Accountable Protocols in Abductive Logic Programming

MARCO GAVANELLI, ENDIF, University of Ferrara
MARCO ALBERTI, DMI, University of Ferrara
EVELINA LAMMA, ENDIF, University of Ferrara

Finding the responsible of an unpleasant situation is often difficult, especially in artificial agent societies. $\mathcal{S}$CIFFis a formalization of agent societies, including a language to describe rules and protocols, and an abductive proof-procedure for compliance checking. However, how to identify the responsible for a violation is not always clear.

In this work, a definition of accountability for artificial societies is formalized in $\mathcal{S}$CIFF. Two tools are provided for the designer of interaction protocols: a guideline, in terms of syntactic features that ensure accountability of the protocol, and an algorithm (implemented in a software tool) to investigate if, for a given protocol, non-accountability issues could arise.

Additional Key Words and Phrases: SCIFF, Accountability, Abductive Logic Programming

1. INTRODUCTION

The current economic world is strongly based on large corporations, that have been able to provide large economic benefits, such as cheaper prices for everyday goods and better employment rates, but that also represented large problems in case of failure. Every person can list problems in his/her own country in which a large firm misbehaved, e.g., polluting the environment, or by failing in a disastrous way causing huge losses for small investors. In many cases, the firm itself cannot be punished for its misbehavior, because a company cannot be sent to jail, and because fines are not always a good deterrent; consider, e.g., that fines cannot be too high, because the company might be forced to fail and shutdown, causing unemployment, and if they are too low they are not a deterrent [Bovens 1998]. One hopes that the culprit of the misbehavior (e.g., the CEO) is punished, but in many cases the complex behavior of an organization depends on the policies established by previous members (that might even be dead), by common practices, or by the fact that many individuals contributed to the disaster each in an inconceivably small amount. This was called *"the problem of many hands"* by Bovens [1998], and the usual result is that nobody is punished.

In the literature of moral responsibility, there exist different notions of responsibility. Van de Poel et al. [2015] distinguish five moral meanings of responsibility: accountability, blameworthiness, liability, obligation and virtue.

> The ascription of responsibility-as-accountability has the following implication: i is responsible-as-accountable for φ implies that i should account for (the occurrence of) φ, in particular for i's role in doing, or bringing about φ, or for i's role in failing to prevent φ from happening, where i is some agent, and φ an action or a state-of-affairs.

and

> Accountability implies blameworthiness unless the accountable agent can show that a reasonable excuse applies that frees her from blameworthiness. So holding an agent i accountable shifts the burden of proof for showing that i is not blameworthy to the agent i: the agent is now to show – by giving an account – that she is not blameworthy.
>
> An agent i is accountable for φ if i has the capacity to act responsibly (has moral agency), is somehow causally connected to the outcome φ (by an action or omission) and there is a reasonable suspicion that agent i did somehow

do something wrong. Agent i may then provide an account that she is not blameworthy.

Chopra and Singh [2014] point out that autonomy and accountability are fundamental concepts in socio-technical systems: agents are autonomous in that they can decide how to behave, but they are also accountable to other agents, in that they need to be able to explain their actions. The authors observe that current methods for requirement engineering lack tools to express that an agent in a socio-technical system is accountable to another, and introduce the notion of accountability requirement for this purpose. This has been later expanded in [Kafalı et al. 2016] where the authors adopt a formal computational representation of accountable norms and requirements.

Baldoni et al. [2016], too, base their work on computational accountability on the concept of accountability as one's obligation to explain their action when requested to, which does not hinder the agent's autonomy.

Clearly, in a world populated not only by human beings, but also by artificial agents, that can perform transactions of money or drive cars, the issues become even more important. The problem has been the object of a multi-disciplinary effort [Boissier et al. 2013]. Several frameworks for specification of agent interaction [Castelfranchi 1995; Singh 1998; Fornara and Colombetti 2003] are based on the notion of commitment, which formalizes an obligation of an agent (the debtor) towards another agent (the creditor), thus providing a built-in notion of responsibility: in case the obligation is not fulfilled, the debtor is clearly responsible.

In this work, we focus on the $\mathcal{S}$CIFF system [Alberti et al. 2008], a complete system for defining and checking the compliance of agents to interaction protocols. It includes a language to define agent interaction protocols and to relate a current state of affairs with one or more expected behaviors of the agents, formalized as a set of expectations. The language was designed to leave freedom to agents, not overconstraining them to follow statically pre-defined paths, but, instead, to assert explicitly the obligatory actions and those that are forbidden, while leaving everything not explicitly stated as a possible action that an agent can perform if it is convenient. An abductive proof-procedure accepts asynchronous events and reasons about them through the protocol definition, generates the expected behavior of the agents, and checks if the actual behavior matches with the expectations. Beside interaction protocols, $\mathcal{S}$CIFF was later used to model Business Processes [Chesani et al. 2011], medical protocols [Ciampolini et al. 2005], careflows [Chesani et al. 2007] and legal reasoning [Alberti et al. 2006a].

Although the $\mathcal{S}$CIFF language and features are sufficiently expressive to develop these applications, $\mathcal{S}$CIFF lacks a concept of responsibility, because expectations, unlike commitments, are not characterized by a debtor, due to different language design objectives (as is discussed briefly in Section 6 and more extensively in [Torroni et al. 2009]): while $\mathcal{S}$CIFF is able to detect violations of the protocols, it is not always clear which agent is responsible for the wrong state of affairs. In other words, the $\mathcal{S}$CIFF way to specify and verify interaction fully supports agent autonomy, but does not guarantee agent responsibility.

In this work, we address the problem by adopting the accountability version of responsibility. Accountability in the proposed setting stands for the possibility to account for the wrong state of affairs of an agent that is the one that performed (or did not perform) the action in its expected behavior. The agent might then, by reasoning on the protocol and the state of affairs (again, using the $\mathcal{S}$CIFF proof procedure), be able to *account for* its own behavior. The agent might be able to find an explanation in which its expected behavior is fulfilled, and in such a case it cannot be held responsible for the violation. In some cases, this might happen because another agent is actually responsible for the violation, but in other cases it might be due to a wrong design of the interaction protocol.

For this reason, we define formally a notion of *accountability* of the interaction protocol (Section 3). The idea is that a protocol is accountable if it allows to identify the agent (or agents) responsible for each possible violation. If the interactions in an organization or society are ruled by an accountable protocol, then, for each possible undesirable state of affairs, one or more agents will be unambiguously held responsible.

The formal definition allows us to provide guidelines and tools for the developer of interaction protocols. The guidelines are syntactic conditions that ensure a-priori the accountability of a protocol. We identify a syntactic characterization of a fairly large class of protocols and prove that protocols in such class are accountable. Various protocols taken from the list of $\mathcal{S}$CIFF applications belong to this class.

However, even protocols that do not belong to the identified class may be accountable. For existing protocols, the user might not want to completely re-design the protocol, and in this case a tool that checks the protocol for accountability might be more suitable. For this purpose, we propose (in Section 5) a tool to detect if a protocol has non-accountability issues. If there exists such an issue, the tool also provides a counterexample: a course of events with protocol violation, but for which no agent can be held responsible. We tested, through such tool, protocols modeled in the past with $\mathcal{S}$CIFF, and we were able to identify non-accountability of two protocols that were completely reasonable for the task for which they were designed. Thanks to the tool and the provided counterexample, it was then easy for the designer to fix the protocol.

Thanks to the available automatic translations from popoular graphical notations, such as ConDec [Pesic and van der Aalst 2006], to $\mathcal{S}$CIFF [Montali 2010], even designers not familiar with $\mathcal{S}$CIFF can take advantage of the presented tools to assess the accountability of their specifications.

2. PRELIMINARIES: THE $\mathcal{S}$CIFF LANGUAGE

The $\mathcal{S}$CIFF language was devised to formalize interaction protocols amongst agents. The objective was to detect possible violations of the protocols, without over-constraining the agents to follow predefined paths. $\mathcal{S}$CIFF uses an abductive semantics to compute, given the actual behavior of the agents, their expected behavior; in case the two match, the current set of events (also called *history*) is compliant.

Like most logic languages, the $\mathcal{S}$CIFF language is built upon a set of a set of function symbols, a set of predicate symbols, and a set of variables. A term is a variable or a function symbol applied to zero or more terms. An atom is a predicate symbol applied to zero or more terms. A literal is an atom or the negation of an atom. A term or an atom are ground if they contain no variables.

In $\mathcal{S}$CIFF the set of events is provided in a (possibly, dynamically growing) set $\mathbf{HAP}$ (the current history). Events are represented as atoms $\mathbf{H}(X, T)$, where X is a term that describes the event and T is the time (numeric) in which the event happened. In the $\mathcal{S}$CIFF proof procedure, the $\mathbf{H}$ events are ground, as it is assumed that full knowledge of the happened events is available.

For example, $\mathbf{H}(do(alice, bob, ask), 1)$ could mean that agent $alice$ sent a request to agent bob at time 1.

