

Model Checking Contracts

– a case study ^{*}

Gordon Pace¹, Cristian Prisacariu², and Gerardo Schneider²

¹ Dept. of Computer Science and AI, University of Malta, Msida, Malta.

² Department of Informatics – University of Oslo,
P.O. Box 1080 Blindern, N-0316 Oslo, Norway.

`gordon.pace@um.edu.mt`, `{cristi, gerardo}@ifi.uio.no`

Abstract. Contracts are agreements between distinct parties that determine rights and obligations on their signatories, and have been introduced in order to reduce risks and to regulate inter-business relationships. In this paper we first show how a conventional contract can be written in the contract language \mathcal{CL} , and then we verify the correctness of the contract with respect to certain desired properties, using the NuSMV model checking tool.

1 Introduction

Internet-based applications involving one or more entities participating in inter-business collaborations, virtual organizations, and web services, usually communicate through service exchanges. Such exchanges are subject to certain understanding on the different roles the participants play, including assumptions on their correct and incorrect behaviors, and their rights and obligations in order to avoid misunderstanding and ambiguities in such business relationships. This motivates the need of establishing an agreement before any transaction is performed, through a *contract*, guaranteeing the rights and duties of each signatory. Such documents must also contain clauses determining penalties in case of contract violations, and be as unambiguous as possible to avoid conflicting interpretations. *Conventional contracts* are documents written in natural language, as one may find in usual judicial or commercial traditional activities. On the other hand, *electronic contracts* (or e-contracts for short) are machine-oriented and as such they must be “understood” by the software responsible for controlling and monitoring the service exchanges. E-contracts might be seen in two different ways: (1) As the executable version of a conventional contract, obtained from the translation of the “paper” version into the electronic one; (2) As contracts by themselves obtained directly from certain software applications, like web services and virtual organizations. For our current purposes, the difference above is irrelevant, though our case study is based on a conventional contract.

^{*} Partially supported by the Nordunet3 project “Contract-Oriented Software Development for Internet Services”.

Ideally, e-contracts should be shown to be contradiction-free both internally, and with respect to the governing policies under which the contract is enacted. Moreover, there must be a run-time system ensuring that the contract is respected. In other words, contracts should be amenable to formal analysis allowing both static and dynamic verification, and thus written in a formal language. In this paper we are interested only in the analysis of the contract itself (statically), and we are not concerned with its relation with policies nor with its enforcement at run-time.

A formal language for writing contracts should be designed as to avoid most of the philosophical problems of deontic logic [1]. Moreover, it should be possible to represent conditional obligations, permissions and prohibitions, as well as *contrary-to-duty obligations* (CTD) and *contrary-to-prohibitions* (CTP). CTDs are statements representing obligations that might not be respected, whereas CTPs are similar statements dealing with prohibitions that might be violated. Both constructions specify the obligation/prohibition to be fulfilled and which is the *reparation/penalty* to be applied in case of violation.

A formal language for writing (untimed) contracts is \mathcal{CL} [2]. The language is tailored to e-contracts, following an action-based approach, and having the following properties: (1) The language avoids most of the classical paradoxes of deontic logic; (2) It is possible to express in the language (conditional) obligations, permission and prohibition over concurrent actions keeping their intuitive meaning; (3) It is possible to express CTDs and CTPs; (4) The language has a formal semantics given in a variant of the modal μ -calculus.

The main contribution of the present paper is to show, on a contract example, how model checking techniques can be applied in the context of contract-oriented software development, in order to determine whether a given contract stipulates what it is supposed to. The method that we put forward in this paper uses \mathcal{CL} as a specification language for both the system and the property to be checked. \mathcal{CL} is used as an intermediary between the contract clauses in plain English and the system specification required by the model checking tool. This use of \mathcal{CL} increases the confidence in the initial formulation of the contract clauses. The model-checking method that we present requires to pursue the following steps:

1. Translate the conventional contract written in English into the formal language \mathcal{CL} ;
2. Translate syntactically the \mathcal{CL} specification into the extended μ -calculus $\mathcal{C}\mu$;
3. Obtain a Kripke-like model (a labeled transition system, LTS) of the $\mathcal{C}\mu$ formulas;
4. Translate the LTS into the input language of NuSMV;
5. Perform model checking using NuSMV;
6. 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;
7. Finally, repair the original contract by adding a corresponding clause, if applicable.

The paper is organized as follows. In Section 2 we start by presenting the language \mathcal{CL} , including an example of the kind of contracts we are dealing with,

This deed of **Agreement** is made between:

1. **[name]**, from now on referred to as **Provider** and
2. **[name]**, from now on referred to as the **Client**.

INTRODUCTION

3. The **Provider** is obliged to provide the **Internet Services** as stipulated in this **Agreement**.

5. **DEFINITIONS**

- 5.1. j) **Internet traffic** may be measured by both **Client** and **Provider** by means of Equipment and may take the two values **high** and **normal**.

OPERATIVE PART

7. **CLIENT'S RESPONSIBILITIES AND DUTIES**

- 7.1. The **Client** shall not:
 - a) supply false information to the Client Relations Department of the **Provider**.
- 7.2. Whenever the Internet Traffic is **high** then the **Client** must pay [*price*] immediately, or the **Client** must notify the **Provider** by sending an e-mail specifying that he will pay later.
- 7.3. If the **Client** delays the payment as stipulated in 7.2, after notification he must immediately lower the Internet traffic to the **normal** level, and pay later twice ($2 * [price]$).
- 7.4. If the **Client** does not lower the Internet traffic immediately, then the **Client** will have to pay $3 * [price]$.
- 7.5. The **Client** shall, as soon as the Internet Service becomes operative, submit within seven (7) days the Personal Data Form from his account on the **Provider's** web page to the Client Relations Department of the **Provider**.

