Playing Nomic using a Controlled Natural Language

John J. Camilleri, Gordon J. Pace and Michael Rosner
Department of Intelligent Computer Systems
University of Malta
Msida, Malta
jcam0003, gordon.pace, mike.rosner@um.edu.mt

ABSTRACT

Controlled natural languages have been used in a variety of domains, to enable information extraction and formal reasoning. One major challenge is that although the syntax is restricted to enable processing, without a similar restricted domain of application, it is typically difficult to extract useful results. In this paper we look at the development of a controlled natural language to reason about contractual clauses. The language is used to enable human players to play a variant of Nomic — a game of changing contracts, whose very nature makes it extremely challenging to mechanise. We present the controlled natural language with its implementation in the Grammatical Framework, and an underlying deontic logic used to reason about the contracts proposed by the players.

Keywords

Controlled natural language, deontic contract logic, Nomic, Grammatical Framework

1. INTRODUCTION

The use of controlled natural languages (CNLs) to enable processing of and formal reasoning about statements in a particular domain is a rather well-established approach. By constraining the language, together with the structural complexity of the grammar, one obtains a language to make statements about the underlying domain, without moving too far off from a natural language description. 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.

WICT 2010 Malta

One area in which CNLs have been applied is that of contracts — agreements between two or more parties regulating their behaviour. Typically contracts state obligations, permissions and prohibitions of actions or states. Contracts have been used in a variety of settings, ranging from game rules and user-level agreements to national legislation and international directives. For instance, a particular game may include clauses in its rules, such as 'Players take turns in a counter-clockwise direction', and 'Upon starting their turn, the player throws two dice'.

In practice, using a CNL for contracts requires not only semantics of the language operators, but also the underlying implicit concepts (for instance, a player may give the dice to the next player at the end of his or her turn, but may not give his or her counter to another player). 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.

2. NOMIC AND BANANOMIC

Nomic is a game of self-amendment [4] — 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 [2], 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.

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, 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. A DEONTIC CONTRACT LANGUAGE FOR BANANOMIC

The contract grammar devised for BanaNomic is based on the deontic logic we have explored in [1]. 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: $throwBanana(Michael, Gordon)$. 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.: $\mathbb{F}(throwBanana(*, John))$ would be the statement saying that everyone is forbidden from throwing a banana at John. In such a logic we would ideally 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 $C_1 \lll q \ggg C_2$ 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)

(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

(v) the conjunction of two clauses $C_1 \wedge C_2$

The syntax of the logic is as follows:

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

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.

4. 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) 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 more intelligent choice of words and phrases.

GF is a specialised functional language for defining grammars, having separate abstract/concrete syntax rules, a strong type system, and inherent support for multilingual grammars. 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 [3].

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

```
cat
  DeoExp ; Time ; Clause ;
fun
  C_Deontic : DeoExp -> Clause ;
  C_Always : Time -> Time -> Clause -> Clause ;
  C_Cond : DeoExp -> 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 better approach in order to produce phrases which

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

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 1) and the "fridge magnets" (figure 2) — 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.

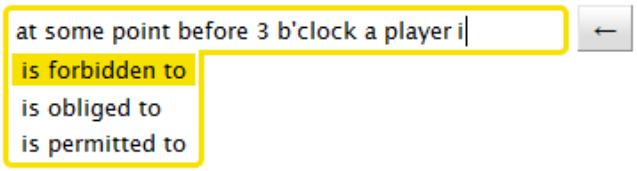

Figure 1: 'Suggest panel' guided input method

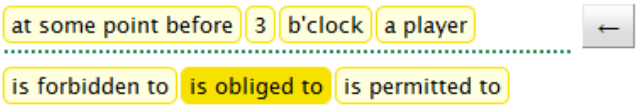

Figure 2: 'Fridge magnets' guided input method

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. We are currently looking into ways to extend the syntactic and semantic domain of the logic whilst keeping a hold on the natural language aspect.

6. REFERENCES

[1] G. J. Pace and M. Rosner. *A Controlled Language for the Specification of Contracts*, volume 5972 of *LNAI*. 2010.

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

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

[4] P. Suber. *Nomic: A Game of Self-Amendment*. Peter Lang Publishing, 1990.