# Controlled Natural Language in a Game for Legal Assistance

John J. Camilleri, Gordon J. Pace and Michael Rosner
University of Malta

**Abstract** This paper addresses the design of an automated legal assistant capable of performing a logical analysis of legal documents and using natural language as a medium of communication with a human client. We focus on the interplay between *natural language* in which the legal document is expressed and the *formal logic* used for reasoning about it — ideally approached using a controlled natural language (CNL) together with an appropriately chosen logic for analysis and reasoning. In translating from CNL to logic, information about the CNL structure is lost. For example, the CNL might contain legal clause numbers, whilst the logic might not. This can lead to problems when for example the reasoning system discovers an inconsistency in the contract and needs to explain its whereabouts to the client. Below we discuss the issues affecting the choice of logic, arguing in favour of keeping certain structural information during formal analysis of legal documents to be able to refer to that structure when interacting with the user.

We present a framework in which to experiment and seek solutions to these issues. Having identified a sufficiently restricted domain of application we also report on the development of a CNL to interact with a variant of the game Nomic — a game based on the notion of contract specification and amendment — and argue how this game provides an ideal platform to explore the use of structure information in the domain of legal analysis.

## 1 Introduction

In practice, the notion of legal assistance is extremely broad, encompassing a range of possible activities involving a legal expert, who offers the assistance, and a client who receives it. These activities will vary according to the type of legal expert in question (e.g. a barrister; a notary; a solicitor), the client (e.g. another lawyer; the victim of a crime; a multi-national company), and the nature of the law in question (criminal; civil; international). A barrister, for example, being a lawyer qualified to present cases in court, must be familiar with court procedures and in a position to offer advice with respect to those procedures. The kind of advice would be different for the victim of a crime, than for the lawyer in charge of the office who is engaging him to work on a case. In the first case, it might be important to warn the victim about tricky questions that might be asked by the defense lawyer, whilst in the second, the cost of the case, its duration, and its chances of success might be more to the point.

In this paper we present our long-range aim towards the creation of an artificial legal assistant that is capable of performing a useful legalistic task on behalf of a human client, and of being able to carry out that task using natural language as a medium of communication. We have had to make a whole series of strong assumptions in order to develop a working model. These centre around (i) the choice of a suitable subarea

within which the notion of assistance can be developed, (ii) a formal model of legalistic reasoning which will underpin the whole enterprise, and (iii), an appropriate setting in which to investigate the practical issues by building a testable artifact. The major challenge lies in the interaction of the natural language descriptions and the formal reasoning about legal tests. While much work has been done in both fields, a legal assistant needs to be positioned in between — on one hand processing natural language legal texts and explaining the results of analysis in natural language, while on the other hand performing formal analyses for issues such as contract inconsistencies and calculating consequences of a course of action. In order to explain consequences with respect to the original text, it is crucial to keep track of its structure.

In section 2 we briefly survey the natural language issues, concentrating on legal contracts. Two particular aspects of legal assistance are developed in this section. One concerns the notion of assistance itself; the other concerns the medium in which contracts are expressed, and with respect to which assistance is offered. For the purpose of putting an upper bound on the complexity of the assistance problem, we have developed a CNL called BanaL, discussed in section 3.3, capable of expressing certain key contract-related concepts. Section 3 relates these issues to the controlled, interactive setting in which we have chosen to carry out initial experiments. The control is enforced in terms of a CNL that is used for communication, and the interactivity is controlled by embedding the interaction within a simple rule-based internet-based game called BanaNomic. This environment has been designed to raise the level of access to a live but non-specialist audience. We believe that such settings not only lower the barrier to participation in a general sense, but provide a good setting in which to carry out much-needed evaluation activities. We then turn our focus to the challenge of dealing with layout and structural information in section 4 — a little-studied phenomenon which plays a crucial role in making written contracts understandable. We thus briefly survey the work that has been carried out from a linguistic perspective on layout and outline a way of dealing with this issue from a semantic point of view.

The aim of the paper is to highlight the challenges in formal reasoning about legal texts through a CNL interface, and set out a framework in which to experiment with and seek solutions to these issues, as stated in the conclusion appearing in section 5.

## 2 Artificial Legal Assistance

Clearly, in order to develop the area of automated legal assistance, we need to further narrow our view and focus our attention upon an area of which offers a set of non-trivial problems and use-cases for us to study in detail, and which is simple enough for a set of computationally plausible methods to be brought to bear. In an earlier paper we took a first step in this direction by focusing on assistance with respect to contracts (Pace and Rosner [PR10]). The reasons for focusing on contracts are first that contracts are less complex than the law in general, and second that they are are *legal documents* agreed between *parties* which regulate the future *behaviours* of those parties[1]. A key

[1] Contracts are defined at http://legal-dictionary.thefreedictionary.com/contract as *"an agreement with specific terms between two or more persons or entities in which there is a promise to do something in return for a valuable benefit known as a consideration"*

characteristic is that they make good candidates for formalisation: attempts have been made at giving a formal analysis to all the italicised words in the present paragraph, i.e.

- *Legal documents* are finitely expressed as texts in natural language. Computational linguistics and natural language processing are devoted to the problem of developing computational machinery to handle just these kinds of text.
- *Parties*, whether simple or complex, animate or inanimate, be represented as logical individuals, to which variables over certain domains can be bound. In this way we have a kind of minimal requirement for further analysis. As individuals, we can given them any properties we please. We can elaborate relationships between such individuals, or between individuals and other kinds of entity that we are prepared to talk about, such as goods and services. We can model intentions in terms of goals that are "possessed" by such individuals – and hence we can involve individuals in different kinds of intentional behaviour or action.
- *Behaviours* along with actions have long been an object of study in Artificial Intelligence (cf. Pat Hayes [Hay71]) where typically, they have been conceptualised as state-to-state functions, or as "events" à la Davidson in his much-cited essay 'The Logical Form of Action Sentences [Dav67].

