Erschienen in: Model Checking Software : 28th International Symposium, SPIN 2022, Virtual Event, May 21, 2022 : Proceedings / Legunsen, Owolabi; Rosu, Grigore (Hrsg.). - Cham : Springer, 2022. - (Lecture Notes in Computer Science ; 13255). - S. 1-23. - ISSN 0302-9743. - eISSN 1611-3349. - ISBN 978-3-031-15076-0
https://dx.doi.org/10.1007/978-3-031-15077-7_1

Automated Consistency Analysis for Legal Contracts

Alan Khoja[2], Martin Kölbl[1(✉)], Stefan Leue[1], and Rüdiger Wilhelmi[2]

[1] Department of Computer Science, University of Konstanz, Konstanz, Germany
Martin.Koelbl@uni-konstanz.de, Stefan.Leue@uni-konstanz.de
[2] Department of Law, University of Konstanz, Konstanz, Germany
Alan.Khoja@uni-konstanz.de, Wilhelmi@uni-konstanz.de

Abstract. Contracts in business life, and in particular company purchase agreements, often comprise a large number of provisions and are correspondingly long and complex. In practice, it is therefore a great challenge to keep track of their regulatory context and to identify and avoid inconsistencies in such contracts. Against this background, we propose a semi-formal as well as a formal logical modeling of this type of contracts, using decidable first-order theories. We also present the tool *ContractCheck*, which performs fully automated inconsistency analyses on the considered contracts using Satisfiability Modulo Theories (SMT) solving.

1 Introduction

Contracts are indispensable in business. They enable the contracting parties to arrange their legal relationships by giving legal effect to their common will and establishing mutual claims. A prominent example is the purchase of a company in a share purchase agreement (SPA). Like any purchase contract, an SPA must specify at least the indispensable *essentialia negotii*: the purchaser and the seller, as well as the purchase object and the purchase price to be transferred. In practice, SPAs regulate all relevant legal issues in the contract and exclude references to statutory law as far as possible. As a consequence, SPAs have a very local semantics that is almost entirely based on the contractual duties agreed upon in the SPA.

Contracts, and especially SPAs, are often very long and complex. This is due, in particular, to the large number and complexity of the issues to be regulated. In addition, a large number of persons are often involved in the drafting. Moreover, the negotiations and the drafting may take long and comprise a large number of amendments of the draft. Length, complexity and frequent changes make a contract prone to errors and inconsistencies, such as references being wrong, missing essentials or unfulfillable claims. Inconsistencies in the form of missing essentials can be found by a simple syntactic analysis. It is significantly more complex to find inconsistencies in the dynamics of several claims. Claims should not contradict each other and be performable in the context of legal facts described in

the contract. Also, the combination of several due dates can be unexpectedly restricted by a statute of limitations. For instance, assume an SPA contains a warranty claim that must be asserted within 14 days after the closing on day 28, then subsequent performance must be made within 28 days, and otherwise damages must be paid within another 14 days. Further assume, that the contract contains a provision that warranty claims are limited to 42 days after closing. In this example, the warranty clause provides that performance can continue until day 84, while the limitation period ends on day 70. The timing in the SPA is inconsistent.

In this paper we propose a concept to model and automatically analyze a textual SPA. As its main contributions, we present a meta-model for the entities of an SPA using a class diagram from the Unified Modeling Language (UML) [Obj17] in Sect. 3, give the modeling a semantics in decidable fragments of first-order logic in Sect. 4 and automatically analyze the SPA for inconsistencies using Satisfiability Modulo Theory (SMT) technology in Sect. 5. The aim is a red-flag system to highlight errors that we compute in a two-step analysis:

- First, our syntactic analysis checks whether the fundamental elements are part of the SPA and every referenced claim does also exist.
- Second, we provide a dynamic analysis to check whether every claim can be performed and whether a feasible execution of the SPA exists.

We have implemented the analyses in a tool called *ContractCheck*, and demonstrate its ability to detect and explain inconsistencies using a case study in Sect. 6. In this paper, we develop the concept and the tool using an SPA under German law, but this does not preclude adaptation to contracts with a different subject matter or under other jurisdictions, as they might be more complex but basically have a comparable structure with similar elements.

When considering examples of SPAs, it can be observed that they often consist of very similar blocks of text that only vary in certain parameters, for instance the agreed price. In the case of international company purchases, it is estimated that about half of the text of the provisions is changed little or not at all [HS16]. In order to obtain a formal logical representation of the SPA, as part of our approach and tool we provide a library of parameterized structured English text blocks that can be freely combined and used to compose the contract textually. These blocks allow greater flexibility than conventional approaches, in which the creation of contracts is dialog-driven and based on decision trees. Each of these blocks has a formal semantics expressed using formulae from a decidable fragment of first-order logic. The conjunction of these conditions then constitutes the logical representation of the contract. The analysis method that we describe in this paper generalizes to the class of all concrete contracts that can be formulated by composing and parameterizing the provided text blocks.

Related Work. We review works addressing the logical modeling and analysis of legal artefacts. We consider the verification of smart contracts [BK19, PF19] which are, in effect, described by executable program code, outside the scope of this paper, since their analysis has more similarity with program analysis.

Several meta-models of various legal domains describing legal entities and their relations have been proposed. A first work representing contracts with UML has been proposed in [EGB+01]. A multi-level hierarchical ontology to model a contract is described in [KJ03], which is expressed using UML class diagrams. A modeling of contracts as business processes has been suggested in [Kab05, WS05]. In [DNS08], a business process is translated into state machines to check whether it is always beneficial for the contracting parties to fulfill their claims while the contract is being executed. LegalRuleML [PGR+11, OAS21, Gru18] is a specialization of the general language RuleML [BPS10] designed to express relationships between legal entities by rules. We are not aware of any extension of LegalRuleML towards the analysis of contract executions for inconsistencies. The Contract Specification Language (CSL) is a modeling language that represents claims as actions in a contract. In [HKZ12], an execution of a given CSL expression is computed as a sequence of actions, which is then analyzed to determine whether any specified commitments are met. In [HLM20], a contract is interpreted and analyzed as a composition of commitments. In [MKK14], the natural language sentences of an e-contract are translated into a dependency graph in order to check whether individual regulations contradict each other.

Deontic logic [VW51] is a family of modal logics designed to express the semantics of claims with operators for obligation, permission, and prohibition. Deontic logic is translated into propositional logic in [CM07] and analyzed for inconsistencies using a tableau calculus in [CM08]. Further analyses using tableau constructions are proposed in [BBB09]. These analyses find contradictions inside of a contract but do not compute potential contract executions. The Contract Lanugage *CL* [PS07, PS12] encompasses Deontic logic. In [PPS07], a contract given in *CL* is translated into a transition system, which is then analyzed by the model checker NuSMV for inconsistencies. In [CS17], a *C-O diagram* [MDCS10], which is a graphical extension of *CL*, is expressed by state machines and analyzed using the real-time model checker UPPAAL. The analyses find syntactic inconsistencies and prove a dynamic inconsistency by one execution but needs an expert to encode a contract as a *C-O diagram*. An extension of *CL* to *RCL* in [MB15, BM21] allows to reason about the persons between whom claims exist using the tool *RECALL*. A further system supporting a Deontic logic based approach is the Linear Time Temporal Logic (LTL) based language *FL* [GMS10, GMS11] together with the model checker *FormaLex* [GMS11, FMS+17]. In the above cited approaches, claims can only be logically connected, so that a prioritization of primary and secondary duties, which are central concepts in SPAs, is impossible.

Comparison to Related Work. The approach that we propose in this paper focuses on the use of SAT and SMT technology in discovering inconsistencies in SPAs. While many of the approaches cited above do have the ability to detect inconsistencies, they do not have the ability to produce models revealing reasons for inconsistencies that lie in data rather than in the execution of actions, which is one of the aspects that we will focus on. Also, only finite executions are of interest in an SPA which is why LTL-style model checking capabilities of infinite

behaviors are not required. In comparison to the approaches relying on Deontic logics, while we do consider claims, they are only one of the many logical facets of contracts that we are interested in. SAT- and SMT-based modeling and analysis turns out to be very flexible in allowing us to directly represent claims in the formalization of SPAs without requiring the overhead of model checking or tableau constructions required to analyze deontic logic fomulae.

