Detecting Conflicts in Legal Systems

Tingting Li[1], Tina Balke[2,1], Marina De Vos[1], Ken Satoh[3], and Julian Padget[1]

[1] University of Bath, Dept. of Computer Science, UK
{t.li,mdv,jap}@cs.bath.ac.uk
[2] University of Surrey, Centre for Research in Social Simulation, UK
t.balke@surrey.ac.uk
[3] National Institute of Informatics, Principles of Informatics Res. Division, Japan
ksatoh@nii.ac.jp

Abstract. When acting in different jurisdictions (e.g. under the laws of different countries) at the same time, it can be of great value for individuals to be able to determine whether disparities among the laws of these different systems exist and allowing them to identify the consequences that may follow from these disparities. For individuals, it is typically not of interest to find all the ways in which these legal systems differ, but rather to establish whether a particular course of action may have different legal interpretations, depending on the jurisdiction. In this paper we present a formal and computational framework that, given specific scenarios (descriptions of courses of action), can automatically detect whether these scenarios could lead to different outcomes. We demonstrate our approach by means of a private international law case-study where a company drafts a contract clause after examining the consequences in the available jurisdictions.

1 Introduction

An individual or company may be used to doing business in a particular way in the jurisdiction(s) with which they are familiar. Unfortunately, the same *modus operandi* may not be interpreted in the same way under the laws of another country and there may be different requirements, sequences of actions, additional actions, unnecessary actions and *caveats*, compared to their usual behaviour, when engaging in an activity that is partly covered by one jurisdiction and partly by another, or even several others.

This "conflict of laws" [14] is an established area of legal research and provides the motivation for the identification of the legal consequences of differences in the legal interpretation of actions in different jurisdictions, in order to be able to take account of them *a priori*. It is not the paper's intention to be able to find all the disparities or potential conflicts between the given systems, although theory and computational methods do allow for it. Rather, our aim is to demonstrate a mechanism that might be used to determine: (i) whether a particular behaviour on their part could be legally interpreted differently in the pertinent jurisdictions, and (ii) could thus lead to a different, possibly detrimental, possibly beneficial, outcome.

To illustrate our methodology we take a real-world case study described in [6]. The case study focuses on two companies, a British software provider and an Italian software purchaser, that wish to conclude a contract about the purchase of some software.

Whereas the focus in [6] is on examples of conflicting laws, our objective here is to show how our methodology can encode aspects of those laws and thereby enable the detection of the existence of those conflicts.

The case study starts when the companies in question are at the stage of drafting the contract and want to include a liquidated-damage-clause specifying the compensations to be paid in case of insufficient performance with respect to the terms of the contract. When drafting such a clause, it is important for the companies to be able to determine that the clause has the desired effect and is not affected by differences in law that might result in unexpected changes in the handling of the clause. By using our mechanism to analyse a formal representation of the legal text on liquidated damages in both countries, the companies could detect that these clauses differ significantly on one point: namely, according to Art. 1384 of the Italian Civil Code, the liquidated-damages penalty can be diminished equitably by a judge, if the principal obligation was executed in part or if the amount of the penalty was apparently excessive, taking into account the performance of the contractors in respect of the contract. This clause has no counterpart in British Law; an important point of variance between the two legal systems, which could result in significant differences in the liquidated-damage demands. This might be of interest to the two companies in drafting their contract. They might want to address this issue by for example by specifying a jurisdiction for handling claims.

In this paper, we present a formal framework and a computational procedure that, given specific scenarios (descriptions of courses of action) can automatically detect whether these scenarios could be affected by differences between legal systems, thus enabling individuals consequently to account for these cases. Throughout the paper, we use the term conflict informally to denote a difference between the legal interpretation in one system and another. The formal notion of conflict is manifested by the presence of a fluent in one system when it is not present in another: this is explained in greater detail shortly. We use the term jurisdiction to refer the extent or range of some judicial or administrative power and legal system to refer to the set of laws and processes that apply in some jurisdiction. We use the term legal framework to refer to our computational model of some part of such a legal system.

The remainder of the paper is structured as follows: based on ideas presented in previous works of the authors [4], we start by outlining our view of legal frameworks and the corresponding computational model of a single legal framework (Section 2). These ideas are then used to capture *comparative* legal frameworks in Section 3. Consequently, in Sections 3.3 and 3.4 we focus on the detection of conflicts between the law of comparative legal frameworks. We use the private international law case-study outlined above to demonstrate our approach. Our approach however is applicable to finding conflicts between laws in general, not just in the specific case study described. The paper ends with a short summary, conclusions and an outline of future work (Section 5).

2 Legal Frameworks

A Formal Model for Legal Frameworks For modelling legal frameworks we use the Inst*AL* language and its tools [2]. The purpose of the model is to formalize the con-

struction of traces that characterize the result of individuals' actions with respect to a given legal corpus. We call such a formal model of (parts of) a legal system a legal framework. In other application domains such a framework is called an institution or normative framework. In [4], we demonstrated how such a formal model can be applied in a legal context. To make this paper self-contained, we briefly review the model but more details can be in found in [4].

To a first approximation, the model comprises an initial state and a state transformer function that, given an action and a state, determines the successor state. To this we add two particular refinements: (i) a function $\mathcal{G}$ that maps an individual's physical actions $\mathcal{E}_{ex}$, subject to conditions on the legal state, to their corresponding legal interpretation $\mathcal{E}_{act}$; for example the signing of a piece of paper when two witnesses are present may denote the signing of a (legal) contract, and (ii) a function $\mathcal{C}$ that transforms the legal state as a result of an action, be it physical or legal, if the current state matches a specified set of conditions; for example the signing of the contract brings about the legal state in which the signatory is bound by the terms of the contract. Apart from physical actions ($\mathcal{E}_{ex}$), traditionally referred to as exogenous events, and their legal action counterparts ($\mathcal{E}_{act}$), the framework also has violation events ($\mathcal{E}_{viol}$). Legal actions and violations together comprise the legal events ($\mathcal{E}_{legal}$) of a legal framework.

The legal state is modelled by a set of fluents $\mathcal{F}$, which are facts about the legal state that are true by their presence and false in their absence. It is useful to identify several disjoint subsets of fluents within the legal state. The first subset contains information about the domain ($\mathcal{D} \subset \mathcal{F}$). For example, indicating whether a contract was signed or not. The remaining subsets convey information about the actions and events of the system. The fluents $\mathcal{P} \subset \mathcal{F}$ indicate which physical and legal events are permitted. The occurrence of events that are not permitted results in a violation. Legal power fluents ($\mathcal{W} \subset \mathcal{F}$) indicate whether an event has currently the legal power to affect the legal state, for example whether an individual has the legal power to witness a signature. An event that is not empowered has no (legal) effect. Furthermore, actions may have consequences for the individual, such as the obligation ($\mathcal{O} \subset \mathcal{F}$) to take some future action before a certain deadline event. Failure to satisfy the obligation results in a violation. Once the obligation is satisfied or violated the obligation is removed from the state. Conditions (ϕ) on a state are expressed over a subset of the set of all fluents and their negation ($\phi \subset \mathcal{F} \cup \neg\mathcal{F}$). The initial state $\Delta \subseteq \mathcal{F}$ is the set of fluents that are true when the legal framework is created. Putting the foregoing together, we define legal frameworks as a quintuple $\mathcal{L} = \langle \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{C}, \Delta \rangle$.

