A Framework for Conflict Analysis of Normative Texts Written in Controlled Natural Language

Krasimir Angelov, John J. Camilleri, and Gerardo Schneider*

Department of Computer Science and Engineering
Chalmers | University of Gothenburg, Sweden

Abstract

In this paper we are concerned with the analysis of normative conflicts, or the detection of conflicting obligations, permissions and prohibitions in normative texts written in a Controlled Natural Language (CNL). For this we present AnaCon, a proof-of-concept system where normative texts written in CNL are automatically translated into the formal language $\mathcal{CL}$ using the Grammatical Framework (GF). Such $\mathcal{CL}$ expressions are then analysed for normative conflicts by the CLAN tool, which gives counter-examples in cases where conflicts are found. The framework also uses GF to give a CNL version of the counter-example, helping the user to identify the conflicts in the original text. We detail the application of AnaCon to two case studies and discuss the effectiveness of our approach.

Key words: normative texts, e-contracts, legal contracts, controlled natural language, CLAN, CL, conflict analysis, grammatical framework

Contents

1 Introduction **2**

2 Background **4**
2.1 The Contract Language $\mathcal{CL}$ 5
2.1.1 Example 6
2.2 CLAN 7
2.3 Controlled Natural Languages (CNLs) 8
2.4 The Grammatical Framework 9

☆The research leading to these results has partly received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-247914.

*Corresponding author's address: Chalmers University of Technology, Dept. of Computing Science and Engineering. SE-412 96 Gothenburg, Sweden.

Email address: {krasimir, john.j.camilleri}@chalmers.se, gerardo.schneider@gu.se (Krasimir Angelov, John J. Camilleri, and Gerardo Schneider)

3 The AnaCon framework **10**
3.1 System workflow 11
3.2 About the CNL 12
3.3 Linearisation and parsing in GF 15

4 Case studies **19**
4.1 Case Study 1: Airline check-in process 20
4.2 Case Study 2: Internet Service Provider 24
4.3 Some reflections concerning the case studies 29

5 Related work **30**

6 Conclusion **32**
6.1 Limitations 33
6.2 Future work 34

A Cases study 1: Airline check-in process **38**
A.1 Contract CNL 38
A.1.1 First version 38
A.1.2 Second version 39
A.1.3 Third version 39
A.2 Generated action dictionaries 40
A.2.1 First version 40
A.2.2 Second version 41
A.2.3 Third version 41

B Case study 2: Internet service provider **42**
B.1 Contract CNL 42
B.1.1 First version 42
B.1.2 Second version 43
B.1.3 Third version 44
B.2 Generated action dictionaries 45
B.2.1 First version 45
B.2.2 Second and third version 45

1. Introduction

Descriptions and prescriptions of procedures and behaviour are normally presented in a language that end users understand, and thus written in *natural languages* (NL) like English, Swedish or Spanish. In particular we refer here to *normative texts*: documents containing a set of *norms* prescribing procedures and behaviours often collected together as the notions of obligation, permission and prohibition. This covers a wide a range of documents including traditional

legal contracts, requirement specifications, work descriptions, regulations and service-level agreements (SLAs), just to mention a few.[1]

Our main interest in this paper is the formal representation and analysis of such documents, in particular being able to detect and solve *normative conflicts*. By this we mean cases where a document contains one or more clauses contradicting themselves in the sense that they stipulate conflicting obligations and/or permissions, or the simultaneous prohibition and obligation/permission of performing a task. Such conflicts in real-world contracts can easily have adverse legal implications, and the motivation for writing contracts which are conflict-free is clear. Yet avoiding such conflicts when writing such documents is non-trivial, in particular because of the often ambiguous language in which they are written.

It is well known that NL texts can have multiple interpretations due to context sensitivity, underspecified terminology, or simply bad use of language. Documents written in NL are thus in general difficult to analyse algorithmically (automatically or semi-automatically). In order to do so, such texts need to be translated into a language having formal syntax and semantics, that is into a *formal language*. The advantage of using formal languages is that they are precise and unambiguous, and in many cases tools are available which provide the possibility of (semi-) automatic analysis. The drawback is that the use of formal languages often requires a high level of expertise, not only at the syntactic level in the use of the language for specifying properties, but also in the interpretation of the results from such analysis tools. This is also the case for the so-called "push-button technologies" such as model checking, where one still needs to write the properties in a logic and interpret the counter-examples, which are usually given as a long formula representing the trace leading to the problematic case.

The ideal situation for an end user would be to have the benefit of both worlds; the simplicity and familiarity of NL combined with the power of formal methods, without having to be involved in the technical details of the latter. The current state-of-practice is however far from this ideal situation. Though the state-of-the-art on NL processing has advanced quite a lot in recent years, we still have to depend on the use of formal languages and techniques to analyse such normative texts. A relatively new trend is to restrict the use of NL in order to get something that "looks like a NL" but having a better structure, and if possible avoiding ambiguity. We call such constrained languages *restricted* or *controlled* natural languages (CNLs) [1]. The syntax of a CNL resembles the syntax of a NL, but it includes only some of the syntactic constructions that are allowed in the full language. This also makes it possible to have a precise semantics for the CNL, which would be impossible for the full NL. By using CNL instead of pure logic, users who are not experts in the underlying logic can easily both create and read new formal contracts, while at the same time

[1] In the rest of the paper we use the word 'contract' to refer not only to legal contracts, but to this more general class of normative texts.

not being forced to deal with the issues that arise when unrestricted natural language is used.

In this paper we present the AnaCon framework as a proof-of-concept system for the analysis of normative texts. We start by considering NL contracts taken from the real world, and describe a CNL which attempts to represent them in a meaningful way. We then explain and demonstrate the use of the AnaCon framework to transform such CNL contracts into expressions in a formal language which can then be analysed with a conflict detection tool. AnaCon also allows the translation of counter-examples (witnessing the existence of conflicting clauses) back into our CNL, facilitating the identification of the problem in the original text.

A conceptual model of AnaCon was first introduced in the workshop paper [2]. In this work we keep the same fundamental idea introduced there and consider the same class of contracts—namely those which can be expressed as formulae in the formal language $\mathcal{CL}$ and thus processed with the CLAN analysis tool. We have thus not changed the name of the framework, though most of the system design and implementation of the individual sub-modules has been changed significantly (*cf.* related work section). In summary, the contributions of this paper are:

1. The definition and implementation of a CNL for writing normative texts. The CNL analyser implemented allows the parsing of full sentences by identifying relevant verbs—in particular those connoting obligations, permissions and prohibitions.
2. A formal syntax for the input file format to AnaCon, along with a parser that automatically extracts action names from the CNL text, taking away from the user the burden of including an action dictionary.
3. A complete implementation of AnaCon. We provide fully-working versions of all the modules described in the framework, including the translation from resulting counter-examples in the formal language $\mathcal{CL}$ back into our CNL.
4. The application of AnaCon to 2 case studies: i) A work description procedure for an airport check-in desk ground crew, and ii) A legal contract between an Internet provider and a client.

The paper is organised as follows. In the next section we recall the necessary technical background the rest of the paper is based on, including $\mathcal{CL}$, CLAN, CNLs and GF. In Section 3 we present our framework in general terms, and provide some details on the implementation of AnaCon. We then go into the application of the framework on two separate case studies in Section 4, as proof-of-concepts to show the feasibility of our approach. Before concluding in the last section, we discuss related work in Section 5.

2. Background

In this section we present the background relevant to understanding the main components of AnaCon. We first introduce the contract language $\mathcal{CL}$, and continue with a description of the conflict analysis tool CLAN. We then discuss

$$
\begin{array}{rcl}
C & := & C_O \mid C_P \mid C_F \mid C \wedge C \mid [\beta]C \mid \top \mid \bot \\
C_O & := & O_C(\alpha) \mid C_O \oplus C_O \\
C_P & := & P(\alpha) \mid C_P \oplus C_P \\
C_F & := & F_C(\alpha) \\
\alpha & := & 0 \mid 1 \mid a \mid \alpha \& \alpha \mid \alpha.\alpha \mid \alpha + \alpha \\
\beta & := & 0 \mid 1 \mid a \mid \beta \& \beta \mid \beta.\beta \mid \beta + \beta \mid \beta^*
\end{array}
$$

Figure 1: $\mathcal{CL}$ syntax.

controlled natural languages in general and finish with a presentation of the Grammatical Framework.

2.1. The Contract Language $\mathcal{CL}$

The formal language $\mathcal{CL}$ has been designed for specifying contracts containing clauses determining the obligations, permissions and prohibitions of the involved parties [3, 4, 5]. $\mathcal{CL}$ is inspired by dynamic, temporal, and deontic logic, and combines concepts from each. Being *action-based*, modalities in $\mathcal{CL}$ are applied to actions and not to *state-of-affairs*. Complex actions can be expressed in the language by using operators for choice, sequence, conjunction (concurrency) and the Kleene star. $\mathcal{CL}$ also allows the expression of what penalties (*reparations*) apply when obligations and prohibitions are not respected, which form a central part of how contracts are defined and used.

For these reasons, $\mathcal{CL}$ was chosen for the underlying representation of the class of contracts in which we are interested. Combined with the availability of the conflict-detection tool CLAN (section 2.2), $\mathcal{CL}$ forms the formal basis of the AnaCon framework.

In what follows we present the syntax of $\mathcal{CL}$, and give a brief intuitive explanation of its notations and terminology, following [4]. A contract in $\mathcal{CL}$ may be obtained by using the syntax grammar rules shown in Fig. 1.

$\mathcal{CL}$ contracts in general consist of a conjunction of clauses representing (conditional) normative expressions, as specified by the initial non-terminal C in the definition. A contract is defined as an obligation (C_O), a permission (C_P), a prohibition (C_F), a conjunction of two clauses or a clause preceded by the dynamic logic square brackets. $\top$ and $\bot$ are the trivially satisfied and violating contracts respectively. O, P and F are deontic modalities; the obligation to perform an action α is written as $O_C(\alpha)$, showing the primary obligation to perform α, and the reparation contract C if α is not performed. This represents what is usually called in the deontic community a *Contrary-to-Duty* (*CTD*), as it specifies what is to be done if the primary obligation is not fulfilled. The prohibition to perform α is represented by the formula $F_C(\alpha)$, which not only specifies what is forbidden but also what is to be done in case the prohibition is violated (the contract C); this is called *Contrary-to-Prohibition* (*CTP*). Both

CTDs and CTPs are useful to represent normal (expected) behaviour, as well as alternative (exceptional) behaviour. $P(\alpha)$ represents the permission of performing a given action α. As expected there is no associated reparation, as a permission cannot be violated.

In the description of the syntax, we have also represented what are the allowed actions (α and β in Fig. 1). It should be noted that the usage of the Kleene star (*)—which is used to model repetition of actions—is not allowed inside the above described deontic modalities, though they can be used in dynamic logic-style conditions. Indeed, actions β may be used inside the dynamic logic modality (the bracket $[\cdot]$) representing a condition in which the contract C must be executed if action β is performed. The binary constructors (&, ., and +) represent (true) concurrency, sequence and choice over basic actions (e.g. "buy", "sell") respectively. Compound actions are formed from basic ones by using these operators. Conjunction of clauses can be expressed using the $\wedge$ operator; the exclusive choice operator ($\oplus$) can only be used in a restricted manner. 0 and 1 are two special actions that represent the impossible action and the skip action (matching any action) respectively.

The concurrency (or *synchrony*) action operator & should only be applied to actions that can happen simultaneously. $\mathcal{CL}$ offers the possibility to explicitly specify such actions by defining the following relation between actions: $a\#b$ if and only if it is not the case that $a\&b$. We call such actions *mutually exclusive* (or *contradictory*). An example of such actions would be *"the ground crew opens the check-in desk"* and *"the ground crew closes the check-in desk"*, which intuitively cannot occur at the same time.

It is worth mentioning that much care has been taken when designing $\mathcal{CL}$ to avoid deontic paradoxes, as this is a common problem when defining a language formalising normative concepts (*cf.* [6]). Besides, $\mathcal{CL}$ enjoys additional properties concerning the relation between the different normative notions, as for instance that obligations implies permissions, and that prohibition may be defined as the negation of permission. It has also been proven that some undesirable properties do not hold, such as that the permission of performing a simple action does not imply the permission of performing concurrent actions containing that simple action (similarly for prohibitions). See [3, 7] for a more detailed presentation of $\mathcal{CL}$, including a proof of how deontic paradoxes are avoided as well as the properties of the language.

