Can Bobby demand delivery? Towards a knowledge-based system for private law

Christoph Beierle[1], Bernhard Freund[1], Gabriele Kern-Isberner[2], Matthias Thimm[2]

[1]Dept. of Computer Science, FernUniversität in Hagen, 58084 Hagen, Germany
[2]Dept. of Computer Science, TU Dortmund, 44221 Dortmund, Germany

Abstract. Whereas in all existing systems, legal reasoning is rule-based, over time legal rules have been amended by exceptions and counter-exceptions. Therefore, legal rules are not strict, but defeasible. We report on an approach of using Defeasible Logic Programming (DeLP) for the development of the LiZ system, a knowledge-based decision support system for private law. A knowledge base containing legal rules together with their corresponding exceptions and counter-exceptions is automatically translated into a defeasible logic program, and legal reasoning is done by exploiting DeLP's reasoning facilities based on dialectical trees. Legal reasoning with DeLP in LiZ is illustrated by a system walkthrough where the plaintiff seeks the remedy of *specific performance*.

1 Introduction

There are two main reasons for classical logic not being an appropriate framework for legal reasoning: First, legal rules have usually been amended, over time, by exceptions and counter-exceptions. Therefore, legal rules are not strict, but *defeasible*, as we can rarely rely on them not having exceptions. Second, there is the nature of the situation in which legal cases typically arise. Usually there are two or more parties involved, one bringing a claim and the other one arguing against it. Since both of them have their stakes in it, they can hardly be expected to be objective in their description of the situation. This raises the question of how to reconcile the two conflicting views presented by the parties to the judge. Since a judge *must* ultimately find a decision, the conflict has to be solved. Thus, a way of dealing with contradictory and missing information has to be found, and the main question is what consequences may be inferred from a set of defeasible, possibly contradicting legal rules.

In this paper, we will show that defeasible logic programming (DeLP) [2] suits the judicial way of reasoning especially well, and present the argumentative system LiZ for decision support in private law that has been implemented based on DeLP. By using DeLP to implement the model, it becomes possible to add defeasible reasoning (cf. [3]) at the level of subsuming (or classifying) cases while at the same time respecting the classical structure of the law. The system is not confined to material law rules, but takes into account the procedural aspects of the law paramount for the work of a judge, particularly, the burden of proof, necessary indications to the parties, intermediary decisions to manage the process and so on.

In Sec. 2, we first recall the basics of DeLP After a a brief introduction to legal reasoning in private law (Sec. 3), we develop a legal knowledge base model and present an automatic translation of the knowledge base model into DeLP (Sec. 4), After an overview on the LiZ system (Sec. 5), Sec. 6 illustrates legal reasoning with DeLP in LiZ, and Sec. 7 concludes and points out further work.

2 Defeasible Logic Programming

The basic elements of *Defeasible Logic Programming* (DeLP) are facts and rules. A single atom h or a negated atom ~h (in DeLP negation is denoted by ~) is called a literal or *fact*. Rules are divided into strict rules of the form $h \leftarrow B$ and defeasible rules of the form $h \prec B$ where h is a literal and B is a set of literals. A literal h is *derivable* from a set of facts, strict rules, and defeasible rules X, denoted by$X \mid\!\sim h$, iff it is derivable in the classical rule-based sense treating strict and defeasible rules equally. A set X is *contradictory*, denoted $X \mid\!\sim \bot$, iff both $X \mid\!\sim h$ and $X \mid\!\sim$ ~h holds for some h. A *defeasible logic program* (*de.l.p.*) P is a tuple $P = (\Pi, \Delta)$ with a non-contradictory set of strict rules and facts Π and a set of defeasible rules Δ. Using rules and facts arguments can be constructed as follows.

Definition 1 (Argument, Subargument). *Let $h \in \mathcal{L}$ be a literal and let $P = (\Pi, \Delta)$ be a* de.l.p.. *Then $\langle \mathcal{A}, h \rangle$ with $\mathcal{A} \subseteq \Delta$ is an* argument *for h, iff $\Pi \cup \mathcal{A} \mid\!\sim h$, $\Pi \cup \mathcal{A} \not\mid\!\sim \bot$, and $\mathcal{A}$ is minimal with respect to set inclusion. An argument $\langle \mathcal{B}, q \rangle$ is a* subargument *of an argument $\langle \mathcal{A}, h \rangle$, iff $\mathcal{B} \subseteq \mathcal{A}$.*

Two literals h and h_1 *disagree* regarding a *de.l.p.* $P = (\Pi, \Delta)$, iff the set $\Pi \cup \{h, h_1\}$ is contradictory. An argument $\langle \mathcal{A}_1, h_1 \rangle$ is a *counterargument* to an argument $\langle \mathcal{A}_2, h_2 \rangle$ at a literal h, iff there is a subargument $\langle \mathcal{A}, h \rangle$ of $\langle \mathcal{A}_2, h_2 \rangle$ such that h and h_1 disagree.

