Proceedings of the 3^{rd} ICAPS Workshop on Hierarchical Planning (HPlan 2020)

Formalising German Legal Opinions as Planning

Gregor Behnke
University of Freiburg
behnkeg@informatik.uni-freiburg.de

Abstract

A legal opinion in the German legal system is a formal piece of writing that investigates whether a given statement of law is true or not given a description of a specific case. Writing these opinions is the central element of German legal education, but is supported only by basic IT technologies, such as text-based search engines. Formalising legal thoughts would enable the creation of various tools that support students, lawyers, and judges in correctly applying the law.

German legal opinions are interesting are interesting from a research perspective as they follow a strictly formalised structure and method of argumentation. In practice, these opinions can (often) be seen as a thorough application of so-called schemata. A schemata provides a fixed way to check whether a specific assertion of law holds or not by providing sub-assertions to check and a rule on how these results should be combined. In essence, these schemata therefore describe a hierarchical (but potentially recursive) structure on legal terms and properties.

We propose a formalisation of these schemata in terms of Hierarchical Task Network (HTN) planning. The modelled domain will describe the application of the law on the specifics of a given case s.t. the resulting plan and its decompositional structure will constitute the structure of a legal opinion on the case.

1 Introduction

Almost any action has some connection to the legal system we are living in. Either by means of civil law (governing the relationship between peoples and between people and objects), public, or criminal law (dealing with the connection between the state and its citizens). Despite its pervasiveness, the legal argumentation is still a relatively manual task. Especially when learning to apply the law, there is almost no automated support – apart from e.g. search engines.

Applying the law is a multi-step process. First, the facts relevant to the case in question must be gathered. Second, these facts are associated with legal concepts – a process called subsumption. For example one may have to decide whether the letter written by a vendor is a "firm summons to pay" (which is required for a notice of default). Third, we have to reason about legal consequence, e.g. who has an obligation to pay. The first two steps deal with the fuzzy and uncertain aspects and are thus more suited for sub-symbolic techniques like machine learning.

The third step may however be viewed as "purely logical". At least for the anglo-american common law system, this is not the case, as legal reasoning here hugely depends on matching prior cases against the facts of the case at hand – which again is a fuzzy process. Thus research for common law systems often deals with reasoning on the basis of precedents (Atkinson and Bench-Capon 2019). In civil law systems, the application of the law depends only on the written law. The constant jurisprudence of the highest courts may still play a role in the interpretation of the written law, but is not as dominant as is the common law. As such, there is a body of work dealing with the formalisation of legal reasoning, e.g. as Deontic Logic (Jones and Sergot 1992), as Answer Set Programming (Aravanis, Demiris, and Peppas 2018), as PROLOG rules (Satoh et al. 2011), as ontologies (Palmirani and Governatori 2018), or as argumentation (Marshall 1989; Prakken and Sartor 2015).

How reasoning about the consequences of given facts under the law is performed depends on the legal system. Germany's legal system uses opinions to derive consequences, which follow a highly formalistic and logic-driven style of argumentation (Kischel 2019, p. 417ff). Notably, there exist pre-determined ways and means for determining whether a statement X follows from the specifics of a given case, so-called schemata. For this, the schema sets out a sequence of other statements that have to be checked and a rule on how to combine them.

In this paper, we present a formalisation of these schemata in terms of planning. With this formalisation we are (1) able to derive the truth of a statement given an appropriately modelled fact (at least to some degree) automatically and (2) lay the foundation for future automated assistance of legal scholars, layers, and students. Notably, the plan generated for a given fact will correspond to the structure of the opinion for checking it. We could, for example, use our formalisation as the basis for teaching student specific concepts of law – by developing opinions in cooperation with

them. Note that we do not aim at fully writing an opinion, we rather aim at representing the structure and argumentation contained in the opinion. Any verbalisation would be out of scope of a conference paper and we consider it future work.

We start by giving a brief introduction into Legal Opinions and their structure and will then introduce our running example that we will consider throughout the paper. We then introduce the concept of HTN planning. Thereafter we discuss how the structure of opinions and their schemata can be represented in terms of HTN planning problems.

