asserta(flist(C,F)),outcome(C,O),
     nl,write(O).

factor(C,F):-flist(C,L),member(F,L).
```

This program does no more than read a case name and a list of factors, apply the rules to this data and display the result. Such a LKBS has a very (early) eighties feel, but is sufficient to demonstrate the principle.

5. Discussion of Case Study

The case study raises a number of questions. We will look at the following:

- Can we say anything about which method of theory construction is the best?
- What is the relation between values and abstract factors as used in CATO?
- Can we say anything about how the "undecided cases" should be decided?
- Do we wish to consider degrees of strength of factors?
- Do we wish to consider any cumulative impact for several factors relating to the same value?

5.1 How should we construct theories?

From the case study we saw that Theory 1 (the safe method) required the most refinement, while Theory 2 (the bold method) required least. We should not, however, conclude from this that the bold method is the best. The reason that it works so well is that it is highly tailored to the data presented. The situation is possibly akin to rules induced from data by techniques such as rule induction and neural nets, which can often give good performance, but which have a tendency to overfit the dataset, and thus to lack robustness. For example if the defendant was an employee who was the sole developer of the product (F3) but had entered into a specific agreement not to disclose (F4), and the plaintiff had taken other security measures (F6), the program derived from Theory 2 would find for the defendant, whereas we might well expect the plaintiff to win such a case, an outcome produced by the other two theories. The problem here is that our selection of cases is silent on F3 and F4, and so their impact may be misinterpreted unless the importance of F4 is recognised via its associated value. We should therefore be wary about using the approach of Theory 2.

In comparing Theories 1 and 3, we may note the following. If comparisons are restricted to one value against another, Theory 3 will supply the correct answers, although we may need to refine the theory by deciding between factors relating to the same value. In contrast, Theory 1 is strong on combinations of values, but leaves many individual comparisons unresolved. Which is best therefore depends on how we believe sets of values should be compared. What Theory 3 suggests is that we always value a set by its most important member, and when the sets being compared contain the same best member, these are the factors which need to be compared. An alternative, involving a small complication to the code generated[1], would allow cancelling of shared values, so that sets of values would be compared on the basis of their best value not present in both sets. Theory 1 offers yet another way of comparing value sets, so that v1 and v2 may together outweigh v3, even though v3 outweighs both v1 and v2 individually. Which method of constructing theories is required therefore depends on how we wish sets of values to be compared. This, we believe, is not a question capable of a general answer, but which needs to be thought about relative to the domain of application. In the domain studied in the case example, we feel that the method produced by Theory 3 is not unreasonable.

5.2 Values and Abstract Factors

In Prakken [5], there was an interesting suggestion that the argument moves of emphasising and downplaying distinctions between cases, which motivates the use of abstract factors in CATO, could be generated using values. We might therefore expect some correlation between abstract factors and values.

In determining the values to be used for factors, we made use only of the description of factors given in Appendix 2 of [1]. If, however, we look at the Factor Hierarchy, we can find

[1] This would involve replacing each factor fa with the conjunction fa and not f1 and … and not fn, where f1 … fn are the factors which relate to the same value as fa, but which favour the opposite side.