### 2.1 Legal Assistance with Contracts

So, how can we elaborate legal assistance with respect to contracts? Daskalopulu and Sergot [DS97] distinguish between two main kinds of activity associated with contracts: contract *formation* and contract *performance*. During contract formation, the parties specify what they want the contract to express, ensure that it is accurately expressed, and finally arrive at an agreement of some kind with respect to a representation of the contract, which is normally a document of some kind. Contract performance, on the other hand, takes place after the contract is in place. Consequently we might envisage two kinds of tool that respectively address these two aspects.

- *Contract formation tools*, which are designed to support the process of producing a document that is correct both in the syntactic sense, with respect to form, and in the semantic sense of being true to the intentions of the parties. Typically, we can imagine an incremental approach in which the final contract emerges out of a dialogue with the client involving a series of successive approximations to the final contract.
- *Contact performance tools*, on the other hand, provide advice about the current state of execution of the contract offering suitable reminders to the parties whenever non-compliance is detected, possibly suggesting remedial action if this has been explicitly foreseen in some part of the contract. The BanaNomic game described in section 3.1 below is an example of a tool that falls into this category.

Central to both kinds of tool is the ability to provide *explanations* to the client about specific contract-related issues.

At the contract formation stage, it is fundamental that all parties to a contract understand the terms included in a contract as well as the rights and responsibilities to which they bind themselves. Thus the client might need different kinds of explanations e.g. concerning

- *terminology:*, Given the complexity of legal terminology, a glossary capability could provide for the explanation of technical terms - with highlighting of instances of those terms in the emerging contract.
- *understandability:*. A contract is understandable if and only if each of its parts are understandable. Therefore if the user does not understand the contract, there must be at least one part that the user does not understand. Therefore a coarse strategy for this kind of explanation would be to (i) identify the part or parts that that are not understood and (ii) explain the content of the part. Of course the kind of explanation would depend on the nature of the part (e.g. a clause; a section; a sentence; a phrase) and the nature of the client's lack of understanding (e.g. incomprehension versus misunderstanding versus disagreement).
- *consistency:* i.e concerning the consistency with respect to possible actions that a party might allow. A party might wish to know whether a contract permits, obliges or forbids a certain action from being executed or a state of affairs from coming into existence. The party might claim that the contract is impossible to satisfy, in which case the expert should demonstrate circumstances that satisfy it (or rephrase the contract if the client turns out to be right!).

At the contract performance stage, the two main activities are *monitoring* and *communication*.

- Monitoring involves keeping track of all obligations and prohibitions that the contract imposes on the parties. The heart of this activity revolves around the detection or measurement of concrete observables, which can be formulated in terms of either *actions* or *states*.
  In the action-based formulation, obligations and prohibitions are phrases in terms of actions that must be carried out - like paying a certain amount or delivering a certain document. In the state-based formulation, it is the presence or absence of properties of states that must be maintained, e.g. the a party shall maintain a positive bank balance. The choice between actions and states depends on the domain being dealt with. In the second case, for example, we are probably more interested in the state of having a negative balance than in the exact action which causes the account to become negative.
  In addition to keeping track, the monitoring process has to take special actions when things go wrong. Many contracts specify contrary-to-duty obligations which are imposed when a party fails to carry out an obligation. In such cases we would expect the monitoring process to keep track not only of the failed obligations, but of the newly imposed ones (this issue is discussed further in Pace and Rosner [PR09]). In the example cited, a negative balance could trigger an obligation to pay interest at an exhorbitant rate.
- Communication is necessary in order to inform the client about the dynamically changing state of current obligations. The bank balance goes negative, so the client must be informed that a new obligation to pay interest has arisen. Several issues surround such an act of communication, not the least of which is the timing. In the case cited, the timing would coincide with the moment at which the change of state took place – in other words *after* the action that broke the existing obligation. In other cases it might be possible or even necessary to issue a warning *before*

the action takes place in order to avoid the imposition of penalties. Another issue concerns explanation. Suppose the client wants a justification or simply further elaboration of the newly imposed obligation. Then the tool would need to provide an explanation, referring back to relevant part of the contract.

The above remarks suggest that for the two kinds of assistance we have considered, contract formation and contract monitoring, explanation enters into the picture. One of the main claims of this paper is that successful explanation requires knowledge of structure and of the way in which the structure relates to the underlying semantics that defines the legal concepts involved. These considerations are further elaborated in the next section.

## 3 CNLs, Contracts and Games

The use of CNLs to enable processing of and formal reasoning about statements in a particular domain is an established approach [Sch08,BSBS09]. By constraining the language, together with the structural complexity of the grammar, one obtains a means to make statements about the underlying domain, without moving too far off from a natural language description, so that it remains understandable to native language speakers. At the same time it is easy to translate into an suitable logic.

In identifying an appropriate domain-specific CNL, one faces two primary challenges — that of identifying the basic underlying concepts in the domain, and secondly that of selecting an appropriate and sufficiently rich grammar through which to combine these basic concepts. Going further, and enabling formal reasoning and manipulation of statements made in the CNL is further hindered by the fact that typically, relating the basic concepts together requires much tedious and error-prone work. Some work has been carried out on CNLs for contracts [PR09]. The primary concepts of interest here are deontic ones, namely obligations, permissions and prohibitions on actions or states. In practice, using a CNL for contracts requires not only semantics of the language operators, but also the underlying implicit concepts that the contract mentions.

Once the CNL has been defined, one also needs to define a setting in which the effectiveness of a CNL for contracts can be investigated empirically. In this paper we investigate the use of a CNL to describe game rules. In particular, to enable interesting cases, and the need for consistency checking of the rules, we apply the technique to implement a variant of the game of Nomic — a game in which changing the rules is part of the game itself.

