A Controlled Language for the Specification of Contracts

Gordon J. Pace[⋆] and Michael Rosner[⋆⋆]

University of Malta, Msida MSD2080, Malta
{gordon.pace,mike.rosner}@um.edu.mt

Abstract. Controlled natural languages have been used to enable the direct translation from natural language specifications into a formal description. In this paper we make a case for such an approach to write contracts, and translating into a temporal deontic logic. Combining both temporal behaviour and deontic behaviour is challenging both from a natural language and a formal logic perspective. We present both a logic and a controlled natural language and outline how the two can be linked.

1 Introduction

Legal contracts are useful artifacts which have evolved to regulate behaviours in agreed ways. An important feature of a contract is that one party makes an offer for an arrangement that another accepts. Once this exchange takes place, constraints are enforced over the parties' future actions. If these constraints are broken, a *breach of contract* is recognised and remedies may be provided. Contracts can operate with varying degrees of formality and complexity. Much of the time, contracts are made orally or are implied by the situation at hand. However, beyond a certain level of complexity, written contracts are the norm. These are typically expressed in natural language without any particular restrictions on the form of language. The use of unrestricted natural language for the formulation of contracts is something of a double-edged sword.

On the one hand, it allows quite complex specifications of contractual properties to be formulated concisely. For example, the clause[1] *"We shall not be held liable for any damages resulting from the disruption or malfunction of the DNS service."* seems pretty clear, relieving one party from the specific obligation to pay damages resulting from a certain class of events. Most of the complexity follows from the meanings of words like "malfunction", and boils down to a three-place relation of the form `"X shall not be held liable for Y resulting from Z"` with appropriate constraints on the variables.

On the other hand, unrestricted natural language brings with it a host of well-known problems relating to ambiguity, syntactic complexity, context sensitivity, etc.

⋆ Dept. Computer Science

⋆⋆ Dept. Intelligent Computer Systems

[1] Extracted from the Terms and Conditions of NIC(Malta), a local domain name registration service.

For example, a certain contract specifies that *"the payment adjustment per tonne will apply to the quantity of asphalt cement in the hot mix accepted into the work during the month for which it is established"*. Amongst the problems are (i) identification of the referent of the word "it" (which could be almost any of the preceding noun phrases), (ii) deciding whether "during . . . established" modifies the application of the payment adjustment or the quantity of asphalt cement in the hot mix.

Common sense will sometimes tell us which of these interpretations is the correct one, but this is not always the case. Certainly important legal cases have been won or lost according to the way in which a scope ambiguity is resolved. Natural language contracts that specify obligations between contracting parties are so complex that we are generally forced to rely upon costly legal specialists for their formulation and for analysis of their implications.

These considerations motivate the idea of a *controlled language* for certain classes of contract. Controlled languages (cf. Pulman [1]) are "designer languages" in the sense of being artificially restricted subsets of natural language that have been designed with a mission in mind. The mission typically involves the facilitation of learning, translation, analysis, or some combination of these as in explanation. The specific restrictions are mission-dependent and might concern vocabulary, syntax and semantics.

In our case we are seeking a controlled language which is rich enough to express the key concepts, simple enough for ordinary people to use, and yet precise enough to have a semantics which is amenable to automated reasoning methods.

Automated reasoning could be the basis for a tantalising range of support tools providing help in the areas of contract formulation, explanation, and applicability with respect to a particular set of circumstances or scenario e.g. deducing whether a given action is obliged, prohibited, or merely licenced.

For the purpose of developing this idea, a great deal will turn on what we mean by a class of contracts and which class to choose. In this article, we focus on contracts which occur within fairly narrow computational settings, where the underlying reality is well enough understood and sufficiently precise to enable a range of contractual issues to be explored.

Contracts appear in computational settings in various guises — from simple pre- and post-condition pairs in programming giving guarantees on outcomes based on assumptions on the initial conditions, to quality of service agreements requesting, for instance, a maximum guaranteed packet-dropping probability. Service-level agreements, contracts used for conformance checking, and system specifications are expressed in a variety of ways, but may still contain interesting deontic nuances. Simple examples of a system specification, combining deontic and temporal notions are given by the following sentences:

- Upon accepting a job, the system guarantees that the results will be available within an hour unless cancelled in the meantime.
- Only the owner of a job has permission to cancel the job.
- The system is forbidden from producing a result if it has been cancelled by the owner.

One major challenge in the use of controlled natural languages (CNLs) is the choice of both the natural sublanguage itself and the logic used to capture its meaning. This choice is crucial to enable us to go back and forth between the two levels when parsing, reasoning about or manipulating objects in the domain, and generating of natural language explaining the results.

In this paper, we present the use of CNL techniques for the specification of contracts. We identify an appropriate logic and controlled language to enable the automated analysis of CNL contracts in a formal and precise manner. One important choice which we discuss at length is in the choice of a complementary pair of a controlled language and logic, so as to enable reasoning and manipulation on either side to be transferable to the other. However, the choice of a compatible logic and language pair still leaves much choice as to the structural abstraction possible in the representations.

One contribution of the paper is a discussion on further choices to make to approach the ideal level of abstraction. Since manipulation and analysis at the logical level may result in changing the structure of the original body of text, referring back to the original content can be challenging. Even keeping cross references between the two entities may not be simple — and we address the issue by trying to keep a one-to-one relationship between the two domains at the syntactic level. One technique to enable easier automated manipulation of the contracts keeping the syntactic structure intact is one developed by the programming language community — using embedded languages. The idea behind the approach is to enable expressions written in a domain specific language to be written as though they were part of the code written in the host language in which one is programming. This allows terms of the logic not only to appear in the host language — but also to be manipulated as data objects. In this way as much as possible of the syntactic structure of the original information is retained in the reasoning at the logical level.