In order to deal with counterarguments, a central aspect of defeasible logic programming is a formal comparison criterion among arguments. Bearing in mind the context of legal reasoning *Generalized Specificity* [2] seems to be the most appropriate choice. According to this criterion an argument is preferred to another argument, iff the former one is more *specific* than the latter, i. e., (informally) iff the former one uses more facts or less rules. For example, $\langle \{c \prec a, b\}, c \rangle$ is more specific than $\langle \{{\sim}c \prec a\}, {\sim}c \rangle$, denoted by $\langle \{c \prec a, b\}, c \rangle \succ \langle \{{\sim}c \prec a\}, {\sim}c \rangle$, cf. [2]. Then an argument $\langle \mathcal{A}_1, h_1 \rangle$ is a *defeater* of an argument $\langle \mathcal{A}_2, h_2 \rangle$, iff there is a subargument $\langle \mathcal{A}, h \rangle$ of $\langle \mathcal{A}_2, h_2 \rangle$ such that $\langle \mathcal{A}_1, h_1 \rangle$ is a counterargument of $\langle \mathcal{A}_2, h_2 \rangle$ at literal h and either $\langle \mathcal{A}_1, h_1 \rangle \succ \langle \mathcal{A}, h \rangle$ (*proper defeat*) or $\langle \mathcal{A}_1, h_1 \rangle \not\succ \langle \mathcal{A}, h \rangle$ and $\langle \mathcal{A}, h \rangle \not\succ \langle \mathcal{A}_1, h_1 \rangle$ (*blocking defeat*).

When considering sequences of arguments, the definition of defeat is not sufficient to describe a conclusive argumentation line since it disregards the dialectical structure of argumentation, cf. [2].

Definition 2 (Acceptable Argumentation Line). *Let $P = (\Pi, \Delta)$ be a* de.l.p.. *Let $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_m, h_m \rangle]$ be a sequence of some arguments. Λ is called an* acceptable argumentation line*, iff 1.) Λ is a finite sequence, 2.) every argument $\langle \mathcal{A}_i, h_i \rangle$ with $i > 1$ is a defeater of its predecessor $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$ and if $\langle \mathcal{A}_i, h_i \rangle$ is a blocking defeater of $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$ and $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ exists, then $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ is a proper defeater of $\langle \mathcal{A}_i, h_i \rangle$, 3.) $\Pi \cup \mathcal{A}_1 \cup \mathcal{A}_3 \cup \ldots$ is non-contradictory (*concordance of supporting arguments*), 4.) $\Pi \cup \mathcal{A}_2 \cup \mathcal{A}_4 \cup \ldots$ is non-contradictory (*concordance of interfering arguments*), and 5.) no argument $\langle \mathcal{A}_k, h_k \rangle$ is a subargument of an argument $\langle \mathcal{A}_i, h_i \rangle$ with $i < k$.*

In DeLP a literal h is *warranted*, if there is an argument $\langle \mathcal{A}, h \rangle$ which is non-defeated in the end. To decide whether $\langle \mathcal{A}, h \rangle$ is defeated or not, every acceptable argumentation line starting with $\langle \mathcal{A}, h \rangle$ has to be considered.