The semantics of the legal framework is defined over a sequence, called a trace, of exogenous events. Starting from the initial state of the legal framework, for each exogenous event in the trace, we take (i) the transitive closure of $\mathcal{G}$ function augmented with a violation generation function for unfulfilled obligations and non-permitted events to generate all events taking place in the framework, and (ii) for each of these events, the $\mathcal{C}$ relation is used to determine the fluents that need initiation and termination in order to derive the next state. We also terminate obligations that were met or violated. A summary of the formal model appears in Figure 1(a).

The Computational Model For a legal framework to be useful, it needs a corresponding computational model. Hence, a user can, for example, verify if a series of actions results

in a violation of the laws encoded in the framework. The Inst*AL* system [2] translates the formal model to an equivalent computational model that uses Answer Set Programming (ASP) [9]. To make the computational model accessible to a wider audience, [1] proposes an intuitive natural-language based action language for the specification of legal frameworks.

ASP is a declarative programming paradigm for logic programs under answer set semantics. A variety of programming languages for ASP exist. We use $AnsProlog$, as several efficient solvers exist for this language. Like all declarative languages $AnsProlog$ has the advantage of describing the constraints and the solutions rather than writing algorithm to find the solutions to the problem.

The basic components of the language are atoms, elements that can be assigned a truth value. An atom can be negated using *negation as failure*. *Literals* are atoms a or negated atoms $\mathrm{not}\ a$. We say that $\mathrm{not}\ a$ is true if we cannot find evidence supporting the truth of a. Atoms and literals are used to create rules of the general form: $a \leftarrow b_1, ..., b_m, \mathrm{not}\ c_1, ..., \mathrm{not}\ c_n$, where a, b_i and c_j are atoms. Intuitively, this means *if all atoms b_i are known/true and no atom c_j is known/true, then a must be known/true*. We refer to a as the head and $b_1, ..., b_m, \mathrm{not}\ c_1, ..., \mathrm{not}\ c_n$ as the body of the rule. Rules with empty body are called *facts*. Rules with empty heads are known as *constraints*, indicating that no solution should be able to satisfy the body. A *(normal) program (or theory)* is a conjunction of rules and is also denoted by a set of rules. The semantics of $AnsProlog$ is defined in terms of *answer sets*, that is, assignments of true and false to all atoms in the program that satisfy the rules in a minimal and consistent fashion. A program may have zero or more answer sets, each corresponding to a solution.

The mapping of a legal framework consists of three parts: a *base component* which is independent of the legal framework being modelled, the *time component* and the *framework-specific component*. The base component deals with the inertia of fluents, generation of violation events from non-permitted actions and unfulfilled obligations. It also terminates fulfilled and violated obligations. The time component defines the predicates for time and generates a single exogenous event at each time instance.

Figure 1 provides the framework-specific translation rules, including the definition of all the fluents and events as facts. A fluent `p` is declared as $\texttt{ifluent}(\texttt{f})$ and $\texttt{event}(\texttt{e})$ is used for an event. The predicate $\texttt{evtype}(\texttt{e}, \texttt{TYPE})$ specifies the type of the event `e`. The time instant is captured by the predicate $\texttt{instant}(\texttt{T})$. According to $\mathcal{C}$ function, a fluent can be initiated [4] ($\texttt{initiated}(\texttt{f}, \texttt{T})$) or terminated [5] ($\texttt{terminated}(\texttt{f}, \texttt{T})$) by the occurrence of a legal event ($\texttt{occurred}(\texttt{e}, \texttt{T})$). For a given expression $\phi \in \mathcal{X}$, we use $EX(\phi, T)$ to denote the translation of ϕ into a set of ASP literals of the form (`not`) $\texttt{holdsat}(\texttt{f}, \texttt{T})$, denoting that some fluent `f` (does not hold) holds at time `T`. The initial state of the framework is encoded as facts ($\texttt{holdsat}(\texttt{f}, \texttt{i00})$). For the detection of conflicts, three atoms are important: $\texttt{occurred}(\texttt{e}, \texttt{i})$ indicates an event e took place at time instance i, $\texttt{observed}(\texttt{e}, \texttt{i})$ that the exogenous action e was observed at time i and $\texttt{holdsat}(\texttt{f}, \texttt{i})$ that fluent f is true at time i.

[4] by being initiated, the fluent `f` holds true at the successor time instant, i.e. $\texttt{holdsat}(\texttt{f}, \texttt{T})$ holds true at `T`.

[5] by being terminated, the fluent `f` holds false at the successor time instant, i.e.**not** $\texttt{holdsat}(\texttt{f}, \texttt{T})$ holds true at `T`.

$\mathcal{L} = \langle \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{C}, \Delta \rangle$, where

1. $\mathcal{F} = \mathcal{W} \cup \mathcal{P} \cup \mathcal{O} \cup \mathcal{D}$
2. $\mathcal{G} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{E}_{legal}}$
3. $\mathcal{C} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}}$
   where $C(X, e) =$
   $(\mathcal{C}^{\uparrow}(\phi, e), \mathcal{C}^{\downarrow}(\phi, e))$ where
   (i) $\mathcal{C}^{\uparrow}(\phi, e)$ initiates a fluent
   (ii) $\mathcal{C}^{\downarrow}(\phi, e)$ terminates a fluent
4. $\mathcal{E} = \mathcal{E}_{ex} \cup \mathcal{E}_{legal}$
   with $\mathcal{E}_{legal} = \mathcal{E}_{act} \cup \mathcal{E}_{viol}$
5. Δ
6. State Formula: $\mathcal{X} = 2^{\mathcal{F} \cup \neg \mathcal{F}}$

(a)

$$p \in \mathcal{F} \Leftrightarrow \texttt{ifluent(p)}. \quad (1)$$
$$e \in \mathcal{E} \Leftrightarrow \texttt{event(e)}. \quad (2)$$
$$e \in \mathcal{E}_{ex} \Leftrightarrow \texttt{evtype(e,obs)}. \quad (3)$$
$$e \in \mathcal{E}_{act} \Leftrightarrow \texttt{evtype(e,act)}. \quad (4)$$
$$e \in \mathcal{E}_{viol} \Leftrightarrow \texttt{evtype(e,viol)}. \quad (5)$$
$$\mathcal{C}^{\uparrow}(\phi, e) = P \Leftrightarrow \forall p \in P \cdot \texttt{initiated(p,T)} \leftarrow \texttt{occurred(e,T)}, EX(\phi, T). \quad (6)$$
$$\mathcal{C}^{\downarrow}(\phi, e) = P \Leftrightarrow \forall p \in P \cdot \texttt{terminated(p,T)} \leftarrow \texttt{occurred(e,T)}, EX(\phi, T). \quad (7)$$
$$\mathcal{G}(\phi, e) = E \Leftrightarrow g \in E, \texttt{occurred(g,T)} \leftarrow \texttt{occurred(e,T)}, \texttt{holdsat(pow(g),T)}, EX(\phi, T). \quad (8)$$
$$p \in \Delta \Leftrightarrow \texttt{holdsat(p,i00)}. \quad (9)$$

(b)

Fig. 1. (a) Formal specification and (b) translation of legal framework rules into $AnsProlog$

Case Study: Liquidated-damage Clause Having outlined the formal and computational models, we look first at the encoding of the case-study. As mentioned in Sec. 1, the case-study illustrates the process of drafting a new contract between a British software provider and an Italian software purchaser, when they want to know any possible conflicts exist regarding liquidated-damage clauses. A conflict occurs in the determination of the compensation amount, because Italian law specifies that judges can reduce the amount if appropriate, but British law has no such provision. Table 1 and 2 give the formal model for both frameworks and the ASP literals for each element in the model.

