Towards Model-Checking Contracts

Cristian Prisacariu and Gerardo Schneider

Department of Informatics – University of Oslo,
P.O. Box 1080 Blindern, N-0316 Oslo, Norway.
{cristi, gerardo}@ifi.uio.no

Abstract

We understand by a *contract* a document written in natural language which engages several parties into a transaction, and which stipulates commitments (obligations, rights, and prohibitions) of the parties. Moreover the contract specifies also reparations in case of contract violation (i.e. some obligations or prohibitions are not respected). Because the human language is ambiguous by nature, contracts (written in plain English, for example) are inherent ambiguous. This ambiguity can, and many times is exploited by the parties involved in the contract. The purpose of our research is to eliminate this ambiguity as much as possible and to automate the process of designing, negotiation and monitoring of contracts. For this purpose contracts should be amenable to formal analysis (including model-checking) and thus should be written in a formal language.

There are currently several different approaches aiming at defining a formal language for contracts. Some works concentrate on the definition of contract taxonomies [1], while others look for formalizations based on logics (e.g. classical [4], modal [3], deontic [6] or defeasible logic [5]). Other formalizations are based on models of computation (e.g. FSMs [7], Petri Nets [2], or process calculi [12]). None of the above has reached enough maturity as to be considered *the* solution to the problems of formal definition of contracts. In our opinion, the most promising approach to formalizing contracts is the one based on logics of actions [13,14] (i.e. actions found in contracts). It has been argued for the need to base deontic logic on a theory of actions which would solve many of the paradoxes deontic logic faces.

The method of *model-checking* is an old and established field of computer science. Model-checking has been applied in several fields of computer science from hardware circuits to concurrent programs. The idea of *model-checking electronic contracts* is extremely new and of great interest. From our knowledge there has been no attempt of using the classical model-checking techniques (or an established model-checking tool) on a real electronic contract example. Model-checking tools usually describe the system as an automata-like structure and the property to be checked in a temporal logic (like CTL, LTL, or μ-calculus). With the contract written in a formal language with semantics based on a Kripke-like structure we wold have the automaton input for the model-checking tools.

Our aim is to define a general framework for describing in a uniform way both the contract and the properties.

In [11] we have provided a formal language for writing contracts, which allows to write (conditional) obligations, permissions and prohibitions over (names of) human *actions* as well as reparations in case of violations. The language is specially tailored for representing statements found in contracts and is proven to avoid major deontic paradoxes (for motivations and design decisions we refer the reader to [11]). Here we briefly sketch and discuss the ideas behind the syntax of the contract language $\mathcal{CL}$.

$$\mathcal{C}ontract := \mathcal{D} \,;\, \mathcal{C}$$
$$\mathcal{C} := \phi \mid \mathcal{C}_O \mid \mathcal{C}_P \mid \mathcal{C}_F \mid \mathcal{C} \wedge \mathcal{C} \mid [\alpha]\mathcal{C} \mid \langle\alpha\rangle\mathcal{C} \mid \mathcal{C}\,\mathcal{U}\,\mathcal{C} \mid \bigcirc \mathcal{C} \mid \Box\mathcal{C}$$
$$\mathcal{C}_O := O(\alpha) \mid \mathcal{C}_O \oplus \mathcal{C}_O \quad \mathcal{C}_P := P(\alpha) \mid \mathcal{C}_P \oplus \mathcal{C}_P \quad \mathcal{C}_F := F(\alpha) \mid \mathcal{C}_F \vee [\alpha]\mathcal{C}_F$$

A contract consists of two parts: *definitions* ($\mathcal{D}$) and *clauses* ($\mathcal{C}$). $\mathcal{D}$ specifies the *assertions* ϕ (which ranges over Boolean expressions including arithmetic comparisons, like *the budget is more than 200$*), and the atomic actions present in the clauses. Actions α are formalized by the $\mathcal{CAT}$ action algebra which we see later. $\mathcal{C}$ is the general *contract clause*. $\mathcal{C}_O$, $\mathcal{C}_P$, and $\mathcal{C}_F$ denote respectively *obligation*, *permission*, and *prohibition* clauses. $\wedge$, $\oplus$, and $\vee$ may be thought as the classical conjunction, exclusive disjunction, and disjunction.