The expected behavior of the agents is described with formulae involving the atoms $\mathbf{E}(X, T)$, meaning that it is expected that X happens at time T, $\mathbf{EN}(X, T)$, meaning that X is expected *not* to happen at time T, and their explicit negations $\neg\mathbf{E}(X, T)$ and $\neg\mathbf{EN}(X, T)$, meaning that such expectations cannot be formulated. Both X and T can be variables, possibly subject to Constraint Logic Programming (CLP) constraints [Jaffar and Maher 1994].

For example, $\mathbf{E}(do(bob, alice, reply), T), 2 \leq T \leq 5$ could mean that bob is expected to reply to $alice$ at any time between 2 and 5. Variables in positive expectations (**E** literals) are existentially quantified, while variables that occur only in **EN** literals are universally quantified. These quantification rules give the natural reading to $\mathbf{EN}(do(A, alice, reply), T), A \neq bob$ that no agent A (except bob) is expected to reply to $alice$, at any time T.

To simplify the notation, we will omit the time parameter when not necessary.

An interaction protocol can be defined through an Abductive Logic Programming (ALP) program [Kakas et al. 1993]. ALP extends logic programming for *abductive reasoning*, i.e. for reasoning from effects to (possible) causes. It is the inference typically used in diagnosis: given a known effect and a set of implications $\mathit{effect} \leftarrow \mathit{causes}$ it tries to hypothesize the possible causes. For example, a physician might have a knowledge base saying that sneezing can be caused by flu or hay fever:

$$\begin{array}{l} sneeze \leftarrow flu. \\ sneeze \leftarrow hay\_fever. \end{array}$$

If the doctor observes a patient sneezing, he might abduce that the patient has a flu, or a hay fever. Formally, an ALP program is a triple $\langle KB, \mathcal{A}, IC \rangle$. The Knowledge Base KB is a set of Horn clauses of the form $H \leftarrow L_1, \ldots, L_n$, whose intuitive meaning is that the atom H (the *head*) is true if all the literals $L_1, \ldots, L_n$ (the *body*) are true. The set of Horn clauses with an atom H as head serve as its definition, because they list all the possible circumstances that make it true. A literal in the body can be a special atom, called an *abducible*, from the set $\mathcal{A}$, which is not defined in the KB, but can be *assumed*. In order to constrain the set of atoms that can be assumed, the user can define a set IC of logic formulae called Integrity Constraints that must be satisfied. In our example, $\mathcal{A} = \{flu, hay\_fever\}$; the physician might have an integrity constraint saying that one cannot have flu during the summer:

$$flu, summer \rightarrow false$$

and the knowledge base contains the fact $summer$. In such a case, the hypothesis flu is ruled out, and the only left hypothesis is hay_fever.

In $\mathcal{S}$CIFF, Integrity Constraints are in the form of implications $body \rightarrow head$ that can contain all the types of literals hereby described (literals of predicates defined in the KB, abducible literals, amongst which we find **E** and **EN** literals, and CLP constraints). Literals built on **H** atoms can occur only in the body (the precondition). The body can be a conjunction of literals, while the head can be a disjunction of conjunctions of literals (for details, and other syntactic restrictions, see [Alberti et al. 2008]). For example, the Integrity Constraint (IC):

$$\mathbf{H}(do(X, Y, ask), T_a) \rightarrow \mathbf{E}(do(Y, X, yes), T_r) \vee \mathbf{E}(do(Y, X, no), T_r) \qquad (1)$$

means that in case an agent X asks a question to agent Y, then agent Y is supposed to reply either yes or no.

Given an ALP program and a goal, the task is usually to find a set of abducibles (an *abductive answer*) that satisfy the integrity constraints and that, together with the knowledge base, ensure the truth of the goal. More formally, we specialize the classical definition [Kakas et al. 1993] for the $\mathcal{S}$CIFF language as follows.

Definition 2.1 (*Abductive answer, adapted from [Kakas et al. 1993]*). Given a goal G and a history $\mathbf{HAP}$, a set $\Delta \subseteq \mathcal{A}$ is an *Abductive Answer* if it entails the goal together with KB, while entailing the integrity constraints together with KB and the history:

$$KB \cup \Delta \models G \tag{2}$$
$$KB \cup \mathbf{HAP} \cup \Delta \models IC. \tag{3}$$

Unless it is stated otherwise, we will assume that the goal G is empty (the *true* goal).

We define a *protocol* as a pair $\mathcal{P} = \langle KB, IC \rangle$. When we describe a protocol only by IC, we will mean that KB is empty.

As already explained, in $\mathcal{S}$CIFF there are abducibles with a special meaning, namely *expectations*. The formal meaning of expectations is considered when dealing with *$\mathcal{S}$CIFF answers*.

Definition 2.2 (*$\mathcal{S}$CIFF answer, adapted from [Alberti et al. 2008]*). An abductive answer is a *$\mathcal{S}$CIFF answer* if it also satisfies the conditions in the following equations (4-8).

The set of abduced expectations should be consistent with respect to explicit negation (i.e., an expectation and its explicit negation cannot belong to the same $\mathcal{S}$CIFF answer):

$$KB \cup \mathbf{HAP} \cup \Delta \models \mathbf{E}(X) \wedge \neg\mathbf{E}(X) \rightarrow \mathit{false} \tag{4}$$
$$KB \cup \mathbf{HAP} \cup \Delta \models \mathbf{EN}(X) \wedge \neg\mathbf{EN}(X) \rightarrow \mathit{false} \tag{5}$$

positive and negative expectations cannot be contradicting (i.e., the same event cannot be expected to happen and not to happen in the same $\mathcal{S}$CIFF answer):

$$KB \cup \mathbf{HAP} \cup \Delta \models \mathbf{E}(X) \wedge \mathbf{EN}(X) \rightarrow \mathit{false} \tag{6}$$

and the set of expectations should be fulfilled by the actual history (i.e., each expectation in the $\mathcal{S}$CIFF answer should match with an event in the history):

$$KB \cup \mathbf{HAP} \cup \Delta \models \mathbf{E}(X) \rightarrow \mathbf{H}(X) \tag{7}$$
$$KB \cup \mathbf{HAP} \cup \Delta \models \mathbf{EN}(X) \wedge \mathbf{H}(X) \rightarrow \mathit{false} \tag{8}$$

If there exists a set Δ of expectations satisfying all conditions 2-8 we write

$$\mathcal{P}_{\mathbf{HAP}} \models G \tag{9}$$

(otherwise we write $\mathcal{P}_{\mathbf{HAP}} \not\models G$ to mean that such a set Δ does not exist).

In this article, we also need to reason about sets of expectations that are not fulfilled, so we introduce the concept of *violation answer*. Intuitively, a *violation answer* is a set of expectations that respects all the conditions required for a $\mathcal{S}$CIFF answer, except for the fact that some of its elements (which we call the *violated set*) are not fulfilled by the history.

Definition 2.3 (*$\mathcal{S}$CIFF Violation Answer*).

A violation answer Δ is an abductive answer that satisfies conditions (4), (5), (6), and that does not satisfy at least one of (7) and (8). The *violated set* is

$$\Delta^v = \{e \in \Delta | e \text{ does not respect at least one of (7) and (8)}\}.$$

In such a case, we write

$$\mathcal{P} \cup \mathbf{HAP} \cup \Delta \models^v G.$$

If there exists such set Δ we write

$$\mathcal{P}_{\mathbf{HAP}} \models^v G. \tag{10}$$

2.1. The $\mathcal{S}$CIFF proof-procedure

The $\mathcal{S}$CIFF proof-procedure computes the $\mathcal{S}$CIFF answers for a $\mathcal{S}$CIFF protocol. It operates by building a proof tree, starting from an initial node (characterized by an empty set of expectations) and applying a set of transitions; the set of expectations in success leaf nodes are $\mathcal{S}$CIFF answers [Alberti et al. 2008] for the given protocol. The core transitions are inherited from $\mathcal{S}$CIFF's predecessor, the IFF proof-procedure [Fung and Kowalski 1997]:

— unfolding: unfolds the clauses of a predicate defined in the KB
— propagation: given an implication $(a(\overline{X}) \wedge body) \rightarrow head$ and an atom $a(\overline{Y})$, produces the new implication $(\overline{X} = \overline{Y} \wedge body) \rightarrow head$
— splitting: deals with disjunctions. Given a formula $(a \vee b)$ opens two branches: one in which a is true and one in which b is true
— case analysis: Given an implication $(\overline{X} = \overline{Y} \wedge body) \rightarrow head$ (e.g., obtained through transition *propagation*) opens two branches: one with $\overline{X} \neq \overline{Y}$ and one in which $\overline{X} = \overline{Y}$ and $body \rightarrow head$
— equivalence rewriting: deals with unification
— logical equivalence: rewrites various logical equivalences, e.g., $false \wedge A$ is rewritten as $false$ etc.