2.1.1. Example

As an example of how $\mathcal{CL}$ can be used to represent contracts, let us consider the following sample clause:

> *The ground crew is obliged to open the check-in desk and request the passenger manifest two hours before the flight leaves.*

Taking a to represent *"two hours before the flight leaves"*, b to be *"the ground crew opens the check-in desk"*, and c to be *"the ground crew requests the passenger manifest"*, then this clause could be written in $\mathcal{CL}$ as $[a]O(b\&c)$. We may also wish to include an additional reparation clause, such as:

If the ground crew does not do as specified in the above clause then a penalty should be paid.

This penalty must be applied in case the ground crew does not respect the above obligations. Assuming that p represents the phrase *"paying a fine"*, one would capture all the above in $\mathcal{CL}$ as $[a]O_{O(p)}(b\&c)$.

This example serves not only to provide samples of normative statements written in $\mathcal{CL}$, but also to highlight the significant gap between natural language descriptions and formal representations of contracts. This paper attempts to bridge this gap through the introduction of an intermediary controlled natural language (CNL) to reconcile these two distinct representations. More background on CNLs can be found in section 2.3.

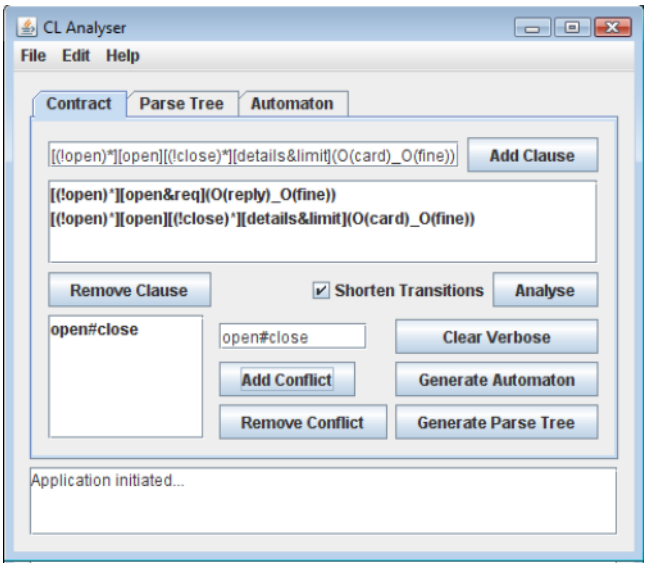
2.2. CLAN

CLAN[2] is a tool aimed at the detection of normative conflicts in contracts written in $\mathcal{CL}$, giving the possibility for automatically generating a monitor for the $\mathcal{CL}$ formula [8]. There are four main kinds of conflicts in normative systems. The first arises when there is an obligation and a prohibition to perform the same action. Such cases will inevitably lead to a violation of the contract, independently of what the performed action is. The second type of conflict happens when there is a permission and a prohibition on the same action, which may or may not lead to a contradicting situation. The other two cases occur when there is an obligation to perform mutually exclusive actions, and when there exist both a permission *and* an obligation to perform mutually exclusive actions.

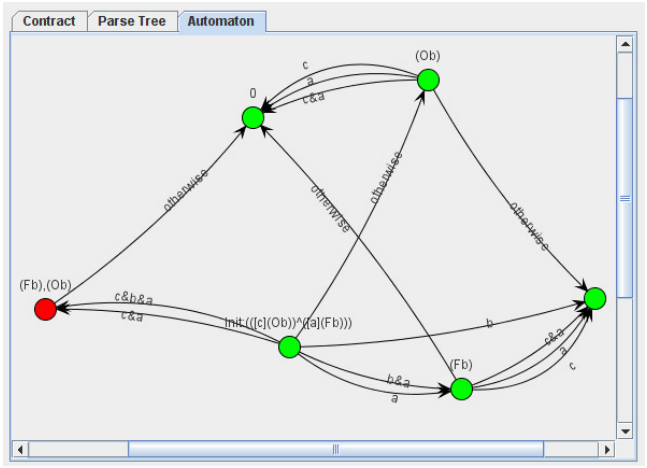
The core of CLAN is implemented in Java, consisting of just over 700 lines of code. The tool provides a graphical user interface as shown in the screen shot depicted in Fig. 2(a). CLAN allows the user to input a $\mathcal{CL}$ contract together with a list of the actions to be considered mutually exclusive. If a conflict is detected, CLAN gives a counter-example trace "explaining" where the conflict arises and giving a sequence of actions realising the path to that conflict state. It is possible to visualise the corresponding automaton, as for instance shown in Fig. 2(b). The complexity of the automaton increases exponentially on the number of actions, since all the possible combinations to generate concurrent actions must be considered.

The analysis provided by CLAN enables the discovery of undesired conflicts. This is particularly useful both when a contract is being written, as well as before adhering to a given contract (to ensure its unambiguous enforcement). AnaCon uses CLAN as its "back-end" conflict analyser, yet abstracts over both the input to and output from CLAN via the CNL interface described in section 3.2.

[2] `http://www.cs.um.edu.mt/~svrg/Tools/CLTool`



(a)



(b)

Figure 2: (a) Screenshot of the CLAN tool; (b) Automaton generated for $[c]O(b) \wedge [a]F(b)$. [8]

2.3. Controlled Natural Languages (CNLs)

CNLs are artificial languages engineered to be simpler versions of "full" (or "plain") natural languages such as English. Such simplified languages are obtained through careful selection of vocabulary and restriction of grammatical rules, and are normally tailored to be used in a particular domain.

Among other applications, CNLs are useful when considering human-machine interactions which aim for an algorithmic treatment of language. Unlike plain natural languages, the simplifications applied to CNLs usually allow them to be

expressed and processed formally, while remaining easy to understand and use for speakers of the original parent natural language. This idea of using a CNL as a natural language-like interface for a formal system is not new [2, 9, 10], and is also the solution chosen in AnaCon.

In general, the richer a CNL is, the more complex is its automation. So, it is a challenge when designing CNLs to find a good trade-off between expressiveness (i.e. how close they are to natural languages) and formalisation. This trade-off is also affected by the richness of the parent NL and the formalism in which the CNL is defined [1].

As an example of the kinds of restrictions found in CNLs, consider again the following natural language clause:

> *The ground crew is obliged to open the check-in desk and request the passenger manifest two hours before the flight leaves.*

Using the CNL introduced later in this paper, such a clause would be re-written as:

```
if {the flight} leaves {in two hours} then both
  - {the ground crew} must open {the check-in desk}
  - {the ground crew} must request {the passenger
      manifest}
```

Even though the structure of the CNL version is noticeably less natural, it is sufficient for our purposes to be merely *close enough* to English as to be understood by any non-technical person, while retaining the possibility of being unambiguously translated into an equivalent $\mathcal{CL}$ expression. It is worth mentioning that the conversion of NL to CNL is not necessarily a trivial process, owing to the ambiguities and potential for misinterpretation in NL. Conversely however, CNLs should be immediately understandable to any speaker of the parent NL, as the former is very much a subset of the other. This means that while it may require some training to convert a contract in NL to CNL, once that conversion has been made then anyone should be able to easily understand that CNL version of the contract. Further details about the design of the CNL for AnaCon is explained in section 3.2.

2.4. The Grammatical Framework

With both the formal language $\mathcal{CL}$ and a controlled natural language for the framework in place, what remains is the software implementation for performing this bi-directional translation between representations. As in [2], we retain the use of the Grammatical Framework (GF) as a grammar formalism and runtime parser/lineariser for converting between CNL and $\mathcal{CL}$.

GF is a logical framework in the spirit of Harper, Honsell and Plotkin [11], which lets us define logics tailored for specific purposes, rather than trying to fit everything in a single model. At the same time, GF is also equipped with mechanisms for mapping abstract logical expressions to a concrete language. This is a distinct feature since most other logical frameworks come with a predefined

syntax. This same feature is a notable characteristic of GF as a linguistic framework. While the logical framework encodes the language-independent structure (ontology) of the current domain, all language-specific features can be isolated in the definition of the concrete language. In other words, the definitions in the logical framework comprise the *abstract syntax* of the domain, while the *concrete syntax* is kept clearly separated [12]. This is a realisation of the separation between *tectogrammatical* and *phenogrammatical* features as was first proposed by Curry [13].

Furthermore, it is usual and actually very common to equip the same abstract syntax with several concrete syntaxes. Since GF has both a *parser* and a *lineariser*, in this case, the abstract syntax can serve as an interlingua. When a sentence is parsed from the source language, then the meaning of the sentence is extracted as an expression in the abstract syntax. The abstract expression then can be linearised back into some other language and this gives us bi-directional translation between any two concrete languages. Most of the time the concrete languages are natural languages, but it is also possible to define a linearisation into some formal language. In AnaCon, we have two concrete syntaxes—one for English (CNL) and one for the source language of CLAN ($\mathcal{CL}$). Thanks to the bi-directionality of GF we can go freely from CNL to logic and vice versa.

Another important advantage of GF from an engineering point of view is the availability of the *Resource Grammar Library* (RGL) [14]. Since every domain is logically different, it is also necessary to define different concrete syntaxes. When these are natural languages, then it means that a lot of tedious low-level details like word order and agreement have to be implemented again and again for each application. Fortunately, RGL provides general linguistic descriptions for several natural languages which can be reused by using a common language independent API. We implemented the AnaCon syntax for English by using this library, which both simplifies the development and makes it easy to port the system to other languages.

The GF runtime system also features an incremental parser, which can parse partial sentences and suggest valid completions according the underlying grammar [15]. While not used in the current version of AnaCon, this feature becomes very useful when composing sentences in CNL, as users do not necessarily need to know the specific grammar rules which define the language. In other words, the incremental parser can be used to provide a guided user-input experience. This feature was a further motivator for choosing GF as the framework for the implementation of AnaCon's CNL.

3. The AnaCon framework

In this section we start with the presentation of our framework, AnaCon, in general terms. We then discuss some issues concerning the particular CNL we are using as an input language for the framework, and present some details on the linearisation and parsing processes via GF.

```
[clauses]
if {the flight} leaves {in two hours} then both
  - {the ground crew} must open {the check-in desk}
  - {the ground crew} must request {the passenger manifest}
[/clauses]
[contradictions]
{the ground crew} open {the check-in desk} # {the ground
    crew} request {the passenger manifest} ;
[/contradictions]
```

Figure 3: Sample contract file in AnaCon format

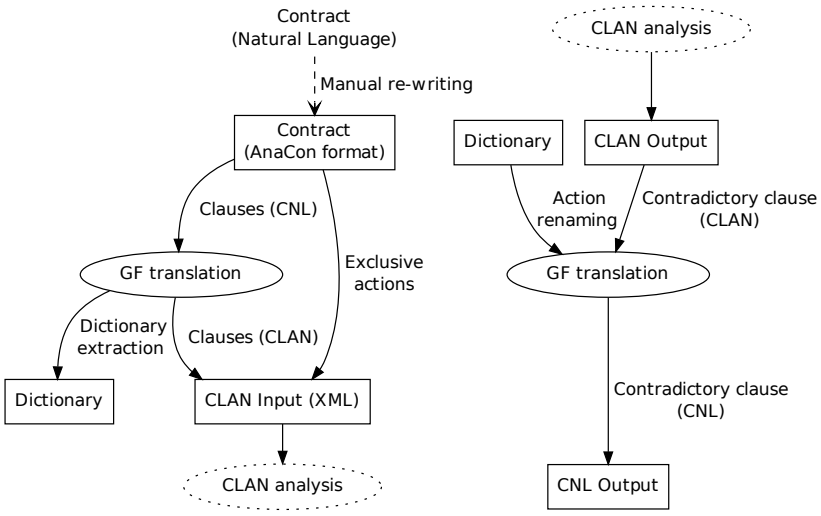


Figure 4: AnaCon processing workflow

3.1. System workflow

AnaCon takes as input a text file containing the description of a contract in two parts: (i) The contract itself written in CNL; (ii) A list of mutually exclusive actions.[3] Fig. 3 shows a sample of the input file to the framework, containing part of the description of what an airline ground crew should do before flights leave (details on the CNL syntax will be given in Section 3.2).

