Journal of Logical and Algebraic Methods in Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Modelling and analysis of normative documents

John J. Camilleri [*], Gerardo Schneider

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

A R T I C L E   I N F O

Article history:
Received 25 February 2016
Received in revised form 3 May 2017
Accepted 4 May 2017
Available online xxxx

Keywords:
Normative documents
Contract analysis
Timed automata
UPPAAL

A B S T R A C T

We are interested in using formal methods to analyse *normative documents* or *contracts* such as terms of use, privacy policies, and service agreements. We begin by modelling such documents in terms of obligations, permissions and prohibitions of agents over actions, restricted by timing constraints and including potential penalties resulting from the non-fulfilment of clauses. This is done using the *C-O Diagram* formalism, which we have extended syntactically and for which we have defined a new trace semantics. Models in this formalism can then be translated into networks of timed automata, and we have a complete working implementation of this translation. The network of automata is used as a specification of a normative document, making it amenable to verification against given properties. By applying this approach to a case study from a real-world contract, we show the kinds of analysis possible through both syntactic querying on the structure of the model, as well as verification of properties using UPPAAL.

1. Introduction

We frequently encounter *normative documents* (or *contracts*) when subscribing to internet services and using software. These come in the forms of terms of use, privacy policies, and service-level agreements, and we often accept these kinds of contractual agreements without really reading them. Though they are written using natural language, understanding the details of such documents often requires legal experts, and ambiguities in their interpretation are commonly disputed. Our goal is to model such texts formally in order to enable automatic querying and analysis of contracts, aimed at benefitting both authors of contracts and their users. To realise this, we are developing an end-to-end framework for the analysis of normative documents, combining natural language technology with formal methods. An outline of this framework is shown in Fig. 1.

Formal analysis requires a formal language: a given syntax together with a well-defined semantics and a state-space exploration technique. Well-known generic formalisms such as first-order logic or temporal logic would not provide the right level of abstraction for a domain-specific task such as modelling normative texts. Instead, we choose to do this with a custom formalism based on the *deontic modalities* of **obligation**, **permission** and **prohibition**, and containing only the operators that are relevant to our domain. Specifically, we use the *Contract-Oriented (C-O) Diagram* formalism [1], which provides both a logical language and a visual representation for modelling normative texts. This formalisation allows us to perform syntactic analysis of the models using predicate-based queries. Additionally, we are able to translate models in this formalism into networks of timed automata (NTA) [2] which are amenable to model checking techniques, providing further possibilities for analysis.

[*] Corresponding author.
E-mail addresses: john.j.camilleri@cse.gu.se (J.J. Camilleri), gerardo@cse.gu.se (G. Schneider).

http://dx.doi.org/10.1016/j.jlamp.2017.05.002
2352-2208/© 2017 Elsevier Inc. All rights reserved.