The paper is structured as follows. In Section 2 we discuss the issues arising in choosing the right controlled natural language and logic pair. We then present the deontic logic in Section 3, and show how it can be embedded in Haskell and what benefits are gained in Section 4. In Section 5, we present the controlled natural language and its transformation into the logic. We finally conclude in Section 7.

2 Controlled Languages and Translation

The end-user system that we are aiming for can be regarded as an expert-system specialised in the domain of computer-oriented contracts. The core expertise of the system resides in its ability to reason logically about the properties of contracts themselves (e.g. concerning a contract's internal consistency) and about the status of actions or events falling within the scope of the contract - for example, whether a certain action is prohibited or obligatory. What makes this system different is that communication with it takes place in natural language. We have already hinted at different kinds of communication task that such a system might undertake, including authoring, explanation, question answering, what-if style querying etc. Underlying these tasks is an object language for the specification of contracts. This underlying language is the CNL that concerns us here.

The idea, then, is that the system can effect two-way communication in CNL. In order to reason about the properties of a contract expressed in CNL, they system must translate it into a representational form which is suitable for performing inferences. To communicate the results of those inferences back to the user, there must be the means to translate relevant elements of the (possibly modified) representational form back into CNL.

Much of the discussion that follows concerns an appropriate choice of logic representation and the level of abstraction that it encodes. The discussion recalls issues that arise with respect to the choice of intermediate levels of representation needed for transfer-based Machine Translation. This is depicted in Figure 1 (a) below which depicts so-called Vauquois triangle (Vauquois [2]).

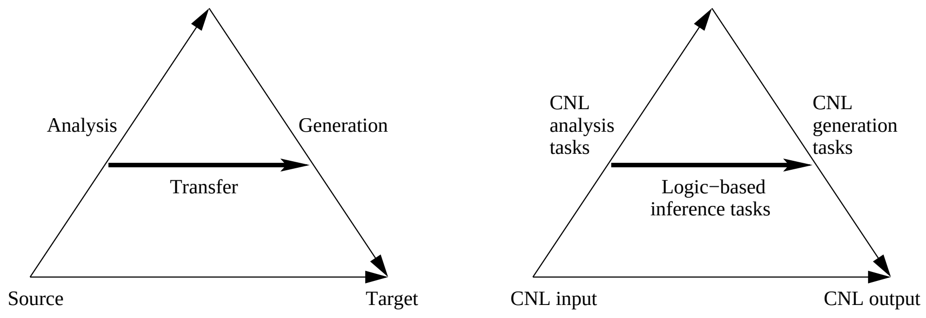


Fig. 1. (a) Vauquois Triangle for machine translation; and (b) Controlled natural language triangle

The bottom left and right vertices of the triangle represent source and target language sentences respectively. In a classic transfer-based system, the translation process comprises three phases: (i) analysis (ii) transfer and (iii) generation. The actual translation is carried out by the transfer phase which applies *transfer rules* to abstract *source sentence representations* to yield *target-language representations*. The great advantage of defining translation over representations rather than sentences or sentence-fragments is that transfer rules are able to translate *classes* of similar linguistic forms and hence to capture linguistic generalisations. The downside is that representations are not sentences, and in order to ground the system to concrete text, appropriate mappings to source and target text have to be defined. These are supplied by the analysis and generation phases, which are respectively responsible for the mapping between source text and representation, and between target representation and target text.

There is a close relationship between transfer-based machine translation and the CNL inference system we are trying to construct. This is illustrated in figure 1(b), where the "source text" is in CNL and expresses at least (i) a contract and also (ii) some other information concerning the particular task at hand, for example concerning the contract's consistency, request for explanation, or applicability of the contract to a particular situation. In this article we will limit the discussion to (i). The right hand "target" vertex of the triangle represents CNL output, i.e. a piece of text that fulfils the task at hand. In between, corresponding to the place where transfer is carried out in the machine translation system, is where the relationship between the task, and the answer, is computed using logical inference. So the representation, whatever it is, encodes expressions in the logic, and logical inference maps between sets of logic expressions encoded as representations. And just as before we had analysis and generation to connect these representations to the concrete world of sentences, so we now require analagous, if rather different processes, to analyse CNL into logic and vice versa.

The geometrical dimensions of the Vauquois triangle provide some insight into the tradeoffs at play between deep and shallow levels of linguistic representation, as shown in figure 2. A shallow level of representation, depicted by the dotted line, is close to the surface text. Accordingly, analysis and generation are relatively simple, since they only need traverse a short (vertical distance). At the same time, the transfer operation is large - since it has to cope with a low level of abstraction, the similarity between similar cases is lost.

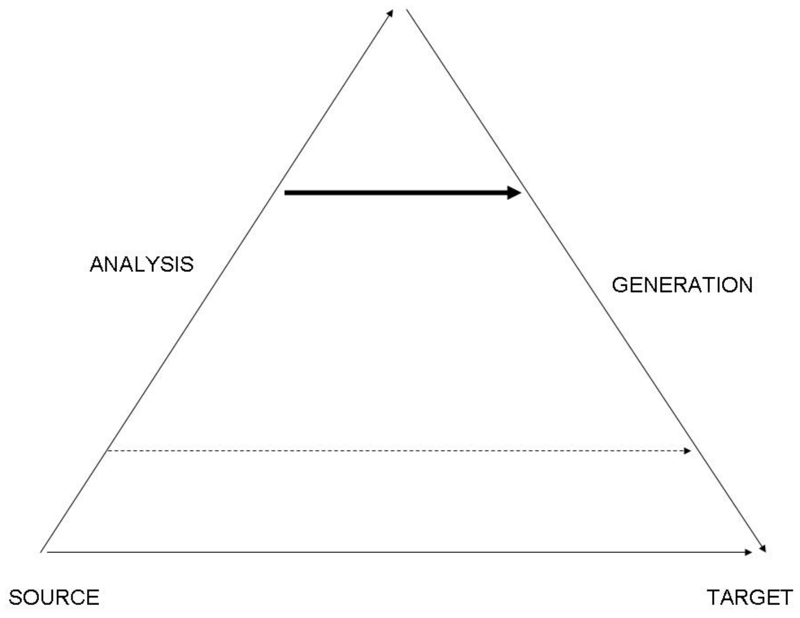
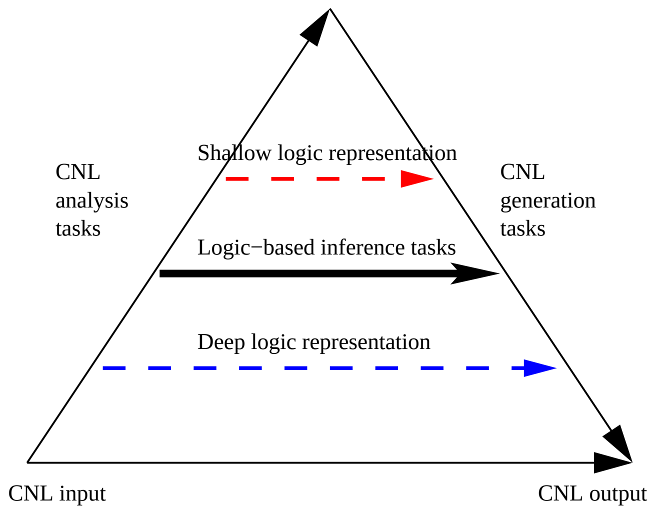