Both frameworks share the same set of exogenous events ($\mathcal{E}_{ex}$) which captures actions performed by contractors in the physical world. The event `makeContract` denotes the action of establishing a contract between `Promisee` and `Promissor`. It also specifies that a payment of amount `Payment` is obliged to be paid before deadline `Time`, otherwise a penalty `Fine` can be claimed, as agreed in the contract. The event `pay` represents the event of making a payment in the real world, with amount `Payment`. Moreover, `clock` is used to generate a time counter and signal the deadline later. With the event `demandComp`, `Promissor` can request compensation value at `Fine` from `Promisee`. It can be noticed that the Italian framework has two additional exogenous events: `setActualDamage` which indicates the damages awarded for insufficient performance by the contractors and `reduceComp` which denotes the action of a judge to reduce the compensation. As shown in the table, each exogenous event is mapped to one legal action in the set $\mathcal{E}_{act}$, which are the legal interpretation of physical events and actually change the legal states. The set $\mathcal{E}_{viol}$ consists of violation events for each exogenous and legal event in order to signal the occurrence of non-permitted events and unfulfilled obligations, e.g. the violation event `paymentViolation` is generated to capture the non-performance of the payment obligation. The domain fluents record when a contract is valid (`contract`), when is the promised deadline of the payment (`deadline`), temporal relations (`next`), penalty amount comparison (`lessThan` and

British Law	**Italian Law**
$\mathcal{E}_{ex} = \{\texttt{makeContract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time}),$ $\texttt{pay}(\texttt{ContractID}, \texttt{Payment}), \texttt{clock}(\texttt{Time}),$ $\texttt{demandComp}(\texttt{Promissor}, \texttt{Promisee}, \texttt{ContractID}, \texttt{Fine})\}$ $\mathcal{E}_{act} = \{\texttt{intContract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time}),$ $\texttt{intPay}(\texttt{ContractID}, \texttt{Payment}), \texttt{intDeadline}(\texttt{Time}),$ $\texttt{intComp}(\texttt{Promissor}, \texttt{Promisee}, \texttt{ContractID}, \texttt{Fine})\}$ $\mathcal{E}_{viol} = \{\texttt{viol}(\texttt{e}) \mid e \in (\mathcal{E}_{ex} \cup \mathcal{E}_{act})\} \cup \{\texttt{paymentViolation}(\texttt{ContractID}, \texttt{Payment})\}$ $\mathcal{D} = \{\texttt{contract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time}),$ $\texttt{next}(\texttt{Time}, \texttt{Time}), \texttt{deadline}(\texttt{Time}), \texttt{lessThan}(\texttt{Payment}, \texttt{Payment}),$ $\texttt{comp}(\texttt{Fine}), \texttt{lessThanFine}(\texttt{Fine}, \texttt{Fine})\}$ $\mathcal{W} = \{\texttt{pow}(\texttt{e}) \mid e \in \mathcal{E}_{act}\}$ $\mathcal{P} = \{\texttt{perm}(\texttt{e}) \mid e \in \mathcal{E}\}$ $\mathcal{O} = \emptyset$	$\mathcal{E}_{ex} = \{\texttt{makeContract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time}),$ $\texttt{pay}(\texttt{ContractID}, \texttt{Payment}), \texttt{clock}(\texttt{Time}), \texttt{setActualDamage}(\texttt{Fine}),$ $\texttt{demandComp}(\texttt{Promissor}, \texttt{Promisee}, \texttt{ContractID}, \texttt{Fine}),$ $\texttt{reduceComp}(\texttt{Judge}, \texttt{ContractID}, \texttt{Fine}, \texttt{Fine})\}$ $\mathcal{E}_{act} = \{\texttt{intContract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time}),$ $\texttt{intPay}(\texttt{ContractID}, \texttt{Payment}), \texttt{intDeadline}(\texttt{Time}), \texttt{intDamage}(\texttt{Fine}),$ $\texttt{intComp}(\texttt{Promissor}, \texttt{Promisee}, \texttt{ContractID}, \texttt{Fine}),$ $\texttt{intReduceComp}(\texttt{Judge}, \texttt{ContractID}, \texttt{Fine}, \texttt{Fine})\}$ $\mathcal{E}_{viol} = \{\texttt{viol}(\texttt{e}) \mid e \in (\mathcal{E}_{ex} \cup \mathcal{E}_{act})\} \cup \{\texttt{paymentViolation}(\texttt{ContractID}, \texttt{Payment})\}$ $\mathcal{D} = \{\texttt{contract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time}),$ $\texttt{next}(\texttt{Time}, \texttt{Time}), \texttt{deadline}(\texttt{Time}), \texttt{lessThan}(\texttt{Payment}, \texttt{Payment}),$ $\texttt{comp}(\texttt{Fine}), \texttt{lessThanFine}(\texttt{Fine}, \texttt{Fine}), \texttt{damage}(\texttt{Fine})\}$ $\mathcal{W} = \{\texttt{pow}(\texttt{e}) \mid e \in \mathcal{E}_{act}\}$ $\mathcal{P} = \{\texttt{perm}(\texttt{e}) \mid e \in \mathcal{E}\}$ $\mathcal{O} = \emptyset$
$\mathcal{G}(\mathcal{X}, \mathcal{E})$: $\langle \emptyset, \texttt{makeContract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time}) \rangle \rightarrow$ $\{\texttt{intContract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time})\}$ $\langle \{\texttt{contract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time}),$ $\neg\texttt{lessThan}(\texttt{PaymentMade}, \texttt{Payment})\}, \texttt{pay}(\texttt{ContractID}, \texttt{PaymentMade}) \rangle \rightarrow$ $\{\texttt{intPay}(\texttt{ContractID}, \texttt{Payment})\}$ $\langle \emptyset, \texttt{demandComp}(\texttt{Promissor}, \texttt{Promisee}, \texttt{ContractID}, \texttt{Fine}) \rangle \rightarrow$ $\{\texttt{intComp}(\texttt{Promissor}, \texttt{Promisee}, \texttt{ContractID}, \texttt{Fine})\}$ $\langle \{\texttt{deadline}(\texttt{Time2})\}, \texttt{clock}(\texttt{Time2}) \rangle \rightarrow$ $\texttt{intDeadline}(\texttt{Time2})$	$\mathcal{G}(\mathcal{X}, \mathcal{E})$: $\langle \emptyset, \texttt{makeContract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time}) \rangle \rightarrow$ $\{\texttt{intContract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time})\}$ $\langle \{\texttt{contract}(\texttt{ContractID}, \texttt{Promisee}, \texttt{Promissor}, \texttt{Payment}, \texttt{Fine}, \texttt{Time}),$ $\neg\texttt{lessThan}(\texttt{PaymentMade}, \texttt{Payment})\}, \texttt{pay}(\texttt{ContractID}, \texttt{PaymentMade}) \rangle \rightarrow$ $\{\texttt{intPay}(\texttt{ContractID}, \texttt{Payment})\}$ $\langle \emptyset, \texttt{demandComp}(\texttt{Promissor}, \texttt{Promisee}, \texttt{ContractID}, \texttt{Fine}) \rangle \rightarrow$ $\{\texttt{intComp}(\texttt{Promissor}, \texttt{Promisee}, \texttt{ContractID}, \texttt{Fine})\}$ $\langle \{\texttt{deadline}(\texttt{Time2})\}, \texttt{clock}(\texttt{Time2}) \rangle \rightarrow$ $\texttt{intDeadline}(\texttt{Time2})$ $\langle \emptyset, \texttt{setActualDamage}(\texttt{Fine}) \rangle \rightarrow$ $\texttt{intDamage}(\texttt{Fine})$ $\langle \{\texttt{pow}(\texttt{intReduceComp}(\texttt{Judge}, \texttt{ContractID}, \texttt{AgreedFine}, \texttt{Fine})),$ $\texttt{perm}(\texttt{intReduceComp}(\texttt{Judge}, \texttt{ContractID}, \texttt{AgreedFine}, \texttt{Fine}))\},$ $\texttt{reduceComp}(\texttt{Judge}, \texttt{ContractID}, \texttt{AgreedFine}, \texttt{Fine}) \rangle \rightarrow$ $\texttt{intReduceComp}(\texttt{Judge}, \texttt{ContractID}, \texttt{AgreedFine}, \texttt{Fine})$