8. **CLIENT'S RIGHTS**

- 8.1. The **Client** may choose to pay either:
 - a) each month; b) each three (3) months; c) each six (6) months;

9. **PROVIDER'S SERVICE**

- 9.2. As part of the Service offered by the **Provider** the **Client** has the right to an e-mail and an user account.
- 9.3. **Provider** is obliged to offer with no limitation and within a period of seven (7) days a password and any other Equipment Specific to Client, necessary for the correct usage of the user account, upon receiving of all the necessary data about the client from the Client Relations Department of the **Provider**.
- 9.4. Each month the **Client** pays the *bill* the **Provider** is obliged to send a Report of Internet Usage to the Client.

10. **PROVIDER'S DUTIES**

- 10.1. The **Provider** takes the obligation to return the personal data of the client to the original status upon termination of the present **Agreement**, and afterwards to delete and not use for any purpose any whole or part of it.
- 10.2. The **Provider** guarantees that the Client Relations Department, as part of his administrative organization, will be responsive to requests from the **Client** or any other Department of the **Provider**, or the **Provider** itself within a period less than two (2) hours during *working hours* or the day after.

11. **PROVIDER'S RIGHTS**

- 11.1. The **Provider** takes the right to alter, delete, or use the *personal data* of the **Client** only for statistics, monitoring and internal usage in the confidence of the **Provider**.
- 11.2. **Provider** may, at its sole discretion, without notice or giving any reason or incurring any liability for doing so:
 - b) Suspend Internet Services immediately if **Client** is in breach of Clause 7.1;

13. **TERMINATION**

- 13.1. Without limiting the generality of any other *Clause* in this *Agreement* the **Client** may terminate this *Agreement* immediately without any notice and being vindicated of any of the Clause of the present Agreement if:
 - a) the **Provider** does not provide the Internet Service for seven (7) days consecutively.
- 13.2. The **Provider** is forbidden to terminate the present Agreement without previous written notification by normal post and by e-mail.
- 13.3. The **Provider** may terminate the present Agreement if:
 - a) any payment due from **Client** to **Provider** pursuant to this **Agreement** remains unpaid for a period of fourteen (14) days;

16. **GOVERNING LAW**

- 16.1. The **Provider** and the present **Agreement** are governed by and construed according to the Law Regulating Internet Services and to the Law of the State.
 - a) The Law of the State stipulates that any **ISP Provider** is obliged, upon request to seize any activity until further notice from the State representatives.

Fig. 1. Part of a contract between an Internet provider and a client.

from which we will extract our case study. Section 3 is the main part of the paper where we first formalize the case study in \mathcal{CL} , and afterwards we show how to use model checking and the NuSMV tool to determine whether the contract is correct w.r.t. certain desired properties, and how to get feedback as to write the “correct” contract. In Section 4 we analyze related works and conclude by discussing our choice of the model-checking tool as well as future work.

2 A Formal Language for Contracts

We present in Fig. 1 a part of a conventional contract between a service provider and a client, where the provider gives access to Internet to the client. We analyze part of this contract in the following section. First we recall the contract language \mathcal{CL} ; for a more detailed presentation see [2].

Definition 1 (Contract Language Syntax). *A contract is defined by:*

$$\begin{aligned}
\text{Contract} &:= \mathcal{D} ; \mathcal{C} \\
\mathcal{C} &:= \phi \mid \mathcal{C}_O \mid \mathcal{C}_P \mid \mathcal{C}_F \mid \mathcal{C} \wedge \mathcal{C} \mid [\alpha]\mathcal{C} \mid \langle \alpha \rangle \mathcal{C} \mid \mathcal{C} \mathcal{U} \mathcal{C} \mid \bigcirc \mathcal{C} \mid \square \mathcal{C} \\
\mathcal{C}_O &:= O(\alpha) \mid \mathcal{C}_O \oplus \mathcal{C}_O \\
\mathcal{C}_P &:= P(\alpha) \mid \mathcal{C}_P \oplus \mathcal{C}_P \\
\mathcal{C}_F &:= F(\delta) \mid \mathcal{C}_F \vee [\delta]\mathcal{C}_F
\end{aligned}$$

The syntax of \mathcal{CL} closely resembles the syntax of a modal (deontic) logic. Though this similarity is clearly intentional since we are driven by a logic-based approach, \mathcal{CL} is *not* a logic. The interpretation of the \mathcal{CL} syntax is given by translating it into an extension of μ -calculus [3] which we call $\mathcal{C}\mu$. In what follows we provide an intuitive explanation of the \mathcal{CL} syntax.

A contract consists of two parts: *definitions* (\mathcal{D}) and *clauses* (\mathcal{C}). We deliberately let the definitions part underspecified in the syntax above. \mathcal{D} specifies the *assertions* (or conditions) and the atomic actions present in the clauses. ϕ denotes assertions and ranges over boolean expressions including the usual boolean connectives, and arithmetic comparisons like “the budget is more than 200\$”. We let the atomic actions underspecified, which for our purposes can be understood as consisting of three parts: the proper action, the subject performing the action, and the target of (or, the object receiving) such an action. Note that, in this way, the parties involved in a contract are encoded in the actions.