2 Legal Opinions and German Civil Law

Legal reasoning is – at least in Germany – most often contained in written so-called opinions. An opinion is a structured collection of arguments what show why a legal result (e.g. a title or a right) is entailed by the law and the specifics of a given case. For example, an opinion may ask whether a person B has the obligation to pay a given amount of money to a claimant A. Writing opinions is the central element of legal training at German universities. Almost all university exams require students to write an opinion on a given case. Further, the First State Exam (the first part of the German bar exam) consists of six written tests, each asking to write a legal opinion. The strict adherence to structured and logical arguments required for opinions sets legal education in Germany apart from the system used in many other countries. Learning to write these opinions is however quite hard for most students. Thus, supporting the process of learning to write opinions with AI technologies may be a fruitful endeavour.

Argumentation in a legal opinion follows a strict style of writing, called the opinion-style. In order to determine the truth of a statement of law, one has to perform the four steps of a syllogism, called the legal syllogism (German: "Justizsyllogismus"):

1. the premise,
2. the definition,
3. the subsumption, and
4. the result.

The premise states that a certain statement (about an obligation, a title, a right, or any other property or legal connection between two persons or a person and an object) might be true. Then one defines the criteria under which the statement is true, citing the relevant law. Within the subsumption, the specifics of the case are mapped to the criteria set out in the definition. Usually, the subsumption is the part of an opinion that requires case-dependent argumentation. As the subsumption is based on the verbal description of the case, it is a highly fuzzy process. Lastly, if the requirements set out in the definition are met, one concludes that the premise is indeed true. If the definition again contains assertions that cannot be directly mapped to specifics of the case, as they are e.g. legal terms, the subsumption will contain further legal syllogisms with these assertions as their premises. These syllogisms are nested recursively.

We do not consider how subsumptions are made, as we want to focus on the logical structure of opinions. Notably, the AI techniques that should be used for performing or assisting subsumptions are quite different from the ones we use, i.e. they might be based on machine learning, natural language processing, and text mining. Instead, we assume that all possible subsumptions of the case have already been made. We assume that we are presented with a given case in terms of a set of logical atoms (instantiated ground predicates) which refer to the most basic concepts of the law. Note that this is not a restriction, but solely a clean separation between the logical and the fuzzy part of the opinion. If we, in the future, are to apply this modelling we of cause have to deal with extracting such a logical description based on non-logical inputs. This may either be done using machine learning techniques, or in collaboration with the user.

Example Case Whenever possible, we will use an example case to illustrate the principles of our formalisation of opinions in terms of planning problems. On June 21st K[1] asked V whether he can buy a set of white floor tiles from V – for 500 EUR. On June 23rd, V agreed. V delivered the tiles to K three days later. K then installed the tiles in his bathroom. Four months later, K noticed that the tiles changed colour from white to greyish. An investigation revealed that this was caused by a production error. K tells V that he wants new tiles delivered to him. Neither K nor V could have noticed the production error before the tiles were installed. Removing the installed tiles and installing the new tiles will cost 400 EUR.

In this case there are two questions: (1) is V required to provide K with new tiles? (2) does V have to pay (additionally) 400 EUR. We will elaborate on the specifics of the case and the answers to the two questions throughout the paper.

3 HTN Planning

In this paper, we will use (totally-ordered) HTN planning (Ghallab, Nau, and Traverso 2004; Erol, Hendler, and Nau 1996; Geier and Bercher 2011) to model the structure of German legal opinions. HTN planning distinguishes two types of tasks: abstract tasks and primitive actions. Both are described via a name and a list of parameter variables, each associated with a type. For example, `(foo ?bar ?baz)` denotes a task named `foo` with parameters `?bar` and `?baz`. We use the syntax of PDDL (McDermott 2000) to denote tasks. Variable names always start with a question mark. A state is described via ground atoms of first order logic, i.e. predicates with constants as their arguments. A state is any subset s of these atoms. As in classical planning, primitive actions in HTN planning carry a state transition semantics, defined via their preconditions $prec$ and effects eff. $prec$ may be any function-free first order formula referring to the variables that are parameters of $prec$'s actions. A (ground) action a is executable in a state s, iff $s \models prec$. The effect eff of an action consists of two sets of atoms add and del which again may refer to the parameters of a. If a is executed in s, it results in the state $(s \setminus del) \cup add$. The execution of sequences of states is defined inductively.

[1] Persons are customarily abbreviated by upper-case letters.