### 3.1 Nomic and BanaNomic

Nomic is a game of self-amendment [Sub90] — starting with an initial rule set, each player takes their turn changing the game's rules through a system of rule proposals and player voting. What makes Nomic so particular is that *everything* is theoretically up for amendment during the game, including the voting system itself and what players need to do to win. Despite the popularity of the game, only one Nomic variant could be found which uses automated rule processing. The game is encoded and played directly in Perl [PB05], and circumvents the contract specification and processing by identifying

the contract with the Perl program governing the voting process — what the program accepts (or rejects) is considered to be the semantics of the program. Encoding the full game of Nomic with natural language contracts is particularly challenging, since it combines challenges in natural language analysis and formal reasoning about contracts. In this section, we show the challenges in combining formal reasoning with controlled natural language reasoning in the game, without the use of (i) layout information; and (ii) macro definitions. We then argue why we believe that the use of these will enhance the game and its interface.

The major challenges in Nomic playing using natural language contracts are twofold: (i) formulating a language in which the contract clauses are expressed — rich enough to be able to reason about notions such as permission and obligation; and (ii) the contracts frequently refer to statements about the real world which require a strong semantic framework ('Players wearing glasses cannot propose amendments to clauses labelled by a prime number'). The former problem we have addressed by developing a CNL called BanaL, which we discuss more concretely in the next section, and the latter was circumvented by reducing the domain of the game to a simpler setting.

BanaNomic is a more concrete version of Nomic, in which players represent monkeys living in a tree, fighting to pick bananas and defend their stash. The constitution corresponds to the rules of the jungle — and can refer to the state of affairs (e.g. how many bananas a player owns) and actions possible in this limited setting (e.g. climbing up the tree). The rules cannot be violated, but the monkeys are allowed to add and remove rules at will. During each turn, the players may carry out actions and modify the constitution. The game is governed by banana-time, thus enabling constitution clauses to refer to time.

### 3.2 A deontic contract logic and language for BanaNomic

Typically, most logics allow reasoning about a state of affairs — studying the consequences of what predicates hold or otherwise in a particular situation. In computer science, one finds various extensions of this notion to deal with the need for interaction with the actual state of the system. For instance, in runtime verification [HR01], one typically identifies properties using an appropriate logic, together with actions to be triggered upon violation of these properties. Reasoning about these extended systems presents an extended challenge due to the feedback between the properties and actions applied to the system behaviour. Similarly, legal reasoning deals with consequences of violations with respect to the state and actions of an entity, thus requiring this additional layer of cognition. In contrast with runtime monitoring, however, most of the consequences of violations are new (or modified) legal statements. For instance, the consequence of violating the obligation to pay one's subscription leads to a permission on the service provider to cancel the service and a further obligation to pay the due amount with additional interest. For millennia, effective reasoning about such contracts and legal texts has proved to be a major challenge for philosophers; more recently computer scientists have also become involved with the issues.

Moral and normative notions such as obligation, permission and intention, have been studied as far back as the time of Aristotle. The first formal analysis can be attributed von Wright [Wri51] in 1951 (although some authors identify Mally's work in 1926 to

be a precursor of this [Mal26]). This family of logics, called deontic logics, allow for reasoning at least about the notions of permission, obligation and prohibition. However, it turns out that even restricting the logic to these notions exposes various challenges and choices [McN06]. One is the inclusion of constructs to deal with contract violation, such as the concepts of *contrary-to-duty obligations* (CTD) and *contrary-to-duty prohibitions* (CTP). CTDs state the consequences of not respecting an obligation while CTPs similarly deal with prohibitions that might be violated. In both cases, one specifies the resulting clauses which are to be fulfilled as reparations in case of violation.

Two main approaches taken in the literature turn on whether the deontic notions are attached to actions or to states. Is one to be prohibited from moving the opponent's pieces (action-based) or is one to be prohibited from having more than 16 pieces on the board at the same time (state-based)? Although both approaches and their combination have been shown to be useful in practice for different domains, in our game setting, we adopt an action-based approach to avoid having to talk about causality of who brought about which parts of the state, and is thus responsible for the consequences.

The contract grammar used for BanaNomic is based on the deontic logic used in [PR10]. The deontic logic is based on three fundamental deontic modal operators: obligation $\mathbb{O}$, permission $\mathbb{P}$ and prohibition $\mathbb{F}$, and is action-based, in that all the deontic operators act on action expressions. Furthermore, all actions are tagged by their subject and object (if relevant) e.g. the action *throwBanana* takes both the name of the monkey throwing the banana, and the monkey at whom it is being thrown — $\mathit{throwBanana}(\mathit{Michael}, \mathit{Gordon})$. These basic actions can be combined together using sequential composition (;) and choice (+) to obtain action expressions such as:

$$(\mathit{throwBanana}(\mathit{Michael}, \mathit{Gordon}); \mathit{eatBanana}(\mathit{Gordon})) + \mathit{eatBanana}(\mathit{Michael})$$

To enable quantification over actors, rather than introducing explicit quantifiers, we borrow the notation used for polymorphic type place-holders from type systems, and enable quantification by using a name placeholder * e.g.: $\mathbb{F}(\mathit{throwBanana}(*, \mathit{John}))$ would be the statement saying that everyone is forbidden from throwing a banana at John. Ideally, in such a logic we would allow for different variable names, allowing us to unify different actors in a statement, but for the sake of a more direct mapping to and from the CNL, we assume that the instances of * all refer to different variables.