\mathcal{C} is the general *contract clause*. \mathcal{C}_O , \mathcal{C}_P , and \mathcal{C}_F denote respectively *obligation*, *permission*, and *prohibition* clauses. $O(\cdot)$, $P(\cdot)$, and $F(\cdot)$, represents the obligation, permission or prohibition of performing a given action. \wedge and \oplus may be thought as the classical conjunction and exclusive disjunction, which may be used to combine obligations and permissions. For prohibition \mathcal{C}_F we have \vee , again with the classical meaning of the corresponding operator. α is a compound action (i.e., an expression containing one or more of the following operators: choice “+”; sequence “.”; concurrency “&”, and test “?” —see [2]), while δ denotes a compound action not containing any occurrence of +. Note that

syntactically \oplus cannot appear between prohibitions and $+$ cannot occur under the scope of F .

We borrow from propositional dynamic logic [4] 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 $\phi \Rightarrow \mathcal{C}$. $\langle\alpha\rangle\phi$ captures the idea that it exists the possibility of executing α , in which case ϕ must hold afterwards. Following temporal logic (TL) notation we have \mathcal{U} (*until*), \mathcal{O} (*next*), and \Box (*always*), with intuitive semantics as in TL [5]. Thus $\mathcal{C}_1\mathcal{U}\mathcal{C}_2$ states that \mathcal{C}_1 holds until \mathcal{C}_2 holds. $\mathcal{O}\mathcal{C}$ intuitively states that \mathcal{C} holds in the next moment, usually after something happens, and $\Box\mathcal{C}$ expressing that \mathcal{C} holds in every moment. We can define $\Diamond\mathcal{C}$ (*eventually*) for expressing that \mathcal{C} holds sometimes in a future moment.

To express CTDs we provide the following notation, $O_\varphi(\alpha)$, which is syntactic sugar for $O(\alpha) \wedge [\bar{\alpha}]\varphi$ stating the obligation to execute α , and the reparation φ in case the obligation is violated, i.e. whenever α is not performed. The reparation may be any contract clause. Similarly, CTP statements $F_\varphi(\alpha)$ can be defined as $F_\varphi(\alpha) = F(\alpha) \wedge [\alpha]\varphi$, where φ is the penalty in case the prohibition is violated. Notice that it is possible to express nested CTDs and CTPs.

In \mathcal{CL} we can write *conditional* obligations, permissions and prohibitions in two different ways. Just as an example let us consider conditional obligations. The first kind is represented as $[\alpha]O(\beta)$, which may be read as “after performing α , one is obliged to do β ”. The second kind is modeled using the test operator $?$: $[\varphi?]O(\alpha)$, representing “If φ holds then one is obliged to perform α ”. Similarly for permission and prohibition. For convenience, in what follows we use the notation $\phi \Rightarrow \mathcal{C}$ instead of the \mathcal{CL} syntax $[\phi?]\mathcal{C}$.

3 A Contract Case Study

In what follows we consider part 7 of the contract given in Figure 1 between a service provider and a client, where the provider gives access to the Internet to the client. We consider two parameters of the service: *high* and *normal*, which denote the client’s Internet traffic. We will consider only the following clauses of the contract.

- 7.1. The **Client** shall not:
 - a) supply false information to the Client Relations Department of the **Provider**.
- 7.2. Whenever the Internet Traffic is **high** then the **Client** must pay $[price]$ immediately, or the **Client** must notify the **Provider** by sending an e-mail specifying that he will pay later.
- 7.3. If the **Client** delays the payment as stipulated in 7.2, after notification he must immediately lower the Internet traffic to the **normal** level, and pay later twice ($2 * [price]$).
- 7.4. If the **Client** does not lower the Internet traffic immediately, then the **Client** will have to pay $3 * [price]$.
- 7.5. The **Client** shall, as soon as the Internet Service becomes operative, submit within seven (7) days the Personal Data Form from his account on the **Provider**’s web page to the Client Relations Department of the **Provider**.

We also add clause 11.2 as it is strongly related to clause 7.1 and the two should be taken together:

- 11.2. **Provider** may, at its sole discretion, without notice or giving any reason or incurring any liability for doing so:
 - b) Suspend Internet Services immediately if **Client** is in breach of Clause 7.1;

In what follows we formalize the above contract clauses. As part of the formalization of a contract in \mathcal{CL} we first have to define the assertions and actions:

- ϕ = the Internet traffic is high
- fi = client supplies false information to Client Relations Department
- h = client increases Internet traffic to *high* level
- p = client pays [price]
- d = client delays payment
- n = client notifies by e-mail
- l = client lowers the Internet traffic
- sfD = client sends the Personal Data Form to Client Relations Department
- o = provider activates the Internet Service (it becomes operative)
- s = provider suspends service

Note that we have the action h which does not appear explicitly in the example clauses. Action h is implicit as it makes the proposition ϕ valid (the Internet becomes *high* only if the client increases it). Action h can be considered as the complement of action l which makes ϕ false (lowers the Internet traffic). The six clauses above are written in \mathcal{CL} as follows:

1. $\Box F_{P(s)}(fi)$
2. $\Box[h](\phi \Rightarrow O(p + (d\&n)))$
3. $\Box([d\&n](O(l) \wedge [l]\Diamond O(p\&p)))$
4. $\Box([d\&n \cdot \bar{l}]\Diamond O(p\&p\&p))$
5. $\Box([o]O(sfD))$

Clause 1 has a concise syntax and represents a *contrary-to-prohibition*. More precisely, the CTP represents the prohibition $F(fi)$ (clause 7.1) and the reparation which should be enforced in case the prohibition is violated (in this case $P(s)$; the right of the provider to suspend the Internet service, clause 11.2).