Abstract tasks represent more complex courses of action and do not carry a (direct) state-transition semantics. Instead, their semantics is defined via so-called decomposition methods $m = (t, tn)$. Here, t is the task the method decomposes and tn is a task network – for totally-ordered HTN planning this is a sequence of other tasks, both primitive and abstract. tn can also be an empty sequence. Applying such a method means to replace an occurrence of t with tn. The objective in HTN planning is given in terms of an initial abstract task t_I. We now maintain a sequence π of tasks and initialise it with t_I. We repeatedly apply decomposition methods to tasks in π, until π contains only primitive actions. This derived plan π is a solution to the HTN planning problem if it is executable in the given initial state s_I.

Additionally, many HTN planners (e.g. SHOP (Nau et al. 1999)) allow for *method preconditions*. Such a precondition $prec$ associated with a method m restricts the application of m to cases where the state prior to the first task originating from m satisfies $prec$. Since we consider only totally-ordered models, this is equivalent to the following restriction. Consider a current sequence of tasks $\pi = t_1 \dots t_{i-1} t_i t_{i+1} \dots t_n$. Then a method precondition for a method m applicable to t_i must hold in the state between t_{i-1} and t_i.

4 Representing the Structure of Opinions

As stated before, we assume that the specifics of the case we are to consider are already converted into a set of atoms. In our modelling, these atoms form the initial state s_I. In our example case, this includes e.g. the facts `(hasCondition whiteTiles beingGrey)`, `(usualCondition whiteTiles beingWhite)`, and `(handover V K whiteTiles June26)`. All of them model (relevant) aspects of the given case.

A legal opinion in its entirety determines whether a given premise holds or not. Throughout the opinion, sub-opinions may discuss different other premises, each asking whether a specific statement of law holds. Thus, we have to represent the notion of a premise in terms of the concepts of HTN planning. A natural way to do so is to model premises as tasks. This way, e.g. the initial abstract task t_I will represent the premise of the whole opinion. In our example case, this would be either `(obligationToProvide K V tiles)` or `(obligationToPay K V 400EUR)`. In an opinion, we have to state the definition pertaining to the premise P. Next, we either have to perform a subsumption or have to start inner opinions that determine the truth of statements made in P's definition.

If we only have to perform a subsumption, the premise P only depends on the *most basic* concepts of the law, which have to be inferred from the textual description of the case. As stated before, we do not deal with the intricacies of extracting knowledge from the case in this paper, but only with modelling a legal opinion based on it. As such, we assume that the facts of the case are already fully modelled as ground first order statements and can thus be checked in a primitive action's precondition. Thus, any premise P that only requires a "pure" subsumption will be modelled as a primitive action who's preconditions will check the necessary facts.

If the premise's definition refers to *non-basic* concepts, we have to introduce inner opinions. We do this via modelling such a premise as an abstract task A and introduce the inner opinions via a method m applicable to A. m's task network contains a task for every premise that needs to be checked in order to be able to determine the truth of the main premise A.

The derivation of a plan π will correspond to the structure of an opinion for the given case. To extract the structure of the opinion, we simply follow the decomposition methods applied for obtaining π. We start out with an opinion containing the premise represented by the initial abstract task t_I. When looking at the method applied to t_I, we know which definition and sub-premises should be inserted into the opinion. Similarly, if a decomposition yields a primitive action, the opinion will contain a subsumption.

5 Schemata for Inner Opinions

We have just stated that decomposition methods will introduce the necessary inner opinions. This assertion is somewhat vague and needs clarification. The central question at this point – and of this paper – is which criteria need to be checked in order to be able to establish the truth of a given statement of law. For that – at least in civil law systems – we have to turn to the written law. Most legal norms describe at their core an implication where a set of premises are mapped to a consequence. There are other types of legal norm, e.g. describing the intention of the legislator (e.g. § 1 of Germany's Nuclear Law: "The purpose of this law is (1.) to end the use of nuclear energy for commercial purposes in an orderly fashion ...") or describe broad abstract statements (e.g. Article 1 of Germany's Basic Law: "Human dignity shall be inviolable."). These types of statements are not used for directly determining whether a given statement of law is true, but (for (1.)) for interpreting under-specified terms in the law. How such interpretations may be considered within a formalised version of the law is out of scope of this paper and may be considered in future work. Thus we are interested in norms that provide as their consequence the statement A – our current premise represented by an abstract task. Of course there can be multiple norms providing the same consequence, and there are often norms that provide exceptions, counter-exceptions, counter-counter-exceptions, and so on for a given assertion. These are known to be quite difficult to handle in a general fashion; see for negation in legal reasoning e.g. (Kowalski 1989).