Table 1. The British and Italian Liquidated-Damage Legal Frameworks Part 1

British Law cnt.	**Italian Law cnt.**
$\mathcal{C}^{\uparrow}(\mathcal{X},\mathcal{E})$: $\langle\emptyset$, intContract(ContractID, Promisee, Promissor, Payment, Fine, Time)$\rangle \rightarrow$ {contract(ContractID, Promisee, Promissor, Payment, Fine, Time), obl(pay(ContractID, Payment), intDeadline(Time), paymentViolation(ContractID, Payment)), perm(pay(ContractID, AnyPayment)), perm(intPay(ContractID, AnyPayment)), pow(intPay(ContractID, AnyPayment)), deadline(Time)} $\langle\emptyset$, paymentViolation(ContractID, Payment)$\rangle \rightarrow$ {perm(demandComp(Promissor, Promisee, ContractID, Fine)), pow(intComp(Promissor, Promisee, ContractID, Fine)), perm(intComp(Promissor, Promisee, ContractID, Fine))} $\langle\emptyset$, demandComp(Promissor, Promisee, ContractID, Fine)$\rangle \rightarrow$ {comp(Fine)} $\langle\emptyset$, intPay(ContractID, PaymentMade)$\rangle \rightarrow$ {contract(ContractID, Promisee, Promissor, Payment, Fine, Time), perm(pay(ContractID, Payment)), perm(intPay(ContractID, PaymentMade)), pow(intPay(ContractID, PaymentMade))} $\langle\emptyset$, paymentViolation(ContractID, Payment)$\rangle \rightarrow$ {perm(pay(ContractID, PaymentMade)), perm(intPay(ContractID, PaymentMade)), pow(intPay(ContractID, PaymentMade))}	$\mathcal{C}^{\uparrow}(\mathcal{X},\mathcal{E})$: $\langle\emptyset$, intContract(ContractID, Promisee, Promissor, Payment, Fine, Time)$\rangle \rightarrow$ {contract(ContractID, Promisee, Promissor, Payment, Fine, Time), obl(pay(ContractID, Payment), intDeadline(Time), paymentViolation(ContractID, Payment)), perm(pay(ContractID, AnyPayment)), perm(intPay(ContractID, AnyPayment)), pow(intPay(ContractID, AnyPayment)), deadline(Time)} $\langle\emptyset$, paymentViolation(ContractID, Payment)$\rangle \rightarrow$ {perm(demandComp(Promissor, Promisee, ContractID, Fine)), pow(intComp(Promissor, Promisee, ContractID, Fine)), perm(intComp(Promissor, Promisee, ContractID, Fine)), perm(setActualDamage(Fine1)), perm(intDamage(Fine1)), pow(intDamage(Fine1))} $\langle\emptyset$, demandComp(Promissor, Promisee, ContractID, Fine)$\rangle \rightarrow$ {compensation(Fine)} $\langle\emptyset$, intDamage(Fine)$\rangle \rightarrow$ {damage(Fine)} $\langle$\{damage(Fine), lessThanFine(Fine, AgreedFine)\}, intComp(Promissor, Promisee, ContractID, AgreedFine)$\rangle \rightarrow$ {perm(reduceComp(Judge, ContractID, AgreedFine, Fine)), pow(intReduceComp(Judge, ContractID, AgreedFine, Fine)), perm(intReduceComp(Judge, ContractID, AgreedFine, Fine))} $\langle\emptyset$, reduceComp(Judge, ContractID, AgreedFine, Fine)$\rangle \rightarrow$ compensation(Fine)} $\langle\emptyset$, intPay(ContractID, PaymentMade)$\rangle \rightarrow$ {contract(ContractID, Promisee, Promissor, Payment, Fine, Time), perm(pay(ContractID, Payment)), perm(intPay(ContractID, PaymentMade)), pow(intPay(ContractID, PaymentMade))} $\langle\emptyset$, paymentViolation(ContractID, Payment)$\rangle \rightarrow$ {perm(pay(ContractID, PaymentMade)), perm(intPay(ContractID, PaymentMade)), pow(intPay(ContractID, PaymentMade))} $\langle\emptyset$, reduceComp(Judge, ContractID, AgreedFine, Fine)$\rangle \rightarrow$ {Comp(AgreedFine)}
$\Delta = \{$perm(clock(Time)), perm(intDeadline(Time)), pow(intDeadline(Time)), perm(makeContract(ContractID, Promisee, Promissor, Payment, Fine, Time)), pow(intContract(ContractID, Promisee, Promissor, Payment, Fine, Time)), perm(intContract(ContractID, Promisee, Promissor, Payment, Fine, Time))$\}$	$\Delta = \{$perm(clock(Time)), perm(intDeadline(Time)), pow(intDeadline(Time)), perm(makeContract(ContractID, Promisee, Promissor, Payment, Fine, Time)), pow(intContract(ContractID, Promisee, Promissor, Payment, Fine, Time)), perm(intContract(ContractID, Promisee, Promissor, Payment, Fine, Time))$\}$