Note that all the clauses are supposed to hold throughout the whole contract because of the \Box . Clause 2 models clause 7.2 of the contract example and it represents the fact that whenever the assertion ϕ holds (the Internet traffic of the client is at the *high* level) then it must be the case that the client is obliged to choose (+) between either paying immediately (p) or delaying the payment by sending the notification ($d\&n$).

Clauses 3 and 4 refer to the clauses 7.3 and 7.4 of the contract example. They both refer to the moment after the client has delayed the payment ($[d\&n]$). Clause 3 states that the client has the obligation to lower the Internet traffic ($O(l)$) and that after lowering the client should pay twice the price. On the other hand, clause 4 specifies the obligation of the client to pay three times the price in case he does not lower the Internet traffic (\bar{l}). The two formulas may be combined in a single formula using CTDs: $\Box([d\&n](O_\varphi(l) \wedge [l]\Diamond(O(p\&p)))$ where $\varphi = O(p\&p\&p)$. Clause 5 formally represents clause 7.5 of the contract example. It represents the obligation of the client to submit the form ($O(sfD)$) after the Internet service becomes operative ($[o]$).

(1)	$f^T(O(\&_{i=1}^n a_i)) = \langle \{a_1, \dots, a_n\} \rangle (\wedge_{i=1}^n O_{a_i})$
(2)	$f^T(\mathcal{C}_O \oplus \mathcal{C}_O) = f^T(\mathcal{C}_O) \wedge f^T(\mathcal{C}_O)$
(3)	$f^T(P(\&_{i=1}^n a_i)) = \langle \{a_1, \dots, a_n\} \rangle (\wedge_{i=1}^n \neg \mathcal{F}_{a_i})$
(4)	$f^T(\mathcal{C}_P \oplus \mathcal{C}_P) = f^T(\mathcal{C}_P) \wedge f^T(\mathcal{C}_P)$
(5)	$f^T(F(\&_{i=1}^n a_i)) = [\{a_1, \dots, a_n\}] (\wedge_{i=1}^n \mathcal{F}_{a_i})$
(6)	$f^T(F(\delta) \vee [\beta]F(\delta)) = f^T(F(\delta)) \vee f^T([\beta]F(\delta))$
(7)	$f^T(\mathcal{C}_1 \wedge \mathcal{C}_2) = f^T(\mathcal{C}_1) \wedge f^T(\mathcal{C}_2)$
(8)	$f^T(\bigcirc \mathcal{C}) = [\mathbf{any}] f^T(\mathcal{C})$
(9)	$f^T(\mathcal{C}_1 \mathcal{U} \mathcal{C}_2) = \mu Z. f^T(\mathcal{C}_2) \vee (f^T(\mathcal{C}_1) \wedge [\mathbf{any}] Z \wedge \langle \mathbf{any} \rangle \top)$
(10)	$f^T(\square \mathcal{C}) = \nu Z. \mathcal{C} \wedge [\mathbf{any}] Z$
(11)	$f^T([\&_{i=1}^n a_i] \mathcal{C}) = [\{a_1, \dots, a_n\}] f^T(\mathcal{C})$
(12)	$f^T([\&_{i=1}^n a_i] \alpha \mathcal{C}) = [\{a_1, \dots, a_n\}] f^T([\alpha] \mathcal{C})$
(13)	$f^T([\alpha + \beta] \mathcal{C}) = f^T([\alpha] \mathcal{C}) \wedge f^T([\beta] \mathcal{C})$
(14)	$f^T([\varphi?] \mathcal{C}) = f^T(\varphi) \Rightarrow f^T(\mathcal{C})$

Table 1. The translation function f^T from \mathcal{CL} to $\mathcal{C}\mu$.

3.1 Translating the \mathcal{CL} specification into $\mathcal{C}\mu$

We extract a model from the \mathcal{CL} clauses by first translating the language specification into the extended μ -calculus $\mathcal{C}\mu$ where the semantics is given as a special labeled transition system. The translation function f^T which takes a \mathcal{CL} formula and returns a formula in the $\mathcal{C}\mu$ is shown in Table 1. The special syntax $[\mathbf{any}]$ (or the dual $\langle \mathbf{any} \rangle$) represents the fact that any action can be executed. To represent obligations and prohibitions of a given action a we need the special propositional constants O_a and \mathcal{F}_a . One of the features of $\mathcal{C}\mu$ is that it uses multisets to represent concurrent actions (i.e. $\langle \{p, p, p\} \rangle$ models the fact that action p is done concurrently three times which in \mathcal{CL} is represented as the concurrent action $p \& p \& p$).

We briefly mention here the semantics of $\mathcal{C}\mu$, see [2] for more details. The formulas are interpreted over a labeled transition system (LTS). The labels of the transitions are represented by multisets of actions (e.g. $\{p, p, p\}$ is a label). The formulas are interpreted over states as usual in modal logics with semantics on Kripke-like structures. For example the expression $\phi \Rightarrow \langle p \rangle O_p$ is interpreted in a state and should be understood as: if the assertion ϕ holds in the state then $\langle p \rangle O_p$ should hold in the same state. $[p] \mathcal{C}$ and $\langle p \rangle \mathcal{C}$ are interpreted as holding in the current state if and only if in the next state reachable by action p the formula corresponding to the translation of \mathcal{C} holds. In $\mathcal{C}\mu$ the difference between the two operators is that $\langle p \rangle \varphi$ requires the existence of at least one next state reachable by p where φ holds, where $[p] \varphi$ is quantified universally, and thus the formula also holds in case the set of states reachable by p is empty.