When writing an opinion the complexities of these logical structures are often simplified by – to some degree – standardised structures for checking whether a given assertion is true or not (Kischel 2019, p. 419f.). Instead of solely applying the word of the law, in many practical cases there are established so-called *schemata* that outline how a specific assertion should be checked, in which order the individual premises should be checked, which exceptions and counter-exceptions should be considered, and in which order. There are even whole books that summarise these schemata for different areas of the law (Maties and Winkler 2018). This apparent "standardisation" serves (in the authors' opinion) the purpose of conformity of expectations: the reader of an

opinion knows what the writer wants to do next and why the author does or does not discuss certain issues at each point while reading the opinion. Schemata also ease a student's understanding of the law, as they are able to "follow them" and by that will obtain the correct result in applying the law. For that however, they first have to understand their meaning, learn them, and be able to extract them from the written law.

Naturally there are situations when applying the law, where no schemata are available, i.e. where the meaning of the law must be interpreted manually. How this is done is a quite complex topic, as it e.g. includes argumentation of the intent of the legislator. Formalising this kind of fuzzy area of applying the law is thus difficult and we consider it to be out of scope of our work. We will restrict our modelling to areas of the law where well-established schemata for applying the law exists. The difficulty in practise lies in correctly applying these schemata and extracting the relevant facts from a description of the case. The work in this paper is however still necessary and useful, as it lays the foundation for being able to connect reasoning about the more fuzzy parts of legal argumentation with those that are more standardised.

6 Logical Structure of Schemata

As discussed in the previous section, when modelling what needs to be checked in order to establish the truth of a given statement of law, we use the well-defined schema for it. Such a schema is essentially a list of conditions under which a given statement holds. As an example, consider the requirements for supplementary performance in German Warranty Law. Supplementary performance is the process or act with which a bought object that is defective is repaired or replaced, i.e. the case where the buyer is given e.g. a non-working object and thus has the right to either get the object repaired or replaced. This is exactly the situation we face in question (1) of our example case: K wants to have the tiles replaced.

Supplementary performance is required if (1) vendor and buyer have a valid sales contract, (2) the purchased object has a "defect" (German: "Sachmangel"), (3) this defect was present when the risk passed from the vendor to the buyer (German: "Gefahrübergang") and (4) the rights arising from defects are not precluded ((Maties and Winkler 2018), Nr. 178). If one is to check whether a right to supplementary performance exists, one has to check these four criteria. Further, there are two types of supplementary performance: repair and replacement. In our example case, K wants the tiles to be replaced. The schemata for replacing an object is this case required (1) that a right to supplementary performance exists, (2) K has selected replacement, and (3) no right to withhold performance because of disproportionality ((Maties and Winkler 2018), Nr. 178). When checking whether K has the right to new tiles, the top-most structure of the opinion is as shown in Fig. 1. In an opinion the conditions just listed always appear in the same fixed order. Further, if one of the conditions is not satisfied, the opinion must discuss all conditions prior to the not satisfied one as well, but not the ones afterwards. If, e.g. the object has no defect under the law, one still has to show that there is a valid sales contract, but can omit discussing the passing of the risk and so on.

However, not all schemata describe conjunctions of conditions. For example, there are in total seven causes for a defect of an object – detailed in § 434 and § 435 BGB.[2] In order for an object to have a defect under the law, one of these causes suffices, i.e. they form a conjunction. One might think that checking only one – namely the one constituting the defect – might be sufficient in an opinion. This is not the case. The causes for a defect have an implicit and customary order in which they have to be discussed ((Maties and Winkler 2018), Nr. 176 and 177). Notably, if e.g. the fourth cause of a defect is present, but not the first three, an opinion must discuss and reject the first three causes. On the other hand, there are also disjunctions, for which it is not necessary to check all possible causes until the first one is successful. Examples are e.g. causes for the invalidity of declarations and causes for a suspension of the statute of limitations. Here, it is sufficient to check the single cause that will lead to the desired result. Lastly, there are also disjunctions for which all conditions must be checked fully, i.e. one has to check any possible criterion causing a statement to become true, irrespective of the truth of the other criteria. One notable example for this is the crime of causing bodily harm ("Körperverletzung"): it requires that the perpetrator to either "physically assault" someone or to cause "damages to health". In an opinion, both criteria must be checked – even if the first one is already fulfilled – while one of the two being fulfilled is sufficient.