The entire system is summarised in Fig. 4 where arrows represent the flow of information between processing stages. AnaCon essentially consists of a translation tool written in GF, the conflict analysis tool CLAN, and some script files

[3] AnaCon can be downloaded from: `http://www.cse.chalmers.se/~gersch/anacon/`.

used to connect these different modules together. The typical system workflow is as follows:

1. The user starts with a contract (specification, set of requirements, etc.) in plain English, which must be rewritten in CNL. This is primarily a modelling task, and it must be done manually. It requires no technical skills from the user, but does demand a knowledge of the CNL syntax and the set of allowed verbs.
2. The CNL version of the contract in AnaCon text format (Fig. 3) is then passed to the AnaCon tool, which begins processing the file.
3. The clauses in the contract are translated into their $\mathcal{CL}$ equivalents using GF. This translation is achieved by parsing the CNL clauses into abstract syntax trees, and then re-linearising these trees using the $\mathcal{CL}$ concrete syntax (see Section 3.3).
4. From the resulting $\mathcal{CL}$ clauses, a dictionary of actions is extracted. Each action is then automatically renamed to improve legibility of the resulting formulae, and a dictionary file is written. The list of mutually exclusive actions from the CNL contract is verified to make sure that each individual action actually does appear in the contract.
5. Using the renamed $\mathcal{CL}$ clauses from the previous step and the list of mutually exclusive actions, an XML representation of the contract is prepared for input into the CLAN tool.
6. This XML contract is then passed for analysis to CLAN via its command-line interface, which checks whether the contract contains any normative conflicts. If no such conflicts are found, the user is notified of the success. If CLAN does detect any potential conflicts, the counter-example trace it provides is linearised back into CNL using the GF translator in the opposite direction. The dictionary file is used to re-instate the original action names.
7. The user must then find where the counter-example arises in the original contract. This last step must again be carried out manually, by following the CNL trace and comparing with the original contract.

3.2. About the CNL

Wyner et al. [1] have identified the following general questions one should ask when designing a CNL: (i) Who are the intended users? (ii) What is the main purpose of the language? (iii) Is the language domain-dependent? In our particular case we have the following answers to these questions: (i) The intended user is any person writing normative texts; (ii) The main purpose of the language is that it is close enough to English as to be understood by any person, yet at the same time structured in such a way that its translation into $\mathcal{CL}$ is feasible; (iii) The language is not specifically tailored for an application domain, however, it should be easy to parse it in such a way that obligations, permissions and prohibitions are easily identified.

Actions. The most primitive element in $\mathcal{CL}$ is the action and this is the starting point in the design of our CNL. While in $\mathcal{CL}$ these are just variable names, in

natural language they correspond to sentences stating who is doing what. As a very rough approximation, every English sentence has the structure:

```
<subject>   <verb>   <object>
```

as for instance in the following sentence:

the ground crew	**opens**	**the check-in desk**
subject	*verb*	*object*

This is what we take as the basic syntax for actions in our CNL. Obviously, if this is taken directly, it will rule out many natural language constructions like the usage of adverbs and the attachment of prepositional phrases. These constructions usually express different moods for performing the action (e.g. *quickly, slowly, immediately*, etc.) or define time and space locations for the action (e.g. *at the airport*). As this kind of information cannot be expressed in $\mathcal{CL}$, we omit it from the CNL altogether. Still, since we permit the subject and the object to be free text, the user has the freedom to include more information than just the noun phrase of the subject or the object. It is also possible to have ditransitive verbs, i.e. verbs with more than one object. In this case we simply insert both objects in the free text slot for the object. If the verb is intransitive (without objects) then we can just leave the object slot empty.

The slot for the verb is not free text and must come from a set of predefined verbs. While we do not have to analyse the subject and the object slots, the ability to analyse the verb is important since we use modal verbs like *must* and *may* to indicate obligation, prohibition and permission. The restriction to use known verbs is not so hard since the grammar has a lexicon with all verbs from the Oxford Advanced Learners Dictionary [16, 17]. A given user will almost certainly find the verb that is needed—or a synonym of it—in the lexicon. The verb should be always in the present tense, and it can be in first, second or third person, in singular or plural. We check the tense but we cannot check the agreement with number and person, since we do not analyse the subject of the sentence. The only exception is when the verb is used with some of the modal verbs, then it must be in the infinitive.

When analysing the action, we must be able to correctly identify the beginning and the end of each slot, which is difficult when there are free text slots. Our simple solution is to require that the object and subject must be surrounded with curly braces, i.e. the user actually writes:

```
{the ground crew} opens {the check-in desk}
```

In some cases, the system can do the splitting even without the help of the curly braces since from the context it knows where each slot starts, and can guess the end of the slot by looking for known words. For instance we can guess the end of the slot for *the ground crew* since the next word *opens* is a known verb. Unfortunately, with the big verb lexicon this is often ambiguous since for instance *ground* is also a verb although here it is used as an adjective. The guessing can be made more sophisticated by using statistical part of a speech tagger which will try to predict whether *ground* is used as a verb or as an

adjective. Unfortunately even the best part of speech taggers are still far from perfect, with precision of about 95%–97% (the precision of the Stanford Tagger, for instance, is 96.86% [18]). Instead, we opted for a solution that is simple and predictable. Integration of statistical tools can be done later, while still keeping bracketing as a safe alternative.

Connectives over actions. The two main operations on actions are concurrency (&) and choice (+). In natural language, they are represented by joining the sentences for the different actions with the conjunctions *and* and *or*. When there are more than two actions the usual English rules apply, i.e. the first actions are separated by comma and the last two with the conjunction. When the same expression mixes concurrency and choice, then in order to avoid ambiguities we use the usual conventions in logic and we give higher priority to the concurrency. In other words, if we have the expression `a and b or c`, then it will be interpreted as `(a and b) or c`. The user can also use parenthesis to override the default priorities.

A sequence of actions (.) in the CNL is introduced with the keyword *first*, followed by a list of actions. The actions are separated by commas except the last two which are separated with a comma followed by the conjunction *then*. For example:

```
first {the ground crew} opens  {the desk},
then  {the ground crew} closes {the desk}
```

We omit from the CNL the two special actions 0 and 1 since they have no obvious equivalent in English. Although they have useful algebraic properties in the logic, they do not appear naturally in any real contracts. A notable exception is the construction $[1^*]C$ which means that the clause C must be enforced at any state. For this purpose, we added the keyword *always* which can be used in front of any clause, which adds the condition $[1^*]$ in the corresponding $\mathcal{CL}$ formula. Similarly we did not include the Kleene star in our CNL, except for its use in relation to *always*.

Deontic modalities. On the next level, from every action, we can construct a clause expressing the obligation, the prohibition or the permission to perform an action. For representing the modalities we use the modal verbs *must*, *shall* and *may*, and the adjectives *required* and *optional*. In this way we implement the Internet recommendation RFC 2119[4] for requirement levels. The only difference is that they also define the verb *should* which is used for recommendations. Since the $\mathcal{CL}$ logic does not support this modality, we do not have it in the CNL either.

More concretely, if we take for example the action *"the ground crew opens the desk"*, then in the different modalities it can be written in one of the following ways:

[4] `http://www.ietf.org/rfc/rfc2119.txt`

- Obligation:
```
{the ground crew} must open {the desk}
{the ground crew} shall open {the desk}
{the ground crew} is required to open {the desk}
```
- Permission:
```
{the ground crew} may open {the desk}
it is optional for {the ground crew} to open {the desk}
```
- Prohibition:
```
{the ground crew} must not open {the desk}
{the ground crew} shall not open {the desk}
```

The two operations on clauses—conjunction ($\wedge$) and the exclusive choice ($\oplus$)—are rendered in English with the keywords *both* (or *each of*) and *either*, followed by a bullet list of clauses. Each list item starts on a new line and begins with a dash. If some of the list items contain clauses which themselves contain conjunction or exclusive choice, then the list items for such clauses must be indented with more spaces than the spaces before the dash of the "parent" clause. Contrary to the case with the concurrency and choice over actions, here we do not have any risk of ambiguity since the indentation level clearly indicates the nested structure of the logical formula.

Reparations. In the case of obligation and prohibition, the user can specify a reparation clause which must be hold if the contract is violated. In the CNL the reparation is introduced with comma and the keyword *otherwise* after the main action. For example:

```
{the ground crew} must open {the desk}, otherwise
{the ground crew} must pay {a fine}
```

Here we can have an arbitrarily long list of clauses, which are applied in the order in which they are written. The last clause is not followed by *otherwise*, which is an indication its reparation is $\bot$. This is also the only way to introduce $\bot$ in the logic. Similarly to 0 and 1 for actions, the clauses $\top$ and $\bot$ cannot be used directly in the CNL.

The last thing to mention about the CNL is the syntax for conditions. As already mentioned, the syntax for the special condition $[1^*]C$ is introduced with the keyword *always* followed by the content of the clause C. The general conditions are introduced with the usual *if . . . then* statements in English, for example:

```
if {the ground crew} opens {the desk}
then {the ground crew} must close {the desk}
```

Note that here the verb *opens* is not used with a modal verb; this is an indication that this is an action and not a clause. In fact the expression between *if* and *then* can be a combination of many actions joined with the different action operators.

3.3. Linearisation and parsing in GF

In what follows we present how the major features of $\mathcal{CL}$ are represented in the abstract syntax, and look at how these features are handled in the concrete

syntax for our CNL and the symbolic language for CLAN. As the chosen CNL covers a *subset* of $\mathcal{CL}$'s full expressivity, some $\mathcal{CL}$ operators are accordingly absent from the grammars—namely $\top$, 0, 1 and a^*. With the GF grammars for our two representations, the framework provides parsing and linearisation to and from the abstract syntax for free. In this way we can achieve two-way translation between the CNL and the CLAN language by having one concrete syntax for each, with shared abstract syntax.

To begin with, we define the following categories based on the BNF of $\mathcal{CL}$ (square brackets denote lists over a category). These correspond to the left-hand-side of the productions in Fig. 1.

```
cat
  Act; [Act]; Clause; [Clause]; ClauseX;
  Clause0; [Clause0]; ClauseP; [ClauseP]; ClauseF;
```

Conjunction of clauses. In the abstract syntax, conjunction over clauses is defined as a function collapsing a list of heterogeneous clauses into one.

```
fun
  andC : [Clause] -> Clause ;
```

Our CNL as defined in Section 3.2 dictates that two or more clauses joined by conjunction should be bulleted and indented (for legibility and to avoid ambiguity), and preceded with a keyword token *both* or *each of*. As there are no other binary operations over clauses, operator precedence is not an issue (unlike for the action operators) and our code is fairly simple:

```
lin
  andC lst = indentS ("both"|"each of")
                     (mkS bullet_Conj lst) ;

oper
  indentS : Str -> S -> S = \keyword,sen -> lin S {
    s = keyword ++ "[" ++ sen.s ++ "]" ;
  } ;
  bullet_Conj = mkConj "-" "-" ;
```

A few different things are happening here. Firstly, the linearisation of `andC` is delegated to the `indentS` operation[5], which prefixes our list with either of the variants *both* or *each of*, and encloses the rest of the term in square brackets. The role of the brackets is to encode the beginning and the end of an indentation level. Since GF grammars work on token level and the spaces and the new lines are ignored, they cannot handle the indentation directly. Instead a *custom lexer* and *unlexer* are used to convert between the square brackets and the indentation levels. In this way the indentation is handled outside of the grammars. We also see the reference to `mkS`, an operation defined in the GF Resource Grammar

[5] **oper** judgements in GF are operations which can be re-used by linearisation judgements, but do not themselves represent linearisations of syntactic constructors.

Library (RGL). This library call does all the work of joining our clauses into a single token list using a hyphen symbol as a delimiter (`bullet_Conj`).