We will now translate the five \mathcal{CL} clauses corresponding to the contract given above, into $\mathcal{C}\mu$. Note that we use the \square and \diamond with their classical interpretation from temporal logics; the last not being included in the Table 1. It is known [6] that $f^T(\diamond \mathcal{C}) = f^T(\top \mathcal{U} \mathcal{C}) = \mu Z. \mathcal{C} \vee ([\mathbf{any}] Z \wedge \langle \mathbf{any} \rangle \top)$. In order to translate the first clause of the \mathcal{CL} representation above we can proceed as follows:

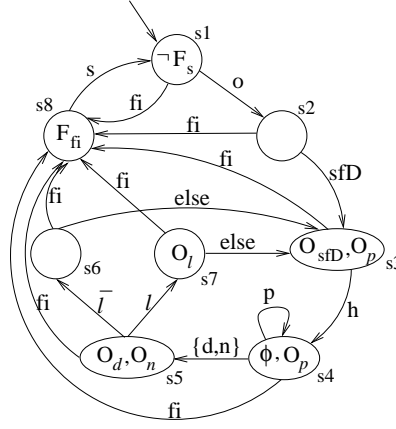


Fig. 2. Example of a model for the five clauses written in \mathcal{CL} .

$$f^T(\Box F_{P(s)}(fi)) = \nu Z. f^T(F_{P(s)}(fi)) \wedge [\mathbf{any}]Z,$$

where: $f^T(F_{P(s)}(fi)) = f^T(F(fi) \wedge [fi]P(s)) = [fi]\mathcal{F}_{fi} \wedge [fi]\langle s \rangle P_s.$

Because all the clauses are wrapped inside the \Box operator, and we have seen its translation above, for the sake of presentation we only show below the translation of the expressions inside the \Box .

1. $[fi]\mathcal{F}_{fi} \wedge [fi]\langle s \rangle P_s$
2. $[h](\phi \Rightarrow (\langle p \rangle O_p \wedge \langle \{d, n\} \rangle (O_d \wedge O_n)))$
3. $[\{d, n\}](\langle l \rangle O_l \wedge [l](\mu Z. \langle \{p, p\} \rangle O_p \vee ([\mathbf{any}]Z \wedge \langle \mathbf{any} \rangle \top)))$
4. $[\{d, n\}][l](\mu Z. \langle \{p, p, p\} \rangle O_p \vee ([\mathbf{any}]Z \wedge \langle \mathbf{any} \rangle \top))$
5. $[o]\langle sfD \rangle O_{sfD}$

In Fig. 2 we have pictured one model of the above clauses where we denote by *else* all other actions different than the ones from the current node (e.g. for the state s_7 in the picture *else* = $\mathbf{any} \setminus \{fi\}$). In the next section we will specify this model using the input language of NuSMV, and prove that it is indeed a model of the \mathcal{CL} formulas. We have to note that in this paper we are not concerned with whether we have a “minimal” model nor if we have the “most desired” model as we do not have those notions for the moment.

Note that because of the semantics of the prohibition $F(fi)$ (i.e., $[fi]\mathcal{F}_{fi}$), we would not need to explicitly add a transition from each state labeled with fi to a state with the propositional constant \mathcal{F}_{fi} . However, in the presence of a CTP, as it is the case with clause 1, we need to do so in order to represent the reparation $P(s)$.

3.2 From the $\mathcal{C}\mu$ LTS to the NuSMV input syntax

In NuSMV [7], a model can be specified in two ways: either using *assignments* or by *direct specification*. We are using the more powerful *direct specification* of

the labeled transition system which uses propositional formulas (as constraints) to represent the initial states (INIT) and the transition relation (TRANS). The disadvantage of this method over the classical *assignments* is that it can give results of logical absurdities if not used carefully. It may happen that no state is chosen as the initial state if the formula is interpreted as \perp , or the transition relation is not total (i.e. contains deadlock states). For us it is more convenient to use INIT and TRANS method as it is more flexible to our not-so-regular transition system, so we can specify exactly the states and the transitions. On the other hand we get a much smaller reachable state space (as we specify the exact transitions from one state to another). A more important motivation is that it is easy to implement our deterministic transitions: the determinism required by \mathcal{CL} refers to the fact that from one state there is only one next state reachable through a certain label. This deterministic aspect is harder to control using the *assignments* method.

NuSMV uses *state variables* to identify states; the number of states is determined by the product of the number of different values each state variable can take. There is also a second kind of variables, *input variables* which are meant to specify labels of a labeled transition system. Since we have actions as labels, we make substantial use of the input variables in our application.

We have defined an input variable for each atomic action of the \mathcal{CL} specification. The type of the input variables is `boolean` so that if the value of `d = false` then `d` is not an active label of the transition. Whenever a variable is left unspecified then NuSMV interprets it as having any value so it creates a transition (or a state in case of state variables) for each value of the variable.