When modelling schemata in an HTN planning problem, we will thus distinguish four types of logical connectors:

1. conjunction,
2. ordered disjunction,
3. free disjunction, and
4. complete disjunction.

Next, we have to consider that it is not only sufficient for an opinion to check whether a given statement of law holds. In many cases it is also necessary to check whether the contrary is true, i.e. that a given statement does not hold. We have already seen two such cases above, namely the conditions (4) no preclusion and (3) no right to withhold. As a further example, a cause for a defect is that the vendor has not provided a manual and the buyer did not successfully assemble the object (§ 434 II 2 BGB, the so-called IKEA-clause). Similarly, a claim can only be pursued as a general rule, if the statute of limitations has not expired. Note that these negations can be nested in complex cases – or if we want to check e.g. the absence of a claim between two persons.

Thus our modelling of schemata has to handle negations. A purely logical approach to negation is not suited as legal opinions require specific structures of argumentation, which are different from a purely logic-based argumentation. Consider the negation of a conjunction, i.e. we want to check that a given statement is not true and the truth of this statement is based on a conjunctive schema. From a logical point-of-view, it is sufficient to find one condition x in the conjunc-

[2] BGB = Bürgerliches Gesetzbuch, Germany's civil code

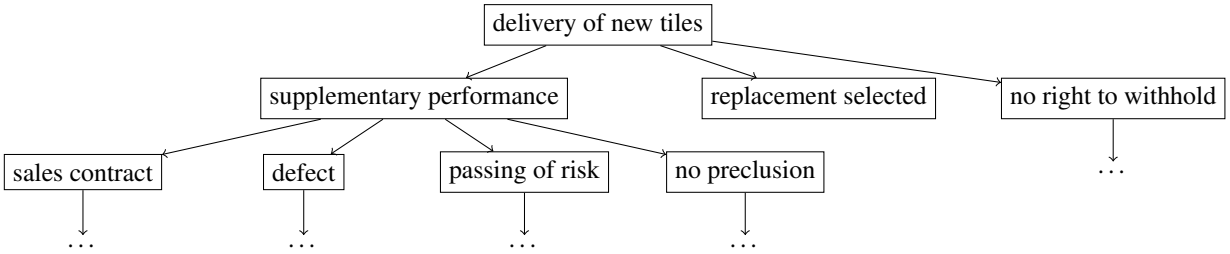

Figure 1: Structure of an opinion for the example case at its top layers.

tion that is not true. However in an opinion, one also has to show that all conditions preceding x are true, i.e. one has to find the *first* failing condition. For all three types of disjunctions, we have no alternative but to show that all its conditions are false.

7 Representing the Logic of Schemata

As we have discussed in Sec. 4, the inner opinions needed to ascertain the truth of a statement A shall be the tasks contained in methods decomposing A. Which assertions we have to check in inner opinions and how their result influences the truth of the overall statement A is determined by A's schema. In our modelling, the method decomposing A will always contain tasks for all conditions $B_1, \ldots, B_n$ contained in the schema. If, in a concrete opinion, a specific condition B_i does not have to be checked, it will be decomposed using an empty decomposition method – denoting that it should not be part of the opinion. Hence, what remains is to provide a mechanism for suitably modelling the logical structure of the schema and its implications on the assertions to be checked in an opinion.

Representing Negation as a Parameter

Next, we note that checking a specific assertion, i.e. an abstract task, may either be done with the objective of proving the assertion or proving the negation of the assertion. Thus it might seem sensible to add a parameter to each abstract task, denoting the mode in which it is to be checked. If we do this though, every task inside a method will need such a parameter. Next, these parameters must be separate variables for each condition in a schema, as if we check a conjunction or the negation of a disjunction, not all conditions are checked with the same objective. Assertions in the methods precondition would have to ensure that the variables are set correctly for each of the subtasks. We don't use such a modelling for three reasons. First, it unnecessarily increases the size of the ground instantiation as the variable determining the objectives for each B_i. Second, such a modelling may make the model harder to solve when using grounded progression search algorithms, which are currently one of the best algorithms for HTN planning (Höller et al. 2018). Since the variables are part of the decomposition method, we have to *guess* which of the conditions $B_1, \ldots, B_n$ will hold when we apply the decomposition method. With the modelling we propose below, we defer this decision until the point where we actually check each of the individual conditions B_i. Third, the rules on determining the conditions' objectives have to be encoded once per method, thus cluttering the model with unnecessary repetitions and making it less readable. With the method that we propose below, these rules are encoded only once in the domain and remain static, e.g. if new material law is added to the domain – thus easing modelling significantly.

