Support for Constructing Theories in Case Law Domains

Alison Chorley and Trevor Bench-Capon

Department of Computer Science, The University of Liverpool, UK
{alison,tbc}@csc.liv.ac.uk

Abstract. Reasoning with cases has been a central focus of work in Artificial Intelligence and Law since the field began in the late eighties. Reasoning with cases is a distinctive feature of legal reasoning and is of interest because such reasoning is both inherently defeasible, and because it is an example of practical reasoning in that it aims to provide a rational basis for a *choice* rather than to deduce some conclusion from premises. As reasoning with cases has developed, it has moved beyond techniques for matching past cases to the current situation to consider how arguments for a position are constructed on the basis of past cases. Recently it has been argued that this should be seen as a process involving the construction, evaluation and application of theories grounded in the phenomena presented by the past cases. Our aim is to develop and refine this idea, with the ultimate goal of building a system which is able to reason with cases in this manner. This paper describes the implementation of a theory construction tool (CATE) to aid in the construction and evaluation of theories to explain the decisions obtained in legal cases, so as to give an understanding of a body of case law. CATE gives a rapid way of creating and testing different theories. Use of CATE is illustrated by showing the construction of alternative theories in a small case study. CATE is useful in itself for anyone wishing to explore their understanding of a set of cases, such as lawyers practising in the domain and knowledge engineers tasked with constructing a rule based system in the domain. We also believe that it offers good prospects for automating the process of theory construction.

1 Introduction

Although there has been some limited success with legal expert systems developed solely on the basis of rules derived from legislation, it is now generally accepted that such systems require knowledge derived from case law if they are to make any real contribution to legal problem solving. Even where there are clear rules, problems of interpretation, under specification and conflict remain. Although cases are used differently in civil and common law jurisdictions this point applies to both styles of legal system. This can be illustrated by [7]: although in their system the Restatement of Torts appear to offer some clear rules, in order to apply these rules to specific cases, the experience of past cases must be drawn upon.

The implication of this is that if one is to understand a piece of law, whether with a view to applying it unaided, or to building a decision support system, it is first necessary to come to an understanding of the relevant case law. Recent work, most fully described in [5], has revived an idea originally proposed by McCarty (eg. [11]) and suggests that coming to this understanding is best seen as represented as the construction of a theory of the domain, developed from, and intended to explain, the phenomena presented by decisions in precedent cases. Once constructed, the theory can be evaluated according to both internal considerations of coherence and by its effectiveness in accounting for the decisions in the precedent cases.

In this paper we describe CATE (CAse Theory Editor), a tool developed to provide support to this process of understanding a legal domain through theory construction. CATE is intended to be useful both to lawyers exploring their understanding of a set of cases, and to knowledge engineers desirous of building an automated system. By providing a means rapidly to develop and execute theories, the task of exploring alternatives and refining initial intuitions is greatly eased.

Section 2 of this paper gives an overview of what we see as a theory of a body of case law, and section 3 describes CATE, which effectively realises the theoretical specification of [5]. Section 4 illustrates the use of CATE through stepping through a well known case study, which although small is sufficiently rich to show many of the important features. Section 5 discusses an extension to CATE which supports a further stage of domain analysis. Section 6 offers some discussion, and considers the prospects for automating the process of constructing legal decision support systems from case data.

2 A Theory of Case Law Domain

Our notion of a theory is taken from [5], which in turn derives from the style of legal case base reasoning developed in the HYPO system and its descendants, e.g. [2], [1] and [7]. The starting point for theory construction is the background of *cases* and the *factors* with which to describe them, which we represent as two files: the *factor background* and the *case background*. Factors are particular patterns of facts which may be present in a case and which if present, will provide a prima facie reason for deciding for one or other of the parties to a case. For example in US Trade Secrets Law, the domain of HYPO, if the plaintiff took security measures his case is strengthened. Thus taking security measures is a factor favouring the plaintiff. Factors are additionally linked to *values*: our account takes a consequentialist view of legal theory, so that we view decisions as justified by the purposes they effect. Here a *value* associated with a factor is some desired purpose, which will be promoted by deciding for the party favoured by the factor when the factor is present. The factor background thus consists of a set of 3-tuples of the form <*factor-name, outcome-favoured, value-promoted*>.