Conditionals. The modality $[\beta]C$ is used to express conditional obligations, permissions and prohibitions, where the condition is a simple or compound action. The abstract syntax declaration and CNL linearisation are given below:

```
fun
  when : Act -> Clause -> Clause ;

lin
  when act c =
    mkS if_then_Conj (act.s ! Default) c ;
```

This example makes use of another version of the overloaded `mkS` operation from the RGL, which constructs an English *if . . . then* sentence given the appropriate arguments. The linearisation of such a clause in the $\mathcal{CL}$ concrete syntax is a simple string concatenation:

```
lin
  when act c = "[" ++ act.s ++ "]" ++ "(" ++ c.s ++ ")" ;
```

Obligations, Permissions and Prohibitions. Obligations, permissions and prohibitions have a similar implementation as they all follow the same pattern. Each is built from an action and a reparation clause (CTP or CTD) where appropriate. Choice over obligations and permissions is defined in the same way as conjunction of clauses above.

```
fun
  O : Act -> ClauseX -> ClauseO ;
  P : Act -> ClauseP ;
  F : Act -> ClauseX -> ClauseF ;
  choiceO : [ClauseO] -> ClauseO ;
  choiceP : [ClauseP] -> ClauseP ;
```

Understanding the linearisation of an obligation also requires a look at the reparation clauses. While $\mathcal{CL}$ uses the bottom symbol $\bot$ to indicate a null CTD, in natural language it sounds very awkward to say something like *"one is obliged to pay a fine, otherwise nothing"*. It is much more natural to simply omit the *"otherwise nothing"* altogether. So, the linearisation of obligations is dependent on the type of the reparation clause (the `ty` field, where `False` indicates a null CTD).

```
lincat
  ClauseO = S ;
  ClauseX = {s : S; ty : Bool} ; -- CTD/CTP
lin
  O act cl = case cl.ty of {
    True  => mkS (mkConj ", otherwise") (act.s ! Obligation)
         cl.s ;
    False => lin S {s = cl.s.s ++ (act.s ! Obligation).s}
```

```
  } ;
  reparation c = { s = c ; ty = True } ;
  failure = { s = lin S {s=""} ; ty = False } ;
```

Actions. Atomic actions are defined as a "triple" containing a subject, a verb and an object, e.g. <*the crew, requests, the boarding pass*>. These are covered by the lexical categories `NP` (noun phrase), `V` (verb) and `NP` respectively:

```
flag
  literal = NP ;
cat
  NP ; V ;
fun
  atom    : NP -> V -> NP -> Act ;
```

By specifying the `literal = NP` flag, the GF compiler is instructed to treat `NP` as a literal category, which means its linearisation is that of a simple string. To achieve a degree of modularity between the logical and the linguistic, all verbs are defined in a separate abstract GF module `Verbs.gf`. In this case, the CNL concrete syntax `VerbsEng.gf` is also imported in the $\mathcal{CL}$ concrete syntax, exhibiting how GF's module system may help avoid duplication of code. The verbs themselves are also defined using the RGL, such that all that is required in our linearisation is a call to the `mkV` smart paradigm:

```
fun
  close_V : V;
  request_V : V;
  ...
lin
  close_V = mkV "close" "closes" "closed" "closed" "closing
      ";
  request_V = mkV "request";
  ...
```

The CNL linearisation of actions is defined as a table parametrised with a *mode*. This essentially reflects the idea that a single action can be realised in four different modalities:

- Default: *the crew requests the boarding pass*
- Obligation *the crew must request the boarding pass*
- Permission: *the crew may request the boarding pass*
- Prohibition: *the crew shall not request the boarding pass*

With this approach, each atomic action internally contains each of these possible linearisations, which must be selected elsewhere in the grammar using the selection operator `!`.

To add a degree of naturalness to the grammar, we also introduce the concept of *linearisation variants*. Variants are a way of adding alternative, non-deterministic linearisations to an abstract syntax tree, and are defined in GF using the pipe symbol `|`. Using variants, we allow the single abstract syntax tree `O (atom (np "the crew") close_V (np "the check-in desk"))` to have any of the following linearisations:

1. *the crew is required to close the check-in desk*
2. *the crew shall close the check-in desk*
3. *the crew must close the check-in desk*

```
param
  Mode = Default | Obligation | Permission | Prohibition ;
lincat
  Act   = {s : Mode => S; p : Prec} ;
lin
  atom = mkAtom 0 | mkAtom 1 | mkAtom 2 ;
oper
  mkAtom : Ints 2 -> NP -> V -> NP -> {s : Mode => S; p :
      Prec} ;
  mkAtom n s p o = {
    s = table {
      Default     => mkS (mkCl s (mkVP (mkV2 p) o)) ;
      Obligation  => case n of {
        0 => mkS ...
        1 => mkS ...
        2 => mkS ...
      } ;
      Permission  => ...
      Prohibition => ...
    } ;
    p = highest
  } ;
```

Operations over actions are defined in the abstract syntax in a way which we are already familiar with. As $\mathcal{CL}$ defines more than one operator over actions, an order of precedence must be enforced to avoid ambiguities in phrases involving compound actions. With the help of the RGL's `Precedence` module, this is achieved by including a precedence field (`p : Prec`) in the linearisation type of actions. The linearisations of the operators are then explicitly given precedence levels, where conjunction is the highest (`p = 2`) and sequence is the lowest (`p = 0`).

```
fun
  andAct, choiceAct, seqAct : [Act] -> Act ;
lin
  andAct    as = {s = \\m => mkS and_Conj  (as!2!m); p=2} ;
  choiceAct as = {s = \\m => mkS or_Conj   (as!1!m); p=1} ;
  seqAct    as = {s = \\m => mkS then_Conj (as!0!m); p=0} ;
```

4. Case studies

In this section we apply AnaCon to two case studies, as a proof-of-concept of the feasibility of our approach. The first is concerned with the workflow description of an airline check-in, including the penalties applicable when the

1. The ground crew is obliged to open the check-in desk and request the passenger manifest from the airline two hours before the flight leaves.
2. The airline is obliged to provide the passenger manifest to the ground crew when opening the desk.
3. After the check-in desk is opened the check-in crew is obliged to initiate the check-in process with any customer present by checking that the passport details match what is written on the ticket and that the luggage is within the weight limits. Then they are obliged to issue the boarding pass.
4. If the luggage weighs more than the limit, the crew is obliged to collect payment for the extra weight and issue the boarding pass.
5. The ground crew is prohibited from issuing any boarding passes without inspecting that the details are correct beforehand.
6. The ground crew is prohibited from issuing any boarding passes before opening the check-in desk.
7. The ground crew is obliged to close the check-in desk 20 minutes before the flight is due to leave and not before.
8. After closing check-in, the crew must send the luggage information to the airline.
9. Once the check-in desk is closed, the ground crew is prohibited from issuing any boarding pass or from reopening the check-in desk.
10. If any of the above obligations and prohibitions are violated a fine is to be paid.

Figure 5: Airline contract case study [19].

work is not carried out as prescribed. The second case study is a legal contract concerning the provision of Internet services. We finish the section with a discussion on the lessons learned from the case studies.

4.1. Case Study 1: Airline check-in process

Our first case study has been taken from [19]. It consists of the description of the check-in process of an airline company, given in Fig. 5.

To show the modelling and re-writing process, we will first consider two clauses from this contract and show their equivalent CNL representations. Note that in our $\mathcal{CL}$ expressions, the actions have been renamed for brevity. This replacement is performed automatically by AnaCon and is completely reversible. You may find a listing of the generated dictionary file in Appendix A.2.

> **Original:** *The ground crew is obliged to open the check-in desk and request the passenger manifest from the airline two hours before the flight leaves.*
>
> **CNL:**
>
> ```
> if {the flight} leaves {in two hours} then {the
>     ground crew} must open {the check-in desk} and {
>     the ground crew} must request {the passenger
>     manifest from the airline}
> ```

For this clause, AnaCon gives the following $\mathcal{CL}$ formula as output:

> $\mathcal{CL}$: `[b3]O(a7&b2)`

where from the dictionary file (see Appendix A.2) we see that:

```
b3 = {the flight} leave {in two hours}
a7 = {the ground crew} open {the check-in desk}
b2 = {the ground crew} request {the passenger
    manifest from the airline}
```

In the example above we see an obligation over two concurrent actions, which only become effective after an initial constraint is met—i.e. *if it is two hours before the flight leaves*. Note how this constraint is moved to the beginning of the clause and expressed using the *if* keyword. As defined by our CNL, conjunction over actions is expressed by joining together the individual actions with the keyword *and*. Conjunction over clauses however must be handled differently, as shown in the second example below:

Original: *Once the check-in desk is closed, the ground crew is prohibited from issuing any boarding pass or from reopening the check-in desk.*

CNL:

```
if {the ground crew} closes {the check-in desk} then
     both
  - {the ground crew} must not issue {boarding pass}
  - {the ground crew} must not reopen {the check-in
      desk}
```

AnaCon gives the following $\mathcal{CL}$ formula as output (again generating the corresponding action names in the dictionary file; see Appendix A.2):

$\mathcal{CL}$: `[b6]((F(a1))∧(F(a4)))`

In this case, using *and* to separate our clauses would be ambiguous with the conjunction over actions (shown above). Thus the bullet syntax is used here to clearly indicate the "level" of the conjunction.

While we have taken the above two examples individually, in real contracts clauses often refer to and depend on each other. When read in NL the reader can easily make the connections between the different clauses, but when it comes to modelling the contract formally these need to be handled explicitly.

Firstly, it is a common assumption that all the individual clauses in a contract are active together and thus there is an implicit conjunction between them. Furthermore, note how clause 10 in the example specifies a CTD for violating *any part* of the contract. Thus combining clauses 1, 8, 9, and 10 from the contract in Fig. 5 we end up with:

CNL:

```
if {the flight} leaves {in two hours} then each of
  - {the ground crew} must open {the check-in desk}
      and {the ground crew} must request {the
      passenger manifest from the airline}
```

```
- if {the ground crew} closes {the check-in desk}
    then each of
  - {the ground crew} must send {luggage
      information to airline}
  - {the ground crew} must not issue {boarding
      pass}
  - {the ground crew} must not reopen {the check-
      in desk}
```

which results in the following $\mathcal{CL}$ formula:

$\mathcal{CL}$: `[b4]((O(b1&a2))∧[b6]((O(b2))∧((F(a1))∧(F(a4)))))`

For full versions of the CNL contracts please refer to Appendix A.1. When processed with AnaCon, the first conflicting state reported was reached after a single action:

```
1 counter example found
Clause:
    (((O(a7&b2))_(Oa3))^(((Oa2)_(Ob1))^(([a7]((O(a6
        .(b4.(a8.a5))))_(Ob7)))^(((F(b5)_(Oa3))^(((
        Ob6)_(Oa3))^(([b6](Oa9))^(([b6](Fa1))^([b6](
        Fa4)))))))))
Trace:
    1. the flight leave in two hours
```

Note that the counter-example above contains 2 parts: (i) a $\mathcal{CL}$ formula, and (ii) a trace in CNL. The first part is the formula representing the state of the automaton where the normative conflict happens, which is not particularly helpful for the end user. The second part is a linearisation of the output of CLAN showing what is the sequence of actions leading to the conflict; in this case only one.

A quick analysis of the original contract reveals that the two mutually exclusive actions *opening the check-in desk* and *closing the check-in desk* were erroneously obliged at the same level in the contract. This is a modelling error, and is corrected in a second version of the case study CNL (see Appendix A.1).

When rewriting the second version we have not only addressed the issue of the arrangement of the actions corresponding to opening and closing the check-in desk, but we have also added more mutually exclusive actions. Such actions are considered mutually exclusive because they are logically contradictory and thus cannot happen at the same time, or because they cannot occur simultaneously due to physical constraints (e.g. *"the check-in crew issue the boarding pass"* and *"the check-in crew check that the passport details match what is written on the ticket"*). By adding such pairs of mutually exclusive (contradictory) actions we are avoiding some possible unnatural traces and at the same time reducing the size of the CLAN automaton, improving its time and space requirements.

By executing AnaCon a third time and analysing the counter-example given, it becomes apparent that there is something wrong with clause 5 (*cf.* Fig. 5). In effect, this clause has two problems: (i) it is ambiguous as the whether the so-named "details" refer to the passport or to the ticket; (ii) it is redundant as it is somehow contained in clause 3. The latter adds some complexity to the analysis, so we decided to eliminate clause 5, without changing the intended meaning of the description.