Contributions. This paper presents the following main contributions:

1. We propose an ontology for SPAs using the UML.
2. We provide a logical formalization of SPAs using a decidable fragment of first order logic. By this we contribute to the extensive research area addressing the use of formal logics in the representation of legal artefacts.
3. We define a number of consistency analyses that are applicable to SPAs.
4. We describe a tool that uses a collection of parameterized natural language building blocks from which textual SPAs can be composed.
5. We present the prototypical tool *ContractCheck* that uses SAT, SMT and satisfiability core technology to perform the consistency analyses and which produces diagnostic information explaining identified inconsistencies.
6. We illustrate the application of this approach to an example SPA.

2 Preliminaries

We describe the modeling of an SPA using class diagrams from the UML and encode both the consistency requirements as well as the consistency analyses in decidable theories of first-order logic.

Modeling. The entities of an SPA are depicted as classes in a class diagram of UML [Obj17]. A class contains attributes and operations. An attribute describes a property of an object and has a name and a type. A class diagram graphically represents classes by rectangles and their associations to each other by lines. The relationship between different classes are described by associations between the classes. An association *generalization* describes the relationship between an abstract class and a more specific class. The generalization is graphically represented by an arrow with a triangle, pointing to the abstract class. The association *aggregation* represents an affiliation and is marked using a diamond symbol attached to the associated object. A set relation can be annotated by an integer value range indicating how many objects of one class are related to how many objects of the other class.

Formalization in Logic. We formalize an SPA and the analyses in decidable fragments of first-order logic. The formalization uses the logic of linear real arithmetic, equality, integers and uninterpreted functions [KS16]. For these fragments, efficient SMT solvers [KS16] exist that automatically decide whether a logical formula from the fragment is satisfiable. In case of satisfiability, they return a satisfying assignment which is also called a model, and otherwise, they return unsat.

For example, an SMT solver calculates that the formula $(x > 0) \wedge (x + y < 0)$ is satisfiable and returns, for instance, the variable assignment $x = 0.9$ and $y = -2.0$ as a possible model.

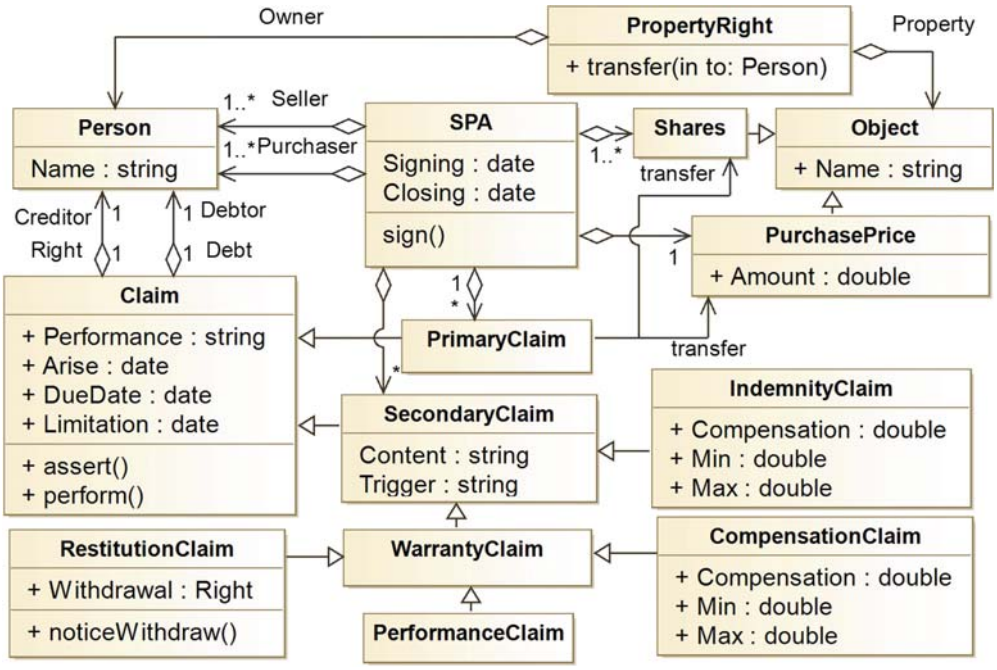

Fig. 1. Class diagram of a Sales Purchase Agreement (SPA)

3 Contract Modeling

We propose the use of UML class diagrams in the ontological modeling of an abstract view on an SPA. An instance of a class diagram describes an legal contract and will be the basis for the analyses encoding in the next sections. We concretize this modeling approach by applying it to the modeling of a concrete SPA for the purchase of a pretzel bakery.

3.1 Modeling of an SPA

For modeling purposes, it is necessary to determine the typical provisions in an SPA. These can be found in numerous legal template books [GS22, MSJ22, Sei18a, WAB20, WK22]. For the present project, we analyzed a wide variety of these books (regarding German law) [vH20a, vH20b, MS22a, MS22b, Pfi22, Sei18b, Sei18c, Sei18d, Sei18e]. We have identified the following provisions as typical for an SPA: contracting parties, purchase object, purchase price provisions, conditions and execution, warranties and indemnities, liability, final provisions. Based on this, an SPA was developed that contains the typical provisions. In this paper, we are primarily concerned with the provisions on *warranties*, *indemnities*

and *liability*[1], supplemented by the indispensable provisions on the contracting parties (*purchaser* and *seller*), the *purchase object* and the *purchase price*.

From these provisions, we derive an ontology that is given as a UML class diagram, depicted in Fig. 1. It represents the typical provisions of an SPA in the form of classes. Notice that this ontology can be extended to other types of contracts. In this paper, we restrict ourselves to considering SPAs since, compared to other types of contracts, they rely much less on implicit legal facts implied by legal dogmatics, which are typically waived in SPAs.

As a purchase contract, an `SPA` consists of at least one person, the `Seller`, who has to transfer an `Object`, in particular the `Shares` of a company, as a *purchase object* to a second person, the `Purchaser`. In return, the `Purchaser` has to transfer another `object`, an `Amount` of money, as `PurchasePrice`. An SPA becomes effective on the date of `Signing`, on which the purchase contract is `sign`ed. The `Shares` and the `Purchase Price` have to be transferred when due on date `DueDate`, that is usually identical to the date of `Closing`.

In the SPA, the purchaser and the seller promise each other to *perform* certain *claims*. The content of a `Claim`, the promised `Performance`, is described as an attribute in the claim. For each *claim*, either the *purchaser* or the *seller* is the *debtor*, who owes the *performance*, and the other one is the *creditor*, respectively. The `Purchaser` is `Creditor` of the *claim* to transfer the company *shares* and the `Seller` is a `Debtor` of the same. Vice versa, the `Seller` is `Creditor` of the *claim* to transfer the `PurchasePrice` and the `Purchaser` is `Debtor` of the same. A `claim` arises on date `Arise`. The creditor can `assert` that the debtor does `perform` the *claim* in rendering the `Performance`. S/he can `assert` the `claim` from the `DueDate` until the `Limitation` date. Any `claim` expires on the day of `Limitation`. In order to fulfill these *claims*, we also need to model the *property rights*. The seller can *perform* the *claim*, when s/he is the `Owner` of the `PropertyRight` on the `Shares` and `transfers` this right to the purchaser.

The *claims* to transfer the *shares* and *purchase price* are `PrimaryClaim`s. A *primary claim* is a *claim* in a contract that has to be performed from the outset by the *due date*. In this, it differs from a `SecondaryClaim`, that only arises with the breach of another *claim* or of separately listed circumstances. A *secondary claim* refers to another claim by a `Trigger`, and we model its circumstances by an attribute `Content`. We call a *secondary claim* without a `Trigger` an *independent claim*.