Table 2. The British and Italian Liquidated-Damage Legal Frameworks Part 2

	British Legal Framework	**Italian Legal Framework**
Effects of paymentViolation	initiated(perm(demandComp(it, gb, contract, 10000)), I) : − occurred(paymentViolation(contract, 500), I), holdsat(live(britishlaw), I), instant(I). initiated(pow(britishlaw, intComp(it, gb, contract, 10000)), I) : − occurred(paymentViolation(contract, 500), I), holdsat(live(britishlaw), I), instant(I). initiated(perm(intComp(it, gb, contract, 10000)), I) : − occurred(paymentViolation(contract, 500), I), holdsat(live(britishlaw), I), instant(I).	initiated(perm(demandComp(it, gb, contract, 10000)), I) : − occurred(paymentViolation(contract, 500), I), holdsat(live(italianlaw), I), instant(I). initiated(perm(intComp(it, gb, contract, 10000), I)) : − occurred(paymentViolation(contract, 500), I), holdsat(live(italianlaw), I), instant(I). initiated(pow(italianlaw, intComp(it, gb, contract, 10000), I)) : − occurred(paymentViolation(contract, 500), I), holdsat(live(italianlaw), I), instant(I). initiated(perm(setActualDamage(contract, 5000)), I) : − occurred(paymentViolation(contract, 500), I), holdsat(live(italianlaw), I), instant(I). initiated(perm(intDamage(contract, 5000)), I) : − occurred(paymentViolation(contract, 500), I), holdsat(live(italianlaw), I), instant(I). initiated(pow(italianlaw, intDamage(contract, 5000)), I) : − occurred(paymentViolation(contract, 500), I), holdsat(live(italianlaw), I), instant(I).
Effects of reduceComp	Not Recognised by British Law	initiated(comp(5000), I) : − occurred(reduceComp(judge, contract, 10000, 5000), I), holdsat(live(italianlaw), I), instant(I). terminated(comp(1000), I) : − occurred(reduceComp(judge, contract, 10000, 5000), I), holdsat(live(italianlaw), I), instant(I).

Table 3. Partial Comparison of British and Italian Legal Framework in $AnsProlog$

lessThanFine), and most importantly for detecting the conflict, the fluent denoting the amount of settled compensation (comp(Fine)).

The action setActualDamage is only meaningful for the Italian framework and the generated legal event intDamage initiates the fluent damage, denoting the value of the actual damage awarded. The permission to perform reduceComp is initiated when the amount of the penalty was excessive compared to the amount of the actual damage, as expressed by damage(Fine) and lessThanFine in the conditions of the rule.

The most significant differences between the two models occurs in the effects after the occurrence of events paymentViolation and demandComp. After observing the violation event paymentViolation, the permission and power of demandComp are initiated in both frameworks, but the Italian framework also initiates the permission of the event setActualDamage. The fluent comp(Fine) is initiated to the amount agreed in the contract by performing event demandComp. However, the event reduceComp in the Italian framework can reduce the penalty amount by terminating the original comp(AgreeFine) and initiating comp(Fine) to denote the new reduced amount of compensation. The differences described above are shown in Table 3.

3 Modelling and Reasoning of Comparative Legal Frameworks

We can now introduce the concept of *comparative legal frameworks*, denoted $C_{\mathcal{L}}$, that comprises a set of individual legal frameworks for conflict analysis, $C_{\mathcal{L}}$ with $\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$. Within $C_{\mathcal{L}}$, all the individual frameworks are still treated as autonomous entities, but $C_{\mathcal{L}}$ provides an interface for the user to interact with the combination, rather than an individual framework.

In this paper we are concerned with the detection of conflicts between different legal frameworks. To be able to compare legal frameworks and the concepts they model, we assume an existing legal ontology that enables semantic alignment between the frameworks. In other words, after semantic alignment, exactly the same fluent or event name is used to express the same concept within all frameworks that are part of $C_{\mathcal{L}}$. This implies that if comp(Fine) denotes the final compensation to be paid in one legal framework, then we assume that exactly the same fluent is used in the other legal frameworks to denote the final compensation settlement.

3.1 Comparative Traces

Once a *comparative legal framework* is formed from a set of individual legal frameworks, users can provide specific cases to analyse. A case is a sequence of exogenous events that captures a sequence of actions performed in the physical world in the context of a comparative legal framework. More precisely, cases can be any possible combination of exogenous events from individual legal frameworks and such cases are defined as *comparative traces*. Each of the exogenous events can be recognised by one or more of the individual frameworks.

Definition 1. *Given a comparative legal framework $C_{\mathcal{L}}$ consisting of legal frameworks $\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$. A* comparative trace *is a sequence $\langle e_1, \ldots e_m \rangle$ such that $\forall e_i, 1 \leq i \leq m : \exists 1 \leq j \leq n : e_i \in \mathcal{E}_{ex}^j$.*

3.2 Null Events

To analyse how a given comparative trace drives the state transition for each individual legal framework, we need to generate individual traces for each individual framework from the given comparative trace. Comparative traces are formed by the events in the union of all possible exogenous events, so some events might be recognised by some legal frameworks, but not by the others, potentially leading to individual traces with different lengths. This implies that even the same event might be associated with different time instances in different individual frameworks. Consequently the state transitions in each individual framework may not be synchronized.

To resolve this technically, we introduce *null events* into our formal model: for each exogenous event in a comparative trace, if it is unknown to some individual framework, then a null event appears in the individual trace. The null events do not change the state, that is null events are not used in either $\mathcal{G}$ or $\mathcal{C}$, but do result in a state transition and therefore guarantee that all individual traces are synchronized. As null events are exogenous events, they need to be permitted in the initial state. For each framework $\mathcal{L}_i \in C_{\mathcal{L}}$, we add $e_{null} \in \mathcal{E}^i_{ex}$ and $\mathrm{perm}(\mathtt{e_{null}}) \in \Delta$ to the formal model.

We now present a simple example from the liquidated-damage clause case study to demonstrate why synchronised traces are necessary for conflict detection. Suppose we have the following comparative trace:

$$CTR = \langle \mathtt{insuffPay}, \mathtt{setDamage(5000)}, \mathtt{demandComp(10000)}, \mathtt{reduceComp(5000)} \rangle.$$

The trace describes a sequence of actions: the Italian company made an insufficient payment `insuffPay`, which caused damage valued at 5,000 Euros `setDamage(5000)`. As a result, the British company demanded compensation of 10,000 Euros from the Italian company `demandComp(10000)`. This is then followed by `reduceComp(5000)`, resulting in the compensation amount being reduced to 5,000. The events `setDamage(5000)` and `reduceComp(5000)` are only recognised by the Italian framework thus the separate traces (without synchronisation) for the British and Italian frameworks respectively are:

$$tr_{gb} = \langle \mathtt{insuffPay}, \mathtt{demandComp(10000)} \rangle$$
$$tr_{it} = \langle \mathtt{insuffPay}, \mathtt{setDamage(5000)}, \mathtt{demandComp(10000)}, \mathtt{reduceComp(5000)} \rangle$$

Therefore, we align the traces by inserting null events `enullGB` at time of the occurrence of event `setDamage(5000)` and `reduceCompen(5000)` for the British trace:

$$tr_{gb} = \langle \mathtt{insuffPay}, \mathtt{enullGB}, \mathtt{demandComp(10000)}, \mathtt{enullGB} \rangle$$
$$tr_{it} = \langle \mathtt{insuffPay}, \mathtt{setDamage(5000)}, \mathtt{demandComp(10000)}, \mathtt{reduceComp(5000)} \rangle$$

Following the occurrence of event `reduceComp(5000)`, Italian framework terminates (and removes) the fluent `comp(10000)` and initiated another fluent `comp(5000)` with new value 5000 that is reduced from 10000. Consequently, we can now detect the expected conflict correctly between fluent `comp(10000)` in S^{GB}_4 and `comp(5000)` in S^{IT}_4 at time instant 4 from the synchronised traces. We call the traces generated from a given CTR and synchronised by means of null events *synchronised traces*.