$\mathcal{S}$CIFF inherits from the CLP operational semantics [Jaffar and Maher 1994] the transitions that deal with constraint propagation. In particular, the dis-unification constraint $(\overline{X} \neq \overline{Y})$ is considered as a CLP constraint.

Other transitions deal with dynamically-growing histories:

— happening: accepts a new event
— closure: terminates the acquisition of new events: the history is declared closed
— non-happening: after the closure, propagates ICs containing $\neg\mathbf{H}$ literals

Finally, some transitions deal with the fulfillment of expectations. For example, if a node contains an expectation $\mathbf{E}(X)$ and an event $\mathbf{H}(Y)$ is in the history, two children nodes are generated:

— one in which the unification $X = Y$ is imposed and the expectation $\mathbf{E}(X)$ is declared fulfilled
— the other in which the dis-unification constraint $X \neq Y$ is imposed.

When no such transitions can be applied to a node, if an $\mathbf{E}(X)$ expectation is not fulfilled, it is marked as *violated*.

A detailed description of all the transitions, together with proofs of soundness, completeness and termination, can be found in [Alberti et al. 2008].

3. ACCOUNTABILITY IN $\mathcal{S}$CIFF

Intuitively, an agent in an interaction is accountable for some undesirable state of affairs if the state of affairs is a consequence of an action (or lack of one) by the agent.

In the context of $\mathcal{S}$CIFF protocols, an undesirable state of affairs is a history $\mathbf{HAP}$ that does not satisfy condition (9). If a violation answer (Def. 2.3) exists, this means that the protocol would have been satisfied if the agents had satisfied the violated expectations regarding them; in this sense, those agents are accountable for the failure.

Definition 3.1 (*Accountable agent*). An agent A is *Accountable* if there exists no $\mathcal{S}$CIFF Answer, and there exists a Violation Answer Δ^v such that[1] $\mathbf{E}(do(A, \_, \_)) \in \Delta^v$ or $\mathbf{EN}(do(A, \_, \_)) \in \Delta^v$.

[1]We use here the underscore as an unnamed variable, as usual in Prolog.

Example 3.2. Consider the protocol

$$\begin{array}{rl}\mathbf{H}(do(alice, bob, a_1)) \rightarrow & \mathbf{E}(do(bob, alice, a_2)) \wedge \mathbf{EN}(do(Z, bob, forbid)) \\ \vee & \mathbf{E}(do(bob, alice, a_3))\end{array} \tag{11}$$

with the history

$$\mathbf{HAP} = \{\mathbf{H}(do(alice, bob, a_1)), \mathbf{H}(do(john, bob, forbid))\}.$$

Clearly, there is no $\mathcal{S}$CIFF answer, because if we consider the first disjunct in the head of IC (11), both expectations are violated (the expectation $\mathbf{EN}(do(Z, bob, forbid))$ is violated because of the event $\mathbf{H}(do(john, bob, forbid))$, while $\mathbf{E}(do(bob, alice, a_2))$ is violated because *bob* did not send message a_2 to *alice*), while if we consider the second disjunct, the expectation $\mathbf{E}(do(bob, alice, a_3))$ has no matching event. We have, instead, the following violation answers:

$$\begin{array}{rcl}\Delta_1^v & = & \{\mathbf{E}(do(bob, alice, a_2)), \mathbf{EN}(do(john, bob, forbid))\} \\ \Delta_2^v & = & \{\mathbf{E}(do(bob, alice, a_3))\}\end{array}$$

In such a case, agents *bob* and *john* are both accountable. However, the positions of the two agents with respect to the society enforcing the protocols are not the same. Agent *john* is able to show that there exists a violation answer in which no expectation on his own behavior is violated, namely Δ_2^v: if *bob* had replied a_3 (as he was supposed to do), there would have been a $\mathcal{S}$CIFF answer, i.e., no violation.

Agent *bob*, instead, cannot provide such an explanation, since in both violation answers there are expectations on *bob*'s behavior that are violated.

One might object that *bob* could try to blame *alice* for sending message a_1: had *alice* avoided sending such message, there would have been no violation. This would be the case if $\mathcal{S}$CIFF did not have a concept of *expected behavior*, and the rules of the protocol were given only in terms of the history. The concept of expected behavior has been very important for legal reasoning applications [Alberti et al. 2006a] in which there exist concepts of obligations, permissions, forbidden actions. The $\mathcal{S}$CIFF language lets the protocol designer define in fine grain such concepts, and relate the actual and expected behavior of agents. If the designer wanted *alice* to be considered liable, (s)he would have written the protocol differently, for example

$$\begin{array}{rl}true \rightarrow & \mathbf{EN}(do(alice, bob, a_1)) \\ \vee & \mathbf{E}(do(bob, alice, a_2)) \wedge \mathbf{EN}(do(Z, bob, forbid)) \\ \vee & \mathbf{E}(do(bob, alice, a_3))\end{array}$$

The previous example hinted also how an agent can exculpate himself: by providing as explanation a violation answer in which the accounted agent is not responsible for any violated expectation.

Definition 3.3. If there exists no $\mathcal{S}$CIFF answer for a given protocol, an accounted agent A can account for its behavior if there exists a violation answer Δ^v (called ***Reply** R to an account*) such that $\nexists \mathbf{E}(do(A, \_, \_)) \in \Delta^v$ and $\nexists \mathbf{EN}(do(A, \_, \_)) \in \Delta^v$.

If there exists no reply to an account, the accounted agent A is *indicted*.

Clearly, an agent A is indicted if in all violation answers there is a violated expectation about the behavior of A.

The previous example shows that the $\mathcal{S}$CIFF semantics in general admits more than one violation answer, which in general can contain violated expectations regarding different agents; in this case, which agents are accountable would depend on which violation answer is selected to explain the violation of the protocol. Thus, for an agent to be indicted for a violation state, it is required that the agent be involved in at least one violated expectation in each violation answer. This is the rationale behind the following definition.

Definition 3.4 (*Set of indicted agents*).
Given a protocol $\mathcal{P}$ and a history **HAP**, if $\mathcal{P}_{\mathbf{HAP}} \not\models G$, the set

$$S = \bigcap_{\Delta|(\mathcal{P}\cup\mathbf{HAP}\cup\Delta\models^v G)} \{X| \begin{array}{l} \mathbf{E}(do(X,Y,Action),T) \in \Delta^v \\ \vee\mathbf{EN}(do(X,Y,Action),T) \in \Delta^v \end{array} \}$$

is called the *Set of indicted agents*.

Example 3.5 (*Example 3.2 continued*). Consider again IC (11) with the following history:

$$\mathbf{HAP} = \{\mathbf{H}(do(alice, bob, a_1)), \mathbf{H}(do(john, bob, forbid)), \mathbf{H}(do(bob, alice, a_2))\}.$$

In this case, there are two violation answers, with the following violation sets:

$$\begin{array}{ll} \Delta_1 = \{\mathbf{E}(do(bob, alice, a_2)), \mathbf{EN}(do(john, bob, forbid))\} & \Delta_1^v = \{\mathbf{EN}(do(john, bob, forbid))\} \\ \Delta_2 = \{\mathbf{EN}(do(john, bob, forbid))\} & \Delta_2^v = \{\mathbf{E}(do(bob, alice, a_3))\} \end{array}$$

In such a situation, $john$ could provide as explanation for his behavior that he intended as correct the answer Δ_2, while bob would choose answer Δ_1.
Each agent is able to provide a reasonable explanation of his behavior, while there is a violation of the protocol. This situation clearly highlights a deficiency of the protocol, that has not been designed in a reasonable way.

We can now provide a definition of accountable protocol: intuitively, a protocol can be characterized as accountable if it is possible to find a set of accountable agents for each possible history that violates the protocol.

Definition 3.6 (*Accountable protocol*).
A protocol $\mathcal{P}$ is accountable if for each history **HAP** such that $\mathcal{P}_{\mathbf{HAP}} \not\models G$ the set of indicted agents is not empty.

When the sets of agents responsible for violations in different violation answers are disjoint, it is not possible to identify a set of indicted agents: in this case, the protocol is not accountable.

4. SYNTACTIC FEATURES THAT ENSURE ACCOUNTABILITY

The concept of accountability that was just formally defined can have various practical uses. One is giving some guidelines on how to define a protocol in such a way that it is known to be accountable. We will provide a syntactic condition that is sufficient to prove accountability; it is interesting to note that the majority of the protocols developed in the past for $\mathcal{S}$CIFF satisfy such syntactic condition.
Before introducing such condition, we show some examples of protocols that are accountable and of some that are not.