Definition 3 (Dialectical Tree). *Let $P = (\Pi, \Delta)$ be a* de.l.p. *and let $\langle \mathcal{A}_0, h_0 \rangle$ be an argument. A* dialectical tree *for $\langle \mathcal{A}_0, h_0 \rangle$, denoted $\mathcal{T}_{\langle \mathcal{A}_0, h_0 \rangle}$, is defined as follows:*

1. *The root of $\mathcal{T}$ is $\langle \mathcal{A}_0, h_0 \rangle$.*
2. *Let $\langle \mathcal{A}_n, h_n \rangle$ be a node in $\mathcal{T}$ and let $\Lambda = [\langle \mathcal{A}_0, h_0 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ be the sequence of nodes from the root to $\langle \mathcal{A}_n, h_n \rangle$. Let $\langle \mathcal{B}_1, q_1 \rangle, \ldots, \langle \mathcal{B}_k, q_k \rangle$ be the defeaters of $\langle \mathcal{A}_n, h_n \rangle$. For every defeater $\langle \mathcal{B}_i, q_i \rangle$ with $1 \leq i \leq k$ such that the argumentation line $\Lambda' = [\langle \mathcal{A}_0, h_0 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle, \langle \mathcal{B}_i, q_i \rangle]$ is acceptable, the node $\langle \mathcal{A}_n, h_n \rangle$ has a child $\langle \mathcal{B}_i, q_i \rangle$. If there is no such $\langle \mathcal{B}_i, q_i \rangle$, the node $\langle \mathcal{A}_n, h_n \rangle$ is a leaf.*

In order to decide whether the argument at the root of a given dialectical tree is defeated or not, it is necessary to perform a *bottom-up*-analysis of the tree. Every leaf of the tree is marked "undefeated" and every inner node is marked "defeated", if it has at least one child node marked "undefeated". Otherwise it is marked "undefeated". Let $\mathcal{T}^*_{\langle \mathcal{A}, h \rangle}$ denote the marked dialectical tree of $\mathcal{T}_{\langle \mathcal{A}, h \rangle}$. We call a literal h *warranted*, iff there is an argument $\langle \mathcal{A}, h \rangle$ for h such that the root of the marked dialectical tree $\mathcal{T}^*_{\langle \mathcal{A}, h \rangle}$ is marked "undefeated". Then $\langle \mathcal{A}, h \rangle$ is a *warrant* for h.

3 Legal Reasoning and the Burden of Proof

Legal reasoning is rule-based. This holds true for all legal systems and all areas of law. Moreover, the structure of the rules, whether contained in judgements, statutes, legal literature or even in practitioners' minds, is more ore less universal. A simple private law case, as it might appear in a student textbook, shall help to illustrate this structure.

Example 1. Consider the following scenario: *Bobby Buyer (B) has contracted with Sally Seller (S), the local car dealer, for the delivery of a brand new car. However, at the agreed date, S does not deliver. Can B demand delivery of the car from S?*

To begin with, we need a rule to decide the case. Let's assume B and S live in land where the *UNIDROIT Principles* [5] apply. Then our case would be governed by the following article (Art. 7.2.2 of the UNIDROIT Principles):

*"Where a party who owes an obligation other than one to pay money does not perform, the other party **may require performance**, unless*
(a) performance is impossible in law or in fact;
(b) performance or, where relevant, enforcement is unreasonably burdensome or expensive;
(c) the party entitled to performance may reasonably obtain performance from another source;
(d) performance is of an exclusively personal character; or
(e) the party entitled to performance does not require performance within a reasonably time after it has, or ought to have, become aware of the non-performance."

Thanks to the clear-cut way in which this provision is designed, it is rather simple to extract the rule incorporated in it, or, more precisely, the conditions and the consequence of the rule. In a semi-formal notation, one might put it like this:

```
[ obligation(X,Y,O) and                     // meaning "X owes O to Y"
not_money(O) and                            // true if O isn't a sum of money
no_performance(X,Y,O) and                   // X hasn't delivered O to Y
not (performance_impossible(X,O) or         // X cannot deliver O
  performance_too_burdensome(X,O) or
  alternative_source_available(O) or
  personal_obligation(X,Y,O) or
  time_limit_exceeded(X,Y,O) ) ]            // Y didn't bring his claim in time
⇝ right_to_performance(Y,X,O)               // Y can demand performance (i. e., delivery)
```

To answer the question of the case, one would aim at gathering information about all atoms mentioned in the body of the rule and then derive formally whether the formula specified in the head holds.