Re-running AnaCon on this new contract also reveals another conflict, relating to the initiation of check-in and the closing of the gate being obliged at the same level in the contract:

CNL:

```
if {the airline crew} provides {the passenger
    manifest to the ground crew} then each of
  - first {the check-in crew} must initiate {the
      check-in process} ...
  - {the ground crew} must close {the check-in desk
      20 mins before flight leaves} ...
  - if {the ground crew} closes {the check-in desk
      20 mins before flight leaves} then ...
```

$\mathcal{CL}$: `[a5]((O(a8&...))∧((O(b5))∧[b5](...)))`

Resulting AnaCon output:

```
4 counter examples found (only showing first)
Clause:
    ((((Oa8)_(Ob6))^([a8]((O(b4.(a7.a6)))_(Ob6))))
        ^(((Ob5)_(Oa3))^(([b5](Ob1))^(([b5](Fa1))^([
        b5](Fa4))))))
Trace:
    1. the flight leave in two hours
    2. the ground crew open the check-in desk 2
        hours before
    3. the ground crew request the passenger
        manifest from the airline
    4. the airline crew provide the passenger
        manifest to the ground crew
```

This leads to yet another re-writing of this final part of the contract, where the closing of the gate is now properly obliged *after* the initiation of the check-in process (note that by adding a new action, the re-written action names have changed):

CNL:

```
if {the airline crew} provides {the passenger
    manifest to the ground crew} then each of
```

```
- first {the check-in crew} must initiate {the
    check-in process} ...
- if {the flight} leaves {in 20 mins} then both
  - {the ground crew} must close {the check-in
      desk}
  - if {the ground crew} closes {the check-in desk
      } then each of
    - {the ground crew} must send {the luggage
        information to the airline}
    - {the ground crew} must not issue {boarding
        pass}
    - {the ground crew} must not reopen {the
        check-in desk}
```

Generated $\mathcal{CL}$: `[a6]((O(a9&...))∧`
`([a5]((O(b6))∧[b6]((O(b2))∧((F(a7))∧(F(a4)))))))`

In order to truly cut down the size of the generated automaton to a bare minimum, a cross product of all possible mutually exclusive actions is generated using a simple shell script. From this, only the actions that *are allowed* to occur concurrently are removed; namely all those including the paying of fines, since a fine can be paid at any time. As this case study turns out to have a highly sequential nature, it makes sense that the list of mutually exclusive actions should be quite large.

Finally, after the iteration process described above we arrive at a final version of the contract without conflicts. This can be found in Appendix A.1. It should be noted that for this case study we modelled the contract as a single instance of a sequence of events, i.e. considering a single airline and ground crew, a single check-in desk and indeed a single passenger. Extending the example with the *always* operator to model multiple check-ins occurring simultaneously introduces a number of difficulties and moreover reveals certain shortcomings of $\mathcal{CL}$ and CLAN. These are discussed further in sections 4.3 and 6.

4.2. Case Study 2: Internet Service Provider

We apply AnaCon here to part of a contract between an Internet provider and a client, taken from [20]. The contract (reproduced in Fig. 6) stipulates the obligations and rights of an Internet provider and a client of the service. For simplicity of presentation we will consider only the following clauses of the contract:

7.1. The **Client** shall not:
  a) supply false information to the Client Relations Department of the **Provider**.
7.2. Whenever the Internet Traffic is **high** then the **Client** must pay [*price*] immediately, or the **Client** must notify the **Provider** by sending an e-mail specifying that he will pay later.
7.3. If the **Client** delays the payment as stipulated in 7.2, after notification he must immediately lower the Internet traffic to the **normal** level, and pay later twice ($2 * [price]$).
7.4. If the **Client** does not lower the Internet traffic immediately, then the **Client** will have to pay $3 * [price]$.
7.5. The **Client** shall, as soon as the Internet Service becomes operative, submit within seven (7) days the Personal Data Form from his account on the **Provider**'s web page to the Client Relations Department of the **Provider**.

We also add clause 11.2 as it is strongly related to clause 7.1 and the two should be taken together:

> 11.2. **Provider** may, at its sole discretion, without notice or giving any reason or incurring any liability for doing so:
> b) Suspend Internet Services immediately if **Client** is in breach of Clause 7.1;

The first clause imposes a prohibition for the client to give false information, while clauses 7.2 through 7.5 stipulate the obligations of the client in what concerns keeping the use of Internet below a certain limit (here specified as *high*) and the penalties to be paid in case these clauses are not respected. Clause 11.2 refers to the right of the provider to suspend the service if the client provides false information.

In what follows we rewrite the above clauses into our CNL and apply AnaCon. Our first attempt to analyse our CNL contract produces a parsing error on the following fragment:

CNL:

```
if {Internet traffic} becomes {high} then either
  - {the Client} must pay {price P}
  - each of
    - {the Client} must notify {the Provider ...}
    - if {the Client} notifies {the Provider ...} then {
        the Client} must lower {Internet traffic to the
        normal level}, otherwise {the Client} is required
        to pay {price 3P}
    - if first {the Client} notifies {the Provider ...},
        then {the Client} lowers {Internet traffic to the
        normal level} then {the Client} must pay {price 2P
        }
```

The syntax error in this example stems from the use of disjunction (*either* on line 1) over clauses, which this is not allowed by $\mathcal{CL}$ and therefore in our CNL. The solution in this case is to treat this disjunction as a reparation, which is indeed the intended meaning in such cases:

CNL:

```
if {Internet traffic} becomes {high} then {the Client}
    must pay {price P}, otherwise first {the Client} must
    notify {the Provider ...}, {the Client} must lower {
    Internet traffic to the normal level}, then {the
    Client} must pay {price 2P}, otherwise {the Client} is
     required to pay {price 3P}
```

$\mathcal{CL}$: `[a4]((O(a8)_((O(a2.b1.a9)_((O(a3)))))))`

Note how rewriting the above clauses actually leads to a neater implementation, both in terms of the CNL and in the underlying $\mathcal{CL}$ expression.

This deed of **Agreement** is made between:
1. [**name**], from now on referred to as **Provider** and
2. [**name**], from now on referred to as the **Client**.

INTRODUCTION

3. The **Provider** is obliged to provide the **Internet Services** as stipulated in this **Agreement**.
5. DEFINITIONS
   5.1. j) **Internet traffic** may be measured by both **Client** and **Provider** by means of Equipment and may take the two values **high** and **normal**.

OPERATIVE PART

7. CLIENT'S RESPONSIBILITIES AND DUTIES
   7.1. The **Client** shall not:
   a) supply false information to the Client Relations Department of the **Provider**.
   7.2. Whenever the Internet Traffic is **high** then the **Client** must pay [*price*] immediately, or the **Client** must notify the **Provider** by sending an e-mail specifying that he will pay later.
   7.3. If the **Client** delays the payment as stipulated in 7.2, after notification he must immediately lower the Internet traffic to the **normal** level, and pay later twice $(2 * [price])$.
   7.4. If the **Client** does not lower the Internet traffic immediately, then the **Client** will have to pay $3 * [price]$.
   7.5. The **Client** shall, as soon as the Internet Service becomes operative, submit within seven (7) days the Personal Data Form from his account on the **Provider**'s web page to the Client Relations Department of the **Provider**.
8. CLIENT'S RIGHTS
   8.1. The **Client** may choose to pay either:
   a) each month; b) each three (3) months; c) each six (6) months;
9. PROVIDER'S SERVICE
   9.2. As part of the Service offered by the **Provider** the **Client** has the right to an e-mail and an user account.
   9.3. **Provider** is obliged to offer with no limitation and within a period of seven (7) days a password and any other Equipment Specific to Client, necessary for the correct usage of the user account, upon receiving of all the necessary data about the client from the Client Relations Department of the **Provider**.
   9.4. Each month the **Client** pays the *bill* the **Provider** is obliged to send a Report of Internet Usage to the Client.
10. PROVIDER'S DUTIES
   10.1. The **Provider** takes the obligation to return the personal data of the client to the original status upon termination of the present **Agreement**, and afterwards to delete and not use for any purpose any whole or part of it.
   10.2. The **Provider** guarantees that the Client Relations Department, as part of his administrative organisation, will be responsive to requests from the **Client** or any other Department of the **Provider**, or the **Provider** itself within a period less than two (2) hours during *working hours* or the day after.
11. PROVIDER'S RIGHTS
   11.1. The **Provider** takes the right to alter, delete, or use the *personal data* of the **Client** only for statistics, monitoring and internal usage in the confidence of the **Provider**.
   11.2. **Provider** may, at its sole discretion, without notice or giving any reason or incurring any liability for doing so:
   b) Suspend Internet Services immediately if **Client** is in breach of Clause 7.1;
13. TERMINATION
   13.1. Without limiting the generality of any other *Clause* in this *Agreement* the **Client** may terminate this *Agreement* immediately without any notice and being vindicated of any of the Clause of the present Agreement if:
   a) the **Provider** does not provide the Internet Service for seven (7) days consecutively.
   13.2. The **Provider** is forbidden to terminate the present Agreement without previous written notification by normal post and by e-mail.
   13.3. The **Provider** may terminate the present Agreement if:
   a) any payment due from **Client** to **Provider** pursuant to this **Agreement** remains unpaid for a period of fourteen (14) days;
16. GOVERNING LAW
   16.1. The **Provider** and the present **Agreement** are governed by and construed according to the Law Regulating Internet Services and to the Law of the State.
   a) The Law of the State stipulates that any **ISP Provider** is obliged, upon request to seize any activity until further notice from the State representatives.

Figure 6: A contract between an Internet provider and a client [20]

Running this corrected version of the contract with AnaCon, we are returned with a list of no fewer than 473 counter-examples which CLAN determined would lead to a state of conflict. An excerpt of the full output from CLAN shown below indicates that there is no proper handling of inherent sequence of the preliminary steps in the contract—i.e. those referring to customer data and the application procedure—which should apply *before* any clauses about the Internet traffic are even considered.

```
473 counter examples found (only showing first)
Clause:
    ((((F(a7)_(Pa1))^([1]([(*1)]((F(a7)_(Pa1)))))^((((Oa9)
        _(Oa3))^(((Oa8)_(((Oa2)_(Oa3))^(([a2]((Ob1)_(Oa3))
        )^([a2]([b1]((Oa9)_(Oa3)))))))^(([a4]((Oa8)_(((Oa2
        )_(Oa3))^(([a2]((Ob1)_(Oa3)))^([a2]([b1]((Oa9)_(
        Oa3))))))))^([1]([(*1)]([a4]((Oa8)_(((Oa2)_(Oa3))
        ^(([a2]((Ob1)_(Oa3)))^([a2]([b1]((Oa9)_(Oa3)))))))
        ))))))^(([a5](Oa6))^([1]([(*1)]([a5](Oa6)))))))
Trace:
    1. Internet traffic become high
    2. the Client provide false information to the Client
        Relations Department of the Provider and the
        Internet Service become operative
    3. the Client notify the Provider ... and the Internet
         Service become operative and the Client submit
        ... the Personal Data Form ...
    4. the Client notify the Provider ... and the Internet
         Service become operative and the Client submit
        ... the Personal Data Form ...
    5. the Internet traffic become high and the Client
        lower Internet traffic to the normal level and the
         Client submit ... the Personal Data Form ...
```

More than a modelling problem, this tends to indicate some underlying assumptions in the original contract which need to be explicitly handled. This leads to the restructuring of the contract, as shown in Appendix B.1. In particular, the new contract was conceptually split into two sections, where all clauses referring to the application process form a prefix to the rest of the contract, which subsequently deals with the service once it has been activated. This is shown below (the corresponding $\mathcal{CL}$ formula has been omitted):

CNL:

```
if {the Client} submits {the data} then each of
  - {the Provider} must check {the data}
  - if first {the Provider} checks {the data}, then {the
      Provider} disapproves {the data} then {the Provider}
       may cancel {the contract}
  - if first {the Provider} checks {the data}, then {the
      Provider} approves {the data} then each of
```

```
- {the Internet Service} must become {operative}
- if {the Internet Service} becomes {operative} then
    always ...
```

It should be noted that this rewriting of the contract may in fact depart from the original meaning of the natural language contract we began with. This however should not be seen as a flaw; indeed the very aim of contract analysis tools like AnaCon is to help identify weaknesses in existing contracts and facilitate their improvement.