Within *secondary claims* in an SPA, a distinction is made between a *warranty claim* and an *indemnity claim* [Wil22]. A `WarrantyClaim` refers to an unknown risk due to a breach of a *primary claim*, or of listed circumstances that are not expected but considered likely enough to require regulation. A *warranty* consists of the *warranty content* as the prerequisite and one or more *warranty claims* as consequence of a breach of warranty. The warranty `content` includes the existence or non-existence of certain circumstances, which usually concern the *purchase object* and especially its properties. The warranty claims are usually *claims* of the *purchaser* for compensation in money, irrespective

[1] We set legal terms in italics and terms referring to UML diagrams in teletype font.

of fault. It is also possible to agree on a *claim* for subsequent performance (`PeformanceClaim`) or a right of `Withdrawal` that in case of a notice of withdrawal (`noticeWithdraw`) terminates the contract and can give the purchaser a *claim* for restitution of the *purchase price* and the seller a *claim* for restitution of the *shares* (`RestitutionClaim`s). A `CompensationClaim` is often limited in the way that the amount of `Compensation` must reach a minimum value `Min` and may not exceed a maximum value `Max`. The limitation may apply only to individual *claims* or to all *claims* under the contract. If the purchaser has a *claim* for compensation, this can be offset against the purchase price claim, so that the total amount of money to be paid is reduced accordingly. An *indemnity* refers to a known risk due to an expected breach of a *primary claim* or listed circumstances. In case of such a breach, it gives the purchaser an `IndemnityClaim` for indemnification by means of an appropriate compensation in money.

Example 1 (Bakery SPA). As a running example, we use an SPA for the fictitious sale of a bakery from the seller `Eva` to the purchaser `Chris`. We instantiate this example by the UML class diagram presented in Fig. 2. In the bakery SPA, Chris agrees to pay the purchase price of €40,000 (`PayClaim`), and Eva agrees to give Chris ownership of the bakery `Shares Bakery` (`TransferClaim`). Both claims are due at the agreed closing 28 days after signing. In case the pretzel bakery is not transferred or the purchase price is not paid at closing, then the other contracting party may withdraw from the purchase contract. In addition to the *primary claims*, Eva warranties in a `WarrantyClaim PretzelWarranty` that the bakery can bake 10,000 of pretzels a day. If the bakery in the example cannot bake 10,000 of pretzels, then the warranty is breached and Chris has to assert this breach within the `DueDate` of 14 days of closing. In case of assertion, Eva has to make good within 28 days because of the `PerformanceClaim Claim1`, otherwise she has to pay within 14 days a compensation of €1,000 per 100 of pretzels that cannot be baked, due to the `CompensationClaim Claim2`. `Claim2` has a minimal compensation of €1000. Any claim under the warranty has a `Limitation` of 42 days of closing. Further crucial facts are represented in the bakery SPA. Due to debts of the Eva, a local bank `Bank` has ownership by way of security in the shares in the bakery `SecurityOwnership`.

Manual inspection reveals the following inconsistencies in the SPA: Eva's *primary claim* according to the SPA is to transfer the bakery to Chris. In order to fulfill her claim, she must be the owner. However, consider that Bank is the owner of the bakery due to the transfer of ownership by way of security. Eva cannot fulfill her claim as she is not the owner of the Bakery, which we consider an inconsistency, or at least a loophole. Another inconsistency is due to the timing of the claims. The `PretzelWarranty` has a `Limitation` of 42 days after Closing, even if an assertion occurs within 14 days after Closing, then followed by the 28 days of `Claim1` and 14 days of `Claim2`. This implies that `Claim2` can take up to 56 days after closing, contradicting the `Limitation` of 42 days.

4 Formalization

We present the logical encoding of the objects of an SPA in decidable fragments of first-order logic and demonstrate the encoding using the bakery SPA case study. A model representing a satisfying assignment of the formalization represents a possible execution of the considered SPA.

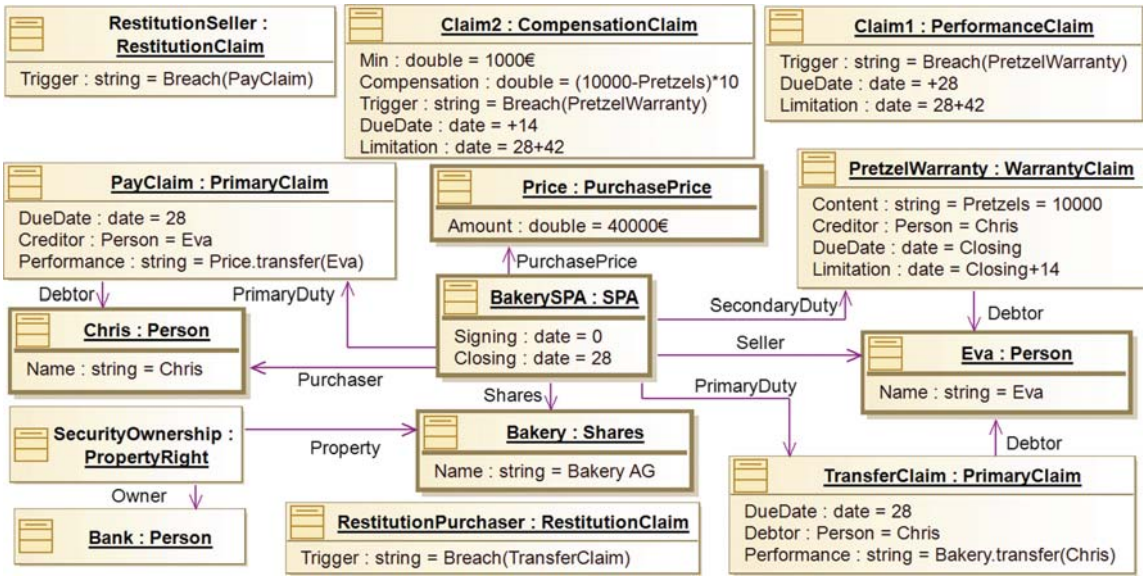

Fig. 2. Object diagram of Bakery SPA

We intend the consistency analyses in Sect. 5 to occur before signing when facts of life are only partially known. In an SPA, for example, a claim can define the performance of a transfer, whereas it is unknown whether and on which day the property relations will change. We use variables to model legal facts, and constrain their possible values according to the constraints specified in the SPA. We formalize an SPA using decidable fragments of first-order logic since it turns out to be ideally suited to accommodate incomplete legal facts.

4.1 Logical Formalization of Contract Entities

An SPA consists of a set of claims with which an execution of the SPA has to comply. Whether a claim is performed in an execution depends on the objective situation and the behavior of the contracting parties that we call *legal facts*. We define an *execution* of an SPA as a combination of legal facts such that for every *primary claim* or *independent claim*, the claim itself or one of its associated *secondary claims* is performed. Notice, that not every claim in an SPA needs to be performed in a specific execution. First, we formalize the legal facts, then the claims and lastly, combine the claims such that a satisfying assignment of the formalization is a possible SPA execution.

Formalization of Legal Facts. An SPA according to the class diagram in Fig. 1 contains a set $\mathcal{P}$ of persons and a set $\mathcal{O}$ of objects. In the formalization, we reference a person p or an object o by the index of p in $\mathcal{P}$ and the index of o in $\mathcal{O}$, which yields unique identifiers.

Dates in an SPA are usually given by calendar dates. Calendar dates are complicated to process due to the many rules that govern them, such as the treatment of a leap year. In the formalization, we simplify the processing and represent dates using integer variables. An execution of an SPA begins on date $d_S = 0$ with the signature of all contract parties and the closing is performed on date d_C by the transfer of the business ownership. For each claim c in an SPA, we create a date d_c for its performance date.

An ownership can only be transferred from the owner to a new owner. Property relations can be explicitly depicted by the class `PropertyRight`, with an association `Owner` towards a `Person` p and an association `Property` towards an `Object` o. We define a set PR and add to it a tuple (p, o) for every instance of `PropertyRight`. We formalize the property rights by an uninterpreted function $owner(Object) : Person$. ϕ_{owner} represents the property relations stated in the SPA.

$$\phi_{owner} = \bigwedge_{(p,o) \in PR} (owner(o) = p) \tag{1}$$

Formalization of the Claims. Having formalized the legal facts in the SPA, we are now ready to formalize the claims in the SPA. We define a set $\mathcal{C}$ which contains the claims in an SPA. The set $\mathcal{C}_I$ contains the *primary claims* and *independent warranty claims* of an SPA. For every *claim* c, the set $\mathcal{C}(c)$ contains every *secondary claim* s with a `Trigger` to c. In the following, we refer to a claim in $\mathcal{C}(c)$ as a *consequence claim* of c.