However, gathering information on a legal case is not always simple. For instance, already the first condition *obligation*(X,Y,O) is problematic. Though it seems clear that S has promised to B to deliver the car and is thereby obliged to do so, so that *obligation*(S,B,CAR) should be true, an obligation per definition belongs to the *legal sphere*. It is not tangible or measurable as a real-life object, there is no physical detector for it. Instead, we need another legal rule to tell us when an obligation arises. In fact, we find such a rule in the UNIDROIT Principles [5, Art. 1.3], a specialized version of which would contain the following rules (again in semi-formal notation):

```
(1) sales_contract(X,Y,PRICE,O) ⇝ obligation(X,Y,money)  // the buyer's obligation
(2) sales_contract(X,Y,PRICE,O) ⇝ obligation(Y,X,O)      // the seller's obligation
```

In applying Art. 7.2.2 to the example, we would take recourse to the second one of the above rules to determine whether a non-monetary obligation between B and S exists. We will call rules which are invoked in this way (i. e., to explain, define or otherwise determine the conditions of another rule) *secondary rules*, as opposed to the *primary rules* that directly raise a legal claim as their consequence (in this case: "*Can B demand delivery of the car?*", which is answered by Art. 7.2.2).

Breaking down a rule by using secondary rules leads to a tree structure, as Fig. 1 illustrates. The conditions of the primary rule are "explained" by secondary rules, the conditions of which may in turn be explained by further secondary rules, and so on; the leafs of the tree should be obtained by collecting appropriate "real-life" facts.

Collecting the relevant facts imposes the next issue that has to be dealt with in order to integrate all information for a given case. As discussed before, usually, there are multiple parties involved in legal reasoning and the question at hand is, what to do when these parties provide contradictory or incomplete information? The approach taken by courts all over the world is the following: First, a kind of *assumption* is created for every single one of the relevant facts. If the parties cannot help to clarify the fact in question, the fact is either assumed to be given or absent. This provides a way of dealing with incomplete information. Second, if the parties do say something but contradict each

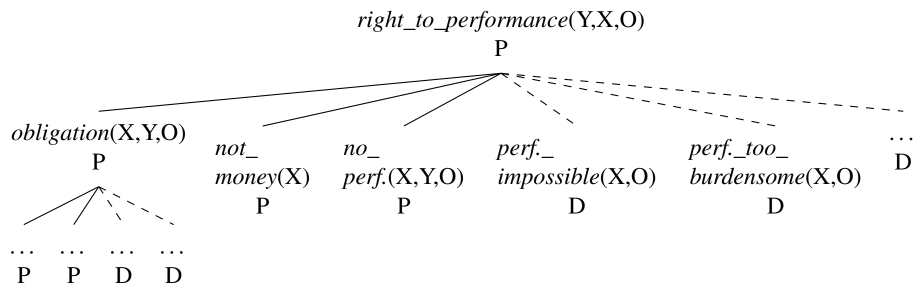


Fig. 1. Part of the *legal tree* for the claim *right_to_performance*(Y,X,O). The solid lines correspond to *if*, a dashed line to *unless*. The labels P (plaintiff) and D (defendant) indicate the *burden of proof*.

other, evidence is collected from both sides. If the evidence suffices to establish the fact (or its absence), this result becomes part of the basis for the decision. If however the evidence remains inconclusive, again recourse is taken to the assumption created earlier to circumvent the contradiction presented by the parties.

The legal term associated with the creation of the assumption is *burden of proof* (see [6], [7], or [4] for an AI and Law perspective on this subject) Saying that a party has the burden to prove fact "A" means that the assumption for this fact is "NOT A", and vice versa.

The concept of burden of proof can be elegantly represented in the *legal tree* introduced in Fig. 1: We start at the root of the tree and mark the root and the conditions of the primary rule with a "P" for plaintiff, and the conditions of exceptions to the primary rule with a "D" for defendant. Then we work our way down the tree: In the illustration given in Fig. 1, this means that the burden of proof is inherited from the parent node along solid lines, but inverted along dashed lines. Thus, not only the relevant facts needed to solve a given case but also the respective burden of proof for each of the facts (and, therefore, the assumptions that follow) can easily and automatically be calculated. The only knowledge necessary is the rules (and their exceptions).

4 Legal Knowledge Modelling

We will now develop a legal knowledge base model based on the observations given above. As a general starting point we will allow rules to be either strict or defeasible, while the latter will be the standard.

Legal provisions In order to represent legal rules in a concise fashion, we will represent exceptions within their corresponding rule. The concept of rules and exceptions correctly reflects the theoretical structure of the law, but at a somewhat low level. For example, a single article from any given statute usually contains more than one rule (and can contain multiple exceptions, cf. Art. 7.2.2). Therefore, a meta-structure comprising all those rules seems convenient. In our model, rules with the same consequence are incorporated in a *legal provision* (cf. Def. 4 below).