Fig. 2. Deep or shallow linguistic representations

Conversely deep level transfer, represented by the upper, thick solid line, is short, meaning that transfer is relatively simple. This is because it is carried out at a high level of generalisation. But the journey to reach that level is long: analysis and generation are both complex.

So from a machine translation perspective, the choice of representation level is by no means straightforward. In arriving at an appropriate choice, a number of different considerations must be borne in mind including in particular the language pairs under consideration, the extent to which the complexity of monolingual processes (analysis and generation) is to be traded off against the bilingual one (transfer).

From the perspective of inference, the issue is the choice of logic to use. Here the primary requirement is that it has to be sufficiently expressive to encompass the domain of the controlled language. However, this leaves a wide spectrum of possibilities of the abstraction level of the logic as shown in figure 3.



Shallow logic representation
CNL analysis tasks
CNL generation tasks
Logic−based inference tasks
Deep logic representation
CNL input
CNL output

Fig. 3. Deep or shallow logical representations

At one extreme, one can choose a logic with a minimal set of orthogonal operators with no overlap in semantic expressiveness; or, at the other extreme, opt for a logic having a rich set of operators with overlapping semantics, but which may allow more concise descriptions in different scenarios. This is usually considered to be a language design choice and challenge, which is faced whenever a new language is designed — e.g. should we have both a `repeat...until` and a `while...do` construct in the same language despite the fact that they are expressible in terms of each other? When designing a programming language one regularly opts for redundancy in the operators, if it aids the description of different algorithms. However, when building a logic, the choice is usually that of limiting the number of operators to simplify giving a semantics to the logic, and to ensure soundness. Other redundant, or higher-level constructs, are simply seen as syntactic-sugar, reducible to the the underlying, syntactically-miserly logic.

Going for a low-level logic with few operators, but complete with respect to the domain of application, is the choice typically made by logicians when designing a new logic, and has various advantages, the biggest of which is that the axiomatisation is limited to just a few, simple operators. Furthermore, analysis from language to logic, and transformations within the logic are typically straightforward to define and prove correct. However, this comes at the price of a loss of information implicitly available in the structure had one to use other operators.

Consider propositional logic, in which all operators can be expressed in terms of the nand ($\uparrow$) operator. However, expression the implication $p \rightarrow q$ using the nand operator results in $p \uparrow (q \uparrow q)$ giving no clue of the original intended term. If the controlled natural language caters for implication, there is the need for a closely corresponding operator in the logic if natural language generation is to be performed in a reasonable manner.

The alternative approach, in which we add various new operators, to have a closer correspondence between the controlled natural language and the logic adds work for the logician, who must ensure that the operators are sound with respect to each other, that their definitions introduce no contradictions and satisfy algebraic laws which the logic is expected to obey.

The solution we adopt in this paper is to go for a logic with few operators (which do however correspond, as closely as possible, to the CNL constructs). As usual, for higher abstraction layers, other operators not included in the basic logic are introduced as syntactic sugar. However, in reasoning about the logic, manipulation of expressions and generation of formulae, we try to retain the structure introduced through the abstraction layers in the logic expressions. For instance, in the propositional logic given earlier, although we may choose to include only nand in the basic logic, and introduce implication as syntactic sugar. However, when storing expressions involving implications, we try to maintain this information throughout the reasoning process, allowing generation of natural language sentences which use implications and not just nand. The reduction to the basic operators is only done when reasoning process requires it. This approach enables the use of a simple and elegant logic, but still enables reasonable generation of natural language. This results in a two-level approach with high-level operators which are only opened up when required.

3 Underlying Logic Representation Language

It is crucial to identify a sufficiently expressive logic to enable the description of and reasoning about contracts. In computer science, contracts have frequently been expressed as properties which should be satisfied by the system bound by the contract. This view enables a straightforward characterisation of contracts, but does not enable (i) reasoning *about* the contract — asking questions such as 'What are the currently undischarged obligations in the contract?'; or (ii) reasoning about exceptional cases in a contract — such as 'whenever clause (a) is violated, the user is prohibited from obtaining the service'. The introduction of explicit prohibition, obligation and permission clauses into the contract is thus essential to enable reasoning about contracts.

An important choice is between adopting an action-based or a state-based deontic logic. In the former, the deontic operators function over actions — one is obliged to deliver a service, as opposed to the latter, in which the operators function over the state, one is prohibited from reaching a state in which the balance is negative. The choice depends mainly on the type of contracts we choose to translate into logic. We choose to adopt the action-based approach.