Each claim c in the SPA contains a `Performance` l_c, which must be performed for the claim to be performed. Either l_c is a logical formula or the name of an operation in Fig. 1. In the latter case, l_c is replaced with the formalization of the pre- and postconditions of the corresponding operation. For example, the conditions of a property transfer p of an object o from the debtor d to a creditor is represented by the constraint $l_c^p \equiv owner(o) = d$. For every claim c there exists a performance date d_c. The value of d_c is either -1, which represents non-performance, or in the interval $[$`DueDate`, `Limitation`$]$. If `DueDate` is undefined, then its default value is the value of date `Arise` of the claim. In case `DueDate` starts with the sign $+$, then we replace it with `Arise` $+$ `DueDate`. We formalize a claim r with $\phi_c \equiv (d_c = -1) \vee ((c.\texttt{DueDate} \leq d_c \leq c.\texttt{Limitation}) \Rightarrow l_c)$.

A `WarrantyClaim` w is a special *secondary claim* since it is independent. For this reason, in our formalization, the constraint $d_w = -1$ encodes that w is met, and otherwise it is breached on a date $d_w \geq 0$. A warranty w is formalized by $\phi_w \equiv (d_w = -1 \Rightarrow l_w) \vee (w.\texttt{DueDate} \leq d_w \leq w.\texttt{Limitation})$.

If a *primary claim* or an *independent claim* is not performed, the `Debtor` is obliged to perform a consequence claim s that is associated to the *claim*. A *primary claim* is breached when it is not performed ($d_c = -1$), while a warranty

is breached on the date d_w of the notification that the warranty `Content` is not met. For the formalization of their consequence claims, we introduce a fresh integer variable d'_c with value -1 when the associated *claim* is performed. The value of d'_c is c.`DueDate` for a breached *primary claim* and the value of d_c for a breached warranty. In case s is a performance claim, its formalization with a performance date d_s and a `Performance` l_c which is defined as $\phi_s \equiv (d'_c < d_s \leq s.\texttt{Limitation}) \Rightarrow l_c$. In case s is a restitution claim, the SPA is withdrawn, formally $\phi_s \equiv d'_c < d_s \leq s.\texttt{Limitation}$. In case s is a compensation claim, the debtor pays a positive compensation l_s to the creditor. The value of l_s is the `Compensation` that specifies the amount of the compensation. The compensation is paid only above a minimum value `Min` and cannot be more than the maximum value `Max`. In the other cases, $l_s =$ ***`Compensation`*** applies, as shown in Formula 2. In every of the above cases, the compensation is performed.

$$\phi_{Compensation} \equiv l_s = \begin{cases} 0, & \textit{if Compensation} < \texttt{Min} \\ \texttt{Max} & \textit{if Compensation} > \texttt{Max} \\ \textit{Compensation}, & \textit{otherwise} \end{cases} \tag{2}$$

The compensation claim is formally a constraint $\phi_s \equiv \phi_{Compensation} \wedge ((d_s = -1 \Rightarrow \textit{Compensation} = 0) \vee (d'_c < d_s \leq s.\texttt{Limitation}))$.

Execution of an SPA. We are now prepared to give the formalization ϕ_{SPA} of SPA executions. ϕ_{SPA} encodes the property rights and the claims in an SPA, and that in an execution of an SPA, for every *primary claim* and *independent claim*, either the claim or one of its consequence claims need to be performed.

$$\phi_{SPA} \equiv \phi_{owner} \wedge \bigwedge_{c \in \mathcal{C}} \phi_c \wedge \bigwedge_{c \in \mathcal{C}_I} (d_c \geq 0 \vee \bigvee_{\forall s \in \mathcal{C}(c)} d_s \geq 0). \tag{3}$$

An SMT solver, such as Z3 [dMB08], can produce a satisfiable model for ϕ_{SPA} in case an execution of the SPA exists. This model then represents an execution of the SPA that is consistent with all constraints specified in the SPA.

For a contracting party, it is preferable to perform the *primary claims* and *independent claims*, and not their consequence claims. We formalize this preference with the help of a special set ϕ_{soft} that is encoded in Z3 using soft-asserts. A constraint in ϕ_{soft} is satisfied if the model still satisfies ϕ_{SPA}, otherwise the constraint is not unsatisfied.

$$\phi_{soft} \equiv \bigwedge_{c \in \mathcal{C}_I} d_c \geq 0 \ \wedge \bigwedge_{s \in \mathcal{C}(c)} d_s = -1 \tag{4}$$

The SMT solver Z3 computes an optimal solution for the partially satisfiable MaxSMT problem $\phi_v \wedge \phi_{soft}$ in which the *primary claims* and *independent claims* are preferably performed. Z3 returns as a model an execution of the SPA that satisfies as few consequence claims as possible.

4.2 Formalization of the Bakery SPA

Legal Facts in the Bakery SPA. The bakery SPA B describes a set of persons $\mathcal{P}^B = \{\texttt{Eva}, \texttt{Chris}, \texttt{Bank}\}$ and an object set $\mathcal{O}^B = \{\texttt{Bakery}, \texttt{Price}\}$ with the shares of the `Bakery` and the purchase price `Price`. `Bakery` is a property of Bank due to the transfer of ownership by way of security, therefore:

$$\phi^B_{owner} \equiv owner(Bakery) = Bank \tag{5}$$

Claims in the Bakery SPA. Eva is obliged by the *TransferClaim* to transfer the shares of the bakery on a day d_u. She has to perform the *TransferClaim* on the day of `Closing` (28) and if she misses that date, `Chris` has a claim on her delivery. Eva can only perform the transfer if she is the owner of `Bakery`. The formalization is the constraint

$$\phi_{TransferClaim} \equiv (-1 = d_u) \vee ((28 \leq d_u) \Rightarrow (owner(Bakery) = Eva)). \tag{6}$$

Furthermore, Chris is obliged by *PayClaim* to pay *Price* to Eva. The condition for the transfer is formally captured by the constraint $owner(Price) = Chris$. The transfer is performed on a day d_z and is due on day 28. The formalization of the *PayClaim* is

$$\phi_{PayClaim} \equiv (-1 = d_z) \vee ((28 \leq d_z) \Rightarrow owner(PurchasePrice) = Chris). \tag{7}$$

If one of these two *primary claims* is not performed, then the related restitution (shorthand: Res.) claim allows the respective creditor on a date ≥ 0 to withdraw from the SPA, formally

$$\phi_{Res.Purchaser} \equiv d_{Res.Purchaser} = -1 \vee PayClaim.DueDate < d_{Res.Purchaser} \tag{8}$$

$$\phi_{Res.Seller} \equiv d_{Res.Seller} = -1 \vee TransferClaim.DueDate < d_{Res.Purchaser}. \tag{9}$$

If the *warranty claim PretzelWarranty* is breached, then Chris notifies this breach on a day $d_g \geq 0$. $d_g = -1$ means that there is no indication of a non-performance and therefore the warranty `Condition` $Pretzels = 10,000$ is met. The formalization for the *PretzelWarranty* is

$$\phi_{PretzelWarranty} \equiv (d_g = -1 \Rightarrow Pretzels = 10000) \vee (28 \leq d_g \leq 28 + 14). \tag{10}$$

In the bakery SPA, the warranty has the consequence `Claim1`, so that the seller can subsequently perform the pretzel guarantee on a date d_n.

$$\phi_{Claim1} \equiv (d_n = -1) \vee (d_g < d_n \leq d_g + 28 \Rightarrow Pretzels = 10000) \tag{11}$$

On the other hand, it may be more advantageous for the debtor to pay a compensation l_s on a date d_s for the `Compensation`. The value of l_s is constrained

by the formula $\phi^s_{Compensation}$. If no compensation occurs ($d_s = -1$), then l_s is 0. The formalization of the *compensation claim* `Claim2` is

$$\phi_{Claim2} \equiv \phi^s_{Compensation} \wedge ((d_s = -1 \Rightarrow l_s = 0) \ \vee (d_g < d_s \leq d_g + 28 + 14)) \tag{12}$$