Representing Negation as a State Variable

Instead of using parameter variables to denote the objective with which we check each of the conditions $B_1, \ldots, B_n$, we use a *mode* that we set in the state. We model the mode as a predicate `(mode ?m)`. Each abstract task A (i.e. its decomposition) will check the objective set out by the mode which holds in the state directly before A. After A has been checked in the given mode, our modelling sets the current mode back to the mode with which A has been checked. If it is not possible to check the assertion of A in the given mode, it will not be possible to decompose it into an executable plan. This way, we can guarantee that the created opinion is in itself consistent: a valid plan can only be found if the result matches the assertion that was set out in the beginning. Further, it allows us to create planning problems where the result of the opinion is unknown. Here, the initial abstract task is a dummy, that decomposes into the actual premise of the opinion preceded with a primitive action that will either set to mode to check the premise positively or negatively.

In addition to two modes representing objectives, we introduce three additional modes we use for controlling which conditions are checked for negated conjunctions and non-negated disjunctions. We use the following five modes with their respective meaning:

- `yes` – we attempt to show that the condition holds
- `no` – we attempt to show that the condition does not hold
- `indifferent` – we have to show that the condition holds or not, but the result is irrelevant
- `ignore` – we do not have to consider this condition, but do not yet know whether the main statement A is true nor not
- `done` – we do not have to consider this condition, but already know whether the main statement A holds

Any method for A uses its method precondition to determine the current mode via a precondition `(mode ?initMode)`. If `?initMode` is `yes`, we know that we have to check the assertion represented by A positively, if it

is `no` negatively. If the mode is `indifferent`, it does not matter whether we check the statement positively or negatively. These three cases are handled by the same decomposition method. This way we have to model the schema for A only once in the planning domain and eliminate unnecessary redundancy. The method precondition of this method checks that `?initMode` is either `yes`, `no`, or `indifferent`.

If `?initMode` is either `ignore` or `done`, we don't have to check the assertion at all. Thus for these cases, the planning domain contains an empty decomposition method (i.e. one without subtasks) that checks whether `?initMode` is either `ignore` or `done` in its precondition. In total, we create only two decomposition methods for each statement A, one for the case where we actually have to execute A's schema and one where we completely forgo checking A. The first type of methods actually applying A's schema will contain a subtask for each of the conditions $B_1, \ldots, B_n$. Whether any condition B_i will be checked in a concrete opinion is determined by the method applied to it – if the second type of method is applied, it will not be contained in the opinion. This way, we can model the schemata fully inside a single method – without any complicated if-then-else structure in the method.

Methods with Mode-Checking

When checking an assertion A via a method of the first type, we have to set the correct checking-mode before each of the tasks representing the conditions $B_1, \ldots, B_n$ contained in A's schema. The corresponding tasks then have to perform their checks according to the set mode. Once the task has been executed, we have to set the mode for the next task, execute it, and repeat until all conditions of the assertion's schema have been handled. Determining which mode can be used for each of the subtasks depends on multiple factors: the type of the logical connector that the schema uses, the mode of the assertion we are currently checking, and the mode with which the previous subtasks have been checked. Firstly, instead of basing the result on the modes for all previous subtasks, we only consider the last one. Its mode will have aggregated all necessary information about the already checked conditions. For example, if a condition B_i was successfully checked with the mode `yes` in a conjunction, all previous conditions B_j with $j < i$ were also checked successfully with the mode `yes`. Here, the modelling capabilities of HTN planning come into play: The method that has checked the previous condition will have set the mode back to the mode it was "called". Via this mechanic, we know in A's method the mode with which the subtask was called without the need for adding additional variables to the method – which would increase the size of the model's grounding and make the method itself less readable. Secondly, we use the variable `?initMode` to determine the checking mode of A itself. This is – once again – only possible as we are using HTN planning. It allows us to enforce that a given set of actions – those setting the modes – share a common parameter. This essentially forms a kind of a brace-like structure, i.e. a context-free structure, which are not expressible with classical planning, but with HTN planning (Höller et al. 2014).