Contracts constrain the behaviour between different agents, and it is thus important to include information about these participating agents. We choose to tag deontic expressions with an actor to identify which entity is being constrained by the clause.

Contracts are obviously dependent on the notion of choices — eg 'if the client connects, then the client is obliged to pay'. Regularly, contracts go further, and make choices based on the enacted clauses in the contract itself eg 'as long as the client is obliged to pay, the client is forbidden from closing the account'. Clearly, this introduces a notion of self reference, which can be dangerous — eg 'if the client is forbidden from closing the account, then the client is permitted to close the account', which would require the analysis of self-referential clauses similar to the ones used in the analysis of hardware with combinational cycles.

The challenge to formalise deontic logic, to reason about normative concepts such as obligations and permissions, has been a hot topic of research for several decades. The main challenge is that it is very easy to describe paradoxical situations using deontic concepts [3]. Paradoxes typically follow from laws which one expects to hold of the deontic operators. For instance, one may consider it natural that obligations should be monotonic. After all, if you are obliged to do something, then you are also obliged to do something weaker (and more): if $p \rightarrow q$, then $O(p) \rightarrow O(q)$ — where $O(p)$ means that there is the obligation to perform p or reach a state satisfying it) However, a naïve introduction of this law results in Ross's Paradox: from the statements 'You are obliged to post the letter', one can infer that 'You are obliged to post the letter or burn it', which is seemingly satisfiable by burning the letter. Without resorting to mutually exclusive actions and states, this interpretation is clearly counterintuitive. Similarly, using monotonicity and allowing reasoning about the state of affairs both inside and outside the deontic operators, leads to the Good Samaritan Paradox: from the statement that 'It ought to be the case that Jones helps Smith who has been robbed' we would be able to conclude that 'It ought to be the case that Smith is robbed,' which is also not expected.

Introducing the concept of time brings further paradoxes [4]. Consider the law of a country which says that: 'You are obliged to hand in Form A on Monday and Form B on Tuesday, unless officials stop you from doing so.' On Monday, John spent a day on the beach, thus not handing in Form A. On Tuesday at 00:00 he was arrested, and brought to justice on Wednesday. The police argue: 'To satisfy his obligation the defendant had to hand in Form A on Monday, which he did not. Hence he should be found guilty.' But John's lawyer argues back: 'But to satisfy the obligation the defendant had to hand in Form B on Tuesday, which he was stopped from doing by officials. He is hence innocent.' Both interpretations seem intuitively acceptable — but are clearly incompatible since they use different interpretations of the moment of violation of *a sequence* of obligations.

Various axiomatizations have been proposed, in an attempt to deal with the paradoxes. However, one of the more effective approaches has been that of restricting the syntax [5]. Since our aim is to reason about contracts derived from natural language texts, and which could thus include paradoxes or contradictions, we opt for a more general logic, which could then be restricted, syntactically or semantically, to weed out potential problems.

The logic we adopt will have operators closely matching those found to be used in the CNL, but expressing some of them using syntactic sugar so to avoid redundancy. The deontic logic we build has three basic underlying deontic concepts: prohibition, permission and obligation. All three of these will always appear tagged with an agent specifying to whom the deontic operator applies. Since we would want our contracts to be phrased in terms of actions — for example, which ones are permitted at a particular point in time, we adopt an action-based deontic logic. The underlying actions can occur concurrently, and we will use the notation $\overline{act}$ for any action different from action act. Action expressions, are defined in terms of regular expression operators, with conjunction. In addition to basic actions, one can also check whether an obligation, permission or prohibition is currently enacted: $O?(A.\alpha)$ is satisfied if agent A is obliged to perform α at this moment in time, similarly $P?(A.\alpha)$ and $F?(A.\alpha)$ are used to check for permission and forbidden actions.

$$\begin{aligned}
\textit{action-expression} ::= \;& \textit{basic-action} \mid \overline{\textit{basic-action}} \\
\mid\;& O?(\textit{agent.action-expression}) \\
\mid\;& P?(\textit{agent.action-expression}) \\
\mid\;& F?(\textit{agent.action-expression}) \\
\mid\;& \textit{action-expression} \cdot \textit{action-expression} \\
\mid\;& \textit{action-expression} + \textit{action-expression} \\
\mid\;& \textit{action-expression} \;\&\; \textit{action-expression} \\
\mid\;& \textit{action-expression}^{*}
\end{aligned}$$

We will use lower case letters (a, b, c) for basic actions, Greek letters for action expressions (α, β, γ) and upper case letters (A, B, C) for agents.

The deontic logic includes obligation, permission and prohibition ($O(A:\alpha)$, $P(A:\alpha)$ and $F(A:\alpha)$), non-deterministic choice (+), conjunction (&), conditional ($c_1 \triangleleft \alpha \triangleright c_2$), generalised sequential composition ($c_2 \blacktriangleleft c \blacktriangleright c_1$, which starts with c, and then follows it up with c_1 or c_2 depending on whether it was satisfied or violated) and timing information ($c_{[b,e]}$):

$$\begin{aligned}
\textit{contract} ::= \;& \top_{\textit{time}} \mid \bot_{\textit{time}} \mid O(\textit{agent} : \textit{action}) \mid P(\textit{agent} : \textit{action}) \mid F(\textit{agent} : \textit{action}) \\
\mid\;& \textit{contract} + \textit{contract} \mid \textit{contract} \;\&\; \textit{contract} \mid \textit{contract} \triangleleft \textit{action} \triangleright \textit{contract} \\
\mid\;& \textit{contract} \blacktriangleleft \textit{contract} \blacktriangleright \textit{contract} \mid \textit{contract}_{[\textit{time},\textit{time}]}
\end{aligned}$$

Note that choice may appear both inside and outside a deontic operator e.g. $O(A : a+b)$ and $O(A : a) + O(A : b)$. These are semantically different: the contract $O(A : a+b)$, allows agent A to choose whether or not to perform a or b — either of the two would satisfy the contract, while the contract $O(A : a) + O(A : b)$ would be non-deterministic.