In NuSMV it is easy to simulate the concurrent labels $\{d, n\}$ of $\mathcal{C}\mu$ which mean that the transition is taken if both actions d and n are executed concurrently: we activate both input variables `d = true ; n = true`. We can also represent the resource-awareness of the labels (i.e. the $p\&p$ of \mathcal{CL} , or the $\{p, p\}$ of $\mathcal{C}\mu$) by defining the input variable with the type *range of integers*. If `p = 0` then the transition is not labeled with the action p ; if `p = 1` then the transition is labeled with one normal action p (like in the case of `boolean` type); but if `p = 2` then we take the transition if two copies of the action p are executed concurrently. We have then the following declaration of variables:

```
IVAR
  d : boolean ;
  n : boolean ;
  p : 0 .. 3 ;
```

Note that we may have *empty transitions* (with no label) by giving to all the input variables the value `false` (or `p = 0`). Moreover, we may represent the special action **any** of $\mathcal{C}\mu$ by leaving all input variables unspecified.

We have defined a state variable named `state` of enumeration type so it can take only eight values, corresponding to the eight states depicted in Fig. 2.

```
VAR
  state : {s1, s2, s3, s4, s5, s6, s7, s8} ;
```

Other variables are declared accordingly (e.g., `high : boolean`). Moreover, we define a state variable of type `boolean` for each input variable. This is required by the $\mathcal{C}\mu$ where we have a propositional constant O_a or \mathcal{F}_a associated to each atomic action a which enters under the scope of an obligation or of a prohibition respectively.

```
F_s : boolean ; F_fi : boolean ;
O_p : boolean ; O_d : boolean ; O_n : boolean ;
O_l : boolean ; O_sfD : boolean ;
```

As an example, we show below the encoding of the initial state, and one of its outgoing transitions, of the automaton in Fig. 2. We call the initial state `s1`.

```
INIT
(state = s1) & !high &
!F_fi & !O_p & !O_d & !O_n & !O_l & !O_sfD & !F_s ;
```

The transitions are specified using the `TRANS` keyword followed by a propositional formula which determines the pairs of states that form the transition relation. The propositional formula contains names of state variable (which are tested in the current state) and `next` expressions which refer to the value of the state variables in the next state. It also contains the input variables to model the labels of the transitions. Remember that any variable that is missing from the formula is interpreted as having any value and will give rise to a number of different transitions equal to the number of values it can take.

```
TRANS
--state variables of the current state
((state = s1) & !high &
!F_fi & !O_p & !O_d & !O_n & !O_l & !O_sfD & !F_s &
--input variables as the labels
(!fi & p = 0 & !d & !n & !l & !negl & !sfD & o & !s) &
--the values of the state variables in the next states
(next(state) = s6) & !next(high) &
next(!F_fi & !O_p & !O_d & !O_n & !O_l & !O_sfD & !F_s))
```

3.3 Model checking the Contract

Proving that the model satisfies the original clauses: First we need to prove that the model of Fig. 2, specified in NuSMV³ respects the five $\mathcal{C}\mathcal{L}$ clauses representing the statements from the contract example. For this we have specified each clause as a special LTL specification in NuSMV and combined them using conjunction in a single big formula:

```
G ((fi -> X F_fi) & (fi -> X (s & X !F_s))) &
G (h -> X (high -> ((p = 1 -> X O_p) & ((d & n) -> X (O_d & O_n)))) &
G ((d & n) -> X ((l -> X O_l) & l -> X F (p = 2 -> X O_p))) &
G ( (d & n) -> X (l -> X F (p = 3 -> X O_p))) &
G (o -> X (sfD -> X O_sfD))
```

³ The NuSMV code we have used is available on Nordunet3 project homepage:
<http://www.ifi.uio.no/~gerardo/nordunet3/software.shtml>

We check the formula using the `check_ltlspec` command. The check proves successful.

We give below some statistical information from NuSMV about the model in the Fig. 2. Notice that though the number of actual states is rather high (2^{11}), only 8 states are reachable since we have given a precise specification of the transition function.

```
NuSMV > print_reachable_states
#####
system diameter: 6
reachable states: 8 (2^3) out of 2048 (2^11)
#####
NuSMV > check_fsm
#####
The transition relation is not total.
However, all the states without successors are
non-reachable, so the machine is deadlock-free.
#####
NuSMV >
```

The NuSMV output also shows that the transition relation is deadlock-free, and it is not total, as we would have obtained using the assignments method instead. On the other hand the fact that the transition is not total is not surprising as there are several states without successors, but these states are of no interest as they are not reachable.

Verifying a property about client obligations: The first desirable property that we want to check on the contract model is expressed in English as: “Is it the case that after the internet is high and the client pays then the client is obliged to pay again?”. The property is expressed in \mathcal{CL} as: $\Box(\phi \wedge \langle p \rangle O(p))$. For model checking purposes we negate the property, so that it reads: “Always it is not the case that after the internet is high and the client pays than the client is obliged to pay again”, and it is translated in LTL as: $G(\neg(\phi \wedge \langle p \rangle O_p))$. By distributing the negation we have: $G(\neg\phi \vee [p][p]\neg O_p)$. The property proves to be false. In the transcript below one can see the encoding of the property in NuSMV, and a counter-example.

```
NuSMV > check_ltlspec
-- specification G (!high | (p = 1 -> X (p = 1 -> X !0_p))) is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
  state = s1
-> Input: 2.2 <-
  o = 1
-> State: 2.2 <-
  state = s2
-> Input: 2.3 <-
  sfD = 1
-> State: 2.3 <-
  state = s3
  0_p = 1
  0_sfD = 1
-> Input: 2.4 <-
  h = 1
-- Loop starts here
-> State: 2.4 <-
  state = s4
```