The contract is modelled as a function from the natural numbers to clauses, and is interpreted as the conjunction of all clauses. The clauses can be (i) deontic statements over action expressions; (ii) temporal operators — $\Box[b, e]C$ says that from time $b$ to time $e$ clause $C$ will always be enforced and $\Diamond[b, e]C$ says that at some time between time $b$ and $e$, clause $C$ will hold; (iii) choice operators — $C_1 + C_2$ says that one of $C_1$ and $C_2$ must hold and $C_1 \twoheadleftarrow q \twoheadrightarrow C_2$ checks whether query $q$ holds (queries are boolean expressions over the state of the game — how many bananas each player has, the height in the tree where each player can be found, etc) and enacts $C_1$ or $C_2$ accordingly; (iv) a consequence operator $C_1 \triangleleft DE \triangleright C_2$ which checks for the existence of deontic clause $DE$ and enacts clause $C_1$ or $C_2$ accordingly; and (v) the conjunction of two clauses

| Player | Rule enacted |
|---|---|
| 1. George | $\mathbb{F}(pickBanana(Paul))$<br>Paul is forbidden to pick a banana |
| 2. Paul | $\Diamond\,[0,9]\,\mathbb{O}(throwBanana(\phi, George))$<br>At some point before time 9 every player is obliged to throw a banana at George |
| 3. George | $\mathbb{F}(abolish(Paul,*)) \triangleleft \mathbb{P}(enact(George,*,*)) \triangleright Ok$<br>If George is permitted to enact a rule then Paul is forbidden to abolish any rule |
| 4. Paul | $\Box\,[0,\infty]\,\mathbb{F}(enact(George,*,*))$<br>At all times George is forbidden to enact a rule |

Table 1: Example of the rules enacted during a four-turn sequence of BanaNomic, showing rule clauses in formal notation along with their natural language linearisations. Other turn actions are omitted.

$C_1 \wedge C_2$. The syntax of the logic is as follows:

$$
\begin{aligned}
ActionExp &::= Action \mid ActionExp; ActionExp \mid ActionExp + ActionExp \\
DeonticExp &::= \mathbb{O}(ActionExp) \mid \mathbb{F}(ActionExp) \mid \mathbb{P}(ActionExp) \\
Clause &::= Ok \mid Fail \mid DeonticExp \mid Clause \wedge Clause \mid Clause + Clause \\
&\mid\ Clause \lll Query \ggg Clause \mid Clause \triangleleft DeonticExp \triangleright Clause \\
&\mid\ \Box\,[Time, Time]\,Clause \mid \Diamond\,[Time, Time]\,Clause
\end{aligned}
$$

Two of the basic actions which can be used in action expressions are *enact* and *abolish*, which refer to a particular player enacting or abolishing an existing clause. When used in conjunction with the deontic operators, one can express clauses about power e.g. $\mathbb{F}(enact(John,*,*))$ says that John is not allowed to enact any clause anywhere in the contract. Using the logic defined above, a few example contracts and their natural language readings are given in table 1.

The clauses are given an operational semantics, defining a relation $C \xrightarrow{a}_{\sigma} C'$ which says that when in game state is $\sigma$ and upon action $a$, contract $C$ evolves to $C'$ advancing forward in time. For example, $\Box\,[0,7]\,\mathbb{O}(a) \xrightarrow{a}_{\sigma} \Box\,[0,6]\,\mathbb{O}(a)$. The main advantage of adopting this approach is the also the reason it is frequently adopted for monitoring of systems: we can progressively consume actions coming from the players, updating the active game contract as required. The semantics of a number of the operators are given below to illustrate how the rules are defined.

**Basic clauses:** The base clauses *Ok* and *Fail* denote the trivial cases of the clause which can never be violated and the clause which will always lead to a violation. The rules for these clauses are rather straightforward:

$$\frac{}{Ok \xrightarrow{a}_{\sigma} Ok} \qquad\qquad \frac{}{Fail \xrightarrow{a}_{\sigma} Fail}$$

**Obligation:** If we are obliged to match an action expression $e$, then upon receiving action $a$, we have three possible situations: (i) $a$ matches the expression $e$, in which case the obligation can be discharged; (ii) after consuming $a$, one is still obliged to match action expression $e'$ (for example receiving $a$ when obliged to perform $a;b + a;c + d;e$ will result in an obligation to perform $b + c$); and (iii) $a$ cannot match action expression $e$, resulting in a violation:

$$\frac{}{\mathbb{O}(e) \xrightarrow{a}_\sigma Ok}\ e \xrightarrow{a} \checkmark \qquad \frac{}{\mathbb{O}(e) \xrightarrow{a}_\sigma \mathbb{O}(e')}\ e \xrightarrow{a} e' \qquad \frac{}{\mathbb{O}(a) \xrightarrow{a}_\sigma Fail}\ e \xrightarrow{a} \times$$

**Conjunction:** For the case of clause conjunction, we have special cases for the different cases when either conjunct reduces to *Ok* or *Fail*:

$$\frac{C_1 \xrightarrow{a}_\sigma Ok \qquad C_2 \xrightarrow{a}_\sigma C_2'}{C_1 \wedge C_2 \xrightarrow{a}_\sigma C_2'} \qquad \frac{C_1 \xrightarrow{a}_\sigma Fail}{C_1 \wedge C_2 \xrightarrow{a}_\sigma Fail}$$

$$\frac{C_1 \xrightarrow{a}_\sigma C_1' \qquad C_2 \xrightarrow{a}_\sigma Ok}{C_1 \wedge C_2 \xrightarrow{a}_\sigma C_1'} \qquad \frac{C_1 \xrightarrow{a}_\sigma C_1' \qquad C_2 \xrightarrow{a}_\sigma C_2'}{C_1 \wedge C_2 \xrightarrow{a}_\sigma C_1' \wedge C_2'}\ C_1',\ C_2' \notin \{Ok,\ Fail\}$$

**Queries:** The rules for queries use the game state $\sigma$ to choose which branch to follow:

$$\frac{}{C_1 \triangleleft q \triangleright C_2 \xrightarrow{a}_\sigma C_1}\ \sigma(q) \qquad \frac{}{C_1 \triangleleft q \triangleright C_2 \xrightarrow{b}_\sigma C_2}\ \neg\sigma(q)$$

The semantics of a contract made up of indexed clauses is then simply the lifting of these semantics over the clause locations.