Using these operators and fix-point definitions, other operators can be defined: (i) one branch conditional: $e \rightarrow c \equiv c \triangleleft e \triangleright \top_0$; (ii) sequential composition: $c_1; c_2 \equiv c_2 \blacktriangleleft c_1 \blacktriangleright \bot_0$; (iii) the always operator: $\Box(c) \equiv c \;\&\; \top_1; \Box(c)$; and (iv) the sometimes operator: $\Diamond(c) \equiv c + \top_1; \Diamond(c)$. Furthermore, the complement of an action or an agent can be expressed in the logic using a bar over the object. The examples given earlier can be written in the following manner:

- Upon accepting a job, the system guarantees that the results will be available within an hour unless cancelled in the meantime:
  $\Box(accept_j \rightarrow O(system : (result_j + cancel_j))_{[0,1hr]})$
- Only the owner of a job has permission to cancel the job:
  $\Box(P(owner_j : cancel_j) \;\&\; F(\overline{owner}_j : cancel_j))$
- The system is forbidden from producing a result if it has been cancelled by the owner: $\Box(cancel_j \rightarrow F(system : (\Diamond(result_j))))$

The logic proposed shares much in flavour with CL [5] and other action-based deontic logics. One can construct observer formulae (one for each actor), using which one can model-check contracts [6], and perform contract analysis. The temporal side of the logic is based on timed regular expressions [7]. Although, as in timed regular expression, a continuous time domain may be used, at the moment we restrict the time to a discrete domain for analysis techniques. Through the use of a trace semantics of the logic, standard model-checking techniques can be used to check for validity.

The deontic logic has been given an operational trace semantics, a portion of the untimed version of which can be seen in Figure 4. Enriched semantics with deontic information (what actions are permitted, prohibited and obliged at every point in time) and with information regarding the actor responsible for an action, and which actor violated or satisfied which contract can be similarly given. The extension of the semantics to deal with time uses the approach adopted in timed regular expressions [7], and adds a time-stamp to each action.

4 Embedding the Contract Logic in Haskell

When building tools to use and manipulate a logic or language, the approach is typically that of building and using a domain-specific library to deal with the concepts and ways of manipulating them. One alternative approach which is becoming more widespread is that of using embedded languages [8, 9] — where one builds a domain-specific *language* within the host language, in which there is a direct correspondence to the syntax of the logic or language one is reasoning about. This approach enables the use of abstraction and modularization techniques from the host language in the embedded language. By building syntactic constructs and a type system of the domain-specific language within the host language, domain-specific programs can be expressed as objects in the host language. Apart from ways of representing syntactically the language in question, one also typically provides semantic interpretations of the domain-specific programs — enabling execution, visualization, analysis, etc. This approach has been used to model various domains, ranging from financial contracts [10] to hardware description [11].

The main difference between a domain-specific library in the general-purpose language and a domain-specific embedded language is that in the embedded language approach, careful design of domain-specific combinators creates the illusion that the domain-specific code fragments are ‘programs’ appearing as parts of the programs in

Obligation:

$$\frac{}{O(p:a) \xrightarrow{a} \top} \qquad \frac{}{O(p:a) \xrightarrow{b} \bot}$$

Prohibition:

$$\frac{}{F(p:a) \xrightarrow{b} \top} \qquad \frac{}{F(p:a) \xrightarrow{a} \bot}$$

Reparation:

$$\frac{}{c_1 \blacktriangleleft \top \blacktriangleright c_2 \longrightarrow c_1} \qquad \frac{}{c_1 \blacktriangleleft \bot \blacktriangleright c_2 \longrightarrow c_2}$$

$$\frac{c \longrightarrow c'}{c_1 \blacktriangleleft c \blacktriangleright c_2 \longrightarrow c_1 \blacktriangleleft c' \blacktriangleright c_2} \qquad \frac{c \xrightarrow{a} c'}{c_1 \blacktriangleleft \top \blacktriangleright c_2 \xrightarrow{a} c_1 \blacktriangleleft c' \blacktriangleright c_2}$$

Conjunction:

$$\frac{}{\bot \,\&\, c \longrightarrow \bot} \qquad \frac{}{c \,\&\, \bot \longrightarrow \bot}$$

$$\frac{}{\top \,\&\, c \longrightarrow c} \qquad \frac{}{c \,\&\, \top \longrightarrow c}$$

$$\frac{c_1 \longrightarrow c_1'}{c_1 \,\&\, c_2 \longrightarrow c_1' \,\&\, c_2} \qquad \frac{c_2 \longrightarrow c_2'}{c_1 \,\&\, c_2 \longrightarrow c_1 \,\&\, c_2'}$$

$$\frac{\begin{array}{c} c_1 \xrightarrow{a} c_1' \\ c_2 \xrightarrow{a} c_2' \end{array}}{c_1 \,\&\, c_2 \xrightarrow{a} c_1' \,\&\, c_2'}$$

Choice:

$$\frac{}{c_1 + c_2 \longrightarrow c_1} \qquad \frac{}{c_1 + c_2 \longrightarrow c_2}$$

Fig. 4. An excerpt of the formal semantics of the contract logic

the host language. By providing appropriate functions, the programmer may then generate, analyse and manipulate these ‘programs’ (written in the domain-specific language) as though they were part of the host language itself and thus, effectively make the host act as a meta-language for the embedded language. Clearly, the more flexible and high-level the host language and its syntax are, the more difficult it becomes to distinguish where the domain-specific program ends and where the rest of the code starts.

The structuring of embedded languages is ideal for the approach we require, in which we have access to the structure of the logic statements, manipulate them, opening up syntactic sugar only when and if required, and we have embedded part of the logic as an embedded-language in Haskell [12] — which has, through various case studies, been shown to be an excellent host language to provide the right modularity and abstraction to develop an embedded language.

