Modelling and analysis of normative contracts

John J. Camilleri and Gerardo Schneider

Chalmers University of Technology and University of Gothenburg, Sweden
john.j.camilleri@cse.gu.se, gerardo@cse.gu.se

1 Introduction

Normative contracts are documents written in natural language, such as English or Swedish, which describe the permissions, obligations, and prohibitions of two or more parties over a set of actions, including descriptions of the penalties which must be payed when the main norms are violated. We encounter such texts frequently in the form of privacy policies, software licenses, and service agreements. These kinds of contracts are often long and difficult to follow for non-experts, and many people agree to such legally-binding documents without even reading them. Our goal is to provide front-end tools for analysing real-world contracts using formal methods. This involves a number of different components, ranging from entity extraction in natural language, the choice and design of a formalism for modelling contracts, textual and visual interfaces for working with contract models, query answering based on syntactic traversal and verification of temporal properties via conversion to timed automata.

2 Front-end

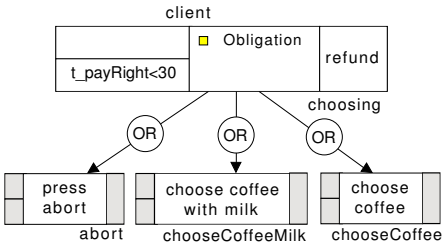
2.1 Extracting partial models

We have built a tool which takes a contract written in English and tries to extract various bits of information from it, in order to bootstrap the modelling process. It uses the Stanford parser [3] to produce dependency trees which we then analyse using some custom heuristics. This involves using the semantic relations between words to determine the subject, object, verb, modality and other elements from each sentence in the contract. After some manual post-editing, the tool's tabular output can be automatically converted to a model in our formalism. Our initial experiments show that the approach is already quite promising, both in terms of accuracy and in the reduction of effort involved for building a contract model.

2.2 Working with models

Diagram editor We visualise contract models as tree-like *C-O Diagrams* [4], an example of which can be seen in Figure 1. We have a web-based tool for working with these diagrams using a drag-and-drop interface along with real-time validation, in order to help the user build syntactically correct diagrams. The tool can import from and export to our XML-based interchange format COML.

CNL A controlled natural language (CNL) is a smaller, unambiguous and formally definable subset of a natural language. CNLs can be particularly useful for specific domains where the coverage of full language is not needed, or when it is possible to abstract away from some irrelevant aspects. We have defined a CNL for our contract models [2], implemented using the Grammatical Framework [6]. Figure 1b shows an example of what a contract clause looks like



(a) *C-O Diagram*

```
choosing : when clock t_payRight less than 30
    ↪ client is required
  - abort : to press abort , or
  - chooseCoffeeMilk : to choose coffee with
      ↪ milk , or
  - chooseCoffee : to choose coffee
  otherwise see refund
```

(b) CNL linearisation

Figure 1: Example of visual and textual representations of an obligation clause.

in this language. As with the diagram editor, we also have a web-based CNL editing tool which comes with syntax checking, inline completion and basic highlighting to aid the user. It too can import and export to the COML format, enabling the user to switch back and forth between editing views.

All these tools are accessible at `http://remu.grammaticalframework.org/contracts/`.

3 Contract formalism

The formalism we use for modelling contracts is based on *C-O Diagrams* [4]. To this we have made a number of syntactic extensions, including a distinction between conditions for enactment and expiry, and the addition of generalised predicates as guards. Our largest contribution has been the definition of a completely novel trace semantics for *C-O Diagrams*, which formally defines what sequences of events can satisfy or violate a given model. We have also worked on a full back-end implementation in Haskell which, by parsing COML files into abstract contract models, can perform the kinds of analysis described in the following section.

4 Analysis

Syntactic Some queries can be checked at a syntactic level, such as identifying obligations without constraints or reparations. We introduce predicates over single clauses, which are the building blocks for defining syntactic properties. The predicate $isObl(C)$ for example is true if the clause C is an obligation. Predicates may also take additional arguments, such as $fromAgent(a, C)$, which is true if agent a is responsible for clause C. Full queries can then be built out of these predicates, and a querying function returns the set of all clauses that satisfy the predicate. This function is defined inductively on the structure of contract models.

Semantic Syntactic analysis alone cannot be used to answer queries about the reachability of a given state. This requires taking into account the conditions applied to each clause, as well as a possible trace of previous events. These kinds of properties are computed using model checking. To do this, we convert contract models into networks of timed automata (NTA) [1] — finite state automata extended with guarded transitions, real-time clocks, and channel-based synchronisation between parallel automata (see example in Figure 2).

Using the Uppaal tool [5], we can then test liveness and safety properties on our translated model using Uppaal's requirement specification language, which is a subset of TCTL including operators for *possibly* ($E\Diamond$) *potentially always* ($E\Box$) and *eventually* ($A\Diamond$). This language allows

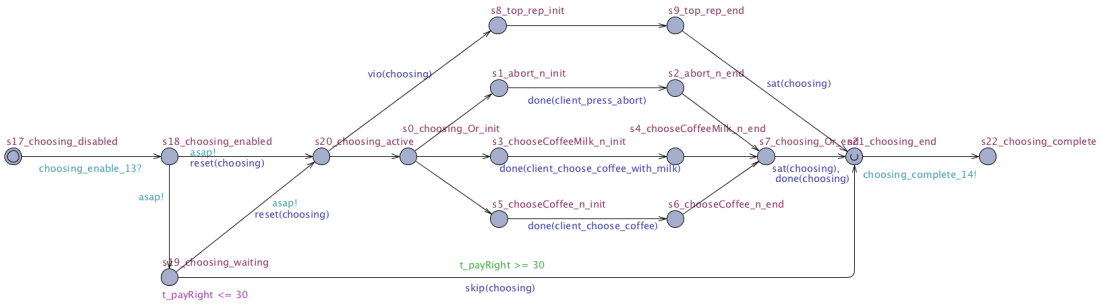


Figure 2: One of the automata produced from the translation of *C-O Diagram* 1a.

us to test for unexpected or undesirable situations which may potentially arise in the execution of a contract. These properties must be written and tested directly in UPPAAL by the user.

Case studies We have applied our methods to a few smaller case studies, including the terms of service for GitHub, Inc.[1] and a service-level agreement from hosting company LeaseWeb Inc.[2]

5 Future work

The largest piece missing from this work is the connection between high-level user questions in natural language and the low-level specification languages used for analysis. For this, we will define a query language (similar to the CNL described above) which can help users build properties for contract analysis by using a human-friendly interface. This will involve classifying the different queries we wish to allow, a method for converting these into logical properties using information from the translation into automata, and using the results of the analysis to produce properly formulated answers to the original query in natural language.

References

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] John J. Camilleri, Gabriele Paganelli, and Gerardo Schneider. A CNL for Contract-Oriented Diagrams. In *Controlled Natural Language*, volume 8625 of *LNCS*, pages 135–146. Springer, 2014.

[3] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *LREC 2006*, pages 449–454, 2006.

[4] Gregorio Díaz, Maria Emilia Cambronero, Enrique Martínez, and Gerardo Schneider. Specification and Verification of Normative Texts using C-O Diagrams. *IEEE TSE*, 40(8):795–817, 2014.

[5] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 2014.

[6] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011. ISBN-10: 1-57586-626-9.

[1] `https://github.com/site/terms`
[2] `https://www.leaseweb.com/legal`