### 3.3 BanaL, a CNL for BanaNomic user input

We have developed BanaL — a CNL for BanaNomic, designed as an application-specific method of natural language representation based on the syntax of the logic. This has the effect of making the conversion from contract logic to natural language and back (*linearisation* and *analysis*, respectively) very simple and deterministic. The Grammatical Framework (GF) [Ran04] was adopted for the guided-input methods it facilitates (see below), and its support for sophisticated forms of language generation — thus future-proofing the design so that subsequent versions of BanaL could easily be extended to include much more intelligent natural language realisation choices.

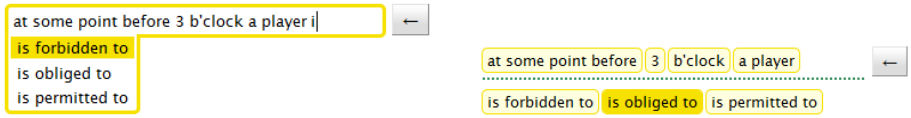
GF is a specialised functional language for defining grammars, having separate abstract/concrete syntax rules, a strong type system, and inherent support for multilinguality. GF grammars are declarative in nature, with a primary focus on the linearisation of syntax trees. By writing an abstract GF grammar and defining *how* it should be expressed in one or more natural languages (concrete grammars), GF is able to derive both a generator *and* a parser for each of those languages [Ran04].

Given the declarative nature of GF grammars, the abstract syntax of BanaNomic could very easily be implemented on the basis of its formal logic. For example, the abstract GF equivalent for the definition of the *Clause* category would be as follows:

```
cat
  DeonticExp ; Time ; Clause ;
fun
  C_Deontic     : DeonticExp -> Clause ;
  C_Always      : Time -> Time -> Clause -> Clause ;
  C_Conditional : DeonticExp -> Clause -> Clause -> Clause ;
  ...
```

For the design of the concrete grammar, each of the functions from the abstract syntax is given a template-like linearisation. While suitable for many cases, certain constructs required a more subtle approach in order to produce phrases which still sound natural. Nested phrases were particularly problematic to express unambiguously ('Paul is allowed to pick a banana and climb the tree or climb down the tree'), and the use of pronouns was avoided altogether.

A major part of GF is its partial evaluation algorithm (or *incremental parser*), which gives rise to interesting guided-input possibilities. By presenting the user with a list of possible words which may come next in a partial sentence, they are able to construct grammatical sentences in an auto-complete fashion. This is highly useful as it ensures that only syntactically-correct phrases are entered first-time round, and will avoid user frustration of trying to construct parseable sentences in free-text. The guided input methods developed for BanaNomic are based on the drop-down suggestions (figure 1a) and the "fridge magnets" (figure 1b) — developed by the GF team. These input methods are of particular interest to the area of CNLs, as they help avoid the problem of users having to know what is grammatical in a particular CNL.




(a) Drop-down suggestions (b) Fridge magnets

Figure 1: Guided input methods used in BanaNomic

### 3.4 Discussion

As should be evident from this section, the combination of CNL and formal reasoning in this setting is crucial, yet extremely challenging. From the implementation of the game, it is evident that the major issue is one of giving feedback and explanations to the users. While parsing of the CNL input and executing the formal semantics of the logic can be done using standard techniques, explaining the consequences of a rule, and how the rules change through the advancement of time is particularly challenging due to the flat nature of the contract language and logic. Allowing the CNL rules to be tagged with layout information is necessary to allow players to deal with longer, and thus more intricate, rules. However, explanations must then be adapted to maintain and refer to this structure as required. Another issue which hampered strategies in the game was the low-level nature of the logic constructs. Advanced players would benefit from the ability to

define macros to be able to build rules of higher complexity, without a proportionate increase in their length. As in the case of layout information, it is however crucial that the use of these macros in the rules is to be appropriately handled in user explanations. It is in this manner that we believe that the use of the techniques proposed in this paper will enable more complex strategies to be more tractable for human players.

## 4 Explanation, Structure and Layout

In the development of Nomic, we kept a simple contract layout, encoded as an enumerated list of clauses. Cross referencing was limited to references to locations of clauses, and due to the nature of the game, the clauses themselves were typically kept simple by the players. However, in real-world contracts, layout and cross-references play a much more important role. Also, certain contract structures recur multiple times, and would ideally be defined as syntactic sugar above the basic logic. Both these layers of non-functional information do not change the behaviour of the contract — whether a conjunction is enumerated or not, and when syntactic sugar is unrolled should not change the semantics of the contract. However, it changes the way natural language would be generated from the clauses in order to interact with the user. In this section we look at the challenges this poses, both from a CNL and logic perspective.

### 4.1 Layout and Presentation

An important aspect of contracts already alluded to is that they have a characteristic structure and layout which explanations must exploit in order to be successful. The dimension of layout arises because written text appears on a page and must therefore have visual characteristics which include not only structuring devices such as paragraph division and indentation but (e.g. in presentations) other structures such as bullet points. Of particular interest are ordered lists, whose internal structure and outward appearance are close the clause and section structure which characterises the presentation of many legal documents.

The question is where to look for further details of that structure. Legal textbooks provide a high-level overview. Under English Law, for example, we know that every contract includes four essential components: an offer, an acceptance, a consideration (the requirement of reciprocal obligations on the parties to a contract) and an intention to create legal relations between the parties. The lower level structure bottoms out in terms of sections and clauses, which, as we have said, closely resemble ordered lists.