For a *compensation claim*, l_s is either 0 or lies in the range of values between a minimum value `Min` and a maximum value `Max` of the compensation. For a paid pretzel compensation, l_s must exceed the value of €1,000 and an upper limit `Max` is not specified. l_s of the compensation claim is calculated according to the constraint $\phi^s_{Compensation}$.

$$\phi^s_{Compensation} \equiv l_s = \begin{cases} 0, & \textit{if } (10.000 - \textit{Pretzels}/100) * 1000 \leq 1.000 \\ (10.000 - \textit{Pretzels}/100) * 1000, & \textit{otherwise} \end{cases} . \tag{13}$$

Execution of Bakery SPA. An execution of the bakery SPA does not need to perform every *primary claim* but can also perform an associated consequence claim. The overall encoding of the claims in the bakery SPA is

$$\phi^B_{SPA} \equiv \phi^B_{owner} \wedge (\phi_{TransferClaim} \vee \phi_{Res.Purchaser}) \wedge (\phi_{PayClaim} \vee \tag{14}$$

$$\phi_{Res.Seller}) \wedge (\phi_{PretzelWarranty} \vee \phi_{Claim1} \vee \phi_{Claim2}) \tag{15}$$

A model of ϕ^B_{SPA} should preferably perform *primary claims* or *independent claims*. This is formalized by

$$\phi^B_{soft} \equiv d_u \geq 0 \wedge d_z \geq 0 \wedge d_{Res.Purchaser} = -1 \wedge \tag{16}$$

$$d_{Res.Seller} = -1 \wedge d_g = -1 \ \wedge \ d_n = -1 \tag{17}$$

The bakery SPA is formalized by the constraint $\phi^B_{SPA} \wedge \phi^B_{soft}$. Each satisfying model of this constraint represents a possible execution of the bakery SPA.

5 Contract Analyses

We now propose static and dynamic analyses that find inconsistencies in the object diagram of an SPA.

5.1 Static Analyses

In an SPA, legal elements can be missing. For instance, an SPA does legally not exist if one of the essential entities (*essentialia negotii*) is missing. For instance, according to the German civil code [BGB, §433], the contract of sale must contain, for example, the contracting parties, the purchase object as well as the purchase price. In an SPA, every *primary claim* also needs at least one consequence claim and every claim needs a `DueDate` by which it is to be performed.

We say a contract is *statically inconsistent* if one of these legal elements is missing. When an SPA is drawn up according to the class diagram in Fig. 1, it is possible to statically check whether every essential legal element is present in the SPA. A static inconsistency analysis is easy to implement using static analyses and can be performed during the parsing (syntactic processing of the contract text) of an SPA.

5.2 Dynamic Analyses

It is more complicated to find a dynamic inconsistency in a contract. An SPA describes both legal facts and claims that the respective contracting parties have agreed upon. For example, Eva agrees to transfer the shares of the bakery. In order to transfer ownership of the shares, she must be the owner. This contradicts the fact that the Bank is the owner of the bakery shares. Despite this fact, the overall SPA can be fulfilled, since Chris can withdraw from the contract. However, an SPA where Chris always has to withdraw is not desirable. For an SPA, it is therefore essential, that each claim can be fulfilled individually, and at least one execution of the SPA exists. We propose two semantic analyses for an SPA:

Analysis I Can each claim be performed?
Analysis II Does there exist an execution of the SPA?

Formalization of the Dynamic Consistency Analyses. We now encode analyses I and II for a given SPA using the constraints defined in Sect. 4. If an encoded analysis is satisfiable, then an SMT solver returns a satisfying variable assignment. If it is unsatisfiable, the SMT solver Z3 can return an unsatisfiability core, which is a minimal subset of conditions that contradicts satisfiability. This subset indicates which claims in the SPA contradict another and, hence, causes an inconsistency in the SPA. These claims need to be changed in the SPA in order to obtain a consistent contract. Remember, a *primary claim* c is performed on its date d_c and a *warranty* w is asserted on day d_w. In the following, we abuse notation and ignore that warranties behave differently to simplify the presentation.

Analysis I is encoded in the constraints Φ_c and Φ_s for every claim in $\mathcal{C}$. We already formalized a claim c by a constraint ϕ_c, which has to perform on a date d_c. For every claim, in addition the property relation represented by ϕ_{owner} has to hold. A *primary claim* or an *independent claim* $c \in \mathcal{C}_I$ can be performed when the following constraint is satisfiable.

$$\Phi_c \equiv \phi_{owner} \wedge \phi_c \wedge d_c \geq 0. \tag{18}$$

A consequence claim $s \in \mathcal{C}(c)$ in an SPA usually needs to be performed if, at the same time, the claim c that is the `Trigger` of s is not performed:

$$\Phi_s \equiv \phi_{owner} \wedge (d_c = -1) \wedge \phi_c \wedge \phi_s \wedge d_s \geq 0. \tag{19}$$

If every constraint Φ_c and Φ_s is satisfiable, then every claim of the considered formalized SPA can be performed.

Analysis II checks whether an execution of the SPA exists by assessing the satisfiability of

$$\Phi_{SPA} \equiv \phi_{SPA} \wedge \phi_{soft}. \tag{20}$$

In addition, Analysis II recognizes whether all *primary claims* and *independent claims* can be performed in the same execution. In this case, every constraint in ϕ_{soft} is satisfied.

Dynamic Consistency Analyses of the Bakery SPA. The bakery SPA is given by the object diagram in Fig. 2. *ContractCheck* generates the following analyses and checks them for satisfiability by the SMT solver Z3.

Analysis I checks whether the individual *primary claims* and *independent claims* are fulfillable. The *TransferClaim* is formalized in the Eq. 6 and the corresponding Analysis I is

$$\begin{aligned} \Phi_{TransferClaim} &\equiv \phi^{B}_{owner} \wedge \phi_{TransferClaim} \wedge \ d_u \geq 0 && (21)\\ &\equiv owner(Bakery) = Bank \wedge (d_u = -1 \vee && (22)\\ &\quad (28 \leq d_u \ \Rightarrow owner(Bakery) = Eva)) \ \wedge \ d_u \geq 0. && (23) \end{aligned}$$

owner is a function, therefore the two constraints $owner(Bakery) = Bank$ and $owner(Bakery) = Eva$ are mutually exclusive. As a consequence, the claim cannot be *performed*, which indicates that the contract contains a logical inconsistency. The SMT solver returns these two inconsistent constraints as an unsatisfiability core.

Analysis I for the *PayClaim* (Eq. 7) is

$$\begin{aligned} \Phi_{PayClaim} &\equiv owner(Bakery) = Bank \wedge (d_z = -1 \vee && (24)\\ &\quad (28 \leq d_z \Rightarrow owner(Price) = Chris)) \wedge d_z \geq 0. && (25) \end{aligned}$$

The SPA states that the closing should be performed on day $d_z = 28$. A possible model that an SMT solver computes contains the assignments $d_z = 28$ and $owner(Price) = Chris$. We can conclude that the `PayClaim` can be *performed*.

Analysis I for the *PretzelWarranty* (Eq. 10) is

$$\begin{aligned} \Phi_{PretzelWarranty} \equiv & owner(Bakery) = Bank \ \wedge \ 28 \leq d_g \leq 28 + 14 \wedge && (26)\\ & (d_g = -1 \Rightarrow pretzels = 10000) \wedge d_g = -1. && (27) \end{aligned}$$

$\Phi_{PretzelWarranty}$ is satisfiable for $d_g = -1$ and $pretzels = 10000$. Thus, `PretzelWarranty` can be *performed*.

The Analysis II is encoded in the constraint Φ^B_{SPA}. It can be used to check whether an execution of the bakery SPA exists.

$$\Phi^B_{SPA} \equiv \phi^B_{SPA} \wedge \phi^B_{soft} \tag{28}$$

The SMT solver that we use in our automated analysis searches for a satisfiable assignment and computes, for instance, a model with the date assignments: $d_u = -1, d_z = 28, d_g = -1, d_n = -1, d_s = -1, d_{ResitutionSeller} = -1$, $d_{ResitutionPurchaser} = 29$. This means that the bakery SPA can be performed, even though it is inconsistent since the *primary claim TransferClaim* cannot be performed. This exemplifies that inconsistency of an SPA does not mean that it cannot be performed. Furthermore, as the SMT solver confirms, if the bakery is not assigned by way of security but instead *owner*(`Bakery`) = *Eva* holds, the pretzel bakery SPA would be consistent.