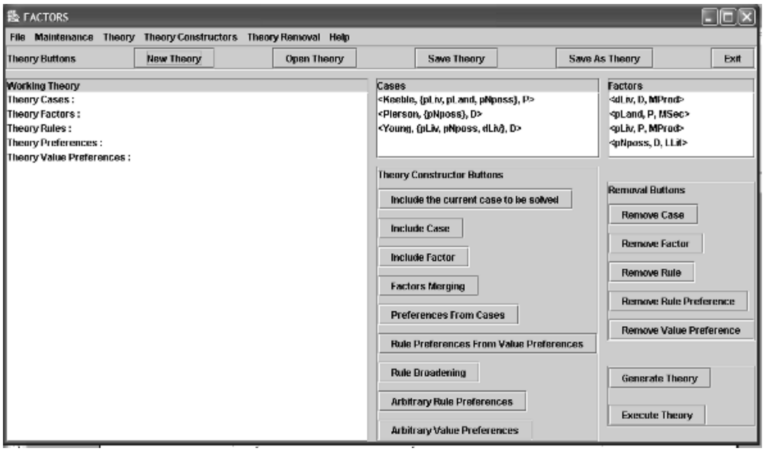
These factors are used to describe the cases which form the case background. Each case will contain a set of factors in the factor background, and as a past case will have an outcome. The case background thus comprises a set of 3-tuples of the form <*case-name, factors-present, outcome*>.

From this background theories are constructed. A theory is a 5-tuple comprising: a selection of cases from the case background; a selection of factors from the factor background; a set of rules linking the factors to outcomes; a set of preferences amongst those rules; and a set of preferences over values promoted by the factors within the rules.

[5] construes the construction of a theory from the background as the application of a number of constructors which enable the inclusion of elements from the background, the formation of rules from included factors and the expression of preferences between rules and between values. It is this process of applying constructors that CATE supports.

3 Description of CATE

Figure 1 gives a screen shot of CATE. It is designed to embody the set of theory constructors as described in [5] and has been implemented in Java. There are panels to show case and factor backgrounds, and the theory under construction. The various theory constructors can be used by clicking on the appropriate button on the screen. For example, to include a case into the theory, the *Include Case* button is selected and the user is prompted to choose which case they want. CATE also provides some checking on the legality of use of the constructors: when a user specifies preferences over rules or values, CATE checks that the resulting theory is consistent and if adding the preference would make the theory inconsistent then a warning is issued and the preference is not added. If the user of CATE still wishes to include the preference then the existing preference causing the conflict must first be removed. CATE also tracks where the rule preferences came from, so that we can distinguish those derived from cases from those derived from a value preference, by labelling each rule preference depending on which theory constructor was used.



Working Theory
Theory Cases :
Theory Factors :
Theory Rules :
Theory Preferences :
Theory Value Preferences :

Cases
<Keeble, {pLiv, pLand, pNposs}, P>
<Pierson, {pNposs}, D>
<Young, {pLiv, pNposs, dLiv}, D>

Factors
<dLiv, D, MProd>
<pLand, P, MSec>
<pLiv, P, MProd>
<pNposs, D, LLit>

Fig. 1. This figure shows CATE before the Theory is constructed

The *Execute Theory* button can be used to generate Prolog code representing the theory which can be executed to give the outcome which results from applying the theory to each case included in the theory. These case outcomes can then be used to evaluate how the theory performs with respect to the actual decisions for the cases, so as to verify that the theory does indeed explain the selected cases.

CATE requires that a domain analysis has already been completed, so as to supply the case and factor backgrounds. CATE is not restricted to any particular domain and so can be used with any domain once the analysis to supply the requisite background has been completed.

4 Wild Animal Case Study

In much of our work we have used as a test domain US Trade Secret law, drawing on the analysis of [2], [1] and [7]. This domain is substantial containing twenty six factors and five values, and up to one hundred and eighty six cases. For the purposes of illustration, however, we will discuss here a small case study. This case study has been discussed in a number of papers in the AI and Law domain, such as [6], [12], [4], and [3], and is widely used as an example for teaching Law students.
The example consists of three cases involving the pursuit of wild animals. In all of those cases, the plaintiff was chasing wild animals, and the defendant interrupted the chase, preventing the plaintiff from capturing those animals. The issue to be decided is whether the plaintiff has a legal remedy (a right to be compensated for the loss of the game) against the defendant or not. In the first case, *Pierson v Post*, the plaintiff was hunting a fox on open land in the traditional manner using horse and hound when the defendant killed and carried off the fox. In this case the plaintiff was held to have no right to the fox because he had gained no possession of it. In the second case, *Keeble v Hickeringill*, the plaintiff owned a pond and made his living by luring wild ducks there with decoys, shooting them, and selling them for food. Out of malice the defendant used guns to scare the ducks away from the pond. Here the plaintiff won. In the third case, *Young v Hitchens*, both parties were commercial fisherman. While the plaintiff was closing his nets, the defendant sped into the gap, spread his own net and caught the fish. In this case the defendant won. The cases are interesting because many people intuitively feel that the final case should have been decided for the plaintiff rather than the defendant, and the challenge is to come up with a convincing rationale of the actual decision.