Running this new contract through AnaCon produces a reduced—though still large—set of counter-examples from CLAN:

```
147 counter examples found (only showing first)
Clause:
    (((Ob1)_(Oa2))^(((Oa6)_(((Oa1)_(Oa2))^(([a1]((Ob2)_(
        Oa2)))^([a1]([b2]((Ob1)_(Oa2)))))))^(([a9]((Oa6)_
        (((Oa1)_(Oa2))^(([a1]((Ob2)_(Oa2)))^([a1]([b2]((
        Ob1)_(Oa2)))))))^([1]([(*1)]([a9]((Oa6)_(((Oa1)_(
        Oa2))^(([a1]((Ob2)_(Oa2)))^([a1]([b2]((Ob1)_(Oa2))
        ))))))))))
Trace:
    1. the Client submit the data
    2. the Provider check the data
    3. the Provider approve the data
    4. the Internet Service become operative
    5. Internet traffic become high
    6. Internet traffic become high
    7. Internet traffic become high and the Client pay
        price P and the Client notify the Provider ...
    8. Internet traffic become high and the Client pay
        price P and the Client notify the Provider ...
    9. Internet traffic become high and the Client pay
        price P and the Client lower Internet traffic to
        the normal level
```

The initial reaction to this large number of counter-examples is to explicitly add more mutually exclusive actions to the contract to reduce the size of the automaton produced. While adding 5 pairs of exclusive actions reduces the number of possible counter-examples to just 18, a new issue with the contract emerges. In the new trace produced by CLAN it can be seen that if the action of the *Internet traffic becoming high* occurs twice (or more) in succession, the contract will always end in conflict, as shown in the counter-example below.

```
18 counter examples found (only showing first)
Clause:
    ((Oa2)^(((Oa6)_(((Oa1)_(Oa2))^(([a1]((Ob2)_(Oa2)))^([
        a1]([b2]((Ob1)_(Oa2)))))))^(([a9]((Oa6)_(((Oa1)_(
        Oa2))^(([a1]((Ob2)_(Oa2)))^([a1]([b2]((Ob1)_(Oa2))
        ))))))^([1]([(*1)]([a9]((Oa6)_(((Oa1)_(Oa2))^(([a1
        ]((Ob2)_(Oa2)))^([a1]([b2]((Ob1)_(Oa2))))))))))))
```

```
Trace:
    1. the Client submit the data
    2. the Provider check the data
    3. the Provider approve the data
    4. the Internet Service become operative
    5. Internet traffic become high
    6. Internet traffic become high
    7. the Client pay price P and the Client notify the
       Provider ...
    8. the Client notify the Provider ...
    9. Internet traffic become high
```

Further analysis of CLAN output indicates that this issue is actually due to the use *always* operator, which essentially allows for parallel branches to be created in the contract automaton which cannot then both be satisfied. This ultimately points to a weakness in $\mathcal{CL}$. In our case we were able to achieve a contract-free contract by removing the *always* keyword on line 10, however this would arguably result in a non-intended meaning. A proper solution would require further remodelling or even augmenting $\mathcal{CL}$ itself.

4.3. Some reflections concerning the case studies

The two case studies examined in this paper come from unrelated domains. However they both share the property that they treat norms, and thus fall into the general group of texts which we are interested in analysing. While we do not claim that AnaCon is yet general enough to handle *any* such contract, we believe that these two case studies serve as a good proof-of-concept of the framework.

Applying AnaCon to the above 2 case studies provides us with some interesting insights on how to improve our framework.

The first observation is that our CNL is quite rich in terms of vocabulary and it is suitable as a high level language to be translated into $\mathcal{CL}$. However, the contract author needs to know the CNL syntax and be able to mentally convert NL clauses into valid CNL. This is for instance the case when writing obligations over sequences: it is not possible to write that in our CNL, and they must instead be written as a *sequence of obligations*, with only one CTD associated to the whole sequence. Though this is a limitation at the CNL level, it is not the case for $\mathcal{CL}$, as sequences of obligations cannot be expressed directly.

A second observation is that it would be desirable to have causal/temporal relationships among actions in addition to the declaration of mutual exclusive actions (#). This would allow a radical reduction in the size of the underlying CLAN automaton and thus improve efficiency and avoid some redundant counter-examples which are eliminated by rephrasing the CNL document. This redundancy is due to the semantics of &, discussed later in this section.

Concerning the output of CLAN, when a conflict is found the output produced by the CLAN tool consists of a list of tuples containing a conflict state and an action trace, as shown in Fig. 7. In this output, CLAN is reporting all possible combinations of actions that would lead to a state of contradictions. As

```
(((Ob1)_(Oa2))^(((Oa6)_(((Oa1)_(Oa2))^(([a1]((Ob2)_(Oa2)))
    ^([a1]([b2]((Ob1)_(Oa2)))))))^(([a9]((Oa6)_(((Oa1)_(Oa2))
    ^(([a1]((Ob2)_(Oa2)))^([a1]([b2]((Ob1)_(Oa2)))))))
    ^([1]([(*1)]([a9]((Oa6)_(((Oa1)_(Oa2))^(([a1]((Ob2)_(Oa2)
    ))^([a1]([b2]((Ob1)_(Oa2))))))))))))
b3,a7,a8,a5,a9,a9,a9&a6&a1,a9&a6&a1,a9&a6&b2

(((Ob1)_(Oa2))^((((Oa1)_(Oa2))^(([a1]((Ob2)_(Oa2)))^([a1]([
    b2]((Ob1)_(Oa2))))))^(((Oa6)_(((Oa1)_(Oa2))^(([a1]((Ob2)_
    (Oa2)))^([a1]([b2]((Ob1)_(Oa2)))))))^(([a9]((Oa6)_(((Oa1)
    _(Oa2))^(([a1]((Ob2)_(Oa2)))^([a1]([b2]((Ob1)_(Oa2)))))))
    )^([1]([(*1)]([a9]((Oa6)_(((Oa1)_(Oa2))^(([a1]((Ob2)_(Oa2
    )))^([a1]([b2]((Ob1)_(Oa2))))))))))))))
b3,a7,a8,a5,a9,a9,a9&a6&a1,a9&a6&a1,a9&b2
```

Figure 7: Sample CLAN output

one can imagine this number could explode exponentially as the total number of actions increases, and for this reason adding multiple mutually exclusive actions to the $\mathcal{CL}$ contract helps to keep this under control. The two traces shown in Fig. 7 end with the action expressions `a9&a6&b2` and `a9&b2` respectively, and it is fairly obvious to notice that in this example the performing of action `a6` along with `a9` and `b2` is, for our purposes, irrelevant. From this observation, it follows that we are not necessarily interested in all possible action combinations which could lead to a state of conflict; rather, we are interested only in the *minimal subset* of them. A fairly simple algorithm could be given to determine which are minimal counter-examples knowing then that any other counter-example would be thus redundant.

In fact, the above problem could easily be solved by eliminating the & action operator in $\mathcal{CL}$. After working on the above (and other small) case studies it would seem that it is not needed, as in most practical cases actions happening simultaneously are either uncommon, or can be expressed using interleaving. The elimination of this action operator will not only simplify the syntax but will radically reduce the complexity of CLAN (the main reason of exponential blow-up in CLAN's execution is due to such concurrent actions).

5. Related work

The basic ideas of this journal paper have appeared on the workshop paper [2], where the conceptual model of AnaCon was first introduced. The only commonalities between our current version of AnaCon and the one in [2] are the use of $\mathcal{CL}$ [4] and CLAN [8], besides the overall idea of the framework. In [2] it was shown that it was possible to relate the formal language for contracts $\mathcal{CL}$ and a restricted NL by using GF [12]. Our CNL, however, is based on a formal grammar inspired from NL sentences (i.e. using a subject, verb and complement) unlike that in [2] which was very much an "if-then-else" language enriched with

keywords for obligation, permission and prohibition. Besides this, we have made extensive use of GF libraries and state-of-the-art constructions to make the definition of the abstract and concrete syntaxes much clearer and modular. We have reimplemented all the modules and implemented the counter-example generation in CNL, not done in the previous paper. Though we do not have (formal) experimental results to show the advantages of this new implementation, we do claim an improvement in performance and clarity of presentation based on its use in the case studies presented in this paper.

Using CNLs as a means to obtain a tractable language which is still human-understandable is not new. To date at least 40 different CNLs have been defined with different purposes and thus following different design decisions (*cf.* [1]).

A notable example of this is Attempto Controlled English (ACE) [21]. The difference between Attempto and our CNL is that while ACE aims to be an universal domain-independent language, we choose to make a language that is specifically tailored for the description of normative texts. Although ACE has syntactic constructions for expressing modalities, it also covers a lot of other constructions that we cannot handle in $\mathcal{CL}$. The proper handling of the whole language would make the underling logic unnecessarily complicated. Furthermore, ACE tries to perform full sentence analysis, while in our case this is not necessary since the semantics of the sentence would not be expressible in the logical fragment of $\mathcal{CL}$. Instead, we combine controlled language with free text which allows us to analyse only the relevant structures, while taking the rest as atomic literals. Another advantage of our choice is that the user does not need, as in ACE, to add new words for each domain since there is already a large lexicon of verbs and the nouns are just literals. A reimplementation of the original ACE grammar in GF has been presented in [22] where this controlled language was also ported from English to French, German and Swedish.

An initial exploratory design of another CNL specifically targeted for contracts is presented in [23], where the underlying logic and a sketch for the language are discussed. The chosen logic is actually close to $\mathcal{CL}$ except that it is more liberal. This broader logic gives flexibility in the translation to and from CNL, but it does not automatically exclude the possibility of paradoxes. In addition, their logic adds to $\mathcal{CL}$ temporal features as well as test operators for querying over the external game state. The logic is implemented with their own custom-build reasoner instead of CLAN. The actual CNL, however, is not implemented yet, and it remains only a sketch. Still, the initial design can be traced in Camilleri et al. [10], where, in their implementation of the game of Nomic, they employed a specialized CNL based on the same logic. The latter also used GF to translate between natural and logical representations, but their CNL involves only predefined actions and thus avoids the treatment of free-text, verbs, and actions as triples as in our approach. This also means that the system in [10] cannot be used for diverse contracts as in our case.

Our work is also similar to [24] where Hähnle et al. describe how to get a CNL version of specifications written in OCL (Object Constraint Language). The paper focuses on helping to solve problems related to authoring well-formed formal specifications, maintaining them, mapping different levels of formality and syn-

chronising them. The solution outlined in the paper illustrates the feasibility of connecting specification languages at different levels, in particular OCL and NL. The authors have implemented different concepts of OCL such as classes, objects, attributes, operations and queries. The difference with our work is that $\mathcal{CL}$ is a more abstract and general logic, allowing the specification of normative texts in a general sense. In addition, we are not interested only in logic to language translation but rather in the use of the formal language to further perform verification (in our case conflict analysis) which is then integrated within our framework by connecting GF's output into CLAN, and vice versa.

It is worth mentioning that there is a general interest in the application of CNL for authoring and maintenance of legislative text. For instance [25] studies the typical linguistics structures in the German laws and relates them to constructions in first-order logic and deontic logic. The ultimate goal is the creation of Controlled Legal German as a human-oriented CNL for defining laws. Similarly [26] studies the legislative drafting guidelines for Austria, Germany and Switzerland, issued by the Professional Association for Technical Communication, from the perspective of controlled language. In both cases, however, the controlled language is aimed for human-to-human communication and its level of formalization is far from what is needed for computer based interpretation.

Rosso et al. [27] have used the passage retrieval tool JIRS to search for occurrences of words from a counter-example in natural language legal texts. In particular, they have applied their technique to a counter-example generated by CLAN on the airline check-in desk case study (the very same we have presented here as Case Study 1). JIRS is fed with a manual translation into English from $\mathcal{CL}$ formulae representing the counter-example given by CLAN, and uses an *n-gram* approach to automatically retrieve those sentences in the contract where the conflict occurs. JIRS does this by returning a ranking list with the passages found to be most similar to each query. We briefly discuss in next section how our work could be combined with passage retrieval tools like JIRS.

Finally, in what concerns deontic logic, and a presentation on the classical paradoxes, please refer to McNamara's article [6], and references therein.

6. Conclusion

We have presented in this paper AnaCon, a framework aimed at analysing normative texts containing obligations, permissions and prohibitions. We introduced a CNL for writing such texts, and provided a new and complete implementation of the AnaCon framework. AnaCon automatically converts normative texts written in CNL into the formal language $\mathcal{CL}$, using GF as a technology to perform bi-directional translations. The analysis performed on such texts is currently limited to the detection of normative conflicts, using the tool prototype CLAN. In line with the aims listed in the beginning of this paper, we have applied our framework to two case studies as a proof-of-concept of the system, detailing the iterative process that writing and revising such contracts involves. These two cases studies have been specifically chosen from unrelated domains (one a document describing the working procedure of a check-in ground crew,

and the other a legal contract on Internet services) in order to demonstrate that the CNL used is a general one. AnaCon is indeed agnostic in what concerns the content or final intention of the document to be analysed; what is important is that it contains clauses that could be analysed for normative conflicts.