Further Dynamic Analyses

Time Analyses. The *due date* and *limitation* of a claim can refer to other time events in the contract. In the bakery SPA, for instance, the `DueDate` of `Claim1` depends on the assert date of `PetzelWarranty`. The combination of several *due dates* and *limitations* can result in an inconsistency where the *due date* of a claim is after the claim expired because of its *limitation*. Analysis $\Phi_{Limitation\_c}$ checks this inconsistency for every *claim* $c \in \mathcal{C}$ with a `Limitation`. For this analysis, we need to substitute $c.\texttt{DueDate} \leq d_c \leq c.\texttt{Limitation}$ with $c.\texttt{DueDate} \leq d_c$.

$$\Phi_{Limitation\_c} \equiv \phi_{SPA}[c.\texttt{DueDate} \leq d_c / c.\texttt{DueDate} \leq d_c \leq c.\texttt{Limitation}] \wedge \tag{29}$$

$$c.Limitation < c.DueDate \tag{30}$$

In the bakery SPA, the claims `Claim1` and `Claim2` have a `Trigger` to `PretzelWarranty` and a `Limitation` of 70 days.

The SMT solver Z3 computes that $\Phi_{Limitation\_}Claim1$ is unsatisfiable, which entails that the time constraints are consistent. For $\Phi_{Limitation\_}Claim2$, Z3 computes a model that, for instance, contains the following assignments: $d_{Claim1} = -1$, $d_{PretzelWarranty} = 30$ and $d_{Claim2} = 71$. With this execution, Chris asserts `PretzelWarranty` on day 29, then Eva tries to perform `Claim1` for 28 days but fails. In this model, the compensation claim is due after 71 days, even the legal basis of the claim is outdated already after 70 days. This shows that there is an inconsistency in the timing between the due dates and the `Limitation` of the `PretzelWarranty`.

6 The *ContractCheck* Tool

An SPA is usually created in natural language and is not formalized. Natural language processing to automatically obtain a semantic model for the SPA is beyond the scope of this paper. Instead, we provide parameterized structured

English text blocks to the user. We provide a formal semantics of these textblocks within the tool *ContractCheck* [CCT22] that we developed to support the SPA formalization and analysis, so that the user is relieved from the need to provide the formalization manually.

The workflow of the tool is depicted in the diagram in Fig. 3. The user only needs to manually select, combine and parameterize the text blocks. A set of blocks represents a contract and is the input into *ContractCheck*. The inconsistency analyses are performed automatically by *ContractCheck*. *ContractCheck* parses the blocks and translates them into an object diagram. The tool extracts the formal representation of the SPA from this object diagram and generates the analysis code as described in Sect. 5, which Z3 then checks for satisfiability. Finally, *ContractCheck* outputs the results.

Fig. 3. Tool analysis workflow, modeled using BPMN [Obj14]

ID:	Block1
Text:	The seller $seller.Name hereby sells shares of $shares.Name, with all rights and obligations pertaining thereto, to the Purchaser $purchaser.Name, who accepts such sale.
Object:	"spa:SPA", "seller:Person","purchaser:Person", "share:Share", "transfer:PrimaryClaim"
Assignment:	"seller.name=Eva", "purchaser.name=Chris", "spa.Seller=$seller", "transfer.Performance=$shares.transfer($purchaser)", ...

Fig. 4. Excerpt from text block encoding of Bakery SPA

SPA Creation by Text Blocks. A user can create and analyze an SPA in the web interface of *ContractCheck*. For every text block, a mapping from text to the elements of the class diagrams in Fig. 1 is defined. A text block consists of a unique *ID*, a natural language *Text* that is parameterized by *Objects* and value *Assignments* to them. Since the variables in Fig. 1 are of a class type, they represent the formalization of an SPA.

A text block example of the bakery SPA in JSON format is given in Fig. 4. The text block with the ID *Block1* defines the essential components of an SPA: A seller, a buyer, the shares to be sold and a price. The $-character in the text indicates the assignment of a variable value, such as the attribute `Name` for a person. For instance, the assignment to *$seller.name* is currently *Eva*, as defined

in the *Assignment* section of the block. A block may reference the variables of another block by using the $-character. For instance, can the property right *prop* assign to its attribute `Property` the value $Block1_share, which references the variable *share* in *Block1*. The library of formalized text blocks can easily be extended. Using the defined text blocks, a user can create an SPA and have it checked automatically using the above devined analyses in *ContractCheck*.

Representation of Results. The results of the analyses are also depicted in the web interface of *ContractCheck*.

The syntactic analysis outputs text messages for each missing legal entity. The result of a dynamic analysis is either a satisfying model, or an unsatisfiability core. *ContractCheck* depicts a satisfying model by a sequence diagram. For the bakery SPA, a computed execution is depicted in Fig. 5. The constraints contained in the unsatisfiability core are being added due to certain blocks, which *ContractCheck* draws side by side in the web interface, as shown in Fig. 6. The figure shows *Block1* and the block with the chattel mortgage. These are the text blocks that created constraints contained in the unsatisfiability core of $\Phi_{TransferClaim}$. The depicted claims help the user to identify the claims that contradict each other.

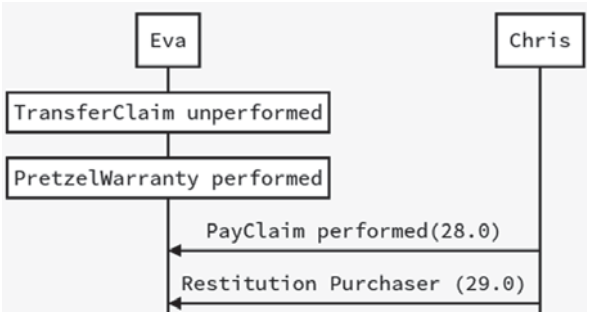

Fig. 5. Bakery SPA execution

Fig. 6. Inconsistence output for the Bakery SPA.

Quantitative Evaluation. We analyzed with *ContractCheck* the bakery SPA and run the dynamic analyses for the claims *TransferClaim*, *PayClaim*, *PretzelWarranty*, *RestitutionSeller* *RestitutionBuyer*, *Claim1*, *Claim2* and the overall contract. The SMT constraint systems that encode the analyses contain at most 21 variables and 63 constraints. The memory demand of the SMT-solver was for every analysis below 18 MB and a model was found within 2 ms. These results show that our analyses encoding is efficient for small SPAs. An analysis of a more realistic size SPA is currently being undertaken.

7 Conclusion

We presented a method for the logical modeling and consistency analysis of legal contracts using the example of an SPA. We provided an ontology for SPAs using UML class diagrams and illustrated the refinement of this ontology to a UML object diagram for a case study. We discussed the logical formalization of the SPA using decidable fragments of first-order logic via SMT solving. We finally introduced the tool *ContractCheck* which allows textual editing of contracts using building blocks, performs the automated derivation of the logical encoding of the contract and the consistency conditions, invokes the Z3 SMT solver, and returns the analysis results to the user. We view this work as an innovative contribution to the enhancement of the quality of complex legal artifacts using logic-based, automated analysis methods.

Future research will increase the scope and complexity of the contract artifacts that we consider. We will also further develop the analysis of the dynamic execution of contracts by introducing state-machine models, among others in order to assess the advantageousness of the contract for different contractual parties in light of the possible dynamic execution scenarios.

Appendix

Text of Bakery SPA

§1 Main Content

1.1 The Seller Anna hereby sells the shares of Bakery AG with all rights and obligations pertaining thereto (including the dividend right for the current financial year), to the Purchaser Chris who accepts such sale. 1.2 The purchaser pays the purchase price 40.000€ to the seller.