Example 4.1 (*Same answer*).

$$\begin{array}{rll} \mathbf{H}(do(alice, bob, ask)) \rightarrow & \mathbf{E}(do(bob, alice, yes)) & \wedge\ \mathbf{E}(do(john, alice, yes)) \\ \vee & \mathbf{E}(do(bob, alice, no)) & \wedge\ \mathbf{E}(do(john, alice, no)) \end{array} \quad (12)$$

In this protocol, two agents should give the same reply. The protocol is clearly not accountable, because in case the replies are different, both bob and $john$ will blame the other agent for not giving the right reply.
One could think that the non-accountability derives from the disjunction in the head of the integrity constraint (12): since there is an explicit choice, represented by the $\vee$

symbol, each agent might choose a different branch. Note, however, that a similar effect can be obtained rewriting the integrity constraint as

$$\mathbf{H}(do(alice, bob, ask)) \rightarrow \mathbf{E}(do(bob, alice, Reply)) \wedge \mathbf{E}(do(john, alice, Reply)).$$

In this second version, the disjunction is not shown explicitly, but, in a history in which the two agents give different replies, the $\mathcal{S}$CIFF proof-procedure will correctly explore two branches: one in which the variable $Reply$ is unified with the actual reply of agent bob (and in which the expectation on $john$ is violated), and one in which it is unified with the answer of $john$ (and in which, symmetrically, it is the expectation on bob's behavior that is violated).

Also, one may think that the non-accountability depends on the fact that there are two expectations in conjunction in the head; however the same protocol could be written with two integrity constraints:

$$\begin{array}{rcl} \mathbf{H}(do(alice, bob, ask)) & \rightarrow & \mathbf{E}(do(bob, alice, Reply)). \\ \mathbf{E}(do(bob, alice, Reply)) & \rightarrow & \mathbf{E}(do(john, alice, Reply)). \end{array}$$

Example 4.2. Consider the protocol

$$\mathbf{H}(do(alice, bob, a_1)) \rightarrow \mathbf{E}(do(bob, \_, b_1)) \vee \mathbf{E}(do(bob, \_, b_2)).$$

Such protocol is accountable.

PROOF. In the protocol, the only agent for which there are expectations is bob. If he does one of the actions b_1 or b_2, then there exists a set of expectations of success. Otherwise, in both branches he is the indicted, so $S = \{bob\}$. $\square$

Since such protocol is accountable, one wonders if a more complex protocol consisting only of integrity constraints with the same structure is necessarily accountable.

Definition 4.3. A protocol is *consequential* if it consists of a set of integrity constraints of the form

$$\begin{array}{lcl} [\neg]\, \mathbf{H}(do(X^b_1, Y^b_1, A^b_1), T^b_1) \ \wedge & & \\ [\neg]\, \mathbf{H}(do(X^b_2, Y^b_2, A^b_2), T^b_2) \ \wedge & & \\ \quad\vdots & & \\ [\neg]\, \mathbf{H}(do(X^b_n, Y^b_n, A^b_n), T^b_n) \ \wedge & & \\ c^b_1 \wedge c^b_2 \wedge \ldots \wedge c^b_p & \rightarrow & [\neg]\, \mathbf{E}/\mathbf{EN}(do(C, Y^h_1, A^h_1), T^h_1) \wedge/\vee \\ & & [\neg]\, \mathbf{E}/\mathbf{EN}(do(C, Y^h_2, A^h_2), T^h_2) \wedge/\vee \\ & & \quad\vdots \\ & & [\neg]\, \mathbf{E}/\mathbf{EN}(do(C, Y^h_m, A^h_m), T^h_m) \\ & & c^h_1 \wedge c^h_2 \wedge \ldots \wedge c^h_q \end{array}$$

where C occurs in at least one $\mathbf{H}$ atom in the body of the integrity constraint, and where c^h_i and c^b_i can be either CLP constraints or predicates that are defined in the KB and that do not depend on abducible predicates. Moreover, we require that also the goal G follows the same restrictions of the head of integrity constraints, and in particular can contain only expectations on a single agent.

Intuitively, in a consequential protocol, the precondition of each integrity constraint depends only on the current history. It can generate an arbitrary set of expectations in which there is only one agent C that is responsible to bring about the expected behavior. C occurs in an $\mathbf{H}$ atom, which is a positive literal and which can unify only with $\mathbf{H}$ atoms in the history $\mathbf{HAP}$, that contains ground atoms, meaning that when the body of a such IC is true, C is always ground.

THEOREM 4.4. *A consequential protocol is accountable.*

Before the proof, we give the definition of subset-minimality, that will be used in the proof.

Definition 4.5. Subset-minimality. Let Δ be a Violation Answer; Δ is subset-minimal if there exists no other Violation Answer Δ' such that $\Delta' \subset \Delta$.

PROOF. of Theorem 4.4. Suppose that, with a given consequential protocol and a history, there is no $\mathcal{S}$CIFF answer. Consider a Violation Answer Δ; without loss of generality, we can consider a subset-minimal set (if it is not subset-minimal, there exists a set $\Delta' \subset \Delta$; consider in this case the set Δ'). Δ is clearly not empty and contains a violated expectation (otherwise there would exist a $\mathcal{S}$CIFF answer).

We prove now that there exists at least an IC whose body is true. Let $e \in \Delta$ be a violated expectation $\mathbf{E}(do(X, Y, A), T)$ (or $\mathbf{EN}(do(X, Y, A), T)$). Clearly, e belongs to the head of some IC whose body is true, indeed, if e is not in the head of any IC whose body is true, then e could be removed from Δ, obtaining a smaller Violation Answer.

Consider an IC whose body is true. Since Δ is an abductive answer, its head must also be true. If all the disjuncts in the head contain a violated expectation, then, by the definition of consequential protocol, all such expectations have the same indicted agent, which means that the set S is not empty and the thesis is proven.

Otherwise, suppose that all ICs whose body is true have a true disjunct in the head without violations; in such a case, we have to prove that, in all possible combinations of disjuncts selected in the various ICs, there is always at least one indicted. In such a case, one may select in each IC the disjunct without violations, and collect all the expectations in the selected disjuncts in a set $\mathbf{A}$. If $\mathbf{A}$ is consistent, then it is also a $\mathcal{S}$CIFF answer (contradicting a previous assumption). If it is inconsistent, then there must be two expectations (possibly, taken from disjuncts of different ICs), that violate one of the equations (4), (5), (6). Without loss of generality, let us assume that the two clashing expectations are taken from the first disjunct of Integrity Constraint IC_i and the first disjunct of IC_j (the extension to more than two ICs is straightforward). In this case, the content of the two expectations must unify, meaning that the agent that should perform the two violated actions is ground and is the same agent. By the definition of consequential protocol, all the expectations in the heads of IC_i and IC_j refer to actions in the expected behavior of the same agent, meaning that, whichever disjuncts are selected in the heads of the two ICs, the set of indicted agents always contains such agent. $\square$

It is worth noting that even if a protocol is not consequential, it can still be accountable, as shown in the following Example 4.6.

Example 4.6. The protocol consisting only of IC (13) is not consequential, nor accountable:

$$\begin{array}{rl} \mathbf{H}(do(alice, bob, a_1)) \;\rightarrow & \mathbf{E}(do(bob, alice, b_1)) \\ \vee & \mathbf{E}(do(john, alice, b_1)) \end{array} \tag{13}$$

It can be interpreted as $alice$ being able to command to one agent amongst bob and $john$ to perform task b_1. On the other hand, the actual agent performing b_1 is not specified; in a sense it is a choice of $john$ and bob, that might not agree on whom is going to perform task b_1. The history $\mathbf{HAP} = \{\mathbf{H}(do(alice, bob, a_1))\}$, in fact, does not have $\mathcal{S}$CIFF answers, and the two violation answers $\Delta^v_1 = \{\mathbf{E}(do(bob, alice, b_1))\}$ $\Delta^v_2 = \{\mathbf{E}(do(john, alice, b_1))\}$ do not have an indicted in common.

The protocol can be made accountable, for example, by adding that one of the two agents cannot do the task, as in

$$\mathbf{H}(do(alice, bob, a_1)) \;\rightarrow\; \mathbf{EN}(do(bob, alice, b_1))$$

The amendment makes the protocol accountable, in fact now Δ_1^v is no longer a violation answer, since it does not satisfy rule (6), so the only indicted is $john$, as the intuition suggests. Note that IC (13) has not been changed, so the new version is still not consequential.

Table I. Consequentiality and accountablity of published protocols

Protocol name	Reference	Consequential	Accountable
Comminunication primitive semantics (assertives, commissives, directives, proposals)	[Alberti et al. 2003]	Yes	Yes
Negotiation (stages 1&2) [Sadri et al. 2003]	[Alberti et al. 2004]	Yes	Yes
FIPA Query-Ref	[Alberti et al. 2006a]	Yes	Yes
E-commerce choreography	[Alberti et al. 2006a]	Yes	Yes
First-Price Sealed Bid Auction	[Alberti et al. 2006b]	No	No
English Auction	[Alberti et al. 2006b]	No	No
Needham-Schroeder	[Alberti et al. 2006c]	Yes	Yes
NetBill	[Alberti et al. 2006c]	Yes	Yes
Software license agreement	[Alberti et al. 2006b]	Yes	Yes
E-commerce protocol	[Alberti et al. 2006]	Yes	Yes
E-commerce protocol	[Alberti et al. 2007]	Yes	Yes

Table I shows that many interaction protocols expressed in the $\mathcal{S}$CIFF language and published over the years are, in fact, consequential and thus accountable, although they were not originally designed for consequentiality. An example of non-consequential (and non-accountable) protocol, published in [Alberti et al. 2006b], is discussed in Example 5.2.