We borrow from Propositional Dynamic Logic (PDL) the syntax $[\alpha]\phi$ to represent that after performing α (if it is possible to do so), ϕ must hold. The $[\cdot]$ notation allows having a *test*, where $[\phi?]\mathcal{C}$ must be understood as $\phi \Rightarrow \mathcal{C}$. The syntax $\langle\alpha\rangle\phi$ captures the idea that there must exist the possibility of executing action α, in which case ϕ must hold afterwards. Following temporal logic (TL) [9] notation we have $\mathcal{U}$ (*until*), $\bigcirc$ (*next*), and $\Box$ (*always*) with intuitive semantics as in TL. Thus $\mathcal{C}_1\,\mathcal{U}\,\mathcal{C}_2$ states that $\mathcal{C}_1$ should hold until $\mathcal{C}_2$ holds. $\bigcirc\mathcal{C}$ intuitively states that the $\mathcal{C}$ should hold in the next moment.

In [10] we have defined a new algebraic structure to provide a well-founded formal basis for $\mathcal{CL}$ and to help give a direct semantics to the language. The main contributions of the algebra are: (1) A formalization of actions found in contracts; (2) The introduction of a different kind of negation over actions; (3) A restricted notion of resource-awareness; and (4) A standard interpretation of the algebra over specially defined guarded rooted trees. The algebra is proven to be complete w.r.t. the interpretation as trees.

We called our algebra $\mathcal{CAT}$ to stand for *algebra of concurrent actions and tests*. $\mathcal{CAT} = (\mathcal{CA}, \mathcal{B})$ is formed of an algebraic structure $\mathcal{CA} = (\mathcal{A}, +, \cdot, \&, \mathbf{0}, \mathbf{1})$ which defines the concurrent actions, and a Boolean algebra $\mathcal{B} = (\mathcal{A}_\mathbf{1}, \vee, \wedge, \neg, \bot, \top)$ which defines the tests. Special care has to be taken when combining actions and tests under the different algebraic operators.

Actions of $\mathcal{A}$ are constructed from atomic actions $\mathcal{A}_B$ by applying: *choice* ($+$), *sequence* ($\cdot$), or *concurrent* composition ($\&$). Tests of $\mathcal{A}_\mathbf{1}$ are constructed with the classical boolean operators ($\vee$, $\wedge$, and $\neg$) and the constants ($\bot$ and $\top$). Moreover, $(\mathcal{A}, +, \cdot, \mathbf{0}, \mathbf{1})$ is an idempotent semiring and $(\mathcal{A}, +, \&, \mathbf{0}, \mathbf{1})$ is a commutative and idempotent semiring. We consider a *conflict relation* over the set of atomic actions $\mathcal{A}_B$ (denoted by $\#_\mathcal{C}$) defined as: $a \#_\mathcal{C} b \overset{def}{\iff} a\&b = \mathbf{0}$. The intuition of the conflict relation is that if two actions are in conflict then the actions cannot be executed concurrently.

With the actions and their interpretation as guarded rooted trees it is simple now to give a direct semantics to the language $\mathcal{CL}$. In [11] the semantics of $\mathcal{CL}$ was given through a translation into a variant of μ-calculus. We have used $\mathcal{CL}$ and the μ-calculus semantics in [8] to do model-checking of an example from a real contract.