1.3 If the transfer is not performed, the Purchaser has the right to withdraw.

1.4 If the pay is not performed, the Seller has the right to withdraw.

§2 The Seller hereby represents and warrants to the Purchaser in the form of an independent guarantee pursuant to Sect. 311 (1) of the German Civil Code and exclusively in accordance with the provisions of this Agreement that the following statements (the "Warranties") are true and correct as of the date of this Agreement and that the warranties set forth in this paragraph will also be true and correct as of the Closing Date:

2.1 The company can produce at least the 10.000 of Pretzels every day (Pretzel Warranty). In case of the breach of the warranty, it needs to be asserted within 14 days.

§3 The Purchaser's rights arising from any inaccuracy of any of the Warranties contained in §1 shall be limited to supplementary performance claims and compensation claims against the Seller, subject to the provisions of

3.1 In case the Pretzel Warranty is not met and then the creditor may demand subsequent performance within 28 business days from the debtor after having transfered the shares.

3.2 In case the Pretzel Warranty is not met and the damage is above 1000€ then a compensation of 100€ per 100 pretzels not baked pretzels is paid within 14 business days.
§4 Claims of §3 expire after 42 business days.
§5 The Bakery AG is transferred by way of security to Bank B.

Text Blocks of Bakery SPA

ID:	Block1
Text:	The seller $seller.Name hereby sells shares of $shares.Name, with all rights and obligations pertaining thereto (including the dividend right for the current financial year), to the Purchaser $purchaser.Name, who accepts such sale.
Object:	"spa:SPA", "seller:Person", "purchaser:Person", "shares:Shares", "transfer:PrimaryClaim"
Assignment:	"purchaser.Name=Chris", "seller.Name=Eva", "spa.Seller=$seller", "spa.Purchaser=$purchaser", "shares.Name=Bakery AG", "spa.Object=$shares", "spa.Claim=$transfer", "spa.Closing=28", "transfer.Performance=Bakery.transfer($purchaser)", "transfer.Debtor=$seller", "transfer.Creditor=$purchaser", "transfer.DueDate=28"

ID:	Block2
Text:	The purchaser pays the purchase price $price.Amount € to the seller on date $payment.DueDate.
Object:	"spa:$SPA", "price:PurchasePrice", "payment:PrimaryClaim"
Assignment:	"spa=$Block1_spa", "spa.Price=$price", "price.Amount=40000", "payment.Debtor=Block1_Purchaser", "spa.Claim=$payment", "payment.Creditor=Block1_Seller", "payment.DueDate=28", "payment.Performance=price.transfer($seller)"

ID:	Block3
Text:	If the $claim is not performed, the $withdraw.Creditor has the right to withdraw.
Object:	"claim:$Claim", "withdraw:RestitutionClaim"
Assignment:	"claim=$Block1_transfer", "withdraw.Name=Restitution Purchaser", "withdraw.Trigger=$claim", "withdraw.Debtor=$claim.Creditor", "withdraw.Creditor=$claim.Debtor"

ID:	Block4
Text:	If the $claim is not performed, the $withdraw.Creditor has the right to withdraw.
Object:	"claim:$Claim", "withdraw:RestitutionClaim"
Assignment:	"claim=$Block2_payment", "withdraw.Name=Restitution Seller", "withdraw.Trigger=$claim", "withdraw.Debtor=$claim.Creditor", "withdraw.Creditor=$claim.Debtor"

ID:	Block5
Text:	The Seller hereby represents and warrants to the Purchaser in the form of an independent guarantee pursuant to Section 311 (1) of the German Civil Code and exclusively in accordance with the provisions of this Agreement that the following statements (the “Warranties) are true and correct as of the date of this Agreement and that the Warranties set forth in this paragraph will also be true and correct as of the Closing Date:

ID:	Block6
Text:	The company can produce at least the $amount of $thing every day. In case of the breach of the warranty, it needs to be asserted within $warranty.Limitation days.
Object:	“warranty:WarrantyClaim”, “count:Integer”, “amount:Integer”, “thing:String”
Assignment:	“warranty.Name=PretzelWarranty”, “warranty.Debtor=$Block1_seller”, “warranty.DueDate=$Block1_spa.Closing”, “thing=Pretzels”, “warranty.Creditor=$Block1_purchaser”, “warranty.Limitation = +14”, “warranty.Performance=(Block6_count=Block6_amount)”, “amount=10000”, “Block1_spa.Claim=$warranty”

ID:	Block7
Text:	The Purchasers rights arising from any inaccuracy of any of the Warranties contained in $block shall be limited to supplementary performance claims and compensation claims against the Seller, subject to the provisions of
Object:	“claim:$Claim”, “per:PerformanceClaim”, “block:$Block”
Assignment:	“block=$Block6”, “claim=$Block6_warranty”, “per.Trigger=$claim”, “per.Name=Claim1”, “per.DueDate=+28”, “per.Debtor=$claim.Debtor”, “per.Creditor=$claim.Creditor”

ID:	Block8
Text:	In case the $claim is not met and then the creditor may demand subsequent performance within $per.DueDate business days from the debtor after having transfered the shares.
Object:	“claim:$Claim”, “per:PerformanceClaim”
Assignment:	“claim=$Block6_warranty”, “per.Name=Claim1”, “per.Trigger=$claim”, “per.DueDate=+28”, “per.Performance=$claim.Performance”, “per.Debtor=$claim.Debtor”, “per.Creditor=$claim.Creditor”

ID:	Block9
Text:	In case the $claim is not met and the damage is above $comp.Min €then a compensation $claim.Performance is paid within $comp.DueDate days.
Object:	“claim:$Claim”, “comp:CompensationClaim”
Assignment:	“claim=$Block6_warranty”, “comp.Name=Claim2”, “comp.Min=1000”, “comp.DueDate=+42”, “comp.Trigger=$claim”, “comp.Compensation=((Block6_amount-Block6_count)/100)*1000”, “comp.Debtor=$claim.Debtor”, “comp.Creditor=$claim.Creditor”

ID:	Block10
Text:	Claims in $block expire after $d business days.
Objects:	“claim:$Claim”, “d:Date”, “block:Block”
Assignment:	“block=Block8”, “d=28+42”, “${//$block//Claim}.Limitation=$d”

ID:	Block11
Text:	The $object is transferred by way of security to $owner.Name.
Objects:	“owner:Person”, “object:$Object”, “prop:PropertyRight”
Assignment:	“owner.Name=Bank”, “object=$Block1_shares”, “prop.Owner=$owner”, “prop.Property=$object”’

References

[BBB09] Balbiani, P., Broersen, J.M., Brunel, J.: Decision procedures for a deontic logic modeling temporal inheritance of obligations. Electron. Notes Theor. Comput. Sci. **231**, 69–89 (2009)

[BGB] Bürgerliches Gesetzbuch, German Civil Code

[BK19] Braegelmann, T., Kaulartz, M.: Rechtshandbuch Smart Contracts. C. H. Beck, Munich (2019)

[BM21] Bonifacio, A.L., Della Mura, W.A.: Automatically running experiments on checking multi-party contracts. Artif. Intell. Law **29**(3), 287–310 (2020). https://doi.org/10.1007/s10506-020-09276-y

[BPS10] Boley, H., Paschke, A., Shafiq, O.: RuleML 1.0: the overarching specification of web rules. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 162–178. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16289-3_15

[CCT22] ContractCheck (2022). https://github.com/sen-uni-kn/ContractCheck

[CM07] Castro, P.F., Maibaum, T.S.E.: A complete and compact propositional deontic logic. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) ICTAC 2007. LNCS, vol. 4711, pp. 109–123. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75292-9_8

[CM08] Castro, P.F., Maibaum, T.S.E.: A tableaux system for deontic action logic. In: van der Meyden, R., van der Torre, L. (eds.) DEON 2008. LNCS (LNAI), vol. 5076, pp. 34–48. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70525-3_4

[CS17] Camilleri, J.J., Schneider, G.: Modelling and analysis of normative documents. J. Log. Algebraic Methods Program. **91**, 33–59 (2017)

[dMB08] de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24

[DNS08] Desai, N., Narendra, N.C., Singh, M.P.: Checking correctness of business contracts via commitments. In: AAMAS (2), pp. 787–794. IFAAMAS (2008)