Thus the mode that should be set of a condition B_i depends on: (1) the logical connector, (2) the overall mode of A, and (3) the mode that was set to B_{i-1}. In Tab. 1 we show how we set the mode for B_i depending on these three inputs. We additionally show how we set the mode initially (the *start mode*). Lastly, we list which mode is required to hold after the last condition B_n has been handled. With this check, we ensure that in the case of a positively checked disjunction one of the conditions was checked with the `yes` mode and for a negatively checked conjunction that one of the conditions was checked with the `no` mode. For the reverse cases (positive conjunction and negative disjunction), the correct modes are already enforced by the mode setting.

To set the modes we use a primitive action `(set-mode ?logic ?initMode)` implementing the state transition function of Tab. 1. Here `?logic` is a variable that can be instantiated with four constants: one for each of the four types of logical connectors. We use actions `(start-mode ?logic ?initMode)` and `(end-mode ?logic ?initMode)` for setting the initial mode, checking the final mode and setting the mode after the last condition back to `?initMode`. We add the `set-mode` action prior to every task B_i in a method and `start-mode` and `end-mode` as the first and last tasks in each method.

The mode-setting in Tab. 1 assumes that the conditions B_i are occurring positively in the schema. For handling negation, we use an additional action `invert`. If executed, it will change the `yes` mode into `no` and vice versa – effectively implementing a negation of a specific check. It will not alter the mode if it is set to anything other than `yes` or `no`. In total, if schema states that A holds if either B, not C, or D hold, its method will contain the following tasks and actions:

1. `(start-mode disj-seq ?m)`
2. `(set-mode disj-seq ?m)`
3. `(B)`
4. `(set-mode disj-seq ?m)`
5. `(invert)`
6. `(C)`
7. `(invert)`
8. `(set-mode disj-seq ?m)`
9. `(D)`
10. `(end-mode disj-seq ?m)`

If we are to use this method in a setting where neither B, nor C, nor D holds, A will hold. The execution and mode-setting in this case is shown in Fig. 2.

8 Modelling

We have modelled (parts of) German Warranty Law using our methodology. Our modelling solves the two questions we set out for our example case in the beginning of the paper.

For this, we also had to model parts of connected areas of civil law, e.g. parts of contracts law, or consumer rights. The domain currently consists of 42 abstract tasks, 84 lifted decomposition methods, and 35 lifted primitive actions. The model is available at https://github.com/galvusdamor/htnlaw. The example case we've outlined in the beginning is solved by an SAT-based HTN planner (Behnke, Höller,

target	last	conjunction	ordered disjunction	free disjunction	complete disjunction
yes	start mode	yes	no	ignore	no
no	start mode	yes	no	no	no
yes	yes	yes	done	done	indifferent
yes	no	–	yes/no	–	yes/no
yes	ignore	–	–	ignore/yes	–
yes	done	–	done	done	–
yes	indifferent	–	–	–	indifferent
no	yes	yes/no	–	–	–
no	no	done	no	no	no
no	done	done	–	–	–
yes	end mode	yes	yes/done	yes/done	yes/indifferent
no	end mode	no/done	no	no	no
indifferent	end mode	yes/no/done	yes/no/done	yes/no/done/ignore	yes/no/indifferent

Table 1: State transitions for modes. Start mode indicates the mode in which checking starts while the last two lines (denoted with "end mode") denotes the states in which checking may end with success. Dashes show transitions that are not allowed.

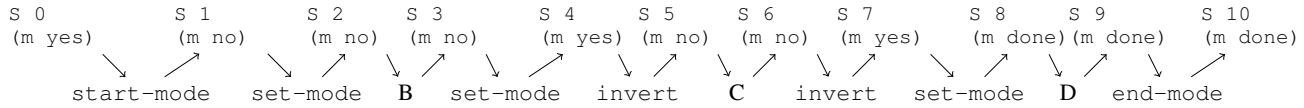

Figure 2: Structure of the evaluation of the assertion: A, if either B, not C, or D. The disjunction is an ordered disjunction. We assume that neither B, C, nor D hold, i.e. A holds because C does not hold. The initial objective is to show that A holds. `S i` denotes that ith state, while `(m x)` indicates the mode `x` set in this state.

and Biundo 2019a; 2018; 2019b) in 13.7 seconds on an Intel i5-4300U.