Definition 2. *Given a comparative trace $CTR = \langle e_1, \ldots, e_t \rangle$ for a comparative legal framework $C_{\mathcal{L}}$ consisting of legal frameworks $\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$, the* synchronised

trace *for any individual legal framework* $\mathcal{L}_i \in C_{\mathcal{L}}$ *w.r.t.* CTR *is the trace* $\langle a_1, \ldots, a_t \rangle$ *with* $a_k = e_k$ *if* $e_k \in \mathcal{E}^i_{ex}$ *and with* $a_k = e_{null}$ *otherwise.*

Based on the definition of comparative trace, we can present the *comparative model* for a comparative legal framework. A comparative model is a set of states sequences over time and each sequence expresses the state transition for each individual framework according to a synchronised trace obtained from CTR.

Definition 3. *Given a comparative trace* CTR *for a comparative legal framework* $C_{\mathcal{L}}$ *with* $\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$*, the corresponding comparative model is the set of models* M_i *with* $1 \leq i \leq n$ *where* M_i *is the model for the synchronised trace of framework* $\mathcal{L}_i$*.*

3.3 Conflict Traces

The problem stated at the outset was to uncover situations whereby a course of actions might be interpreted differently depending on which jurisdiction is used.

This mechanism is formalized here as a comparative trace CTR and the conflicting states can be detected by analysing the corresponding comparative model. A comparative trace CTR is a *conflict trace*, if any two frameworks have a shared fluent holding true in the state of one and false in the corresponding state of the other.

Definition 4. *Given a comparative legal framework* $C_{\mathcal{L}}$ *with a comparative trace* CTR*.* CTR *is a* conflict trace *iff:*

- $\exists \mathcal{L}_i, \mathcal{L}_j \in \mathcal{L}$ *with synchronised models* $M_i = \langle S^i_0, \ldots S^i_t \rangle$ *and* $M_j = \langle S^j_0, \ldots S^j_t \rangle$ *such that*
- $\exists f \in (\mathcal{F}^i \cap \mathcal{F}^j)$ *such that*
- $\exists k, 0 \leq k \leq t$ *such that*
- $f \in S^i_k$ *and* $\neg f \in S^j_k$

We also say a comparative legal framework is conflict-free when none of the possible comparative traces are conflict traces.

3.4 Law Conflict Discovery

In this section, we present an implementation of the theory introduced above. Firstly, we represent each individual framework in InstAL and then translate them automatically into ASP, which sets the stage for conflict detection. One might initially think that we could simply merge the $AnsProlog$ programs for each framework and add the constraints (i) $\texttt{conflict} : -\texttt{holdsat}(\texttt{F}, \texttt{T}), \textbf{not}\ \texttt{holdsat}(\texttt{F}, \texttt{T})$, and (ii) : $-\textbf{not}\ \texttt{conflict}$ to find conflicts with respect to fluent F. However, the same fluent has exactly same representation in each framework, which implies that even if there is a conflict for the fluent F, it can never be detected since the first rule will never hold. We resolve this problem by a simple technical solution in which the events and fluents are consistently renamed. We construct a mapping such that $\texttt{rename}(\texttt{F}, \texttt{FInst}, \texttt{Inst})$ indicates that fluent/event F corresponds to FInst in framework Inst. For example, the fluent $\texttt{comp}(\texttt{Fine})$ is $\texttt{compGB}(\texttt{Fine})$ in britishlaw and $\texttt{compIT}(\texttt{Fine})$ in italianlaw. Two facts establish the mapping: (i) $\texttt{rename}(\texttt{comp}(\texttt{Fine}), \texttt{compGB}(\texttt{Fine}), \texttt{britishlaw})$, and (ii) $\texttt{rename}(\texttt{comp}(\texttt{Fine}), \texttt{compIT}(\texttt{Fine}), \texttt{italianlaw})$.

We define two different conflict atoms, one with zero arguments and the other with three. The first is because we are interested in the occurrence of conflicts. Therefore, the constraint ensures the generation only of answer sets containing conflicts. Thus, if there are no answer sets, there are no conflicts in the comparative legal framework. We could test the following program with all possible comparative traces CTR (programs to generate all CTRs come shortly) to determine whether a comparative legal framework is conflict-free:

$$\begin{aligned}\texttt{conflict} : -\ & \texttt{holdsat}(\texttt{FInX}, \texttt{I}), \textbf{not}\ \texttt{holdsat}(\texttt{FInY}, \texttt{I}),\\ & \texttt{rename}(\texttt{F}, \texttt{FInX}, \texttt{InX}), \texttt{rename}(\texttt{F}, \texttt{FInY}, \texttt{InY}),\\ & \texttt{ifluent}(\texttt{FInX}), \texttt{ifluent}(\texttt{FInY}),\\ & \texttt{instant}(\texttt{I}), \texttt{inst}(\texttt{InX}; \texttt{InY}).\\ : -\ & \textbf{not}\ \texttt{conflict}.\end{aligned}$$

On the other hand, it is also of great value to determine whether a provided CTR will cause any conflicts and what the conflicts are. Thus we also define an atom conflict/3 with the first argument denoting the fluent occurring positive, the second argument indicating the occurrence of the negative counterpart and the third argument being time instant in which the conflict occurs. Compared with the conflict atom with no argument, this one carries more information about a specific conflict the system detected and would be of great use for further analysis.

$$\begin{aligned}\texttt{conflict}(\texttt{FInX}, \texttt{FInY}, \texttt{I}) : -\ & \texttt{holdsat}(\texttt{FInX}, \texttt{I}), \textbf{not}\ \texttt{holdsat}(\texttt{FInY}, \texttt{I}),\\ & \texttt{rename}(\texttt{F}, \texttt{FInX}, \texttt{InX}), \texttt{rename}(\texttt{F}, \texttt{FInY}, \texttt{InY}),\\ & \texttt{ifluent}(\texttt{FInX}), \texttt{ifluent}(\texttt{FInY}),\\ & \texttt{instant}(\texttt{I}), \texttt{inst}(\texttt{InX}; \texttt{InY}).\end{aligned}$$

In order to determine whether a comparative legal framework is conflict-free, we could apply conflict detection to all possible comparative traces. The rules below are designed for this purpose by generating all possible comparative traces CTR as the set of all answer sets.