While the mapping between $\mathcal{CL}$ and our CNL may seem trivial, we believe that the use of an intermediary CNL has some important benefits. As the CNL is more human-focused than the purely logical $\mathcal{CL}$, certain unnatural logical constructions have no equivalent representation in the CNL. In this sense, the CNL is strictly *less expressive* than $\mathcal{CL}$. Yet the nearness of CNL to regular unrestricted natural language, when compared to a purely formal language like $\mathcal{CL}$ can go a long way towards making the authoring of such contracts easier. The use of our CNL also allows actions names to contain arbitrary strings, which may convey valuable information for the human reading of the contract. They can also be very helpful when it comes to understanding the output of the conflict analysis step and identifying the source of conflicts within a contract. This is in fact a general property of CNLs; while it is true that constructing valid sentences in a CNL does require *some* training (although still less than is required to write pure logical formulas), *understanding* something written in CNL should be effortless for any speaker of the parent NL. In other words, the benefits of using CNL as a verbalisation for some formal language can be felt by both authors and readers.

6.1. Limitations

The intention for AnaCon is that it can become a general framework for analysis of any text which contains normative clauses. While the two case studies presented in this paper make a good argument for the framework's generalisability, we recognize that more extensive work would be required for it to reach that stage. Aside from this, we also identified a number of smaller issues with the current implementation.

First, though $\mathcal{CL}$ can be used as a formal language to specify normative texts in general, many aspects have to be abstracted away from, such as for instance timing constraints. Other limitations of $\mathcal{CL}$, and similar contract languages, are described in [28].

Secondly, there is the issue of CLAN efficiency. The current version is not optimised to obtain small non-redundant automata. The tool is very much a specialised explicit model checker, where a high number of transitions is generated due to the occurrence of concurrent actions. One practical way to reduce the size of the automaton created by CLAN is to try to identify and list as many mutually exclusive actions as possible. Note that some of the actions in our case studies are obviously mutually exclusive from the logical point of view (e.g. *open the check in desk* and *close the check in desk*), while others are mutually exclusive in a pragmatic sense, that is we know that they cannot occur at the same time (for instance, *issue a fine* and *issue the boarding pass*, if we consider that these actions are done by the same person). The performance of CLAN might be considerably improved by reducing the size of the automaton

while building it, though a more fundamental way of improving it would be by eliminating & from $\mathcal{CL}$ as discussed in Section 4.3.

Third, CLAN is limited to conflict analysis and clearly it could be replaced by a more general model checker to check richer properties of normative documents in general, and contracts in particular.

As noted in sections 4.1 and 4.2, during the modelling and analysis of our two case studies problems were encountered with the *always* operator, expressed in $\mathcal{CL}$ as the prefix `[1*]`. While conceptually it is convenient and easy to think of a clause applying at all times, when modelled in $\mathcal{CL}$ and interpreted in CLAN it becomes clear that the true meaning of *always* in the natural sense is harder to formalise than anticipated. In order to overcome these issues, in this paper we were forced to exclude the use of this operator and instead model each contract as only covering a single "instance". The justification behind this is that if a contract holds for a single sequence of events, then it could later be generalised to run on concurrent instances of such sequences. In particular, we could consider adding features to the language to being able to distinguish between different instances of a contract, as done in the language FLAVOR [29].

6.2. Future work

Though in this paper we are not directly concerned with the translation from NL into CNL, it is worth mentioning that such translations could be carried out in a semi-automatic manner using guided-input techniques, or even better by using machine-translation.

In what concerns the ease of using CNL (vs. the use of a formal language) it could be very informative to perform experiments on different groups of users to have a qualitative analysis on the use of CNL and $\mathcal{CL}$. Evaluating CNL is not easy in general, and any experiment to do so should be carefully designed [30].

Another interesting future work concerns the use of passage retrieval tools like JIRS [27, 31] to help finding the counter-examples in the original English contract. This could be done by sending the CNL output from AnaCon to JIRS to automatically get a list of possible clauses where a conflict may arise. We envisage in this way a big increase in efficiency and precision when analysing counter-examples.

Finally, we believe that the development of a legal corpus could improve our CNL, giving the possibility to get a richer language even closer to natural language and enhancing the potential for obtaining a semi-automatic translation from NL documents into CNL.

Acknowledgements. We would especially like to thank Seyed Montazeri and Nivir Roy for their work on implementing a first version of AnaCon. We also thank Aarne Ranta for his help concerning GF, and Stephen Fenech for his input concerning CLAN.

References

[1] A. Wyner, K. Angelov, G. Barzdins, D. Damljanovic, B. Davis, N. Fuchs, S. Hoefler, K. Jones, K. Kaljurand, T. Kuhn, M. Luts, J. Pool, M. Rosner, R. Schwitter, J. Sowa, On controlled natural languages: properties and prospects, in: CNL'09, Vol. 5972 of LNCS/LNAI, Springer-Verlag, 2010, pp. 281–289.

[2] S. M. Montazeri, N. Roy, G. Schneider, From Contracts in Structured English to CL Specifications, in: FLACOS'11, Vol. 68 of EPTCS, 2011, pp. 55–69.

[3] C. Prisacariu, G. Schneider, A Formal Language for Electronic Contracts, in: FMOODS, Vol. 4468 of LNCS, Springer, 2007, pp. 174–189.

[4] C. Prisacariu, G. Schneider, CL: An Action-based Logic for Reasoning about Contracts, in: WOLLIC'09, Vol. 5514 of LNCS, Springer, 2009, pp. 335–349.

[5] C. Prisacariu, A dynamic deontic logic over synchronous actions, Ph.D. thesis, Department of Informatics, University of Oslo, Oslo, Norway (2010).

[6] P. McNamara, Deontic Logic, in: Gabbay, D.M., Woods, J., eds.: Handbook of the History of Logic, Vol. 7, North-Holland Publishing, 2006, pp. 197–289.

[7] C. Prisacariu, G. Schneider, A dynamic deontic logic for complex contracts, Journal of Logic and Algebraic Programming 81 (4) (2012) 458–490.

[8] S. Fenech, G. J. Pace, G. Schneider, CLAN: A tool for contract analysis and conflict discovery, in: ATVA'09, Vol. 5799 of LNCS, Springer, 2009, pp. 90–96.

[9] N. E. Fuchs, K. Kaljurand, T. Kuhn, Attempto Controlled English for Knowledge Representation, in: Reasoning Web, Vol. 5224 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2008, pp. 104–124.

[10] J. J. Camilleri, G. J. Pace, M. Rosner, Controlled Natural Language in a Game for Legal Assistance, in: Controlled Natural Language, Vol. 7175 of LNCS, Springer Berlin / Heidelberg, 2012, pp. 137–153.

[11] R. Harper, F. Honsell, G. Plotkin, A framework for defining logics, J. ACM 40 (1993) 143–184.

[12] A. Ranta, Grammatical Framework: Programming with Multilingual Grammars, CSLI Publications, Stanford, 2011.

[13] H. B. Curry, Some logical aspects of grammatical structure, in: Structure of Language and its Mathematical Aspects: Proceedings of the Twelfth Symposium in Applied Mathematics, American Mathematical Society, 1963, pp. 56–68.

[14] A. Ranta, The GF resource grammar library, Linguistic Issues in Language Technology 2 (2).

[15] K. Angelov, Incremental parsing with parallel multiple context-free grammars, in: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL '09), Association for Computational Linguistics, 2009, pp. 69–76.

[16] A. S. Hornby, Oxford Advanced Learner's Dictionary of Current English, Third Edition, Oxford University Press, 1974.

[17] R. Mitton, A partial dictionary of English in computer-usable form, Literary & Linguistic Computing 1 (1986) 214–215.

[18] K. Toutanova, C. D. Manning, Enriching the knowledge sources used in a maximum entropy part-of-speech tagger, in: Proceedings of the 2000 Joint SIGDAT Conference EMNLP/VLC, Vol. 13 of EMNLP, Association for Computational Linguistics, 2000, pp. 63–70.

[19] S. Fenech, G. J. Pace, G. Schneider, Automatic Conflict Detection on Contracts, in: ICTAC'09, Vol. 5684 of LNCS, Springer, 2009, pp. 200–214.

[20] G. J. Pace, C. Prisacariu, G. Schneider, Model checking contracts –a case study, in: ATVA'07, Vol. 4762 of LNCS, Springer-Verlag, 2007, pp. 82–97.

[21] N. E. Fuchs, U. Schwertel, R. Schwitter, Attempto controlled english (ACE) language manual, version 3.0, Tech. Rep. 99.03, Department of Computer Science, University of Zurich (August 1999).

[22] K. Angelov, A. Ranta, Implementing controlled languages in GF, in: CNL'10, Vol. 5972 of LNCS, Springer, 2010, pp. 82–101.

[23] G. J. Pace, M. Rosner, A controlled language for the specification of contracts, in: Workshop on Controlled Natural Language (CNL'09), Vol. 5972 of LNCS, Springer, 2010, pp. 226–245.

[24] R. Hähnle, K. Johannisson, A. Ranta, An authoring tool for informal and formal requirements specifications, in: FASE, Vol. 2306 of LNCS, Springer, 2002, pp. 233–248.

[25] S. Höfler, A. Bünzli, Designing a controlled natural language for the representation of legal norms, in: Second Workshop on Controlled Natural Languages, 2010, p. online.

[26] S. Höfler, Legislative drafting guidelines: How different are they from controlled language rules for technical writing?, in: Controlled Natural Language - Third International Workshop, CNL 2012, no. 7427 in Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg, 2012, pp. 138–151, cNL 2012.

[27] P. Rosso, S. Correa, D. Buscaldi, Passage retrieval in legal texts, Journal of Logic and Algebraic Programming 80 (35) (2011) 139 – 153.

[28] G. J. Pace, G. Schneider, Challenges in the specification of full contracts, in: Integrated Formal Methods (iFM'09), Vol. 5423 of LNCS, 2009, pp. 292–306.

[29] R. Thion, D. L. Métayer, Flavor: A formal language for a posteriori verification of legal rules, in: POLICY'11, IEEE Computer Society, 2011, pp. 1–8.

[30] T. Kuhn, An evaluation framework for controlled natural languages, in: Proceedings of the 2009 conference on Controlled natural language (CNL'09), Vol. 5972 of LNCS, Springer, 2010, pp. 1–20.

[31] D. Buscaldi, P. Rosso, J. Gómez-Soriano, E. Sanchis, Answering Questions with an n-gram based Passage Retrieval Engine, Journal of Intelligent Information Systems 34 (2) (2009) 113–134.

A. Cases study 1: Airline check-in process

A.1. Contract CNL

A.1.1. First version

```
[clauses]
if {the flight} leaves {in two hours}
then each of
  - {the ground crew} must open {the check-in desk 2 hours
      before} and
  {the ground crew} must request {the passenger manifest
      from the airline}, otherwise
  {the ground crew} must pay {a fine}
  - {the airline crew} must provide {the passenger manifest
      to the ground crew}, otherwise
  {the airline crew} must pay {a fine}
  - if {the ground crew} open {the check-in desk 2 hours
      before}
  then first {the check-in crew} must initiate {the check-in
       process},
  {the check-in crew} must check {that the passport details
      match what is written on the ticket},
  {the check-in crew} must check {that the luggage is within
       the weight limits}, then
  {the check-in crew} must issue {the boarding pass},
      otherwise
  {the check-in crew} must pay {a fine}
  - {the ground crew} must not issue {any boarding passes
      without inspecting that the details are correct
      beforehand}, otherwise
  {the ground crew} must pay {a fine}
  - {the ground crew} must close {the check-in desk 20 mins
      before flight leaves}, otherwise
  {the ground crew} must pay {a fine}
  - if {the ground crew} closes {the check-in desk 20 mins
      before flight leaves} then
  both
    - {the ground crew} must send {the luggage information
        to the airline}
    - {the ground crew} must not issue {boarding pass}
    - {the ground crew} must not reopen {the check-in desk}
[/clauses]
[contradictions]
  {the ground crew} open {the check-in desk 2 hours before}
      # {the ground crew} close {the check-in desk 20 mins
      before flight leaves} ;
  {the ground crew} open {the check-in desk 2 hours before}
      # {the ground crew} reopen {the check-in desk} ;
  {the check-in crew} check {that the passport details match
       what is written on the ticket} # {the check-in crew}
```

```
      check {that the luggage is within the weight limits} ;
  {the check-in crew} issue {the boarding pass} # {the check
      -in crew} check {that the passport details match what
      is written on the ticket} ;
[/contradictions]
```