Describing contracts: Although internally, the embedded-language uses an abstract datatype to keep information about the structure of the contract, the end user is provided with a set of functions and operators to describe contracts. For instance, the infix operators `<|` and `|>` correspond to the conditional operators $\triangleleft$ and $\triangleright$, while the operators `<<` and `>>` correspond to the reparation operators ◀ and ▶. Deontic operators are similarly defined using the functions `permission`, `obligation` and `forbidden`, all of which take two operands — the agent and the action. Agents and actions are defined using the constructors `agent` and `action` which take a string.

As a simple example of writing a contract using the embedded language directly, we show below a simple contract which branches on whether the agent picks up an item in a shop. If the item is not picked up, then the contract terminates successfully, but if it is, an obligation to pay is enacted, which can either be satisfied, thus permitting the agent to leave the shop, or not, in which case an obligation to return the object on the shelf is enacted:

```
stroll :: Agent -> Contract
stroll a = shop a <| pickupItem |> success

shop :: Agent -> Contract
shop a =
  permission(a,leaveShop)
    << obligation(a,pay) >>
      obligation(a,returnObject)
```

Note that in our setting, most of this code would be generated directly from the CNL.

Families of contracts: One advantage of using the embedded language approach is that regular contracts can be described using higher-order contracts. This approach of writing parametrised contracts is particularly useful to enable syntactic sugar to be handled in a direct manner. Consider an operator to retry satisfying a contract a number of times. The combinator can be expressed in the embedded language in the following manner:

```
retry :: Integer -> Contract -> Contract
retry 1 c = c
retry n c = success << c >> retry (n-1) c
```

Contract transformations: The embedded language also enables the writing of transform contracts syntactically. Using functions to check the type of operator and its operands, users can write their own transformations of contracts. The following code extract shows how one can change a contract selectively, allowing a softer form of obligations for a particular action:

```
retryObl :: Action -> Contract -> Contract
retryObl a c
  | isConditional c =
      retryObl a (leftBranch c)
        <| condition c |>
          retryObl a (rightBranch c)
  | isObligation c =
       if action c==a then (success << c >> c) else c
  | ...
```

Contract interpretations: Access to the structure of contracts allows us to generate different interpretations, including monitoring of a trace according to a contract, output to external analysis tools and natural language generation. As to natural language generation, as yet we only have an *ad hoc* translator, but we eventually plan to reuse the grammars used for generation.

Syntactic sugar: In order to retain syntactic features of the controlled natural language operators which are expressed using syntactic sugar, whenever possible, a *deep embedding* is used — with higher level reasoning done using *algebraic manipulation* using the abstract operators (e.g. look for conflicts, and if any are discovered, explain them in terms of natural language) while using lower level reasoning at the basic logic level obtained by opening up the syntactic sugar (e.g. just reporting whether a contract is satisfiable or not). Essentially the approach is to keep the abstract operators for as long as possible and take the one-way road down to the basic logic only when we cannot do otherwise.

5 Remarks on a Possible Controlled Language

5.1 Some key examples

In this section we take the examples of section 1 and propose some simplifications which reduce both syntactic complexity and, more importantly, the potential for ambiguity.[2]

1. **original:** Upon accepting a job, the system guarantees that the results will be available within an hour unless cancelled in the meantime.

[2] At this exploratory stage such simplifications are guided primarily by our own intuitions rather than through conformity to any existing CNL that includes deontic operators (e.g. SBVR structured English).

controlled: `if SYSTEM accepts Job, then during one hour it is obligatory that SYSTEM make available results of Job unless SOMEONE cancels Job.`

logic: $\Box(accept_j \rightarrow O(system : (result_j + cancel_j))_{[0,1hr]})$

comment: There are three events: accepting a job, results being available, and a cancellation. There is also a contractual obligation concerning the second event, but this is discharged if the cancellation takes place. This sentence displays the classical problem of *attachment ambiguity*. Typically this concerns the point of application of a modifer expressed by a prepositional phrase. The classical example is "I shot an elephant in my trousers" where the modifier "in my trousers" can either apply to elephant (unlikely, given the physical dimensions involved) or to the shooting event.

In the case at hand the analagous problem is the attachment of the time adverbial *within an hour* which could be to the availability of the results, or to the obligation concerning the availability of results (cf. *within an hour I promise to go* vs. *I promise to go within an hour*).

There is also a problem of ellipsis: in the phrase "unless cancelled in the meantime", the syntactic object of the the cancellation has been omitted, and this creates further ambiguity, since it could be the job or the results. We should note that in section 3, both these ambiguities have been resolved: attachment is to the obligation has been favoured, whilst the object of the cancellation is assumed to be the job.

Several problems can be solved by allowing proper names into the language. Predefined ones, like `SYSTEM`, are notated using all capital letters, whilst arbitrary ones, like `Job`, are used to enforce coreference, have an initial capital letter. A second major change is a rationalised event syntax based on a simple agent-action-object format. The easy cases are underlined above. The complex case revolves around `is obliged`, which can usefully be treated as an event (strictly it is a state) whose object is itself an event. We have carefully controlled the syntax and placement of the time adverbials, and, last but not least, the position of `unless` is immediately after the statement of the obligation.

The phrase up to the first comma defines an interval according to a strictly controlled syntax for time expressions. During that interval there are the two possible outcomes. The event syntax has also been rationalised to a simple agent-action-object format.

2. **original:** Only the owner of a job has permission to cancel the job.

   controlled: `it is permitted that only owner of Job cancels Job.`[3]

   logic: $\Box(P(owner_j : cancel_j) \; \& \; F(\overline{owner}_j : cancel_j))$

   comment: A tricky issue in this sentence is the anaphoric status of the determined noun phrases "a job" and "the job". These somehow have to be tied to referents. Informally, "a job" does not have a referent, so a referent is created, and this same referent is referenced by "the job". Although it would be possible, following Fuchs-et-al. [13], to deal with this using Discourse Representation

[3] the CNL grammar avoids the complexity of natural language quantifiers by treating "only" as a simple determiner.