We must first give a domain analysis. There are several variants in the literature. Here we follow that given in [5]. First we identify four factors:

1. whether the plaintiff had possession of the animal,
2. whether the plaintiff owned the land on which the chase was taking place,
3. whether the plaintiff was engaged in earning his living and
4. whether the defendant was engaged in earning his living.

We abbreviate these factors to *pNposs* (plaintiff had no possession), *pLand* (it was the plaintiff's land), *pLiv* (plaintiff earning his living) and *dLiv* (defendant earning his living.

We can now identify the values associated with these factors. By requiring the plaintiff to be actually in possession of the animal we give a clear line following which will tend to reduce litigation in this area. The first factor thus promotes this value, which we abbreviate as *LLit* (less litigation). The second factor promotes respect for property rights, offering more security (*MSec*). The final two factors will protect economic activity, which should enable more production (*MProd*).

Our initial starting point thus comprises a case background consisting of:

```
<Keeble, {pLiv, pLand, pNposs}, P>
<Pierson, {pNposs}, D>
<Young, {pLiv, pNposs, dLiv}, D>
<Young, {pLiv, pNposs, dLiv}, P>
```

and a factor background consisting of:

```
<dLiv, D, MProd>
<pNposs, D, LLit>
<pLand, P, MSec>
<pLiv, P, MProd>
```

Two versions of the as yet undecided Young case are included in the background, as suggested in [5], so that the outcome being argued for is available for use in the theory.

We now use CATE to construct the theories. Figure 1 shows the initial state. *Theory 1* is constructed for the defendant. First the *Include Case* Constructor is used to include the *Pierson* case and the defendant based *Young* case. Then the *Include Factor* Constructor is used to include the *pNposs* factor in the theory.

The theory produced by CATE is this:

```
Theory Cases :
  <Pierson, {pNposs}, D>
  <Young, {pLiv, pNposs, dLiv}, D>
Theory Factors :
  pNposs
Theory Rules :
  <{pNposs}, D>
Theory Preferences :
Theory Value Preferences :
```

This theory can be used to generate Prolog code which is executed to produce the outcome for the cases according to the theory. We show the case, the outcome and the decisive rule.

```
pierson |  d  |  outcome(X, d) :- factor(X, pnposs).
young   |  d  |  outcome(X, d) :- factor(X, pnposs).
```

Both cases have been decided for the defendant as there was only a defendant factor present in the theory. This theory adopts a very straightforward approach: considering only a single factor, possession being seen as the whole of the law. *Theory 2* is next constructed for the plaintiff. It extends *Theory 1* by including the *Keeble* case using the *Include Case* Constructor and then the *Include Factor* Constructor is used to include the *pLiv* factor. This gives us two conflicting rules applying to *Keeble*. We know, however, how this conflict was resolved: *Keeble* was found for the plaintiff. We can therefore use this case to give a rule preference which prefers *pLiv* factor over the *pNposs* factor. This rule preference also generates a value preference which is also included in the theory.

```
Theory Cases :
  <Young, {pLiv, pNposs, dLiv}, P>
  <Pierson, {pNposs}, D>
  <Keeble, {pLiv, pLand, pNposs}, P>
Theory Factors :
  pLiv
  pNposs
Theory Rules :
  <{pLiv}, P>
  <{pNposs}, D>
Theory Preferences :
  pref(<{pLiv}, P>, <{pNposs}, D>)     <|<Keeble, {pLiv,
pLand, pNposs}, P>|>
Theory Value Preferences :
  valpref({MProd}, {LLit})
```

The theory can be executed to produce the following outcomes for the cases. *Young* and *Keeble* have been decided for the plaintiff because the plaintiff factor is preferred over the defendant factor.

```
pierson |  d  |  outcome(X, d) :- factor(X, pnposs).
keeble  |  p  |  outcome(X, p) :- factor(X, pliv).
young   |  p  |  outcome(X, p) :- factor(X, pliv).
```