Subsumption When applying a legal rule, after breaking it down as far as possible to its constituents (i. e. the conditions of the lowest secondary rules in the "legal tree"), there inevitably comes the task of *subsuming* the given case under these constituents. Although there are no more legal rules available to decide whether these conditions are fulfilled, one will usually find further lexical rules to do so. Taking a trivial example, to decide whether an obligation is "non-monetary" (see Art. 7.2.2), "money" could be looked up in a dictionary, which would probably enumerate the forms in which money comes along or its characteristics. Such definitions are nothing else than further rules. Our model therefore allows for *subsumption rules* to be added to the database. They have the same form as legal rules and are treated likewise in the inference process. They are only named differently to make clear that they do not share the same authority due to their non-legal origin.

Open concepts and antagonistic rules As mentioned, legal rules in the knowledge base are usually meant to be *defeasible*, i. e. open to exceptions and refutation. For instance, as "adequate" denotes a very broad and fuzzy concept, indicators have been developed for the adequacy and inadequacy of damages to make the handling of the law easier and more foreseeable. Such arguments can be seen as *antagonistic* rules, where the consequence of one rule is the negation of the consequence of the other.

In our legal knowledge base, not only rules with the same consequence, but also their antagonistic counterparts become part of one single *legal provision*:

Definition 4 (legal provision). *A* legal provision *is a triple* $P = (c, \mathcal{B}, \overline{\mathcal{B}})$ *with* $\mathcal{B} = \{B_1, \ldots, B_k\}$ *and* $\overline{\mathcal{B}} = \{B_{k+1}, \ldots, B_{k+l}\}$ *where* $k \geq 1$ *is the number of legal rules having* c *as their consequence, and* $l \geq 0$ *is the number of corresponding antagonistic rules. Each* B_i *is of the form* $B_i = ((b_1, \ldots, b_n), E_1, \ldots, E_m)$ *where the* b_j *are the* conditions *and* E_j *sequences of* exceptions *for the* i*-th rule with* $n \geq 1$*, and* $m \geq 0$*.*

For instance, in an attribute-value pair notation for legal provisions (in LiZ, an XML representation is used), Art. 7.2.2 is given by the following knowledge base entry:

```
Art. 7.2.2 = (consequence = right_to_performance(Y,X,O),
              rules = ( (conditions = (obligation(X,Y,O), not_money(X),
                                       no_performance(X,Y,O) ),
                     exception = ( performance_impossible(X,O) ),
                     exception = ( performance_too_burdensome(X,O) ),. . . ) ) )
```

A *legal knowledge base* is a set $KB = \{P_1, \ldots, P_n\}$ of legal provisions. For each claim c, such a knowledge base uniquely determines a legal tree $ltree(c, KB)$ with root c and whose further nodes and arrows are constructed from KB. For the general case, we extended the legal tree construction as illustrated in Fig. 1 accordingly:

- Multiple rules with the same consequence: Outgoing arrows of a node are grouped together if they belong to the same rule.
- Conjunctions of exceptions: The respective arrows are grouped accordingly.
- Antagonistic rules: Introduce *not-if* and *not-unless* arrows.

Note that the computation of the burden of proof within the legal tree carries over smoothly to antagonistic rules: The burden of proof is inverted along *not-if* arrows, but inherited along *not-unless* arrows.

One advantage of the representation for legal knowledge model described above is the similarity to a *de.l.p.*. A legal knowledge base can be transformed into DeLP rules straightforwardly; the relevant facts of the case can then be added as DeLP facts. To define the translation to DeLP, let $P = (c, \mathcal{B}, \overline{\mathcal{B}})$ be a legal provision as in Def. 4 and let $B = ((b_1, \dots, b_n), E_1, \dots, E_m)$ with $E_i = (e_{i,1}, \dots, e_{i,r_i})$. If $B \in \mathcal{B}$, the following $m + 1$ DeLP rules are generated:

$$\begin{aligned} c &\prec b_1, \dots, b_n. \\ {\sim}c &\prec b_1, \dots, b_n, e_{1,1}, \dots, e_{1,r_1}. \\ &\dots \\ {\sim}c &\prec b_1, \dots, b_n, e_{m,1}, \dots, e_{m,r_m}. \end{aligned}$$

For $B \in \overline{\mathcal{B}}$, the DeLP rules generated are obtained from the rules above by replacing c (resp. ${\sim}c$) in each rule head by ${\sim}c$ (resp. c).