$$\begin{aligned}\texttt{compEvent}(\texttt{E}) : -\ & \texttt{rename}(\texttt{E}, \texttt{EIn}, \texttt{In}), \texttt{evtype}(\texttt{EIn}, \texttt{ex}), \texttt{evinst}(\texttt{EIn}, \texttt{In}), \texttt{inst}(\texttt{In}).\\ \{\texttt{compObserved}(\texttt{E}, \texttt{I})\} : -\ & \texttt{compEvent}(\texttt{E}), \texttt{instant}(\texttt{I}), \textbf{not}\ \texttt{final}(\texttt{I}).\\ \texttt{ev}(\texttt{I}) : -\ & \texttt{compObserved}(\texttt{E}, \texttt{I}), \texttt{instant}(\texttt{I}).\\ : -\ & \textbf{not}\ \texttt{ev}(\texttt{I}), \texttt{instant}(\texttt{I}), \textbf{not}\ \texttt{final}(\texttt{I}).\\ : -\ & \texttt{compObserved}(\texttt{E1}, \texttt{I}), \texttt{compObserved}(\texttt{E2}, \texttt{I}), \texttt{E1} != \texttt{E2},\\ & \texttt{instant}(\texttt{I}), \texttt{compEvent}(\texttt{E1}), \texttt{compEvent}(\texttt{E2}).\end{aligned}$$

The first rule translates all exogenous events from the individual frameworks to comparative events compEvent/1, from which we then form comparative traces by means of the second rule. $\{\texttt{compObserved}(\texttt{E}, \texttt{I})\}$ indicates a choice to generate the atom or not. If generated, then a $\texttt{ev}(\texttt{I})$ is provided accordingly. The last two constraints guarantee that only one event can be observed/produced at each non-final time instant.

As discussed in Section 3.2, having obtained comparative traces, we now need to separate the traces for each individual legal framework. The observed/2 atoms are produced from compObserved/2 by generating renamed events for each framework that recognises the events or null events otherwise. The program fragment is as follows:

$$\begin{aligned}\texttt{observed}(\texttt{EIn}, \texttt{I}) : -\ & \texttt{compObserved}(\texttt{E}, \texttt{I}), \texttt{rename}(\texttt{E}, \texttt{EIn}, \texttt{In}), \texttt{evinst}(\texttt{EIn}, \texttt{In}),\\ & \texttt{inst}(\texttt{In}), \texttt{instant}(\texttt{I}).\\ \texttt{observed}(\texttt{NullEvent}, \texttt{I}) : -\ & \texttt{compObserved}(\texttt{E}, \texttt{I}), \texttt{rename}(\texttt{E}, \texttt{EIn}, \texttt{In}), \textbf{not}\ \texttt{evinst}(\texttt{EIn}, \texttt{In}),\\ & \texttt{inst}(\texttt{In}), \texttt{instant}(\texttt{I}), \texttt{nullEvent}(\texttt{NullEvent}, \texttt{In}).\end{aligned}$$

3.5 Case Study Conflicts

We can now apply the conflict detection mechanism to the case study presented in Section 2. A comparative trace is provided as follows to detect any conflicts that occur:

$$CTR = \langle \texttt{makeContract}(\texttt{gb}, \texttt{italy}, 1000, 10000, \texttt{deadline}), \texttt{makePayment}(500), \texttt{demandComp}(\texttt{italy}, \texttt{gb}, 10000), \texttt{setActualDamage}(5000), \texttt{reduceComp}(\texttt{judge}, 10000, 5000) \rangle$$

The trace shows that a British company and Italian company have signed a contract specifying the Italian company is obliged to pay the full purchase amount of 1,000 Euros before a certain deadline, otherwise a liquidated-damage penalty of 10,000 Euros will be demanded. However, the Italian company only paid 500 Euros before the deadline, which caused damage valued at 5,000 Euros for the British company. As a result, the British company demands compensation of 10,000 Euros as the amount agreed in the contract. As discussed in Section 2, at this point, referring to different legal texts produces different interpretations and results. For example, the judge may decide to reduce the compensation amount to 5,000 Euros instead according to Italian Civil Code.

The conflict detection program find two conflicts with respect to `compensation`. As can be seen from the result, conflicts occur because of different values are produced to the same fluent from British law and Italian law.

$$\texttt{conflict}(\texttt{compGB}(10000), \texttt{compIT}(10000), 8)$$
$$\texttt{conflict}(\texttt{compIT}(5000), \texttt{compGB}(5000), 8)$$

4 Related Work

In this paper we presented a computational approach for detecting conflicts between different legal systems with the help of legal frameworks. Modelling and reasoning about legal systems with the help of legal frameworks is not a new idea, but has been subject of research for several decades (see [12] for a comprehensive discussion). Of these works, we highlight Dung and Sartor [6] who present a logic-based approach to model private international law and Governatori [10] who proposed the use of *RuleML* for representing and reasoning about clauses of business contracts. Dung and Sartor's work is of particular interest because not only do they focus on private international law as the example in this paper does, but they also analyse the interactions between different legal systems to explain how these can be coordinated. In particular, they adopt *modular argumentation* in which each legal system is modelled as a module, allowing relevant modules to deal with specific queries.

However, there are major differences between our work and theirs: we present a generalised methodology to model legal systems and illustrate how different jurisdictions change their legal state in response to the exogenous events and from which, as a result, conflicts might emerge and can be detected. In contrast, they focus on the representation of individual examples and their specific resolution.

There are also some other legal modelling methodologies in the literature addressing different foci and motivations. For example, Governatori et al [11] present a legal modelling approach based on *temporal defeasible logic (TDL)*. They focus on capturing the temporal properties of legal effects, such as persistence and retroactivity. We capture similar concepts by inertial and non-inertial fluents in our model specification.Governatori [10] converts business contracts from natural language into executable

rules based on *RuleML* with the aim of monitoring contract execution. Moreover, conflicting rules are assumed to be identified beforehand in order to be able to establish precedence relations between them. No mechanism is provided to identify conflicts automatically, in contrast to our approach.

Several other works address potential conflicts in legal or normative settings, such as: [8,13,14,18,17]. Of these, [8,13,14] assume that the legal specifications of the systems to be analysed can be altered over time and proposes mechanisms to deal with conflicts detected. Our paper takes the viewpoint of the designer who wishes to check that the system is conflict-free. Although [8] and [14] present a formal definition of a conflict that is similar to the one presented here, no computational mechanisms for detecting conflicts are provided, and it is assumed that all conflicts are known *a priori*. This is unlikely in the situation described in this paper, because actors (companies in this case) are unlikely to have detailed legal knowledge but need to explore their specific business situation with respect to the consequences arising from interacting with different legal systems.

Vasconcelos et al. [18,17] concentrate on both the detection and the resolution of conflicts between legal systems. They apply *first-order unification* to discover overlapping substitutions to the variables of laws/norms in which legal/norm conflicts may occur. Conflicts are then resolved by annotating a norm with an *undesirable* set specifying values its variables should not have in order to avoid overlaps and hence conflicts. This approach allows for the detection of conflicts between norms relating to single actions. In contrast to their work, we aim to uncover a broader class of conflicts between normative frameworks, namely those that may emerge as consequences of a sequence of events (here specified as a trace). We contend that our approach is more suitable for legal reasoning, because many conflicts between laws are not easily observed by the static comparison of legal texts, but rather may only arise as a consequence of specific legal cases and can be only precisely detected through continuous comparison of the changing legal states and consequences.