considerable similarities to our values. We can associate the following abstract factors with our values:

Table 3: Values and Abstract Factors

F102 Efforts to Maintain Secrecy	Reasonable Efforts
F111 Questionable Means	Questionable Means
F105 Info Known or Available	Legitimate Means
F114 Confidential Relationship	Confidentiality Agreement
F104 Info Valuable	Material Worth
F112 Info Used	Material Worth
F124 Defendant Ownership	Legitimate Means

In most cases the value we assigned to a factor corresponds to the related abstract factor, although we always assigned factors to a single value, whereas CATO associates several factors with two or more abstract factors. Note also that we have conflated two abstract factors into a single value in two cases. There are, however, some differences: we assigned F7, *Brought-tools*, to *Questionable Means*, whereas CATO relates it to Info-Used. This factor appears only in Yokana, and reassigning it does not alter that case, or substantially change the theory. More interesting is *Legitimate Means*. Two factors we assigned to *Legitimate Means*, F17, *Info Independently Generated* and F25 *Info Reverse Engineered*, are related to *Questionable Means* in CATO (favouring the defendant by showing that the means were not questionable). F25 does not occur in our cases, but F27 occurred in Ferranti, where the pro-plaintiff factor was F2 *Bribe Employee*, and so it was likely that it was used in that case to show that the means were not questionable. Even so, on a literal reading of the factor names, they suggest to us that they are better seen as indicating that the information was legitimately obtained. In CATO, F105 and F111 come together under the more abstract factor *Information Legitimately Obtained or Obtainable.*

Thus, although there is some scope for differing interpretations, it does seem that abstract factors and values are quite highly correlated.

5.3 Mason and National Rejectors

As can be seen from Table 2, the various LKBS give varying answers for these "undecided" cases. Before refinement Theory 1 decided both for the defendant, explaining Mason through F1 and National Rejectors through F27. After refinement, however, Mason is decided for the plaintiff on the basis of F6 and now National Rejectors is decided on the basis of F19. Not too much stress should be placed in the change of explanation for National Rejectors; both F19 and F27 relate to the same value, and so their ordering is of no real significance. The precedence of F6 over F1 which reverses the Mason decision was introduced as a refinement motivated by Bryce.

Theory 2 consistently decides both cases for the plaintiff on the basis of F15, the uniqueness of the product. Theory 2 is based on the idea that *Legitimate Means* is the most important value: but the use of F15 rather than F16, which appears in the very next clause, is again determined by the way the code is generated and is merely consistent with rather than determined by the theory. Since if these clauses were reversed, both cases would be decided for the defendant, it is difficult to place much confidence in these decisions.

Theory 3 decides Mason for the plaintiff using F21, and National Rejectors for the defendant using F16. That F16 appears before F15, which is what leads to the decision for the plaintiff, is again a matter of chance.

Thus, although the performance of the three LKBSs is entirely similar for the cases used to produce them, the sharp differences in the various underlying theories are revealed when we turn to the "new" cases. Theory 1 focuses on the factors relating to the security measures, whereas theory 2 focuses on the legitimacy of the acquisition of the information. Theory 3 first relies on the confidentiality of the relationship, before turning to consider the legitimacy of the acquisition of the information. Most threatening to the overall approach, however, is that may of the decisions turn on the precise ordering of the clauses for factors relating to the same value, which is a by-product of the automated generation of rules. This could be avoided if we determined the order of such clauses by a prior ranking of the factors within each value. This in turn suggests that we ascribe different strengths to factors. We consider this in the next subsection.

5.4 Strengths and Cumulative Impact

In the original theory of [2] all factors were supposed to relate to values with equal strength, and the presence of a value was considered to have the same weight, no matter how many factors drove it. Do we wish to stay with this view?

The considerations in 5.3 strongly suggest that we do need to order factors within a value: otherwise we cannot determine the outcome in a case where both the plaintiff and the defendants have factors relating to a particular value. This, however, is only necessary when a value has factors favouring both sides, and so we need not worry about *Material Worth* and *Questionable Means*.