5 Overview of the LiZ System

The structure of the LiZ system is illustrated in Fig. 2. It interactively follows the course of a civil action from the judge's point of view. First, the user enters general information about the case, especially the kind of claim. Then, LiZ searches the knowledge base for rules that might support such a claim, collects all the secondary rules needed to interpret them, and constructs and shows the legal tree(s). It also determines the burden of proof for the relevant facts. The user can then successively enter the relevant facts as presented by plaintiff and defendant through the GUI. Whenever a new fact is entered, LiZ updates the three fact sets (view of the plaintiff/defendant, view to be taken by the judge), transforms them to DeLP facts and passes them on to the DeLP interpreter, along with a query as to whether the claim would be supported under each of the fact sets. The results are transformed into plain text as used by judges, the user can base his or her intermediary decisions on them (e. g., ask the defendant to produce further evidence to avoid losing the case). Thus, the system shows the decision that would have to be made at any given time, which becomes the final decision once all facts have been entered. Currently, the LiZ database contains rules from the law of obligations of the German Civil Code, but the design of LiZ is in no way specifically tailored to the German jurisdiction.

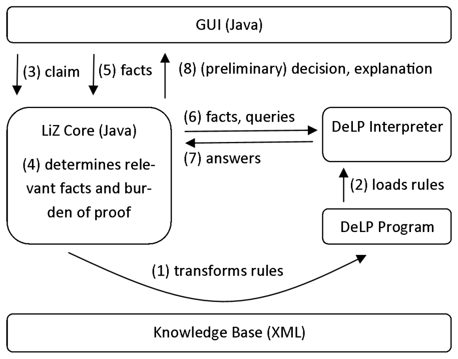
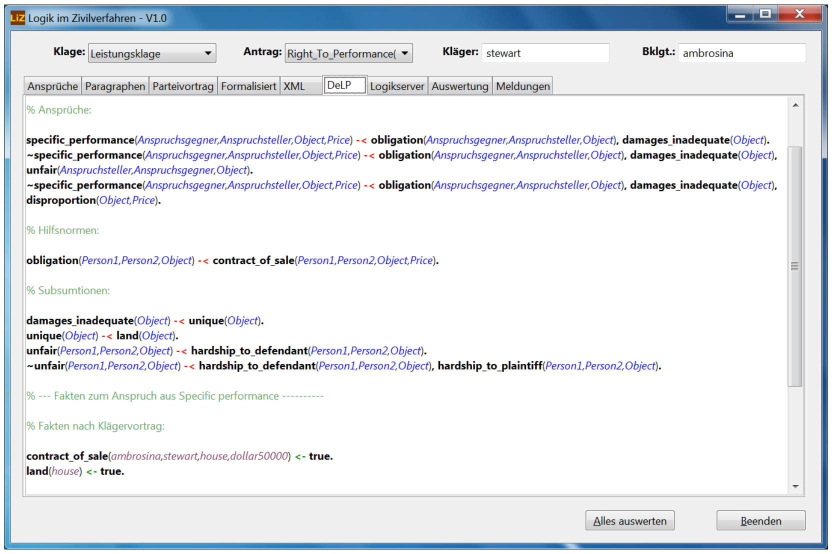


Fig. 2. Illustration of the LiZ system. The system is interactive, steps 5 to 8 are repeated during the legal procedure until no more relevant facts are submitted.

6 Legal Reasoning with DeLP in the LiZ System

For a system walkthrough, we will use some rules of the common law regarding *specific performance* in the context of a sale:



```
% Ansprüche:

specific_performance(Anspruchsgegner,Anspruchsteller,Object,Price) -< obligation(Anspruchsgegner,Anspruchsteller,Object), damages_inadequate(Object).
~specific_performance(Anspruchsgegner,Anspruchsteller,Object,Price) -< obligation(Anspruchsgegner,Anspruchsteller,Object), damages_inadequate(Object),
unfair(Anspruchsteller,Anspruchsgegner,Object).
~specific_performance(Anspruchsgegner,Anspruchsteller,Object,Price) -< obligation(Anspruchsgegner,Anspruchsteller,Object), damages_inadequate(Object),
disproportion(Object,Price).

% Hilfsnormen:

obligation(Person1,Person2,Object) -< contract_of_sale(Person1,Person2,Object,Price).

% Subsumtionen:

damages_inadequate(Object) -< unique(Object).
unique(Object) -< land(Object).
unfair(Person1,Person2,Object) -< hardship_to_defendant(Person1,Person2,Object).
~unfair(Person1,Person2,Object) -< hardship_to_defendant(Person1,Person2,Object), hardship_to_plaintiff(Person1,Person2,Object).

% --- Fakten zum Anspruch aus Specific performance ----------

% Fakten nach Klägervortrag:

contract_of_sale(ambrosina,stewart,house,dollar50000) <- true.
land(house) <- true.
```