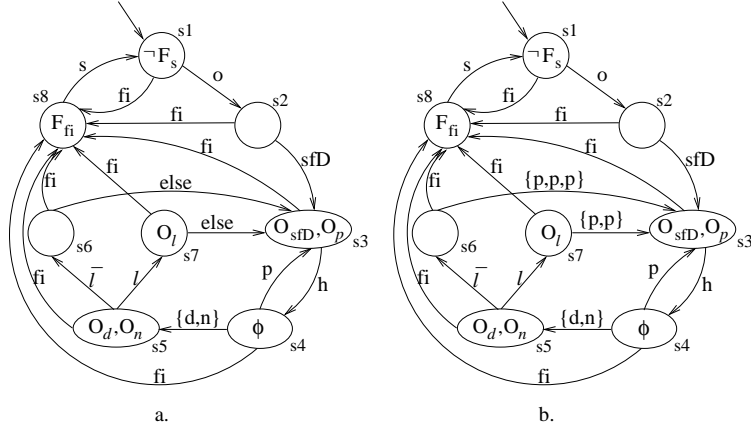


Fig. 3. The model of Fig. 2 corrected.

```

high = 1
0_sfD = 0
-> Input: 2.5 <-
p = 1
-- Loop starts here
-> State: 2.5 <-
-> Input: 2.6 <-
p = 1

```

The above counter-example shows that in state s_4 of Fig. 2 the client must fulfill one of the following obligations: or to pay (p), or to delay payment and notify (d, n). However, after paying once, the automaton is still in a state with high traffic (state s_4), and thus the client is still obliged to pay again.

We give in Fig. 3-a the new model, which is proved correct with respect to the above property. The difference is the transition $s_4 \xrightarrow{p} s_3$ which replaces the one labeled with p from s_4 to itself. From this it is easy now to modify the original contract by introducing the following clause: “The provider guarantees that if the Internet traffic of the Client reaches a high level and the Client pays the [price] then it will not be obliged to pay the [price] again”.

Notice that though we have obtained a new model that satisfies the property (and a clause in the original contract solving the above problem), the solution is still not satisfactory, as the contract does not specify what happens after the client pays but does not decrease the Internet traffic. In the new model shown in Fig. 3-a this is reflected by the fact that after taking the new added transition (from s_4 to s_3), there is an implicit assumption that the Internet traffic is low. For brevity we do not further analyze the contract in order to obtain the right contract concerning this problem, though it can be done following a similar approach as above.

Verifying a property about payment in case of increasing Internet traffic: The checking of the previous property was done for the benefit of the client. We now

perform model checking in order to increase the confidence of the provider of the service. In particular, we are interested in knowing whether “if the Internet is high and the client delays and notifies, and afterward lowers the Internet traffic, can it happen that the client does not pay twice until the internet traffic is high again”. This complicated English clause is specified in \mathcal{CL} as:

$$\Box(\phi \wedge [d\&n][l](\overline{\langle p\&p \rangle} \top \mathcal{U}\phi)).$$

Using the translation of the property into NuSMV we check it on the model of Fig. 3-a where it proves false. Though it was not apparent at first sight, and confirmed by the result given by the tool, the clauses we are analyzing allow the client to go from normal to high Internet traffic many times and pay the penalty ($2 * [price]$) only once. The problem is that after the client lowers the Internet traffic, he might get a high traffic again and postpone the payment till a future moment. This problem comes from the ambiguity of the language. Note that the \mathcal{CL} formalization in the clauses 3 and 4 use the \diamond to model the fact that a statement will hold eventually in the future but not necessarily *immediately* (expressions “pay later” in clause 7.3 and “will have to pay” in clause 7.4 are the ambiguities). The *eventually* was translated with the help of the special syntax *else* that we see in Fig. 3-a. We use the counter-example given by NuSMV to construct the model in Fig. 3-b where the property holds. The difference is at the transition from s_7 to s_3 where we have changed the label to the multiset label $\{p, p\}$. In \mathcal{CL} this solution might be expressed by changing the clause 3 above as:

$$\mathbf{3}' \quad \Box([d\&n](O(l) \wedge [l](\neg\phi \mathcal{U} O(p\&p)))).$$

In order to remedy the original contract, we should add the following clause: “after getting a high Internet traffic, if the client postpones the payment then the client can get a high traffic again only after having paid”. Note that a similar property can be stated for the clause 4 for which we have given the solution in Fig. 3-b also by replacing the label of the transition from s_6 to s_3 by the multiset label $\{p, p, p\}$.

4 Conclusion

In this paper we have shown how model checking techniques and tools can be applied to analyze contracts. In particular, we have used NuSMV [7] to model check conventional contracts specified using the language \mathcal{CL} . The main idea behind having \mathcal{CL} as a formal contract language is to use not only model checking but also logical deduction techniques, operating on the underlying semantics in $\mathcal{C}\mu$, to analyze contracts.

About NuSMV: NuSMV [7] is the successor of the milestone symbolic model checker SMV [8]. Symbolic model checking [9] is based on the clever encoding of the states in the data structure called binary decision diagram (BDD), but still relies on the classical model checking algorithm. NuSMV has had great success

in verifying hardware circuits and communication protocols, where the BDDs give a great state space minimization.