Contracts display what might be called a *macro structure* and a *micro structure*. The macro structure deals with the main sections etc. and depends on the kind of contract under consideration. Vella [Vel10] for instance, studied several hundred property sale contracts under Maltese law and found that they always comprised a number of *sections* including

- a description of the parties;
- a description of the property;
- a description of the financial arrangements;
- the signatures of the parties;

– a date.

Apart from the fact that each of these sections played a distinct legal role (and can be related to the essential high-level components just mentioned), they were each characterised by the use of particular phrases, and also certain layout conventions including the use of certain keywords, phrases, fonts and capitalisation.

Clearly, different kinds of contract will have different macro structures, and these to a large extent could be successfully represented in the form of document templates, style sheets, etc. These can be very concrete (e.g. word template files) or else more abstract structures which have the advantage of being platform independent. There is a movement towards the formal representation of legal documents using XML[2].

The micro structure, on the other hand, concerns a level of presentation corresponding roughly to a collection of legal clauses. This transcends sentence structure, since a single clause be realised by two simple sentences in preference one complex sentence. It includes the order in which sentence forms occur, since this can reflect the order in which actions might have to be carried out. For example, under Maltese Law, you are obliged to pay all outstanding fines *before* applying for a road licence. It has to be in that specific order else the application will fail.

The micro structure also addresses punctuation. One of the problems with punctuation is that nobody can agree on the range of phenomena it includes. Is it just the use of certain punctuation symbols, or does it also include, say, list markers, indentation, text styles, fonts, etc? This is not just a theoretical discussion, since the way we answer this question will determine whether it can properly be referred to in explanations.

Opinions differ as to the importance of punctuation marks in legal texts, where traditionally, they are used sparsely. Nunberg [Nun90] argued strongly against a "transcriptional" view (Crystal and Davy [CD69]) according to which punctuation has an essentially secondary role. Instead, he claims that punctuation properly belongs to a distinct level of linguistic structure that is peculiar to the written language - *text structure* - that no less important that traditional sentence structure and exists alongside it. Nunberg introduces what we feel is an important distinction between this abstract text structure and the concrete graphical devices used to express it.

A practical application based on Nunberg's ideas is offered by Power-et-al [PSBA03], which describes a system, ICONOCLAST, for the generation of text structures of the kind mentioned from a meaning representation that includes not just propositional information, but the *rhetorical function* of that information.

The input to ICONOCLAST takes the form of a tree whose leaves are propositions, and whose nodes indicate the rhetorical relation between them. The latter is based on Mann and Thompson's Rhetorical Structure Theory (RST) [Man88], an attempt to provide a formal and computational basis for the description of rhetorical function. A simple example of such a structure is

**concession**(`ban(fda,elixir),approve(fda,exlixirplus)`)

[2] Legal XHTML - see `http://www.hypergrove.com/legalxhtml.org`.

where **concession** is the rhetorical relation holding between the propositions `ban(fda,elixir)` (the FDA[3] banned the drug Elixir) and `approve(fda,elixirplus)` (the FDA approved the drug Elixirplus).

The process of transforming such a rhetorical-semantic message into a document involves not only realising the basic propositions, but solving a series of realisation constraints holding between nodes of rhetorical tree and text-structures. Several solutions are possible and are in fact generated by ICONOCLAST:

- The FDA approves ElixirPlus, although it bans Elixir.
- Although the FDA bans Elixir, it approves ElixirPlus.
- The FDA bans Elixir, but it approves ElixirPlus.
- The FDA bans Elixir; but it approves ElixirPlus.
- The FDA bans Elixir. But it approves ElixirPlus.

The main claim underlying Power *et. al.*'s work is that to handle the phenomena present in the above sentences it is necessary to distinguish a third level of structure – rhetorical structure. This level is over and above the abstract and concrete levels of text structure postulated by Nunberg.

Finally, no discussion of the issue of layout could be complete without mentioning the ambitious research agenda of Bateman and colleagues [BKKR01]. Bateman's work addresses "the desirability of combining text, layout, graphics, diagrams, 'punctuation' and typesetting" for the most effective presentation of information. The scope is thus wider than Power-*et. al.*'s, but the general methodology is similar insofar as in both cases, the message to be conveyed is expressed using a rhetorical structure tree. The main difference is that whilst Power is concerned with transforming this into a text-structure, Bateman's concerns a wider-ranging *layout structure* whose nodes correspond to blocks that will be realised as regions on the printed page. Blocks *may* be realised by text-structures, but may also be expressed with different kinds of graphics such as images. Furthermore, blocks can participate in graphic relations, such as proximity, similarity of style, etc.

Power and Bateman have effectively both added a semantic basis in the form of the underlying layer of RST. It is the RST structure which defines *what* any potential realisation is supposed to express. The two approaches differ in the range of realisation phenomena considered. Power only considers text structures, whilst Bateman includes in addition other kinds of layout device.

### 4.2 Syntactic Sugar and Semantics

Well-formed contracts abiding by the syntactic rules of the domain may have multiple semantic interpretations depending on their application. For instance, to print out a contract, layout information plays a crucial role while the distinction between an obligation and a prohibition is limited to a superficial change in the symbol used. On the other hand, to monitor behaviour with respect to a contract, we require a semantic interpretation in which the distinction between obligation and prohibition is a major one, while layout information can be discarded without any loss of information. This approach

[3] Federal Drug Association

of having multiple semantic interpretations of the same syntax is a well-known one in other domains, such as embedded languages [Hud96]. For instance, in [CSS03], circuit descriptions have multiple interpretations, depending on how they are intended to be used. To print out a circuit, layout information is sufficient, while for the simulation or model checking of circuits, an interpretation which discards layout information, but uses the semantics of logic gates and latches is required. For other analysis, such as signal delay propagation (due to the gates and wire length) both types of information must be retained. The solution usually adopted, is to give independent semantics for each of these applications.