Fig. 3. DeLP representation of legal rules and facts in LiZ

R1: Specific performance of an obligation (e.g. delivery in the case of a sale) can be demanded if money damages would be an inadequate compensation for non-performance.
R2: By concluding a sales contract, a seller enters an obligation to deliver a certain object to the buyer.
R3: Damages are inadequate if the object to be delivered is unique.
R4: Land is unique.
R5: Specific performance will not be granted if this would be unfair and unreasonable.
R6: Specific performance would be unfair and unreasonable if it amounts to hardship for the defendant (e.g. the seller).
R7: Hardship to the defendant will not be considered unfair and unreasonable if there would also be hardship on the plaintiff's side in case of non-performance.
R8: Specific performance of a sale is to be denied if the price and the real value of the object are grossly in disproportion.

Note that R5 and R8 are exceptions to R1, while R2, R3, R4, R6 and R7 are secondary rules. After formalizing these rules in legal provisions (Def. 4), the transformation presented in Sec. 4, yields the eight DeLP rules shown in the upper part of Fig. 3.

Let us now concider the Canadian case Stewart v. Ambrosina (*Stewart v Ambrosina* (1976) 63 DLR (3d) 595, Ontario High Court) where the plaintiff sought the remedy of *specific performance*, i.e. she asked the court to force the defendant to fulfill a certain obligation (namely the obligation to deliver a house the defendant had agreed to sell). In a first step, the names of the parties as well as the nature of the claim are entered into the LiZ system. LiZ returns a list of rules that can support such a claim. The user (e.g. the judge) selects the rules appropriate for further consideration. Here, the list contains only one such rule (i.e. R1), representing the traditions of the common law regarding specific performance (as a so-called equitable remedy). After selection of the rule to be investigated, LiZ constructs the corresponding legal tree by expanding the rule with the help of secondary rules from the knowledge base. The resulting legal tree, showing all conditions of the rule and therefore all information necessary to solve the case, is presented within the GUI of the LiZ system.

In the next phase, the information presented by the parties is entered. Whenever new facts are entered, LiZ calls the DeLP interpreter to re-evaluate the case. In court proceedings as modeled by the LiZ system, the parties usually take turns asserting new facts or presenting new evidence. Therefore, the evaluation (whether the remedy should be granted or not at the present stage) usually shifts from one side to the other, along with what is sometimes called the *tactical burden of proof*. This shifting happens when new facts support new arguments that support the claim or act as counterarguments to existing arguments. In the example case, the first facts about the case presented by the plaintiff are that a contract of sale has been concluded. We also enter as a fact that the object of the sale - a house with the corresponding estate - falls into the category of *land*. Internally, LiZ transforms these information to DeLP facts, adds them to the DeLP programme representing the rule for specific performance (cf. Fig. 3; the facts are given in the lower part of the screenshot), and calls the DeLP-interpreter to evaluate whether the claim is now supported.

With the two facts given, we have a first valid DeLP argument supporting the claim: ⟨{R1,R2,R3,R4}, *right_to_performance*(*stewart*,*ambrosina*,*house*)⟩. The DeLP-interpreter therefore answers that, according to the facts brought forward by the claimant, the claim is supported. LiZ translates this into language intelligible for the user being a legal expert.

However, in Stewart v. Ambrosina, the defendant also had some arguments on his side. Mrs. Ambrosina was in a rather unfortunate situation. She had concluded the contract selling her home together with her husband, who, before the contract was executed, had committed suicide. After this unexpected tragedy, the widow had to raise six children on her own relying on benefits. In this situation, forcing her to abandon her home and deliver it to the plaintiff was to be considered *hardship*. In LiZ, we will therefore enter the additional fact that the defendant has established *hardship_to_defendant*.