A.1.2. Second version

```
[clauses]
if {the flight} leaves {in two hours} then both
  - {the ground crew} must open {the check-in desk 2 hours
      before}
  - if {the ground crew} open {the check-in desk 2 hours
      before} then both
    - {the ground crew} must request {the passenger manifest
        from the airline}, otherwise {the ground crew} must
       pay {a fine}
    - if {the ground crew} requests {the passenger manifest
       from the airline} then both
      - {the airline crew} must provide {the passenger
          manifest to the ground crew}, otherwise {the
          airline crew} must pay {a fine}
      - if {the airline crew} provides {the passenger
          manifest to the ground crew} then each of
        - first {the check-in crew} must initiate {the check
            -in process},
        {the check-in crew} must check {that the passport
            details match what is written on the ticket},
        {the check-in crew} must check {that the luggage is
            within the weight limits}, then
        {the check-in crew} must issue {the boarding pass},
            otherwise
        {the check-in crew} must pay {a fine}
        - {the ground crew} must close {the check-in desk 20
            mins before flight leaves}, otherwise {the
            ground crew} must pay {a fine}
        - if {the ground crew} closes {the check-in desk 20
            mins before flight leaves} then each of
          - {the ground crew} must send {the luggage
              information to the airline}
          - {the ground crew} must not issue {boarding pass}
          - {the ground crew} must not reopen {the check-in
              desk}
[/clauses]
[contradictions]
...
[/contradictions]
```

A.1.3. Third version

```
[clauses]
```

```
if {the flight} leaves {in two hours} then both
  - {the ground crew} must open {the check-in desk 2 hours
      before}
  - if {the ground crew} open {the check-in desk 2 hours
      before} then both
    - {the ground crew} must request {the passenger manifest
        from the airline}, otherwise {the ground crew} must
       pay {a fine}
    - if {the ground crew} requests {the passenger manifest
       from the airline} then both
      - {the airline crew} must provide {the passenger
         manifest to the ground crew}, otherwise {the
         airline crew} must pay {a fine}
      - if {the airline crew} provides {the passenger
         manifest to the ground crew} then both
        - first {the check-in crew} must initiate {the check
           -in process},
        {the check-in crew} must check {that the passport
           details match what is written on the ticket},
        {the check-in crew} must check {that the luggage is
           within the weight limits}, then
        {the check-in crew} must issue {the boarding pass},
           otherwise
        {the check-in crew} must pay {a fine}
        - if {the flight} leaves {in 20 mins} then both
          - {the ground crew} must close {the check-in desk
             }, otherwise {the ground crew} must pay {a fine
             }
          - if {the ground crew} closes {the check-in desk}
             then each of
            - {the ground crew} must send {the luggage
               information to the airline}
            - {the ground crew} must not issue {boarding
               pass}
            - {the ground crew} must not reopen {the check-
               in desk}
[/clauses]
[contradictions]
...
[/contradictions]
```

A.2. Generated action dictionaries

A.2.1. First version

```
b4 = {the check-in crew} check {that the passport details
    match what is written on the ticket}
b2 = {the airline crew} pay {a fine}
b7 = {the check-in crew} pay {a fine}
b3 = {the flight} leave {in two hours}
b6 = {the ground crew} close {the check-in desk 20 mins
    before flight leaves}
```

```
a1 = {the ground crew} issue {boarding pass}
b5 = {the ground crew} issue {any boarding cards without
    inspecting that the details are correct beforehand}
a3 = {the ground crew} pay {a fine}
a2 = {the ground crew} request {the passenger manifest from
    the airline}
a5 = {the airline crew} provide {the passenger manifest to
    the ground crew}
a4 = {the ground crew} reopen {the check-in desk}
a7 = {the check-in crew} check {that the luggage is within
    the weight limits}
a6 = {the check-in crew} issue {the boarding pass}
a9 = {the ground crew} open {the check-in desk 2 hours
    before}
a8 = {the check-in crew} initiate {the check-in process}
b1 = {the ground crew} send {the luggage information to the
    airline}
```

A.2.2. Second version

```
b4 = {the check-in crew} check {that the passport details
    match what is written on the ticket}
b2 = {the airline crew} pay {a fine}
b3 = {the flight} leave {in two hours}
b6 = {the check-in crew} pay {a fine}
a1 = {the ground crew} issue {boarding pass}
b5 = {the ground crew} close {the check-in desk 20 mins
    before flight leaves}
a3 = {the ground crew} pay {a fine}
a2 = {the ground crew} request {the passenger manifest from
    the airline}
a5 = {the airline crew} provide {the passenger manifest to
    the ground crew}
a4 = {the ground crew} reopen {the check-in desk}
a7 = {the check-in crew} check {that the luggage is within
    the weight limits}
a6 = {the check-in crew} issue {the boarding pass}
a9 = {the ground crew} open {the check-in desk 2 hours
    before}
a8 = {the check-in crew} initiate {the check-in process}
b1 = {the ground crew} send {the luggage information to the
    airline}
```

A.2.3. Third version

```
b4 = {the flight} leave {in two hours}
b2 = {the ground crew} send {the luggage information to the
    airline}
b7 = {the check-in crew} pay {a fine}
b3 = {the airline crew} pay {a fine}
b6 = {the ground crew} close {the check-in desk}
```

```
a1 = {the ground crew} issue {boarding pass}
b5 = {the check-in crew} check {that the passport details
    match what is written on the ticket}
a3 = {the ground crew} pay {a fine}
a2 = {the ground crew} request {the passenger manifest from
    the airline}
a5 = {the flight} leave {in 20 mins}
a4 = {the ground crew} reopen {the check-in desk}
a7 = {the check-in crew} issue {the boarding pass}
a6 = {the airline crew} provide {the passenger manifest to
    the ground crew}
a9 = {the check-in crew} initiate {the check-in process}
a8 = {the check-in crew} check {that the luggage is within
    the weight limits}
b1 = {the ground crew} open {the check-in desk 2 hours
    before}
```

B. Case study 2: Internet service provider

B.1. Contract CNL

B.1.1. First version

```
[clauses]
always each of
  - {the Client} shall not provide {false information to the
       Client Relations Department of the Provider},
      otherwise
  {the Provider} may suspend {the Internet service
      immediately}
  - if {the Internet traffic} becomes {high} then {the
      Client} must pay {price P}, otherwise
  first {the Client} must notify {the Provider by sending an
       e-mail specifying that he will pay later},
  {the Client} must lower {the Internet traffic to the
      normal level}, then
  {the Client} must pay {price 2P}, otherwise
  {the Client} is required to pay {price 3P}
  - if {the Internet Service} becomes {operative} then {the
      Client} shall submit {within seven days the Personal
      Data Form from his account on the Provider web page to
      the Client Relations Department of the Provider}
[/clauses]
[contradictions]
  {the Internet traffic} become {high} # {the Internet
      Service} become {operative} ;
  {the Provider} suspend {the Internet service immediately}
      # {the Internet traffic} become {high} ;
  {the Client} provide {false information to the Client
      Relations Department of the Provider} # {the Internet
      traffic} become {high} ;
```

```
  {the Client} pay {price P} # {the Client} pay {price 2P} ;
  {the Client} pay {price 2P} # {the Client} pay {price 3P}
      ;
  {the Client} pay {price 3P} # {the Client} pay {price P} ;
  {the Client} pay {price P} # {the Client} notify {the
      Provider by sending an e-mail specifying that he will
      pay later} ;
  {the Client} pay {price 2P} # {the Client} notify {the
      Provider by sending an e-mail specifying that he will
      pay later} ;
  {the Client} pay {price 3P} # {the Client} notify {the
      Provider by sending an e-mail specifying that he will
      pay later} ;
  {the Client} provide {false information to the Client
      Relations Department of the Provider} # {the Client}
      submit {within seven days the Personal Data Form from
      his account on the Provider web page to the Client
      Relations Department of the Provider} ;
[/contradictions]
```

B.1.2. Second version

```
[clauses]
{the Client} must submit {the data} ;
if {the Client} submits {the data} then each of
  - {the Provider} must check {the data}
  - if first {the Provider} checks {the data}, then {the
      Provider} disapproves {the data} then {the Provider}
      may cancel {the contract}
  - if first {the Provider} checks {the data}, then {the
      Provider} approves {the data} then each of
    - {the Internet Service} must become {operative}
    - if {the Internet Service} becomes {operative} then
        always
      if {the Internet traffic} becomes {high} then {the
        Client} must pay {price P}, otherwise
      first {the Client} must notify {the Provider by sending
        an e-mail specifying that he will pay later},
      {the Client} must lower {the Internet traffic to the
        normal level}, then
      {the Client} must pay {price 2P}, otherwise
      {the Client} is required to pay {price 3P} ;
[/clauses]
[contradictions]
  {the Client} submit {the data} # {the Provider} check {the
      data} ;
  {the Provider} approve {the data} # {the Provider}
      disapprove {the data} ;
  {the Client} submit {the data} # {the Internet Service}
      become {operative} ;
```

```
  {the Provider} approve {the data} # {the Internet Service}
      become {operative} ;
  {the Provider} disapprove {the data} # {the Internet
     Service} become {operative} ;
  {the Client} pay {price P} # {the Client} pay {price 2P} ;
  {the Client} pay {price 2P} # {the Client} pay {price 3P}
     ;
  {the Client} pay {price 3P} # {the Client} pay {price P} ;
[/contradictions]
```

B.1.3. Third version

```
[clauses]
{the Client} must submit {the data} ;
if {the Client} submits {the data} then each of
  - {the Provider} must check {the data}
  - if first {the Provider} checks {the data}, then {the
     Provider} disapproves {the data} then {the Provider}
     may cancel {the contract}
  - if first {the Provider} checks {the data}, then {the
     Provider} approves {the data} then each of
    - {the Internet Service} must become {operative}
    - if {the Internet Service} becomes {operative} then
       always
    if {the Internet traffic} becomes {high} then {the
       Client} must pay {price P}, otherwise
    first {the Client} must notify {the Provider by sending
       an e-mail specifying that he will pay later},
    {the Client} must lower {the Internet traffic to the
       normal level}, then
    {the Client} must pay {price 2P}, otherwise
    {the Client} is required to pay {price 3P} ;
[/clauses]
[contradictions]
  {the Client} submit {the data} # {the Provider} check {the
      data} ;
  {the Provider} approve {the data} # {the Provider}
     disapprove {the data} ;
  {the Client} submit {the data} # {the Internet Service}
     become {operative} ;
  {the Provider} approve {the data} # {the Internet Service}
      become {operative} ;
  {the Provider} disapprove {the data} # {the Internet
     Service} become {operative} ;
  {the Client} pay {price P} # {the Client} pay {price 2P} ;
  {the Client} pay {price 2P} # {the Client} pay {price 3P}
     ;
  {the Client} pay {price 3P} # {the Client} pay {price P} ;
  {the Client} pay {price P} # {the Internet traffic} become
      {high} ;
```

```
  {the Client} pay {price 2P} # {the Internet traffic}
     become {high} ;
  {the Client} pay {price 3P} # {the Internet traffic}
     become {high} ;
  {the Internet traffic} become {high} # {the Client} notify
      {the Provider by sending an e-mail specifying that he
     will pay later} ;
  {the Internet traffic} become {high} # {the Client} lower
     {the Internet traffic to the normal level} ;
[/contradictions]
```

B.2. Generated action dictionaries

B.2.1. First version

```
a1 = {the Provider} suspend {the Internet service
    immediately}
a3 = {the Client} pay {price 3P}
a2 = {the Client} notify {the Provider by sending an e-mail
    specifying that he will pay later}
a5 = {the Internet Service} become {operative}
a4 = {the Internet traffic} become {high}
a7 = {the Client} provide {false information to the Client
    Relations Department of the Provider}
a6 = {the Client} submit {within seven days the Personal
    Data Form from his account on the Provider web page to
    the Client Relations Department of the Provider}
a9 = {the Client} pay {price 2P}
a8 = {the Client} pay {price P}
b1 = {the Client} lower {the Internet traffic to the normal
    level}
```

B.2.2. Second and third version

```
b2 = {the Client} lower {the Internet traffic to the normal
    level}
b3 = {the Client} submit {the data}
a1 = {the Client} notify {the Provider by sending an e-mail
    specifying that he will pay later}
a3 = {the Provider} disapprove {the data}
a2 = {the Client} pay {price 3P}
a5 = {the Internet Service} become {operative}
a4 = {the Provider} cancel {the contract}
a7 = {the Provider} check {the data}
a6 = {the Client} pay {price P}
a9 = {the Internet traffic} become {high}
a8 = {the Provider} approve {the data}
b1 = {the Client} pay {price 2P}
```