9 Conclusion and Outlook

In this paper, we showed that the structure of German legal opinions can be formalised in terms of HTN planning. This formalisation represents objectives to be checked as abstract tasks and uses decomposition methods to check whether the assertion holds using the established schemata for doing so. We showed how the various types of logical structures occurring in these schemata can be modelled within decomposition methods by exploiting the high expressiveness of HTN planning. The presented modelling relies on a prior formalisation of the facts of a given case in terms of appropriate first order atoms. In the future, we may use the modelled domain to enable a semi-supervised formalisation of subsumption. We may, in turn use this to assist law students in learning to structure their thoughts and to write opinions themselves. For example, we could verify that the structure of options written by students complies with the schemata using HTN plan verification (Behnke, Höller, and Biundo 2017; Barták, Maillard, and Cardoso 2018). Based upon the detected errors, we could develop techniques to provide useful hints to the students on how to improve their opinions.

References

Aravanis, T.; Demiris, K.; and Peppas, P. 2018. Legal reasoning in answer set programming. In *Proceedings of the 30th International Conference on Tools with Artificial Intelligence (ICTAI 2018)*, 302–306. IEEE Computer Society.

Atkinson, K., and Bench-Capon, T. 2019. Reasoning with legal cases: Analogy or rule application? In *Proceedings of the 17th International Conference on Artificial Intelligence and Law (ICAIL 2019)*. ACM.

Barták, R.; Maillard, A.; and Cardoso, R. C. 2018. Validation of hierarchical plans via parsing of attribute grammars. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2017. This is a solution! (... but is it though?) – Verifying solutions of hierarchical planning problems. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 20–28. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT – Totally-ordered hierarchical planning through SAT. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*, 6110–6118. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2019a. Bringing order to chaos – A compact representation of partial order in SAT-based HTN planning. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, 7520–7529. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2019b. Finding optimal solutions in HTN planning – A SAT-based approach. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5500–5508. IJCAI.

Erol, K.; Hendler, J.; and Nau, D. 1996. Complexity results for HTN planning. *Annals of Mathematics and AI* 18(1):69–93.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1955–1961. AAAI Press.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language classification of hierarchical planning problems. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, volume 263, 447–452. IOS Press.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, B. 2018. A generic method to guide HTN progression search with classical heuristics. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*, 114–122. AAAI Press.

Jones, A., and Sergot, M. 1992. Deontic logic in the representation of law: Towards a methodology. *Artificial Intelligence and Law* 1(1):45–64.

Kischel, U. 2019. *Comparative Law*. Oxford University Press, 1. edition.

Kowalski, R. 1989. The treatment of negation in logic programs for representing legislation. In *Proceedings of the 2nd International Conference on Artificial Intelligence and Law (ICAIL 1989)*, 11–15. ACM.

Marshall, C. 1989. Representing the structure of a legal argument. In *Proceedings of the 2nd International Conference on Artificial Intelligence and Law (ICAIL 1989)*, 121–127. ACM.

Maties, M., and Winkler, K. 2018. *Schemata und Definitionen Zivilrecht: mit Arbeits-, Handels-, Gesellschafts- und Zivilprozessrecht*. C.H.Beck, 1. edition.

McDermott, D. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):35–55.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 1999)*, 968–973.

Palmirani, M., and Governatori, G. 2018. Modelling legal knowledge for GDPR compliance checking. In *Legal Knowledge and Information Systems - JURIX 2018: The Thirty-first Annual Conference*, 101–110.

Prakken, H., and Sartor, G. 2015. Law and logic: A review from an argumentation perspective. *Artif. Intell.* 227:214–245.

Satoh, K.; Asai, K.; Kogawa, T.; Kubota, M.; Nakamura, M.; Nishigai, Y.; Shirakawa, K.; and Takano, C. 2011. Proleg: An implementation of the presupposed ultimate fact theory of japanese civil code by prolog technology. In Onada, T.; Bekki, D.; and McCready, E., eds., *JSAI International Symposium on Artificial Intelligence – New Frontiers in Artificial Intelligence (JSAI 2011)*, 153–164. Springer Berlin Heidelberg.