5. CHECKING ACCOUNTABILITY

The class of consequential protocols is fairly large; widely known protocols, such as the Needham-Schroeder [Needham and Schroeder 1978] and the NetBill [Cox et al. 1995] were formalized in $\mathcal{S}$CIFF [Alberti et al. 2006c], and both satisfy the conditions of a consequential protocol.

On the other hand, some protocols do not satisfy Definition 4.3; in such a case the accountability of the protocol is not guaranteed. Some protocols are accountable and not consequential; one example was presented in Example 4.6, another is given in Example 5.1:

Example 5.1. $alice$ assigns a task to either bob or $john$, as in Example 4.6:

$$\begin{aligned}\mathbf{H}(do(alice, bob, a_1)) \;\rightarrow\;& \mathbf{E}(do(bob, alice, b_1)) \\ \vee\;& \mathbf{E}(do(john, alice, b_1))\end{aligned}$$

Assume that bob is more important than $john$, and he is entitled to delegate the task to $john$:

$$\begin{aligned}\mathbf{H}(do(alice, bob, a_1)) \;\rightarrow\;& \mathbf{E}(do(bob, alice, b_1)) \qquad (14)\\ \vee\;& \mathbf{E}(do(bob, john, delegate))\\ \mathbf{H}(do(bob, john, delegate)) \;\rightarrow\;& \mathbf{E}(do(john, alice, b_1)) \qquad (15)\end{aligned}$$

The protocol is clearly not consequential, because the first IC contains two alternative expectations, one on the behavior of bob and the other on that of $john$. Nevertheless, it is accountable: in case no action is done after $alice$'s request a_1, *bob* is indicted, since he did neither do b_1 nor delegate it to $john$; if *bob* delegates but action b_1 is not done, $john$ is indicted by IC (15).

Example 5.2. First-Price Sealed Bid Auction (adapted from [Alberti et al. 2006b]) In [Alberti et al. 2006b], two protocols are defined for detecting possible violations in auctions; two types of auction are considered: a First-Price Sealed Bid (FPSB) auction, and a classical English Auction.

The protocol for the First-Price Sealed Bid auction contains various Integrity Constraints that deal with the various phases of the auction (opening, bids, declaration of the winner, payment, and delivery of the good). One of the Integrity Constraints, simplified, can be stated as follows:

$$\begin{array}{ll} & \mathbf{H}(do(Auctioneer, Bidders, opauc(Item, T_{dead}, T_{notify}, fpsb)), T_{open}), \\ & \mathbf{H}(do(Bidder_1, Auctioneer, bid(Item, Q_1)), T_1), T_1 < T_{dead} \\ \rightarrow & \mathbf{E}(do(Bidder_2, Auctioneer, bid(Item, Q_2)), T_2), \\ & Q_2 > Q_1, T_2 < T_{dead} \\ \vee & \mathbf{E}(do(Auctioneer, Bidder_1, answ(win, Item, Q_1)), T_{win}), \\ & T_{win} < T_{dead} + T_{notify} \end{array} \qquad (16)$$

Intuitively, the IC (16) defines one of the rules the Auctioneer should follow to declare a winning bid: if there exists at least a bid, for a price Q_1, then either another agent $Bidder_2$ makes a bid for a higher price, or the auctioneer must declare Q_1 as winning.

IC (16) shows that the FPSB protocol is not consequential. Whether the protocol is accountable or not depends also on the other ICs (and KB) defining the protocol.

An automatic checking procedure can be devised by using the formal definition of accountable protocol; in principle, one should check the protocol with all the possible histories, compute all the indicted agents for each violation answer and, possibly, discover the non-accountability of the protocol as sketched in Algorithm 1. Clearly, as the number of histories is infinite, such procedure can at most prove non-accountability, but never prove that the protocol is accountable.

Even when considering finite histories, the $\mathcal{S}$CIFF proof-procedure might not terminate. This is not surprising, since the $\mathcal{S}$CIFF language is a superset of Prolog (e.g., terms can be built on function symbols without restrictions) so it is Turing-complete. Even in the propositional case without integrity constraints, the complexity of finding if there exists an abductive explanation is Σ_2^P-complete [Eiter and Gottlob 1995].

ALGORITHM 1: Procedure to check if a protocol is accountable

```
Data: A protocol
Result: $accountable$
$accountable \leftarrow true$;
while $accountable$ do
    generate a history HAP;
    run $\mathcal{S}$CIFF;
    if not obtained $\mathcal{S}$CIFF answer then
        compute all $\mathcal{S}$CIFF violation answers;
        $S \leftarrow \bigcap\{$indicted agents of violation answers$\}$;
        if $S = \emptyset$ then
            $accountable \leftarrow false$;
        end
    end
end
```

Fortunately, generating all histories is not necessary: a variation of the $\mathcal{S}$CIFF proof-procedure that considers $\mathbf{H}$ events as abducible atoms can handle also event sets that are not ground. As already said, the $\mathcal{S}$CIFF proof-procedure can handle abducibles that contain variables, and which can, themselves, be subject to CLP constraints. In this way, classes of histories can be generated intensionally. Also, many practical protocols consider only finite histories, so the actual number of histories to be considered is, in some interesting cases, finite.

Even when the history length is unbounded, it can still be of interest to prove, or refute, accountability for histories of bounded maximal length, as in Bounded Model Checking [Biere et al. 2003]: we follow this approach, considering histories of increasing maximal length.

Unluckily, considering abducible the $\mathbf{H}$ events means that those transitions that rely on the fact that the $\mathbf{H}$ atoms are ground can no longer be applied. Fortunately, only one such transition exists, namely the *not-happened* transition [Alberti et al. 2008], that is activated when a $\neg\mathbf{H}$ literal exists in an integrity constraint. In the rest of this section, we will consider only protocols that do not contain integrity constraints with negative $\mathbf{H}$ literals.

We name $g\mathcal{S}$CIFF$^-$ the aforementioned variation of the $\mathcal{S}$CIFF proof-procedure. We recap the differences with $\mathcal{S}$CIFF:

— in $g\mathcal{S}$CIFF$^-$, $\mathbf{H}$ literals are considered as abducibles; the possible variables occurring in $\mathbf{H}$ literals are existentially quantified;
— in $g\mathcal{S}$CIFF$^-$, $\neg\mathbf{H}$ literals cannot occur in Integrity Constraints;
— in $g\mathcal{S}$CIFF$^-$, by default the intended semantics is that of violation answers (while, by default, for $\mathcal{S}$CIFF one is usually interested in $\mathcal{S}$CIFF answers, i.e., violations are not accepted).

The $g\mathcal{S}$CIFF$^-$ proof procedure tries to match each event in the history with the body of each integrity constraint. If the event contains variables, two derivation branches are generated: one in which the event unifies with the $\mathbf{H}$ literal in the body of the IC, and one in which a dis-unification constraint is imposed. This means that, if a $\mathbf{H}$ event containing variables is abduced, the $g\mathcal{S}$CIFF$^-$ proof-procedure will generate several alternative branches: one for each $\mathbf{H}$ atom occurring in each integrity constraint.

Example 5.3. For instance, consider the protocol in Example 4.1, defined with the integrity constraint in Eq. (12), in which we make explicit the time parameter of events and expectations:

$$\begin{array}{rll} \mathbf{H}(do(alice,bob,ask),T) & \rightarrow & \mathbf{E}(do(bob,alice,yes),T_b) \ \wedge\ \mathbf{E}(do(john,alice,yes),T_j) \\ & \vee & \mathbf{E}(do(bob,alice,no),T_b) \ \wedge\ \mathbf{E}(do(john,alice,no),T_j) \end{array} \quad (17)$$

By asking the $\mathcal{S}$CIFF proof procedure to abduce one atom $\mathbf{H}(X_1,T_1)$ (where both X_1 and T_1 are variables), two alternative branches are generated

— in one $\mathbf{H}(X_1,T_1)$ is unified with $\mathbf{H}(do(alice,bob,ask),T)$, i.e., $X_1 = do(alice,bob,ask)$ and $T_1 = T$;
— in the other, the previous unification is forbidden by imposing a dis-unification constraint, resulting in $X_1 \neq do(alice,bob,ask) \vee T_1 \neq T$ in this case.

In this way, two histories are generated, and they intensionally represent all the histories composed of exactly one event that are significant for the protocol in consideration:

$$\begin{array}{rcl} \mathbf{HAP}_1 & = & \{\mathbf{H}(do(alice,bob,ask),T)\} \\ \mathbf{HAP}_2 & = & \{\mathbf{H}(X_1,T_1)\} \text{ such that } X_1 \neq do(alice,bob,ask) \vee T_1 \neq T. \end{array}$$

Now, these two intensional histories (including the CLP constraints, such as the disunification $\neq$) can be fed to the $\mathcal{S}$CIFF proof procedure, that will generate the corresponding sets of expectations.