In the case of contracts, the main challenge comes with the need for explanation. Clearly, the original contract one starts off with, fully annotated with layout information, may use the annotations for explanations. However, as already discussed, contract performance requires the keeping track of obligations, prohibitions and permissions which might appear over time as the contract evolves. For instance, consider a clause which states that after three consecutive occurrences of action *bad-password*, the user is obliged to answer a *captcha*[4]. Explaining the state of the contract after two consecutive bad passwords, one would have to say that if one more wrong password is given, the user will have the obligation to answer a captcha. If this clause occurs within an extensive contract, giving this explanation, without giving a reference to the location of the clause, or presenting the 'evolved' clause as part of the contract as a whole will not be of much help to the person reading the explanation. Similarly, consider a conflict analysis procedure, which given a contract returns whether the contract may lead to a conflicting state e.g. having an action obliged and prohibited at the same time. Such potential conflicts are typically explained by giving a trace of actions which leads to the problem, and the conflict itself. However, in the case of extensive contracts, simply telling the user 'After the actions $\langle a,\ b,\ c\rangle$, action $d$ is both obliged and prohibited', is not of much use, since which parts of the contract give rise to these clauses may not be obvious. A better explanation would include information as to which parts of the contract led these clauses, or better still, allow the visualisation of how the contract evolves as the trace is traversed, finally highlighting the parts which have led to a conflict.

A related issue with human readability of contracts is that of syntactic sugar. Although the core logic one reasons in may just have a handful of constructs, one would typically have various compound constructs used in the contract, and which are defined in terms of the underlying basic ones. For instance, in a contract logic which allows for sequential composition, one may define the repetition of a contract $n$ times by unrolling it using sequential composition. As in the case of layout information, one would thus lose all such high-level constructs at the reasoning level thus hampering the explanation given back to the user when the need arises.

Both layout information and macro-definitions typically have no consequences on whether a trace leads to a violation of a contract, or whether a contract contains a conflict. Due to this, semantics of violation and conflict-analysis typically apply Occam's razor, ignoring all such information. The dilemma is that, although the layout inform-

[4] A captcha is a transformed visual representation of text, intended to identify non-human access to a resource. They are frequently used to avoid posting of spam and repeated password attempts from automated scripts.

ation is irrelevant to answer questions such as *'Does a conflict exist?',* it is relevant in *explaining* the result. One would like to be able to ensure that (i) the semantics extended to handle layout and macro-definition context information does not change the trace (and hence conflict) interpretation of a contract; and (ii) the logic retains as much of this information as possible, in order to aid the explanation process.

Adding layout information within the context-free structure of the logic involves having an additional constructor $tag_t(C)$, which tags contract $C$ with tag $t \in Tag$.

$$Clause ::= tag_{\text{Tag}}(Clause) \mid Ok \mid Fail \mid \ldots$$

Thus, as a simple example, we can add information about enumerated clauses using tags in a formula of the form: $tag_{lst}(tag_{lst:1}(\mathbb{O}(a)) \wedge tag_{lst:2}(\mathbb{F}(b)))$. Tags can be given a semantics to enable the transformation of such a clause into an enumerated list. Note that, even if one applies some analysis which commutes the conjunction operator to transform the clause into $tag_{lst}(tag_{lst:2}(\mathbb{F}(b)) \wedge tag_{lst:1}(\mathbb{O}(a)))$, the originally intended description can still be achieved.

Although these tags may have a layout semantics of their own, the addition of these tags will not change the normative semantics of the contract — which can be handled by the following rule:

$$\frac{C \xrightarrow{a}_{\sigma} C'}{tag_t(C) \xrightarrow{a}_{\sigma} tag_t(C)}$$

For instance, upon receiving an action $c$, one can show that the enumerated contract we saw earlier, behaves as follows:

$$tag_{lst}(tag_{lst:1}(\mathbb{O}(a)) \wedge tag_{lst:2}(\mathbb{F}(b))) \xrightarrow{c} tag_{lst}(tag_{lst:1}(Fail) \wedge tag_{lst:2}(Ok))$$

With the tags included one can still provide information regarding which of the subclauses failed due to action $c$. On its own, however, this rule is not sufficient to handle tags as one would expect. The problem is that by keeping the tag, other rules may be inhibited from firing. For example, the expression we obtain after action $c$ cannot be reduced to $tag_{lst}(Fail)$, as one would it expect it to. The only solution is such cases is to discard a tag within any syntactic context $\alpha$ by performing an internal action $\tau$:

$$\frac{}{\alpha(tag_t(C)) \xrightarrow{\tau}_{\sigma} \alpha(C)}\ deadlock_{\sigma}(\alpha(C))$$

The side-condition $deadlock(\alpha(C))$ is used delay untagging until no other external actions are possible[5].

In this manner, we can extend the syntax and semantics of a logic in such a manner to retain as much information as possible to be used for explanations. By refining and generalising the approach proposed here, one can retain much information, without the need for redefining the semantics from scratch. If the semantics of contracts remains unchanged under the addition of tags, one can perform all analysis in the untagged logic, and use the tagged inference rules only once a scenario or counter-example is discovered, which is to be explained in a controlled natural language. This approach can be ideal in a setting where the transitions system semantics of the logic with no annotations is automatically extended to maintain the necessary information for explanations.

[5] The predicate $deadlock_{\sigma}(C)$ can be defined as $\neg\exists a, C' \cdot C \xrightarrow{a}_{\sigma} C'$.

A related problem is that of the use of syntactic sugar or macros which is also ideally kept for explanations. For instance, one can define a macro *soon*($C$) which allows contract $C$ to be satisfied either now or in 5 time units: $soon(C) \stackrel{df}{=} C + \Box[5,5]C$. As with tags, one can add new syntax to the logic and allow for keeping the macros intact. However, more caution needs to be put in to avoid unsound inferences from being made.