Theory 3 is constructed for the defendant and adds *pLand* to *Theory 2*. This factor is merged with *pLiv* to produce a rule with *(pLand, pLiv)* as antecedent. The preference in *Keeble* can now be explained in terms of this rule, giving the rule preference of *(pLand, pLiv)* over *pNposs* instead of *pLiv* over *pNposs* as in *Theory 2*. However, this will not explain how *Young* should be decided. To do this we need to include an arbitrary rule preference of *pNposs* preferred over *pLiv*.

```
Theory Cases :
  <Pierson, {pNposs}, D>
  <Keeble, {pLiv, pLand, pNposs}, P>
  <Young, {pLiv, pNposs, dLiv}, D>
Theory Factors :
  pLand
  pLiv
  pNposs
Theory Rules :
  <{pLand, pLiv}, P>
  <{pLand}, P>
  <{pLiv}, P>
  <{pNposs}, D>
Theory Preferences :
  pref(<{pLand, pLiv}, P>, <{pNposs}, D>)
<|<Keeble, {pLiv, pLand, pNposs}, P>|>
  pref(<{pNposs}, D>, <{pLiv}, P>)     <|Arbitrary Rule
Preference|>
Theory Value Preferences :
  valpref({LLit}, {MProd})
  valpref({MProd, MSec}, {LLit})
```

This theory decides *Young* in the way we wanted but has resorted to an arbitrary rule preference, which is not desirable.

```
  pierson |  d  |  outcome(X, d) :- factor(X, pnposs).
  keeble  |  p  |  outcome(X, p) :- factor(X, pland), factor(X,
  pliv).
  young   |  d  |  outcome(X, d) :- factor(X, pnposs).
```

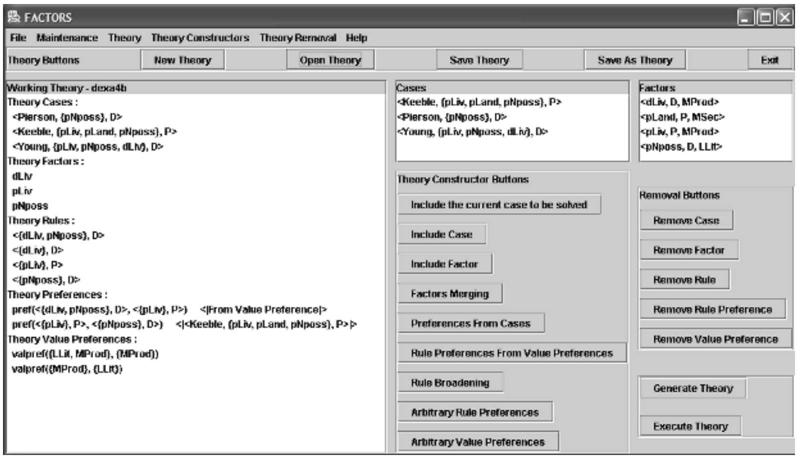
An alternative to *Theory 3* is *Theory 4* which is constructed by including *dLiv* instead of *pLand* and merging it with the *pNposs* factor to give a rule with antecedent *(dLiv, pNposs)*. We now add the value preference of *(LLit, MProd)* over *MProd* (which seems justifiable as the preferred value is a superset of the less preferred value) and from this we derive the rule preference of *(dLiv, pNposs)* over *pLiv*.

```
  Theory Cases :
    <Pierson, {pNposs}, D>
    <Keeble, {pLiv, pLand, pNposs}, P>
    <Young, {pLiv, pNposs, dLiv}, D>
  Theory Factors :
    dLiv
    pLiv
    pNposs
  Theory Rules :
    <{dLiv, pNposs}, D>
    <{dLiv}, D>
    <{pLiv}, P>
    <{pNposs}, D>
  Theory Preferences :
    pref(<{dLiv, pNposs}, D>, <{pLiv}, P>)      <|From Value
  Preference|>
    pref(<{pLiv}, P>, <{pNposs}, D>)      <|<Keeble, {pLiv,
  pLand, pNposs}, P>|>
  Theory Value Preferences :
    valpref({LLit, MProd}, {MProd})
    valpref({MProd}, {LLit})
```