Among the features of the NuSMV are the use of a SAT-based bounded model-checking method (BMC) [10] and the possibility to check properties specified in CTL, LTL, or PSL (property specification language). More recently NuSMV has included *input variables* with which it is possible to specify not only a finite state machine (FSM) but also a labeled transition system. This feature of NuSMV has been very useful in our context, and was one of the reasons to chose NuSMV as a model checking tool. On the other hand, translations between LTS and Kripke structure (without labels) semantics are known (see [11]), but they are more involved and usually increase the state space.

Related Work: To our knowledge, model checking contracts is quite an unexplored area where only few works can be found [12,13]. The main difference with our approach is that in [12] there is no language for writing contracts, instead automata are used to model the different participants of a contract, i.e. there is no model of the contract itself but only of the behavior of the contract signatories. Many safety and liveness properties identified as common to e-contracts are then verified in a purchaser/supplier case study using SPIN [14]. Similarly, in [13] Petri nets are used to model the behavior of the participants of a contractual protocol. Though in [12] it is claimed that modeling the signatories gives modularity, adding clauses to a given contract implies modifying the automata. In our case, adding clauses to a contract is done as in any declarative language, without changing the rest. Though in our current implementation we would also need to rewrite the verification model, this should not be seen as a disadvantage; given that \mathcal{CL} has formal semantics in $\mathcal{C}\mu$ the model could be obtained automatically after the modifications. An advantage of our approach is the possibility of explicitly writing conditional obligations, permissions and prohibitions, as well as CTDs and CTPs. We are not aware of any other work on model checking e-contracts. When speaking only about formalizing contracts, discussions and other approaches besides \mathcal{CL} can be found in [2] or [12], and references therein.

Future Work: We have presented a manual translation from the $\mathcal{C}\mu$ semantics of the contract written in \mathcal{CL} into the input language of NuSMV. We plan to implement a prototyping tool taking as input a contract and a property written in \mathcal{CL} and giving as output *yes* or *no* depending on whether the property is satisfied or not. In case of a negative answer, we can benefit from the counterexample generation to rewrite the original contract, as we have briefly shown in Section 3.3. The underlying model checker of such tool could be NuSMV or another existing μ -calculus model checker (e.g., [15,16]).

With such a tool the whole model checking process will be accelerated facilitating its use and thus making it easy to prove other interesting general properties about e-contracts, as suggested in [12]. Besides such classical liveness or safety properties we are also interested in properties more specific to e-contracts, like: finding the obligations or prohibitions of one of the parties in the contract; listing of all the rights that follow after the fulfilling of an obligation; what are

the penalties for whenever violating an obligation or prohibition; determining whether a given participant is obliged to perform contradictory actions.

The generation of the (automata-like) model that we did by hand in Section 3 could be done automatically with a LTL-to-Büchi automata translator (like `ltl2smv` or `ltl2ba`) modified to comply with the syntax of \mathcal{CL} . See [17] for a comprehensive overview of the state-of-the-art of such tools.

In the current state of development, the language \mathcal{CL} cannot explicitly express timing constraints. We intend to extend the language with such features in order to be able specify and verify real-time properties.

References

1. McNamara, P.: Deontic logic. Volume 7 of Handbook of the History of Logic. North-Holland Publishing (2006) 197–289
2. Prisacariu, C., Schneider, G.: A formal language for electronic contracts. In: FMOODS'07. Volume 4468 of LNCS., Springer (2007) 174–189
3. Kozen, D.: Results on the propositional mu-calculus. *Theor. Comput. Sci.* **27** (1983) 333–354
4. Fischer, M.J., Ladner, R.E.: Propositional modal logic of programs. In: STOC'77, ACM (1977) 286–294
5. Pnueli, A.: The temporal logic of programs. In: FOCS'77, IEEE Computer Society Press (1977) 46–57
6. Bradfield, J., Stirling, C. In: Modal Logics and mu-Calculi: an Introduction. Elsevier (2001) 293–330
7. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: CAV 2002. Volume 2404 of LNCS., Springer (2002) 359–364
8. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
9. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. In: LICS'90, IEEE Computer Society (1990) 428–439
10. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: TACAS'99. Volume 1579 of LNCS. (1999) 193–207
11. De Nicola, R., Vaandrager, F.W.: Action versus state based logics for transition systems. In Guessarian, I., ed.: Semantics of Systems of Concurrent Processes. Volume 469 of LNCS., Springer (1990) 407–419
12. Solaiman, E., Molina-Jiménez, C., Shrivastava, S.K.: Model checking correctness properties of electronic contracts. In: ICSOC'03. Volume 2910 of LNCS. (2003) 303–318
13. Daskalopulu, A.: Model checking contractual protocols. In: JURIX'00. Frontiers in Artificial Intelligence and Applications Series, IOS Press (2000) 35–47
14. Holzmann, G.: The Spin Model Checker, Primer and Reference Manual. Addison-Wesley, Reading, Massachusetts (2003)
15. Biere, A.: mu-cke - efficient mu-calculus model checking. In: CAV'97. Volume 1254 of LNCS. (1997) 468–471
16. Mateescu, R., Sighireanu, M.: Efficient on-the-fly model-checking for regular alternation-free mu-calculus. *Sci. Comput. Program.* **46** (2003) 255–281
17. Rozier, K.Y., Vardi, M.Y.: LTL satisfiability checking. In Bosnacki, D., Edelkamp, S., eds.: SPIN'07. Volume 4595 of LNCS., Springer-Verlag (2007) 182–200