## 5 Conclusions

One of the primary challenges we have found when supporting reasoning through the use of a CNL is the domain to which the language is applied — controlling the structure of the sub-language ensures that a mapping to and from the operators of the formal underlying representation is possible. On the other hand, if the domain of the basic terms is not carefully controlled, the reasoning one can perform is strictly limited. In this paper we have investigated the use of a controlled domain of application for BanaL, a CNL to specify contract clauses as input to the game BanaNomic, in which the basic actions and state queries are limited. The CNL has been used as a front end input to a web-based version of BanaNomic, with players taking turns to change the constitution and take actions — as regulated by the current contract.

The tractability of legal documents from a human perspective much depends on the way in which the legal clauses are organised, and the use of definitions to avoid lengthy identical descriptions in different contexts. Collapsing even a fraction of a legal agreement into one long paragraph of text, transforms a document from one which can be followed by a human expert to one which is unintelligible. However, since this structure is void of legal meaning, formal reasoning typically discards this information. The main challenge is to maintain this structure in such a manner as to enable feedback and explanation of the outcome of formal reasoning back in a natural language setting. We are currently looking into how these notions can be incorporated into BanaNomic so as to enrich the game, by enabling more complex rule sets which are still tractable to human players.

The conclusion here is that assistance comes in two flavours, so to speak. Assistance with the strictly logical properties requires an underlying semantic model. However, when assistance takes the form of *explanation* it is invaluable to be able to refer to identifiable parts of the document structure. Here layout is crucial.

## References


[BKKR01] John Bateman, Jörg Kleinz, Thomas Kamps, and Klaus Reichenberger. Towards Constructive Text, Diagram, and Layout Generation for Information Presentation. *Computational Linguistics*, 27:409–449, 2001.

[BSBS09] Jie Bao, Paul R. Smart, Dave Braines, and Nigel R. Shadbolt. A Controlled Natural Language Interface for Semantic Media Wiki Using the Rabbit Language. In *CNL*, 2009.

[CD69] D. Crystal and D. Davy. *Investigating English style*. Studies in the History and Theory of Linguistics. Indiana University Press, 1969.

[CSS03] Koen Claessen, Mary Sheeran, and Satnam Singh. Functional Hardware Description in Lava. In *The Fun of Programming*, Cornerstones of Computing, pages 151–176. Palgrave, 2003.

[Dav67] Donald Davidson. The Logical Form of Action Sentences. In Nicholas Rescher, editor, *The Logic of Decision and Action*. University of Pittsburgh Press, 1967.

[DS97] Aspassia Daskalopulu and Marek Sergot. The Representation of Legal Contracts. *AI and Society*, 11:6–17, 1997.

[Hay71] Patrick J. Hayes. A Logic of Actions. In B. Meltzer and D. Michie, editors, *Machine Intelligence 6*, pages 495–520. Edinburgh University Press, 1971.

[HR01] Klaus Havelund and Grigore Rosu. Monitoring Programs Using Rewriting. In *Proceedings of the 16th IEEE international conference on Automated software engineering*, ASE '01, pages 135–, Washington, DC, USA, 2001. IEEE Computer Society.

[Hud96] Paul Hudak. Building domain-specific embedded languages. *ACM Computing Surveys*, 28:196, 1996.

[Mal26] Ernst Mally. *Grundgesetze des Sollens. Elemente fer Logik des Willens*. Graz: Leuschner & Lubensky, 1926.

[Man88] Mann, William C. and Thompson, Sandra A. Rhetorical Structure Theory: Toward a functional theory of text organization. *Text*, 8(3):243–281, 1988.

[McN06] Paul McNamara. Deontic Logic. In Dov M. Gabbay and John Woods, editors, *Handbook of the History of Logic*, volume 7, pages 197–289. North-Holland Publishing, 2006.

[Nun90] Geoffrey Nunberg. *The Linguistics of Punctuation (Center for the Study of Language and Information - Lecture Notes)*. Center for the Study of Language and Inf, August 1990.

[PB05] Mark E. Phair and Adam Bliss. PerlNomic: Rule Making and Enforcement in Digital Shared Spaces. In *Online Deliberation 2005 / DIAC-2005*, Stanford, CA, USA, 2005.

[PR09] G. Pace and M. Rosner. A Controlled Natural Language for the Specification of Contracts. In Norbert E. Fuchs, editor, *Proceedings of the Workshop on Controlled Natural Languages, CNL 2009, Marettimo Island, Sicily*. Springer, isbn: 978-3-642-14417-2 edition, June 2009.

[PR10] Gordon J. Pace and Michael Rosner. A controlled language for the specification of contracts. In *Proceedings of the 2009 conference on Controlled natural language*, CNL'09, pages 226–245, Berlin, Heidelberg, 2010. Springer-Verlag.

[PSBA03] Richard Power, Donia Scott, and Nadjet Bouayad-Agha. Document Structure. *Computational Linguistics*, 29:211–260, 2003.

[Ran04] Aarne Ranta. Grammatical Framework: A Type-Theoretical Grammar Formalism. *Journal of Functional Programming*, 14(02):145–189, 2004.

[Sch08] R. Schwitter. A Controlled Natural Language for the Semantic Web. *Journal of Intelligent Systems*, 17(1-3):125–141, 2008.

[Sub90] Peter Suber. Nomic: A Game of Self-Amendment. In *The Paradox of Self-Amendment*. Peter Lang Publishing, 1990.

[Vel10] G. Vella. Automatic Summarisation of Legal Documents. Master's thesis, Univerisity of Malta, Dept Intelligent Computer Systems, University of Malta, Msida MSD2080, Malta, 2010.

[Wri51] Georg Henrik Von Wright. Deontic Logic. *Mind*, 60:1–15, 1951.