When executed, this theory gives the required decision for *Young*.

```
  pierson |  d  |  outcome(X, d) :- factor(X, pnposs).
  keeble  |  p  |  outcome(X, p) :- factor(X, pliv).
  young   |  d  |  outcome(X, d) :- factor(X, dliv),
  factor(X, pnposs).
```

A screen shot of CATE in this final state is shown in Figure 2. This example illustrates the benefits of using CATE to assist in coming to an understanding of the domain. By incrementally constructing the theory we can develop a broad understanding and then refine it to accommodate cases not yet explained. At any point we have a clear statement of the theory and can check its implications by executing it. This ensures that we can recognise when additional preferences are required to complete the theory; and that the theory has the desired effects, and if it does not, the reasons for the undesired effects are identified. Adding preferences is constrained by the need to keep the theory consistent. The ability to experiment with different theories also helps to identify false moves: in [6] it is said that students are often misled into incorporating the fact about the ownership of land in *Keeble,* which leaves them unable to explain *Young*. This move corresponds to *Theory 3* above, in which the need for an arbitrary preference shows the deficiencies of the theory, allowing us to explore the more acceptable alternative of *Theory 4*.



```
Working Theory - dexa4b
Theory Cases :
 <Pierson, {pNposs}, D>
 <Keeble, {pLiv, pLand, pNposs}, P>
 <Young, {pLiv, pNposs, dLiv}, D>
Theory Factors :
 dLiv
 pLiv
 pNposs
Theory Rules :
 <{dLiv, pNposs}, D>
 <{dLiv}, D>
 <{pLiv}, P>
 <{pNposs}, D>
Theory Preferences :
 pref(<{dLiv, pNposs}, D>, <{pLiv}, P>)    <|From Value Preference|>
 pref(<{pLiv}, P>, <{pNposs}, D>)    <|<Keeble, {pLiv, pLand, pNposs}, P>|>
Theory Value Preferences :
 valpref({LLit, MProd}, {MProd})
 valpref({MProd}, {LLit})
```

```
Cases
<Keeble, {pLiv, pLand, pNposs}, P>
<Pierson, {pNposs}, D>
<Young, {pLiv, pNposs, dLiv}, D>
```

```
Factors
<dLiv, D, MProd>
<pLand, P, MSec>
<pLiv, P, MProd>
<pNposs, D, LLit>
```

Fig. 2. CATE displaying the complete Theory 4

5 Dimensions

As described above, CATE supports case based reasoning in the style of CATO [1] and IBP [7]. The original conception, however, as described in HYPO [2] used dimensions instead of factors. A factor is either present in the case or not, and if present, favours either the plaintiff or the defendant, However, the features of a case can be thought of more as a particular point on a range or dimension. For example, above the factor *pNposs* represents the fact that the plaintiff was not in possession of the animal and favours the defendant. Possession can, however, be seen as a point on a range of positions starting from just seeing the animal, through chasing it, all the way to having caught and killed it. One end of the dimension favours the defendant strongly and as one moves along the dimension the defendant is less strongly supported and the plaintiff is more strongly supported

CATE has been extended to deal with dimensions and works in much the same way as the factor based version. The starting point is now a dimension background, giving a range of possibilities for each of the four factors.

```
<pControl, <noContact, seen, started, wounded, mortally-
Wounded, captured>, <D, P>, <LLit, Prop>>
```

```
<pOwn, <dProperty, dLease, otherPeopleProperty, communalProp-
erty, pLease, pProperty>, <D, P>, <MFreedom, Prop>>
<pMotive, <pMalice, pSport, pLivelihood>, <0, P>, <0, MProd>>
<dMotive, <dMalice, dSport, dLivelihood>, <0, D>, <0, MProd>>
```

The case background is now expressed in terms of <*dimension, point*> pairs instead of factors.

```
<Keeble, {<pMotive, pLivelihood>, <pOwn, pProperty>, <pCon-
trol, mortallyWounded>}, P>
<Pierson, {<pControl, mortallyWounded>}, D>
<Young, {<pMotive, pLivelihood>, <pControl, mortallyWounded>,
<dMotive, dLivelihood>}, D>
<Young, {<pMotive, pLivelihood>, <pControl, mortallyWounded>,
<dMotive, dLivelihood>}, P>
```

Now instead of being presented with a fixed set of factors, we can choose different points on the dimensions so as to produce our own set of factors. The factor background therefore now has the factors described in terms of the dimension they belong to and the point on the dimension where they are.

```
<dLiv, dMotive, dLivelihood, D, MProd>
<pLand, pOwn, pProperty, P, MSec>
<pLiv, pMotive, pLivelihood, P, MProd>
<pNposs, pControl, mortallyWounded, D, LLit>
```

Once we have selected the desired set of factors from the available dimensions, we can proceed as above. The importance of this extension is that it allows for disagreement as to where the crucial points on dimensions should be fixed. The need to be able to represent arguments of this sort is argued in [3]. This extension thus allows a further level of theory construction to be supported.