We have used the state-of-the-art tool NuSMV and followed the steps: (1) first translate the conventional contract into the formal language $\mathcal{CL}$; (2) translate syntactically the $\mathcal{CL}$ specification into the extended μ-calculus $\mathcal{C}\mu$ to obtain a Kripke-like model (a labeled transition system, LTS) of the $\mathcal{C}\mu$ formulas (representing the semantics of the $\mathcal{CL}$ specification); (3) translate the LTS into the input language of NuSMV and perform model checking using NuSMV; (4) in case of a counter-example given by NuSMV, interpret it as a $\mathcal{CL}$ clause and repeat the model checking process until the property is satisfied; (5) finally, repair the original contract by adding a corresponding clause, if applicable.

Besides classical properties like liveness, safety, or response which can be specified using LTL or CTL we are also interested in properties more specific to electronic contracts. The first kind of properties is *search properties*. Examples of search properties include: listing of the obligations, or prohibitions of one of the parties in the contract; listing of the rights that follow after the fulfilling of an obligation; or what are the penalties for whenever violating an obligation or prohibition, and so on. A second kind of reasoning about a contract (formalized in our contract language) is quantitative reasoning with respect to the amount of obligations or of rights a contract imposes. Examples of such properties include: the client would like to know if the contract can be changed so that the client has a smaller number of obligations, or a greater number of rights; if quantitative measures are inserted into the language (like time or amounts) one could ask to minimize time between certain events, or try to change the contract so that a minimal(maximal) amount is reached.

References

1. A. Beugnard, J.-M. Jézéquel, and N. Plouzeau. Making components contract aware. *IEEE Computer*, 32(7):38–45, 1999.
2. A. Daskalopulu. Model Checking Contractual Protocols. In L. Breuker and Winkels, editors, *Legal Knowledge and Information Systems, JURIX 2000*, pages 35–47. IOS Press, 2000.
3. A. Daskalopulu and T. S. E. Maibaum. Towards Electronic Contract Performance. In *12th International Conference and Workshop on Database and Expert Systems Applications*, pages 771–777. IEEE C.S. Press, 2001.
4. H. Davulcu, M. Kifer, and I. V. Ramakrishnan. CTR-S: A Logic for Specifying Contracts in Semantic Web Services. In *Proceedings of WWW2004*, pages 144–153, May 2004.
5. G. Governatori. Representing business contracts in RuleML. *International Journal of Cooperative Information Systems*, 14:181–216, 2005.
6. G. Governatori and A. Rotolo. Logic of violations: A gentzen system for reasoning with contrary-to-duty obligations. *Australatian Journal of Logic*, 4:193–215, 2006.
7. C. Molina-Jiménez, S. K. Shrivastava, E. Solaiman, and J. P. Warne. Run-time Monitoring and Enforcement of Electronic Contracts. *Electronic Commerce Research and Applications*, 3(2), 2004.
8. G. Pace, C. Prisacariu, and G. Schneider. Model checking contracts - a case study. In K. Namjoshi and T. Yoneda, editors, *ATVA'07*, volume 4762 of *LNCS*, pages 82–97. Springer, 2007.
9. A. Pnueli. Temporal logic of programs, the. In *FOCS'77*, pages 46–57. IEEE Computer Society Press, 1977.
10. C. Prisacariu and G. Schneider. An algebraic structure for the Action-Based Contract language CL - Theoretical Results. Technical Report 361, Department of Informatics, University of Oslo, Oslo, Norway, July 2007.
11. C. Prisacariu and G. Schneider. A formal language for electronic contracts. In M. Bonsangue and E. B. Johnsen, editors, *FMOODS'07*, volume 4468 of *LNCS*, pages 174–189. Springer, 2007.
12. G. Salaün, L. Bordeaux, and M. Schaerf. Describing and reasoning on web services using process algebra. In *ICWS '04*, page 43. IEEE Computer Society, 2004.
13. G. H. V. Wright. Deontic logic. *Mind*, 60:1–15, 1951.
14. G. H. V. Wright. Deontic logic: A personal view. *Ratio Juris*, 12(1):26–38, 1999.