For $\mathbf{HAP}_1$, there exist two possible sets of expectations, corresponding to the two alternatives in the head of IC (17):

$$\begin{aligned}\Delta_{1,1} &= \{\mathbf{E}(do(bob, alice, yes), T_b), \mathbf{E}(do(john, alice, yes), T_j)\}\\ \Delta_{1,2} &= \{\mathbf{E}(do(bob, alice, no), T_b),\ \mathbf{E}(do(john, alice, no), T_j)\}.\end{aligned}$$

Since no other events are in the history, all the generated expectations are violated, and in both branches there are two indicted agents: $S = \{bob, john\}$.

For $\mathbf{HAP}_2$, no expectation is generated, since the only event does not match with the body of any Integrity Constraint, so we obtain a $\mathcal{S}$CIFF answer.

We can say that the protocol defined with Eq. (17) is accountable for all histories consisting of exactly one event.

In a similar way, the $g\mathcal{S}$CIFF$^-$ proof-procedure tries to match each expectation with each event in the history, to prove fulfillment of positive expectations and violation of negative ones.

Example 5.4 (*Example 5.3 continued*). Let us now suppose that $g\mathcal{S}$CIFF$^-$ is asked to abduce a history consisting of two generic events: $\mathbf{H}(X_1, T_1)$ and $\mathbf{H}(X_2, T_2)$. We have already seen the histories generated for one event; let us suppose that $\mathbf{H}(X_1, T_1) = \mathbf{H}(do(alice, bob, ask), T)$: since the body of the only Integrity Constraint is true, one of the disjuncts is chosen. If the first choice is taken, both bob and $john$ should reply yes:

$$\Delta_{1,1} = \{\mathbf{E}(do(bob, alice, yes), T_b), \mathbf{E}(do(john, alice, yes), T_j)\}.$$

Now, the $g\mathcal{S}$CIFF$^-$ proof-procedure tries to match each expectation with each event in the history. No expectation matches with the first event $\mathbf{H}(do(alice, bob, ask), T)$. The following options are explored for $\mathbf{H}(X_2, T_2)$

— (X_2, T_2) unifies with $(do(alice, bob, ask), T)$
— (X_2, T_2) unifies with $(do(bob, alice, yes), T_b)$
— (X_2, T_2) unifies with $(do(john, alice, yes), T_b)$
— (X_2, T_2) does not unify with any of the previous.

the generated histories are

$$\begin{array}{ll}\mathbf{H}(do(alice, bob, ask), T), & \mathbf{H}(do(alice, bob, ask), T)\\ \mathbf{H}(do(alice, bob, ask), T), & \mathbf{H}(do(bob, alice, yes), T_b)\\ \mathbf{H}(do(alice, bob, ask), T), & \mathbf{H}(do(john, alice, yes), T_b)\\ \mathbf{H}(do(alice, bob, ask), T), & \mathbf{H}(X_2, T_2)\end{array}$$

where, in the last one, $\mathbf{H}(X_2, T_2)$ must not unify with any of the other events $\mathbf{H}(do(alice, bob, ask), T)$, $\mathbf{H}(do(bob, alice, yes), T_b)$, and $\mathbf{H}(do(john, alice, yes), T_b)$. These four histories can then be submitted to the $\mathcal{S}$CIFF proof-procedure to obtain the possible sets of expectations. Considering the second history, the sets of expectations are

$$\begin{aligned}\Delta_1 &= \{\mathbf{E}(do(bob, alice, yes), T_b),\ \mathbf{E}(do(john, alice, yes), T_j)\}\\ \Delta_2 &= \{\mathbf{E}(do(bob, alice, no), T_b),\ \ \mathbf{E}(do(john, alice, no), T_j)\}\end{aligned}$$

None of them is a $\mathcal{S}$CIFF answer, as in both there are violated expectations. The agents responsible of the expected behavior are respectively $\{john\}$ and $\{bob, john\}$, with a non-empty intersection: in both cases, $john$ is indicted.

No non-accountability was proven. By reasoning on the remaining cases, it can be easily seen that in all histories consisting of exactly two events the issue of non-accountability does not occur.

In the same way, one can try all histories with three events: in such a case, one of the generated histories is

$$\begin{array}{l} \mathbf{H}(do(alice, bob, ask), T), \\ \mathbf{H}(do(bob, alice, yes), T_b) \\ \mathbf{H}(do(john, alice, X_3), T_3) \text{ such that } X_3 \neq yes \end{array} \tag{18}$$

and represents all the histories in which $john$ gives a reply that does not match with the yes reply given by bob. When this history is fed to the $\mathcal{S}$CIFF proof-procedure, amongst the sets of expectations there are

$$\begin{array}{l} \Delta_y = \{\mathbf{E}(do(bob, alice, yes), T_b),\ \mathbf{E}(do(john, alice, yes), T_j)\} \\ \Delta_n = \{\mathbf{E}(do(bob, alice, no), T_b),\ \mathbf{E}(do(john, alice, no), T_j)\} \end{array}$$

where, as already noted, in Δ_y the violated expectation is on agent $john$'s behavior, while in Δ_n the violated expectation is on bob's behavior, meaning that the intersection $S = \emptyset$. This not only proves that the protocol is not accountable, but also provides as counterexample the history (18), for which no agent can be blamed for the violation.

Algorithm 2 details, in a Prolog-like pseudocode, how histories are generated. Predicate generate_goal provides as $Goal$ an empty history; upon backtracking it provides histories containing an increasing number of generic events. The $g\mathcal{S}$CIFF^{-} proof-procedure executes the $Goal$, in this case abducing the set of generic events, specializing them for the protocol, as in Examples 5.3-5.4. Finally, a grounding phase takes care of grounding the history such that each action has a ground agent that performs it. In such a phase, it takes as input a set $Agents$ of names of agents.

ALGORITHM 2: Generate a history **HAP**

```
Data: Agents: set of names of considered agents
Result: HAP: a history
generate_history(Agents,HAP):-
    generate_goal(Goal),
    $g\mathcal{S}$CIFF$^{-}$(Goal,HAP),
    ground_agents(Agents,HAP).
generate_goal([]).
generate_goal([$\mathbf{H}(X,T)$ | Goal]):-
    generate_goal(Goal).
```

The previous Examples 5.3 and 5.4 hinted how events are generated; we can now provide an upper bound to the number of generated histories.

THEOREM 5.5. *Suppose that all ICs contain only* $\mathbf{H}$ *atoms in the body and only* $\mathbf{E}$ *atoms in the head.*

Let $N_{\mathbf{E}}$ *be the number of* $\mathbf{E}$ *atoms occurring in the heads,* $N_{\mathbf{H}}$ *the number of* $\mathbf{H}$ *atoms occurring in the body and* N_A *the number of agents.*

The number of histories with size n *generated by Algorithm 2 is* $O\left(N_A^n(2^{N_{\mathbf{H}}+nN_{\mathbf{E}}})^n\right)$.

PROOF. Each event in the history might trigger an IC, that can generate up to $N_{\mathbf{E}}$ expectations; thus the number of expectations raised for a history of n events is at most $nN_{\mathbf{E}}$.

Each generic atom $\mathbf{H}(A,T)$ is checked against each body literal and each generated expectation, generating a binary tree of maximum height $N_{\mathbf{H}} + nN_{\mathbf{E}}$; the number of leafs are then $2^{N_{\mathbf{H}}+nN_{\mathbf{E}}}$ at most.

Table II. Experimental results (✓= accountable, ✗= not accountable, TO = timeout, - = unnecessary; runtimes in parentheses are in milliseconds)

	History depth					
Protocol	0	1	2	3	4	5
FPSB Auction (Original)	✓(2)	✓(33)	✗(48)	-	-	-
FPSB Auction (Revised)	✓(1)	✓(23)	✓(249)	✓(3727)	✓(335073)	TO(1000000)
English Auction	✓(1)	✓(44)	✗(103)	-	-	-

The number of non-ground histories of n events cannot exceed $(2^{N_{\mathbf{H}}+nN_{\mathbf{E}}})^n$. The grounding phase tries each agent with each element in the history, so there are at most N_A^n groundings for each non-ground history; in total the number of generated histories is $O\left(N_A^n(2^{N_{\mathbf{H}}+nN_{\mathbf{E}}})^n\right)$. $\square$

We ran the devised algorithm on a collection of protocols that were developed in $\mathcal{S}$CIFF in the past. The experimental results are reported in Table II. The code for the experiments, with instructions for running them, is available at the repository https://bitbucket.org/malbertife/bmc_accountability.