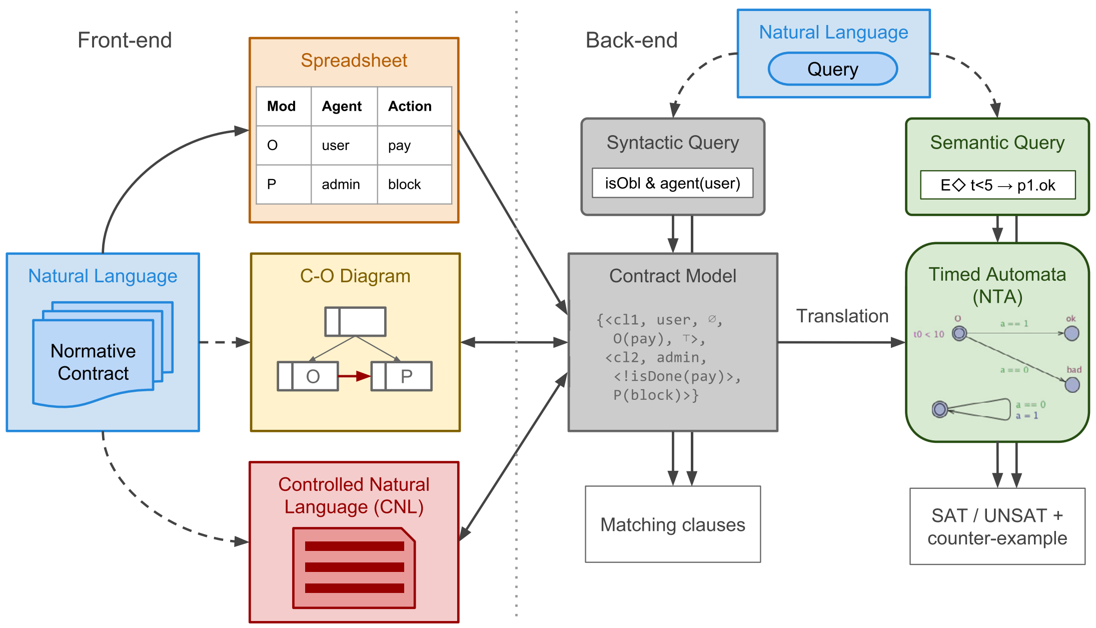

Fig. 1. Overview of our contract processing framework, separating the front-end concerns of model-building from the back-end tasks related to analysis. Dashed arrows represent manual interaction, while solid ones represent automatic steps.

Table 1
Legend of symbols and functions used. Where relevant, we have also included references to their definitions.

$\mathcal{N}$	set of names	ϕ, ψ	predicate name placeholders
$\mathcal{A}$	set of agents	ϵ	empty conditions
Σ	set of actions	$\emptyset$	empty guard/constraint list
$\mathcal{V}$	set of integer variables	ε	empty bound in interval
$\mathcal{C}$	set of clocks	Γ	environment
$\mathcal{B}$	set of Boolean flags	$get_{v/c/b}$	getters (17), (18), (19)
$\mathbb{N}$	type of natural numbers	$set_{v/b}$	setters (20), (21)
$\mathbb{Z}$	type of integers	$reset_c$	reset clock (22)
$\mathbb{T}$	type of time stamps	*lookup*	lookup clause by name (23)
σ	event trace	τ	combine interval with constraints (24)
$\mathcal{T}$	set of event traces	*lst*	find lowest satisfying time stamp (28)
σ_U	Uppaal timed trace	*check*	check a set of constraints (26)
$\mathcal{T}_U$	set of timed traces	*eval*	evaluate a constraint (27)
$\mathcal{T}_U^C$	set of timed traces corresponding to the satisfaction of C	*trf*	translate from *C-O Diagram* to Uppaal model (Section 3)
$\vDash$	*respects* relation between traces and contracts (Fig. 6)	*abstr*	translate from timed trace to event trace
		$\mathcal{Q}$	syntactic query function (53)

Building such models from natural language texts is a non-trivial task which can benefit greatly from the right tool support. In previous work [3] we presented front-end user applications for working with *C-O Diagram* models both as graphical objects and through a controlled natural language (CNL) interface (shown on the left-hand side of Fig. 1). The ability to work with models in different higher-level representations makes the formalism more attractive for real-world use when compared to other purely logical formalisms. The present work is concerned with the back-end of this system, focusing on the details of the modelling language and the different kinds of analysis that can be performed on these models.

Contributions and outline The paper is layed out as follows. In Section 2 we first present an extended definition of the *C-O Diagram* formalism, introducing an updated syntax and a novel trace semantics. Section 3 then describes our own translation function from the extended *C-O Diagram* formalism into Uppaal timed automata, which is more modular and fixes a number of issues with respect to the previous translation given in [4]. Our contribution includes the first fully-working implementation of this translation, written in Haskell. We also prove the correctness of this translation function with respect to our trace semantics. Section 4 covers the analysis processes that we can perform on this formalism, discussing our methods for syntactic querying and semantic property checking of contract models. We demonstrate these methods by applying them to a case study from a real-world contract in Section 5. Finally, we conclude with a comparison of some related work in Section 6 and a final discussion in Section 7.

Notation Table 1 presents the symbols and function names used throughout the rest of this article.