Obviously it may prove to be a difficult and debateable task to form a total order over the factors for a value. We could, however, propose a general principle. In our case study the values all seem to lean towards one of the parties (just as abstract factors favour one of the parties). Thus *Confidentiality Agreement*, *Questionable Means* and *Material Worth* seem natural plaintiff values, and *Reasonable Efforts* and *Legitimate Means* seem natural defendant values. *Reasonable Efforts* is the most debateable here, but we could resolve any doubts by saying that a value favours the party favoured by the greater number of factors within it. But if we look at the factors favouring the other party, they seem, in general, to represent exceptions that should be considered before the more typical factors. Suppose then we designed our knowledge bases so that these exceptional factors always appeared first. Now, for *Reasonable Efforts*, F6 would appear before F1 and so the refinement of Theory 1 occasioned by Bryce would not have been needed. Also F23 would be favoured over F21, rendering the refinement of Theory 3 to accommodate Ecologix unnecessary. Refinement of Theory 1 to satisfy Space Aero would still have been needed, but note that this was done as a change to the *theory*, not the *program*. Of course, it is quite possible that this is an over generalisation of

the case example, but it appears to us to be an idea worth testing further. Certainly, when applied to the case study it is effective.

Thus we feel that identifying factors as exceptions to the prevailing trend of a value, and prioritising them, may give sufficient control to the order of factors in values without needing to resort to notion of assigning different strengths to factors.

Turning to whether we wish to see factors as having a cumulative effect when promoting a value, we can see the preference expressed in Robinson as a test case. Here three *Reasonable Efforts* factors were held to outweigh *Questionable Means* and *Material Worth*. Would we need to see Robinson as weakened were two of the *Reasonable Efforts* factors absent? We feel not, as either F10 or F27 would probably have served on their own. F1, however, would not. This need not, however, convince us that a cumulative effect is essential: F1 in CATO is related to two abstract factors, *Efforts to Maintain Security* and *Questionable Means*. It is possible that we have chosen the wrong value to which to assign the factor.

6. CONCLUDING REMARKS

In this paper we have proposed a methodology for constructing an LKBS from a set of cases, through the construction of a theory to explain those cases. We have described a tool which supports this process and some experiments on a case study to show how it works in practice.

The case study pointed to three things in particular

- It is possible to construct a variety of quite different theories to explain a given set of cases: the resulting LKBSs may well give different results in "new" cases;
- Typically a theory under-determines the LKBS, especially with regard to factors relating to the same value, but favouring different parties to the dispute. We have tentatively proposed a solution to this problem;
- Given a suitable analysis of the cases to provide a background it is a straightforward matter to develop theories and generate LKBS. Potentially this enables considerable scope for rapid prototyping and experimentation to test and refine theories.

For future work, we wish to refine the tool to improve its usability, to incorporate the refinement to the code generation outlined here, and to provide a more integrated package. Secondly we wish to explore the scope for the automation of the process. Following the methodology of Theory 3, it would be possible to automatically generate theories corresponding to every possible ordering of values. We presume that only a few such theories would perform acceptably, and these could then be presented for consideration and refinement. Finally we need to examine more domains, to determine the extent to which the conclusions we have drawn from this single case study are generally applicable.

ACKNOWLEDGEMENT

Alison Chorley is supported by the EPSRC funded Doctoral Training Account of the Department of Computer Science at the University of Liverpool.

REFERENCES

[1] Aleven, V. (1997). *Teaching Case Based Argumentation Through an Example and Models.* PhD Thesis. The University of Pittsburgh.

[2] Bench-Capon, T.J.M., and G. Sartor (2001). Theory Based Explanation of Case Law Domains. In *Proceedings of the Eighth International Conference on AI and Law*, 12-21. ACM Press: New York.

[3] van Kralingen, R., Visser, P.R.S., Bench-Capon, T.J.M., and van der Herik, J., (1999). *A Principled Methodology for the Development of Legal Knowledge Systems*, International Journal of Human Computer Studies, (51) pp 1127-54.

[4] Prakken, H., and G. Sartor 1998. Modelling Reasoning with Precedents in a Formal Dialogue Game. *Artificial Intelligence and Law* 6: 231-287.

[5] Prakken, H., 2000. An Exercise in Formalising Teleological Reasoning, in Breuker, J., Leenes, R., and Winkels, R., (eds), Proceedings of JURIX 2000, IOS Press, Amsterdam, pp49-58.