Theory ((cf. Kamp and Reyle [14]), the introduction of proper names, notated in the CNL with an initial capital, offers a simpler solution since it allows the user to establish the coreference explicitly.

We should also note that the syntactic subject of "cancel" is missing. However, it is understood to be the same as the subject of "has permission" as a result of the syntactic behaviour of the verb "to have permission". For example, if "John has permission to leave", then John is the subject of leave. This behaviour is known as *subject-control* and can be handled within the framework of unification grammar. However, our CNL version of this sentence makes use of the "it is permitted that..." formulation in which the object of the permission is fully specified as an agent-action-object triple.

Another aspect concerns the initial noun-phrase, the syntactic structure of which includes "only" as a specifier. Note that semantically this usage makes sense in relation to a set of *referents* (here a singelton: the owner) and a set of *alternative referents* (here a complement set of other agents who might otherwise cancel the job), as reflected in the logical representation suggested in section 3

$\Box(P(owner_j : cancel_j)\ \&\ F(\overline{owner}_j : cancel_j))$

which is straightforward to engineer using techniques from unification grammar.

3. **original:** The system is forbidden from producing a result if it has been cancelled by the owner.

   controlled `If owner of Job cancels Job, it is forbidden that SYSTEM produces result of Job`

   logic: $\Box(cancel_j \rightarrow F(system : (\Diamond(result_j))))$

   comment: The main problem with the original sentence is that it the syntax is complex, and the word *it* ambiguously refers to either *the system* or *result* or *producing a result"*, or, if we consider all three sentences to be part of a discourse, the job mentioned in the second sentence. Although it has been argued that this particular case could be resolved by semantic constraints or or by the use of heuristics (cf. ACE [13]), we feel that all these problems can be avoided by adopting the controlled language being suggested.

5.2 The Grammar

We are currently designing a grammar using PC-PATR, a version of the PATR2 (Shieber [15]) formalism that is available free of charge from the Summer Institute of Linguistics [4].

Such unification grammar formalisms facilitate the transformation of parse tree fragments shown into attribute-value structures that can be regarded as notational variants of the formulae presented in section 3 of this paper. The grammar is currently in the design phase, and our starting point has been the sentences discussed in section 5.1.

A central idea is that a sentence can describe a *simple event* and that an unadorned version of such an event — syntactically and semantically — is a agent-action-object

[4] see http://www.sil.org

triple of the kind used by the Semantic Web community as a building block within RDF[5].

```
s            -> simple-event
simple-event -> agent action object
```

Typically an agent is nominal entity of some kind, an action is described by a verb-group, and an object is another nominal entity, although we allow for a sentential object in order to handle the object of verbs like “allow”, “prohibit” etc.

```
agent  -> nom
action -> vg
object -> nom | s
```

The simplest kind of nominal is a *noun phrase* which can reduce to a proper noun. More complex kinds of noun phrase can involve determiners, prepositional phrases, and hence, subordinate sentences. For example “the hour after SYSTEM make available the result”

```
               np
      __________|____________
     np                   pp
   ___|____      __________|__________
   d      n      p                   s
  the    hour   after                 |
                               simple-event
                        _____________|______________
                      agent     action          object
                        |          |               |
                       nom        vg              nom
                        |      ____|_____          |
                       np      v       aj         np
                        |    make   available   ___|____
                        n                       d      n
                     SYSTEM                    the  result
```

Certain adornments to simple events, such as time adverbials are foreseen. More general adverbials are possible at the end of the simple event[6].

```
s -> t-adv simple-event
s -> simple-event adv
```

Time adverbials can also range from simple one-word specification (“tomorrow”) to complex constructions involving subordinate sentences such as “after SYSTEM make available the result”.

In order to handle modalities we also include a rules of the following type

[5] RDF is a language for expressing simple propositions - see http://www.w3.org/RDF/

[6] an `adv` reduces to a `t-adv`

```
s -> mode s_1
```

where `mode` is defined to include phrases such as “it is forbidden/obligatory that” etc.

To conclude this section, we illustrate these ideas in context with two examples of parse trees for key example sentences 1 and 3:

```
PART OF EXAMPLE 1
=================
                              s
        ______________________|_______________________
      t-adv                                          s
   _____|_____                      _________________|__________________
   p        np                    mode                                 s
during    ___|____      ____________|_____________                     |
          d      n     pro          vg           comp               simple-event
         one   hour    it       ____|_____       that     _______________|_______________
                              aux     ppart                agent     action            object
                              is   obligatory               |          |                  |
                                                           nom        vg                 nom
                                                            |      ____|_____             |
                                                           np      v       aj            np
                                                            |    make   available    _____|_____
                                                            n                       np        pp
                                                     SYSTEM                      |      ___|____
                                                                                 n      p     np
                                                                              results  of      |
                                                                                               n
                                                                                              Job
EXAMPLE 3
=========

                                     s
   __________________________________|___________________________________
 conj                      s                                            s
  if                       |                              ______________|______________
                     simple-event                       mode                           s
               ____________|____________      ____________|____________                |
             agent           action object   pro        vg          comp         simple-event
               |                |      |     it     ____|_____      that     __________|___________
              nom             vg      nom           aux     ppart              agent  action       object
               |                |      |            is    forbidden             |       |            |
              np                v     np                                    nom     vg           nom
          _____|_____        cancels   |                                     |       |            |
         np        pp                  n                                    np       v           np
          |      ___|____             Job                                    |   produces    _____|_____
          n      p     np                                                    n              np        pp
        owner   of      |                                                 SYSTEM             |      ___|____
                        n                                                                    n      p     np
                       Job                                                                result   of      |
                                                                                                           n
                                                                                                          Job
```