[EGB+01] Engers, T., Gerrits, R., Boekenoogen, M., Glassée, E., Kordelaar, P.: Power: using UML/OCL for modeling legislation - an application report. In: 8th International Conference on Artificial Intelligence and Law, pp. 157–167. Association for Computing Machinery (2001)

[FMS+17] Faciano, C., et al.: Performance improvement on legal model checking. In: ICAIL, pp. 59–68. ACM (2017)

[GMS10] Gorín, D., Mera, S., Schapachnik, F.: Model checking legal documents. In: JURIX, Frontiers in Artificial Intelligence and Applications, vol. 223, pp. 151–154. IOS Press (2010)
[GMS11] Gorín, D., Mera, S., Schapachnik, F.: A software tool for legal drafting. In: FLACOS, EPTCS, vol. 68, pp. 71–86 (2011)
[Gru18] Grupp, M.: Wie baut man einen rechtsautomaten? In: Hartung, M., Bues, M.-M., Halbleib, G. (eds.) Legal Tech, edge number: 1110. C.H. Beck (2018)
[GS22] Gebele, A., Scholz, K.-S. (eds.): Beck'sches Formularbuch Bürgerliches, Handels- und Wirtschaftsrecht, 14th edn. C.H. Beck (2022)
[HKZ12] Hvitved, T., Klaedtke, F., Zalinescu, E.: A trace-based model for multiparty contracts. J. Log. Algebraic Methods Program. **81**(2), 72–98 (2012)
[HLM20] Henglein, F., Larsen, C.K., Murawska, A.: A formally verified static analysis framework for compositional contracts. In: Bernhard, M., et al. (eds.) FC 2020. LNCS, vol. 12063, pp. 599–619. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-54455-3_42
[HS16] Hill, C.A., Solomon, S.D.: Research Handbook on Mergers and Acquisitions. Edward Elgar Publishing, Cheltenham (2016)
[Kab05] Kabilan, V.: Contract workflow model patterns using BPMN. In: EMMSAD, CEUR Workshop Proceedings, vol. 363, pp. 171–182 (2005). CEUR-WS.org
[KJ03] Kabilan, V., Johannesson, P.: Semantic representation of contract knowledge using multi tier ontology. In: Cruz, I.F. Kashyap, V., Decker, S., Eckstein, R. (eds.) Proceedings of SWDB 2003, The First International Workshop on Semantic Web and Databases, Co-Located with VLDB 2003, Humboldt-Universität, Berlin, Germany, 7–8 September 2003, pp. 395–414 (2003)
[KS16] Kroening, D., Strichman, O.: Decision Procedures - An Algorithmic Point of View. Texts in Theoretical Computer Science. An EATCS Series, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-540-74105-3
[MB15] Mura, W.A.D., Bonifácio, A.L.: Devising a conflict detection method for multi-party contracts. In: SCCC, pp. 1–6. IEEE (2015)
[MDCS10] Martínez, E., Díaz, G., Cambronero, M.-E., Schneider, G.: A model for visual specification of e-contracts. In: IEEE SCC, pp. 1–8. IEEE Computer Society (2010)
[MKK14] Madaan, N., Radha Krishna, P., Karlapalem, K.: Consistency detection in e-contract documents. In: ICEGOV, pp. 267–274. ACM (2014)
[MS22a] Meyer-Sparenberg, W.: Unternehmenskaufvertrag (gmbh-anteile) - käuferfreundlich. In: Gebele, A., Scholz, K.-S. (eds.) Beck'sches Formularbuch Bürgerliches, Handels- und Wirtschaftsrecht. C.H. Beck (2022)
[MS22b] Meyer-Sparenberg, W.: Unternehmenskaufvertrag (gmbh-anteile) - verkäuferfreundlich. In: Gebele, A., Scholz, K.-S. (eds.) Beck'sches Formularbuch Bürgerliches, Handels- und Wirtschaftsrecht. C.H. Beck (2022)
[MSJ22] Meyer-Sparenberg, W., Jäckle, C. (eds.): Beck'sches M&A-Handbuch: Planung, Gestaltung, Sonderformen, regulatorische Rahmenbedingungen und Streitbeilegung bei Mergers & Acquisitions, 2nd edn. C.H. Beck (2022)
[OAS21] OASIS Standard: LegalRuleML, version 1.0 (2021). https://docs.oasis-open.org/legalruleml/legalruleml-core-spec/v1.0/os/legalruleml-core-spec-v1.0-os.pdf
[Obj14] Object Management Group: Business Process Model and Notation 2014. https://www.omg.org/spec/BPMN
[Obj17] Object Management Group: Unified Modelling Language, Specification 2.5.1 (2017). http://www.omg.org/spec/UML

[PF19] Boris, P.: Paal and Martin Fries. Smart Contracts, Mohr Siebeck (2019)
[Pfi22] Pfisterer, B.: Share deal. In: Weise, S., Krauß, H.-F. (eds.) Beck'sche Online-Formulare. C.H. Beck (2022)
[PGR+11] Palmirani, M., Governatori, G., Rotolo, A., Tabet, S., Boley, H., Paschke, A.: LegalRuleML: XML-based rules and norms. In: Olken, F., Palmirani, M., Sottara, D. (eds.) RuleML 2011. LNCS, vol. 7018, pp. 298–312. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24908-2_30
[PPS07] Pace, G., Prisacariu, C., Schneider, G.: Model checking contracts – a case study. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 82–97. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75596-8_8
[PS07] Prisacariu, C., Schneider, G.: A formal language for electronic contracts. In: Bonsangue, M.M., Johnsen, E.B. (eds.) FMOODS 2007. LNCS, vol. 4468, pp. 174–189. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72952-5_11
[PS12] Prisacariu, C., Schneider, G.: A dynamic deontic logic for complex contracts. J. Log. Algebraic Methods Program. **81**(4), 458–490 (2012)
[Sei18a] Seibt, C.H. (ed.): Beck'sches Formularbuch Mergers & Acquisitions, 3rd edn. C.H. Beck (2018)
[Sei18b] Seibt, C.H.: GmbH-Anteilskaufvertrag - ausführlich, käuferfreundlich. In: Beck'sches Formularbuch Mergers & Acquisitions, pp. 324–456. C.H. Beck (2018)
[Sei18c] Seibt, C.H.: Gmbh-anteilskaufvertrag - ausführlich, verkäuferfreundlich, deutsch. In: Beck'sches Formularbuch Mergers & Acquisitions, pp. 233–323. C.H. Beck, München (2018)
[Sei18d] Seibt, C.H.: Gmbh-anteilskaufvertrag - knapp, ausgewogen. In: Beck'sches Formularbuch Mergers & Acquisitions, pp. 515–525. C.H. Beck (2018)
[Sei18e] Seibt, C.H.: Gmbh-anteilskaufvertrag - knapp, verkäuferfreundlich. In: Beck'sches Formularbuch Mergers & Acquisitions, pp. 457–514. C.H. Beck (2018)
[vH20a] von Hoyenberg, P.: Share deal (GmbH, fester kaufpreis). In: Weipert, L., Arnhold, P., Baltus, M. (eds.) Münchener Vertragshandbuch, pp. 228–233. C.H. Beck (2020)
[vH20b] von Hoyenberg, P.: Share deal (GmbH, mit stichtagsbilanzierung). In: Weipert, L., Arnhold, P., Baltus, M. (eds.) Münchener Vertragshandbuch, Beck-online Bücher, pp. 203–227. C.H. Beck (2020)
[VW51] Wright, G.H.V.: Deontic logic. Mind **60**(237), 1–15 (1951)
[WAB20] Weipert, L., Arnhold, P., Baltus, M. (eds.): Münchener Vertragshandbuch: Band 2, 8th edn. C.H. Beck (2020)
[Wil22] Wilhelmi, R.: §453. In: Gsell, B., Krüger, W., Lorenz, S., Reymann, C. (eds.) Beck'scher Online Großkommentar, edge note: 744–782. C.H. Beck (2022)
[WK22] Weise, S., Krauß, H.-F. (eds.): Beck'sche Online-Formulare: Vertrag. C.H. Beck (2022)
[WS05] Wan, F., Singh, M.P.: Formalizing and achieving multiparty agreements via commitments. In: AAMAS, pp. 770–777. ACM (2005)