Thus, ⟨{R5,R6}, ~*right_to_performance*(*stewart*,*ambrosina*,*house*)⟩ becomes a counterargument to ⟨{R1,R2,R3,R4}, *right_to_performance*(*stewart*,*ambrosina*,*house*)⟩ at *right_to_performance*(*stewart*,*ambrosina*,*house*). The evaluation of DeLP and the ouput of LiZ switch accordingly to show that given these facts, the defendant would prevail.

However, the case wasn't over yet. The plaintiff, Mrs. Ambrosina, brought forward additional information to the effect that she - herself a widow with five children and little income - had sold her old home in expectation of the newly bought one. She could therefore establish hardship in case of non-performance (*hardship_to_plaintiff*). Adding this in LiZ yields ⟨{R7}, ~*unfair*(*stewart*,*ambrosina*,*house*)⟩ as a counterargument to ⟨{R5,R6}, ~*right_to_performance*(*stewart*,*ambrosina*,*house*)⟩ at *unfair*(*stewart*,*ambrosina*,*house*). The resulting dialectical tree is given in Fig. 4. Since the argument at the root is undefeated, LiZ suggests to grant the remedy sought, as was done by the court in the real case of Stewart v. Ambrosina.

Finally, it should be pointed out that, whenever LiZ calls the DeLP interpreter, it sends the query corresponding to the claim three times, along with three different fact sets. The first two sets represent the *picture of the case* as painted by plaintiff and defen-

$$\begin{tikzcd}
\langle\{\mathrm{R1,R2,R3,R4}\},\ \mathit{right\_to\_performance}(\mathit{stewart},\mathit{ambrosina},\mathit{house})\rangle \quad \text{(undefeated)} \arrow[d, dash] \\
\langle\{\mathrm{R5,R6}\},\ {\sim}\mathit{right\_to\_performance}(\mathit{stewart},\mathit{ambrosina},\mathit{house})\rangle \quad \text{(defeated)} \arrow[d, dash] \\
\langle\{\mathrm{R7}\},\ {\sim}\mathit{unfair}(\mathit{stewart},\mathit{ambrosina},\mathit{house})\rangle \quad \text{(undefeated)}
\end{tikzcd}$$

Fig. 4. Dialectical tree generated by DeLP

dant respectively, and the third set of facts represents the combination of these two sets according to the rules of procedure, especially the burden of proof. The burden of proof is calculated by LiZ based on the legal tree built from the rules in the knowledge base. The answer of the DeLP interpreter to the third query, based on the fact set according to the rules of procedure, is the most important one. It tells LiZ how the case should be decided.

7 Conclusions and Further Work

LiZ is an interactive system providing decision support for private law cases. It takes into account the viewpoint of a judge and follows the legal procedures established in the judical domain. The implementation in DeLP exploits the reasoning facilities provided by defeasible logic programming as legal reasoning is not strict, but inherently defeasible. In [1], various criteria for legal reasoning to be justifiable are investigated, a more extensive discussion of the burden of proof is given, and, taking a broader judical and argumentation based point of view, it is shown how this relates to the approach taken in the LiZ system. Our future work includes the modelling of further legal cases with LiZ and a corresponding broader evaluation of the system.

References

1. C. Beierle, B. Freund, G. Kern-Isberner, and M. Thimm. Using defeasible logic programming for argumentation-based decision support in private law. In *Proc. of the Third Int. Conf. on Computational Models of Argument (COMMA 2010)*. IOS Press, 2010. (to appear).
2. Alejandro J. García and Guillermo R. Simari. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming*, 4(1):95–138, 2004.
3. D. M. Godden and D. Walton. Defeasibility in judicial opinion: Logical or procedural? *Informal Logic*, 28(1):6–19, 2008.
4. T. F. Gordon and D. Walton. Proof burdens and standards. In I. Rahwan and G. Simari, editors, *Argumentation in Artificial Intelligence*, pages 239–260–144. Springer, 2009.
5. International Institute for the Unification of Private Law. *UNIDROIT Principles of International Commercial Contracts.* UNIDROIT, Rome, 2nd edition, 2004.
6. H. Prakken. A formal model of adjudication dialogues. *AI and Law*, 16(3):333–359, 2008.
7. G. Sartor. Defeasibility in legal reasoning. In Z. Bankowski, I. White, and U. Hahn, editors, *Informatics and the Foundations of Legal Reasoning*, pages 119–157. Kluwer, 1995.