6 Semantic Constuction

The last section outlined the grammar which builds syntactic trees out of sentences. The next phase is to build logic expressions. The idea is to use constraint equations to build representation of such expressions expressed as feature structures which are essentially attrubute-value matrices. Example 1, for instance has the following logical representation:

$$\Box(accept_j \rightarrow O(system : (result_j + cancel_j))_{[0,1hr]})$$

but this can also be expressed as a feature structure of the following kind:

$$\begin{bmatrix} \text{modality} & \text{always} \\ \text{action} & \begin{bmatrix} \text{type} & \text{conditional} \\ \text{ante} & \begin{bmatrix} \text{agent} & \text{SYSTEM} \\ \text{act} & \text{accept} \\ \text{object} & \text{J} \\ \text{time} & \begin{bmatrix}\text{start} & \boxed{1}\end{bmatrix} \end{bmatrix} \\ \text{conseq} & \begin{bmatrix} \text{mode} & \text{oblig} \\ \text{time} & \begin{bmatrix} \text{type} & \text{duration} \\ \text{start} & \boxed{1} \\ \text{end} & \boxed{1}+1 \end{bmatrix} \\ \text{agent} & \text{SYSTEM} \\ \text{action} & \begin{bmatrix} \text{type} & \text{choice} \\ \text{action1} & \text{give\_result} \\ \text{action2} & \text{cancel}\end{bmatrix} \\ \text{object} & \text{J} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Such structures are built by annotating grammar rules with constraint equations which the system attempts to satisfy as the grammar rules are invoked.

A successful parse then yields a feature structure of the above type as a possible solution to the constraint equations that have been satisfied. To give a very intuitive example of this, the first line of the above representation could be produced by the following annotation on the respective grammar rule:

```
s -> mode s_1
<s modality> = <mode type>
```

In other words, the value of the modality attribute of the s is the same as the type attrubute of the mode – which in this case would be the constant "always".

7 Conclusions and Future Work

In this article we have outlined the proposal for a controlled natural language and a deontic logic to enable specification and reasoning about contracts in an automatic fashion. A number of challenges still remain to be addressed from both linguistic and logic perspectives.

On the linguistic side the priorities are to firm up the existing grammar specification and to write the rule annotations which will carry out the semantic construction. If this is done correctly, the "translation" between the feature structure and logic representation will be trivial because they should be notational variants.

On the logic side, we plan to use contract analysis techniques similar to ones we have recently developed for CL to enable contract sanity checking, including discovery of conflict analysis and superfluous clauses. These techniques can provide feedback to the contract translation from the natural sublanguage into logic, for example, for the resolution of ambiguities.

The main aim of transforming logic to language is to make it more understandable to those unfamiliar with logic. The transformation is clearly a problem of Natural Language Generation (NLG) which is generally recognised (cf. Reiter and Dale [16]) to involve large numbers of realisation choices. These can of course be by-passed by adopting a suitably strict generation algorithm. The extent to which this compromises naturalness is a factor that will need to be carefully evaluated in the contracts domain.

Once a basic system is in place we will start to experiment with CNL generation tasks, the very first of which will be to translate a logic representation of a contract back into CNL. This will then be compared to the orginal contract. In parallel we will define some simple contract-related inference tasks which will result in new logic expressions. A challenge will then be to investigate the issues involved in adapting the generation algorithm so that the system will report the inferences made back to the user using CNL.

References

1. Pulman, S.: Controlled language for knowledge representation. In: Proceedings of the First International Workshop on Controlled Language Applications, Leuven, Belgium (1996)
2. Vauquois, B.: A survey of formal grammars and algorithms for recognition and transformation in mechanical translation. In: IFIP Congress (2). (1968) 1114–1122
3. Meyer, J.J.C., Dignum, F., Wieringa, R.: The paradoxes of deontic logic revisited: A computer science perspective (or: Should computer scientists be bothered by the concerns of philosophers?). Technical Report UU-CS-1994-38, Department of Information and Computing Sciences, Utrecht University (1994)
4. Pace, G.J., Schneider, G.: Challenges in the specification of full contracts. In: Proceedings of Integrated Formal Methods (iFM'09). LNCS, Springer (2009)
5. Prisacariu, C., Schneider, G.: A formal language for electronic contracts. In: 9th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'07). Volume 4468 of LNCS., Springer (2007)
6. Pace, G., Prisacariu, C., Schneider, G.: Model checking contracts –a case study. In: 5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07). Volume 4762 of LNCS., Tokyo, Japan, Springer-Verlag (2007)
7. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. Journal of the ACM **49**(2) (2002) 172–206
8. Hudak, P.: Building domain-specific embedded languages. ACM Computing Surveys **28** (1996) 196
9. Hudak, P.: Modular domain specific languages and tools. In Devanbu, P., Poulin, J., eds.: Proceedings of the 5th International Conference on Software Reuse, IEEE Computer Society Press (1998) 134–142
10. Jones, S.P., Eber, J.M., Seward, J.: Composing contracts: an adventure in financial engineering (functional pearl). In: ICFP '00: Proceedings of the 5th ACM SIGPLAN international conference on Functional programming, New York, NY, USA, ACM (2000) 280–292
11. Claessen, K., Sheeran, M., Singh, S.: Functional hardware description in Lava. In: The Fun of Programming. Cornerstones of Computing. Palgrave (2003) 151–176
12. Jones, S.P.: Haskell 98 Language and Libraries: The Revised Report. Cambridge University Press (2003)
13. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S., eds.: Reasoning Web, Fourth International Summer School 2008. Number 5224 in Lecture Notes in Computer Science, Springer (2008) 104–124

14. Kamp, H., Reyle, U.: From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory. Studies in Linguistics and Philosophy. Springer (1993)
15. Shieber, S.: An Introduction to Unification-Based Approaches to Grammar. CSLI Publications, Stanford University, California (1986)
16. Reiter, E., Dale, R.: Building natural language generation systems. Studies in natural language processing. Cambridge University Press, Cambridge, U.K. ; New York (2000)