The algorithm was able to detect in 0.048s (on an Intel core i5-6200U 2.4GHz, running SWI Prolog 7.2.3 [Wielemaker et al. 2012] on a virtual Linux Debian 8 64bit machine with two allocated cores and 4GB of RAM) the non-accountability of the FPSB auction and provided as a counterexample the history

$$\mathbf{HAP} = \{\mathbf{H}(do(a, opauc), T_{open}), \mathbf{H}(do(b, bid(Q_1)), T_{bid})\} \tag{19}$$

that says that if the auctioneer opened an auction and the only other action is a bid (i.e., there is no declaration of a winner), then there is a violation, but it is unclear who is to blame for the violation. In fact, one option is that the auctioneer should declare a winner, the other option being that a third agent makes a better bid. Note that the provided counterexample (19) contains variables (the price Q_1 of the bid and the times in which the events are executed), since any history matching with the given one would be non-accountable. In this experiment, we considered a set of three agents (named a, b and c).

After looking at the counterexample (19), the designer decided that in such a situation the *auctioneer* should be indicted; he then was able to update the protocol, by adding the following rule:

$$\begin{aligned} &\mathbf{H}(do(Auctioneer, opauc), T_{open}) \wedge \mathbf{H}(do(Bidder_1, bid(Q_1)), T_{bid}) \\ &\quad \rightarrow \mathbf{E}(do(Auctioneer, answ(Bidder_2, win, Q_2)), T_{win}). \end{aligned} \tag{20}$$

that, intuitively, states that if there exists (at least) one bid, the auctioneer must always declare some winner. By adding IC (20), in case there is no winner declaration, in all violation sets there will be the expectation $\mathbf{E}(do(Auc, answ(Bidder_2, win, Q_2)), T_{win})$, so the history (19) becomes accountable. As expected, the algorithm confirms the accountability for the history depths (up to 4) examined before the timeout.

It is worth noting that the protocol was already able to identify the violation state (so it was correct for the task taken into consideration at that time); thanks to the new definition of accountability and to the checker we developed, it was possible to improve it and make it more apt toward new challenges.

The same experiment was also run on the English auction [Alberti et al. 2006b], and the result was similar.

6. RELATED WORK

Kailar [1996] proposes a framework for the analysis of protocol accountability, but for a notion of accountability that is different from the one analyzed in this article, i.e., the ability for a participant interaction to prove the origin of an action; for example, a customer in an electronic commerce transaction might want to prove that the message she or he received was in fact sent by the seller. Such property, while obviously important, is orthogonal to the one studied in this article, in that we assume that the originator of each action (communicative or physical) has been established, and we focus on whether the action has caused a violation of the interaction protocol.

Cederquist et al. [2005] propose a framework in which agents can distribute data along with usage policies, and they are accountable for their actions. Agent actions can be logged securely. The only actions allowed are those explicitly permitted by policies; agents are free to deviate from the policies, but their violations can be tracked by an auditing authority that can observe (some of) their logged actions. The paper [Cederquist et al. 2005] defines a language for agent actions and policies, and a logic to reason about actions and, in particular, to prove that an action was permitted by a policy; an implementation of the reasoner is also provided. In our work, we start from a different normative premise, which is more realistic for the open systems to which $\mathcal{S}$CIFF has been applied: all actions are allowed if not explicitly forbidden; also, the policy language in [Cederquist et al. 2005] is tailored for access control scenarios, while the $\mathcal{S}$CIFF language supports a variety of open systems, as shown in the aforementioned application papers.

Baldoni et al. [2012] address a problem similar to the one addressed by the present work: they developed a tool to visualize all the possible enactments of a protocol, in order to visualize possible violations. They formalize interaction protocols by means of responsibility and *commitments* [Castelfranchi 1995], a successful framework for defining and reasoning about interaction protocols. Their case study is the Markets in Financial Instruments Directive (MiFID) issued by the European Union. While Baldoni et al. [2012] adopt a framework that supports responsibility and propose a tool to detect visually violations, we propose tools that address accountability, with the aim of detecting automatically (either by means of checking syntactic features, or by means of software tools) if a protocol is not accountable.

In [Baldoni et al. 2016], the authors introduce the term “Computational Accountability” as the application of Artificial Intelligence techniques to address the problems of “traceability, evaluation, and communication of values and good conduct”. They describe a work in progress aimed to analyze agent accountability in a society ruled by commitment-based interaction protocols: since each commitment has a debtor and a creditor (and intuitively, the debtor is accountable to the creditor), and each commitment is originated by an action, the idea is to establish an agent’s accountability by following the causal chain that led to a violated commitment.

Our work also differs from [Baldoni et al. 2012] and [Baldoni et al. 2016] in that our proposed formalization of the notion of accountability is based on expectations, as defined in the $\mathcal{S}$CIFF framework [Alberti et al. 2008], rather than commitments. Both formalisms are aimed at providing semantics to interaction protocols without hindering agent autonomy, but expectations differ from commitments in two important aspects: first, they do not have a mutable state, but collectively describe an evolution of an open system conforming to a protocol; second, an expectation is not characterized by a debtor and a creditor (although, loosely speaking, the agent whose behavior is described by the expectation can be viewed as the debtor, and the society as the creditor). An in-depth comparison between commitments and expectations is available in [Torroni et al. 2009].

Kafalı et al. [2016] present a framework, named Revani, for Revision and Verification of Normative Specifications for privacy. They adopt a formal computational representation of norms, also incorporating the social (e.g., humans) and technical (e.g., computers) elements peculiar of privacy. They consider three types of norms: Authorization, Commitment, and Prohibition. Each norm is in practice an atom with predicate A, C, or P, and indicates who is accountable to whom. Assumptions, in the form $Head \leftarrow Body$ rules, then describe the operating environment and what can or cannot happen. Revani also provides a mechanism which, on the basis of truth of enabling conditions, makes some effects taking place. Basically, effects are addition or deletion of propositional atoms. To perform formal verification, model checking for CTL logic is exploited in order to check if the specified norms, assumptions, and mechanisms, entails requirements (i.e., properties) specified as CTL formulas. If not, the model checker adopted (i.e., NuSMV) provides a counterexample. An iterative design process for specifications is also provided, in order to revise specifications until requirements are entailed. This design process is sound, even if not complete (might not always compute a revise specification satisfying the requirements). In our approach, instead, we exploit an abductive proof procedure (rather than a model checker) to check accountability of a $\mathcal{S}$CIFF protocol. The proof procedure is also able to provide a counter-example useful to fix the protocol.

7. CONCLUSIONS

Accountability is an important property in human societies, and it will be more and more important in software agent societies. In this work, we consider a notion of accountability, taken from the field of ethics and philosophy [Van de Poel et al. 2015], and we provided a formal definition based on the $\mathcal{S}$CIFF framework [Alberti et al. 2008]. $\mathcal{S}$CIFF is a framework grounded on logic programming and including a language for defining interaction protocols and a proof-procedure to check the compliance of the agents to the protocols.

The formal definition of accountability given in this work can be used to verify if a protocol is accountable, i.e., if in any violation state there exists at least one agent that can be asked to account for the violation.

In order to help the designer of interaction protocols, we provided a guideline and a tool. The guideline is a defined a static syntactic property of a protocol (called *consequential protocol*) that ensures that a protocol is accountable. We found that many of the protocols defined in the past for $\mathcal{S}$CIFF fall within the class of consequential protocols, and are by construction accountable.

Nevertheless, there are accountable protocols that are not consequential, as shown in various examples. We proposed a tool, based on the $\mathcal{S}$CIFF proof-procedure, that checks if an existing protocol is not accountable. By running the tool on the previously defined protocols, we were able to spot some cases of non accountability; the tool also provided a counterexample showing a possible course of events in which, although a violation was detected, no agent was liable. After this check it was possible to fix and improve the protocol toward accountability.

REFERENCES

Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Marco Montali. 2006a. An Abductive Framework for A-Priori Verification of Web Services. In *Proceedings of the 8th Symposium on Principles and Practice of Declarative Programming*, Michael Maher (Ed.). ACM Press, New York, USA, 39–50.

Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, Sergio Storari, and Paolo Torroni. 2006. Computational Logic for Run-Time Verification of Web Services Choreographies: Exploiting the SOCS-SI Tool. In *Web Services and Formal Methods: Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006 Proceedings*, Mario Bravetti, Manuel Núñez, and Gianluigi Zavattaro (Eds.). Springer, Berlin, Heidelberg, 58–72. DOI:http://dx.doi.org/10.1007/11841197_4

Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, and Paolo Torroni. 2007. A rule-based approach for reasoning about collaboration between smart Web services.

In *Proceedings of the 1st International Conference on Web Reasoning and Rule Systems (RR) (Lecture Notes in Artificial Intelligence)*, Massimo Marchiori, Jeff Z. Pan, and Christian de Sainte Marie (Eds.), Vol. 4524. Springer-Verlag, Innsbr uck, 279–288.

Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. 2006b. Compliance Verification of Agent Interaction: a Logic-Based Software Tool. *Applied Artificial Intelligence* 20, 2-4 (Feb.-April 2006), 133–157.

Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. 2006c. Security protocols verification in Abductive Logic Programming: a case study. In *Proceedings of ESAW'05*, Oguz Dikenelli, Marie-Pierre Gleizes, and Andrea Ricci (Eds.). Lecture Notes in Artificial Intelligence, Vol. 3963. Springer, Kusadasi, Turkey, 106–124.

Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. 2008. Verifiable Agent Interaction in Abductive Logic Programming: the SCIFF Framework. *ACM Transactions on Computational Logic* 9, 4 (2008), 29:1–29:43.

Marco Alberti, Anna Ciampolini, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. 2003. A Social ACL Semantics by Deontic Constraints. In *Multi-Agent Systems and Applications III. Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003 (Lecture Notes in Artificial Intelligence)*, V. Mařík, J. Müller, and M. Pěchouček (Eds.), Vol. 2691. Springer Verlag, Prague, Czech Republic, 204–213.

Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, Giovanni Sartor, and Paolo Torroni. 2006a. Mapping Deontic Operators to Abductive Expectations. *Computational and Mathematical Organization Theory* 12, 2–3 (Oct. 2006), 205 – 225. DOI:http://dx.doi.org/10.1007/s10588-006-9544-8

Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. 2004. Modeling Interactions Using *Social Integrity Constraints*: A Resource Sharing Case Study. In *Declarative Agent Languages and Technologies*, João Alexandre Leite, Andrea Omicini, Leon Sterling, and Paolo Torroni (Eds.). Lecture Notes in Artificial Intelligence, Vol. 2990. Springer-Verlag, 243–262. First International Workshop, DALT 2003. Melbourne, Australia, July 2003. Revised Selected and Invited Papers.

Marco Alberti, Marco Gavanelli, Evelina Lamma, Giovanni Sartor, and Paolo Torroni. 2006b. Un Sistema Basato su Logica Computazionale per il Trattamento degli Operatori Deontici. In *La Gestione e la Negoziazione Automatica dei Diritti sulle Opere dell'Ingegno Digitali: Aspetti Giuridici e Informatici*, Silvia Bisi and Claudio di Cocco (Eds.). Gedit, Bologna, Chapter 1, 1–33.

Matteo Baldoni, Cristina Baroglio, Elisa Marengo, and Viviana Patti. 2012. *Supporting the Analysis of Risks of Violation in Business Protocols: The MiFID Case Study*. Physica-Verlag HD, Heidelberg, 545–553. DOI:http://dx.doi.org/10.1007/978-3-7908-2789-7_59

Matteo Baldoni, Cristina Baroglio, Katherine M. May, Roberto Micalizio, and Stefano Tedeschi. 2016. Computational Accountability. In *URANIA 2016 Deep Understanding and Reasoning: A challenge for Next-generation Intelligent Agents (CEUR Workshop Proceedings - AI*IA Series)*, Paola Mello, Michela Milano, and Federico Chesani (Eds.).

Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. 2003. Bounded Model Checking. *Advances in Computers* 58 (2003), 117 – 148. DOI:http://dx.doi.org/10.1016/S0065-2458(03)58003-2

Olivier Boissier, Marco Colombetti, Michael Luck, John-Jules Ch. Meyer, and Axel Polleres. 2013. Norms, organizations, and semantics. *Knowledge Eng. Review* 28, 1 (2013), 107–116. DOI:http://dx.doi.org/10.1017/S0269888912000367

Mark Bovens. 1998. *The quest for responsibility - Accountability and Citizenship in Complex Organizations*. Cambridge University Press, Cambridge.

C. Castelfranchi. 1995. Commitments: From individual intentions to groups and organizations. In *Proceedings of the First International Conference on Multiagent Systems, San Francisco, California, USA*. AAAI Press, San Francisco, California, 41–48.

J. G. Cederquist, R. Conn, M. A. C. Dekker, S. Etalle, and J. I. den Hartog. 2005. An audit logic for accountability. In *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*. IEEE, 34–43. DOI:http://dx.doi.org/10.1109/POLICY.2005.5

Federico Chesani, Paola Mello, Marco Montali, and Sergio Storari. 2007. Testing Careflow Process Execution Conformance by Translating a Graphical Language to Computational Logic. In *Proceedings of the 11th Conference on Artificial Intelligence in Medicine (AIME '07)*. Springer-Verlag, Berlin, Heidelberg, 479–488. DOI:http://dx.doi.org/10.1007/978-3-540-73599-1_64

Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. 2011. Modeling and verifying business processes and choreographies through the abductive proof procedure SCIFF and its extensions. *Intelligenza Artificiale* 5, 1 (2011), 101–105. DOI:http://dx.doi.org/10.3233/IA-2011-0011

Amit K. Chopra and Munindar P. Singh. 2014. The thing itself speaks: Accountability as a foundation for requirements in sociotechnical systems. In *IEEE 7th International Workshop on Requirements Engineering and Law, RELAW 2014, 26-26 August, 2014, Karlskrona, Sweden*, Daniel Amyot, Annie I. Antón, Travis D. Breaux, Aaron K. Massey, and Alberto Siena (Eds.). IEEE Computer Society, Karlskrona, Sweden, 22. DOI:http://dx.doi.org/10.1109/RELAW.2014.6893477

Anna Ciampolini, Paola Mello, Marco Montali, and Sergio Storari. 2005. Using Social Integrity Constraints for On-the-Fly Compliance Verification of Medical Protocols. In *18th IEEE Symposium on Computer-Based Medical Systems (CBMS 2005), 23-24 June 2005, Dublin, Ireland*. IEEE Computer Society, Trinity College Dublin, Ireland, 503–505. DOI:http://dx.doi.org/10.1109/CBMS.2005.102

Benjamin Cox, J.D. Tygar, and Marvin Sirbu. 1995. NetBill Security and Transaction Protocol. In *Proceedings of the First USENIX Workshop on Electronic Commerce*. USENIX Association, New York.

Thomas Eiter and Georg Gottlob. 1995. The Complexity of Logic-based Abduction. *J. ACM* 42, 1 (Jan. 1995), 3–42. DOI:http://dx.doi.org/10.1145/200836.200838

N. Fornara and M. Colombetti. 2003. Defining Interaction Protocols using a Commitment-based Agent Communication Language. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003)*, J. S. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo (Eds.). ACM Press, Melbourne, Victoria, 520–527.

T. H. Fung and R. A. Kowalski. 1997. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming* 33, 2 (1997), 151–165.

J. Jaffar and M.J. Maher. 1994. Constraint Logic Programming: a Survey. *Journal of Logic Programming* 19-20 (1994), 503–582.

Özgür Kafalı, Nirav Ajmeri, and Munindar P. Singh. 2016. Revani: Revising and Verifying Normative Specifications for Privacy. *IEEE Intelligent Systems* 31, 5 (2016), 8–15. DOI:http://dx.doi.org/10.1109/MIS.2016.89

Rajashekar Kailar. 1996. Accountability in Electronic Commerce Protocols. *IEEE Trans. Softw. Eng.* 22, 5 (May 1996), 313–328. DOI:http://dx.doi.org/10.1109/32.502224

A. C. Kakas, R. A. Kowalski, and Francesca Toni. 1993. Abductive Logic Programming. *Journal of Logic and Computation* 2, 6 (1993), 719–770.

Marco Montali. 2010. *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach*. Lecture Notes in Business Information Processing, Vol. 56. Springer.

Roger M. Needham and Michael D. Schroeder. 1978. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM* 21, 12 (December 1978), 993–999.

Maja Pesic and Wil M. P. van der Aalst. 2006. A Declarative Approach for Flexible Business Processes Management. In *Business Process Management Workshops: BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4-7, 2006. Proceedings*, Johann Eder and Schahram Dustdar (Eds.). Springer, Berlin, Heidelberg, 169–180. DOI:http://dx.doi.org/10.1007/11837862_18

Fariba Sadri, Francesca Toni, and Paolo Torroni. 2003. Minimally Intrusive Negotiating Agents for Resource Sharing. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 796–801. http://dl.acm.org/citation.cfm?id=1630659.1630774

Munindar P. Singh. 1998. Agent Communication Languages: Rethinking the Principles. *IEEE Computer* 31, 12 (Dec. 1998), 40–47. DOI:http://dx.doi.org/10.1109/2.735849

Paolo Torroni, Federico Chesani, Paola Mello, Pinar Yolum, Munindar P. Singh, Marco Alberti, Marco Gavanelli, and Evelina Lamma. 2009. Modeling Interactions via Commitments and Expectations. In *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, Virginia Dignum (Ed.). Information Science Reference, Hershey, PA, 263–284. http://www.igi-global.com/reference/details.asp?ID=33141

Ibo Van de Poel, Lambèr Royakkers, and Sjoerd D. Zwart. 2015. *Moral Responsibility and the Problem of Many Hands*. Routledge.

Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. 2012. SWI-Prolog. *Theory and Practice of Logic Programming* 12, 1-2 (2012), 67–96. DOI:http://dx.doi.org/10.1017/S1471068411000494