5 Conclusion and Future Work

Legal conflicts are a common issue when different jurisdictions are applicable in the same case. It is of great value for individuals to be able to determine whether any disparities exist between pertinent jurisdictions in general, or whether a course of actions would result in any conflicts of law leading to unforeseen outcomes, in particular in the case of penalties. In this paper, we present a formal and computational framework to model jurisdictions and then automatically detect possible conflicts between them. The approach presented has, we believe, no intrinsic limitation to our private international case study, but is generally applicable to conflict-finding between different jurisdictions, or indeed any other type of framework that may be composed or compared. Examples of such frameworks could be the detection of conflicts between revised and existing laws, or the comparison of two different existing contracts. Our approach performs as good as the traces the users provide. The more details (events) provided with the traces, the more accurate conflicts can be detected. If users are interested in all conflicts among legal systems, then all possible traces need to be examined. This might bring some

concern on computational complexity because generating all possible traces is computationally expensive. However, it is possible to compute the reachable state space of comparative legal framework.

We identify several issues for future work. The first concerns extensions of the ideas set out here, where we consider only conflicts between permission and prohibition. Another typical sources of conflict arises between obligation and prohibition. A comprehensive overview of legal conflicts is provided in [14], which points out more directions for the future applications. We should note that our system can detect all types of legal conflicts presented in [14].

The second is to address a potentially useful and novel application domain, in that it would be interesting and valuable to detect conflicts between existing laws and proposed revisions: some may be intentional, others not.

The third issue is that of semantic alignment. In this paper we make an assumption that the legal frameworks are already semantically aligned as a result of using a common established legal ontology. More precisely, we assume that there is a one-to-one mapping between concept and representation. This assumption can be relaxed by means of a legal ontology that unifies a set of representations with a legal concept. This topic has been studied for the last two decades and is regarded as the bridge between legal theory and AI & Law in [16]. Generally speaking, legal ontology is able to provide a means to establish a shared conceptualisation way of any legal systems, by which the legal entities and notions can be represented by disambiguate logical propositions. In the future, we will try to tackle this problem based on some developed ontological frameworks. For example, *JurWordNet* [7] and *LOISWordNet* [5] are typical semantic matching systems specially designed for legal domain. They enables the search in legal documents by using layman's search terms and in response with legal professional terms. These two systems are actually extensions to a generic ontological framework *WordNet* [15]. Inspired by this, we can consider how to establish an ontological framework by linking legal codes to Inst*AL* representations. Using this ontology we could then interpret consistently elements that have different representations but the same intended meaning in different legal frameworks.

A fourth point for future work is conflict resolution. While not applicable to the case-study, as the laws modelling in the frameworks cannot be altered by the participants, other application domains could benefit from this. An example would be when new laws cause unintentional conflict with existing laws. In that case, more revisions might be needed. Some work on revision already exists. Sartor [14] proposes classic ordering strategies over laws using either a belief change function or a non-monotonic reasoning approach. An alternative resolution approach is proposed by García-Camino et. al [8]. The authors resolve conflicts by removing laws with lower priority. Our intention is to work out how to revise laws, rather than deleting or ignoring them. With a formal model and corresponding AnsProlog encoding, we believe that Corapi et al.'s [3] can provide us with both the theoretical and computational model to do so.

References

1. O. Cliffe, M. De Vos, and J. Padget. Specifying and reasoning about multiple institutions. In P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, and E. Mat-

son, editors, *Coordination, Organization, Institutions and Norms in Agent Systems II*, volume 4386 of *LNCS*, pages 67–85. Springer, 2007.
2. O. Cliffe, M. De Vos, and J. A. Padget. Answer set programming for representing and reasoning about virtual institutions. In K. Inoue, K. Satoh, and F. Toni, editors, *Computational Logic in Multi-Agent Systems*, volume 4371 of *LNCS*, pages 60–79. Springer, 2007.
3. D. Corapi, A. Russo, M. D. Vos, J. A. Padget, and K. Satoh. Normative design using inductive learning. *TPLP*, 11(4-5):783–799, 2011.
4. M. De Vos, J. Padget, and K. Satoh. Legal modelling and reasoning using institutions. In T. Onoda, D. Bekki, and E. McCready, editors, *New Frontiers in Artificial Intelligence (JSAI-isAI 2010 Workshop)*, pages 129–140. Springer, 2011.
5. L. Dini, W. Peters, D. Liebwald, E. Schweighofer, L. Mommers, and W. Voermans. Cross-lingual legal information retrieval using a wordnet architecture. In *Proceedings of the 10th international conference on Artificial intelligence and law*, pages 163–167. ACM, 2005.
6. P. M. Dung and G. Sartor. The modular logic of private international law. *Artificial Intelligence and Law*, 19:233–261, 2011.
7. A. Gangemi, M. Sagri, and D. Tiscornia. Metadata for content description in legal information. In *Procs. of LegOnt Workshop on Legal Ontologies*, 2003.
8. A. García-Camino, P. Noriega, and J.-A. Rodríguez-Aguilar. An algorithm for conflict resolution in regulated compound activities. In *Engineering Societies in the Agents World VII*, volume 4457, pages 193–208. Springer, 2007.
9. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.
10. G. Governatori. Representing business contracts in ruleml. *International Journal of Cooperative Information Systems*, 14(2–3):181–216, 2005.
11. G. Governatori, A. Rotolo, and R. Rubino. Implementing temporal defeasible logic for modeling legal reasoning. *New Frontiers in Artificial Intelligence*, pages 45–58, 2010.
12. A. J. I. Jones and M. Sergot. On the characterization of law and computer systems: the normative systems perspective. In J.-J. C. Meyer and R. J. Wieringa, editors, *Deontic logic in computer science*, pages 275–307. John Wiley & Sons Ltd., 1993.
13. M. J. Kollingbaum, T. J. Norman, A. Preece, and D. Sleeman. Norm refinement: Informing the re-negotiation of contracts. In *ECAI 2006 Workshop on Coordination, Organization, Institutions and Norms in Agent Systems, COIN@ECAI 2006*, pages 46–51, 2006.
14. G. Sartor. Normative conflicts in legal reasoning. *Artificial Intelligence and Law*, 1:209–235, 1992.
15. M. Stark and R. Riesenfeld. Wordnet: An electronic lexical database. In *Proceedings of 11th Eurographics Workshop on Rendering*. Citeseer, 1998.
16. A. Valente and J. Breuker. Ontologies: The missing link between legal theory and ai & law. In *JURIX*, volume 94, pages 139–149. Citeseer, 1994.
17. W. Vasconcelos, M. Kollingbaum, and T. Norman. Normative conflict resolution in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 19(2):124–152, 2009.
18. W. Vasconcelos, M. J. Kollingbaum, and T. J. Norman. Resolving conflict and inconsistency in norm-regulated virtual organizations. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 644–651. ACM, 2007.