6 Prospects for Automation

We believe that it should be possible to automate the process of constructing and evaluating theories. The theory constructors of CATE are available for an automated program to use, although it will need guidance on how to construct the theory, so that the theory constructors and the cases, factors and preferences to use can be selected according to some fixed strategy. In previous work [8], [9] and [10] we have explored and evaluated different principles for theory construction. Current work involves determining suitable heuristics for case and factor selection. These methods will be used to construct a program which will drive the CATE so as to create its own theories. These theories will then be evaluated to see how well they explain the outcomes for the cases in both the training set and a set reserved to test how well the theory generalises to new cases.

7 Conclusion

In this paper we have described CATE, a tool which supports the construction of theories to explain a set of legal cases. We see this process as a central part of the work both of a lawyer who wishes to apply the cases, and of a knowledge engineer constructing a system to represent and apply knowledge of cases. The process of theory construction is not simple, and we believe there is a real need for support tools. CATE embodies a particular theory of case based reasoning using factors (or dimensions) and of theory construction (as described in [5]). It enables the rapid development of theories, and enhances the process through checks for consistency and by providing a means to execute the theory so as to explore its implications. Although CATE is designed for human use, we hope in future to add functions to automate the process, by providing strategies for theory construction and heuristics for case selection. This could be deployed either as a stand alone program capable of reasoning with legal cases, or as part of a semi-automated knowledge engineering methodology.

References

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

[2] Ashley. K.D., (1990). Modelling Legal Argument. Bradford Books, MIT Press, Cambridge, Mass.

[3] Bench-Capon, T.J.M., and E.L., Rissland, 2001. Back to the Future: Dimensions Revisited. In B. Verheij, A.R. Lodder, R.P. Loui, and A. J. Muntjewerff (eds.) *Legal Knowledge and Information Systems*, IOS Press, Amsterdam, pp41-52.

[4] Bench-Capon, T.J.M., 2002. The Missing Link Revisited: The Role of Teleology in Representing Legal Argument, *Artificial Intelligence and Law*, volume 10, 1-3, pp 79-94.

[5] Bench-Capon, T., and Sartor, G. 2003. A model of legal reasoning with cases incorporating theories and values. *Artificial Intelligence*. Vol 150 1-2 pp97-143.

[6] Berman, D.H., and C.L. Hafner. 1993. Representing Teleological Structure in Case Based Reasoning: The Missing Link. In *Proceedings of the Fourth International Conference on AI and Law,* 50-59. ACM Press, New York.

[7] Brüninghaus, S and Ashley, K.D. 2003. Predicting Outcomes of Case-based Legal Arguments. *Proceedings of the Ninth International Conference on AI and Law*: ACM Press, New York, 2003, pp233-42.

[8] Chorley, A and Bench-Capon, T. 2003. *Developing Legal Knowledge Based Systems Through Theory Construction.* Technical Report ULCS-03-013, Department of Computer Science, The University of Liverpool, 2003.

[9] Chorley, A., and Bench-Capon, T. 2002. Developing Legal Knowledge Based Systems Through Theory Construction. *Proceedings of the Ninth International Conference on AI and Law*: ACM Press, New York, 2002, pp85-6.

[10] Chorley, A., and Bench-Capon, T. 20023. Reasoning With Legal Cases as Theory Construction: Some Experimental Results. In D. Bourcier (ed.) *Legal Knowledge and Information Systems: Jurix 2003*. IOS Press: Amsterdam.

[11] McCarty, L.T. 1995. *An Implementation of Eisner v Macomber.* In Proceedings of the Fifth International Conference on AI and Law, 276-286. ACM Press: New York.

[12] Prakken, H. 2000. An exercise in formalising teleological case based reasoning. In J. Breuker, R. Leenes and R. Winkels (eds.*), Legal Knowledge and Information Systems: Jurix 2000,* 49-57. IOS Press, Amsterdam.