Available online at www.sciencedirect.com

ScienceDirect

Artificial Intelligence 171 (2007) 776–804

Artificial Intelligence

www.elsevier.com/locate/artint

An executable specification of a formal argumentation protocol

Alexander Artikis[a,*], Marek Sergot[b], Jeremy Pitt[c]

[a] *Institute of Informatics & Telecommunications, NCSR "Demokritos", Athens, 15310, Greece*
[b] *Department of Computing, Imperial College London, SW7 2AZ, UK*
[c] *Department of Electrical & Electronic Engineering, Imperial College London, SW7 2BT, UK*

Received 8 November 2006; received in revised form 3 April 2007; accepted 16 April 2007

Available online 29 April 2007

Abstract

We present a specification, in the action language $\mathcal{C}+$, of Brewka's reconstruction of a theory of formal disputation originally proposed by Rescher. The focus is on the procedural aspects rather than the adequacy of this particular protocol for the conduct of debate and the resolution of disputes. The specification is structured in three separate levels, covering (i) the physical capabilities of the participant agents, (ii) the rules defining the protocol itself, specifying which actions are 'proper' and 'timely' according to the protocol and their effects on the protocol state, and (iii) the permissions, prohibitions, and obligations of the agents, and the sanctions and enforcement strategies that deal with non-compliance. Also included is a mechanism by which an agent may object to an action by another participant, and an optional 'silence implies consent' principle. Although comparatively simple, Brewka's protocol is thus representative of a wide range of other more complex argumentation and dispute resolution procedures that have been proposed. Finally, we show how the 'Causal Calculator' implementation of $\mathcal{C}+$ can be used to animate the specification and to investigate and verify properties of the protocol.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Argumentation; Disputation; Protocol; Norm; Multi-agent system; Specification; Action language

1. Introduction

One of the main tasks in the formal specification and analysis of (open) multi-agent systems (MAS) is the representation of the protocols and procedures for agent interactions, and the norms of behaviour that govern these interactions. Examples include protocols for exchanging information, for negotiation, and for resolving disputes.

It has been argued that a specification of systems of this type should satisfy at least the following two requirements: first, the interactions of the members should be governed by a formal, declarative, verifiable and meaningful semantics [64]; and second, to cater for the possibility that agent behaviour may deviate from what is prescribed, agent interactions can usefully be described in terms of permissions and obligations [26].

We have been developing a theoretical framework for the executable specification of open agent systems that addresses the aforementioned requirements [3–5]. We adopt the perspective of an external observer, thus taking into account only externally observable behaviours and not the internal architectures of the individual agents, and view

[*] Corresponding author.
E-mail addresses: a.artikis@acm.org (A. Artikis), mjs@doc.ic.ac.uk (M. Sergot), j.pitt@imperial.ac.uk (J. Pitt).

0004-3702/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2007.04.008

agent systems as instances of *normative systems* [26] whereby constraints on agents' behaviour (or social constraints) are specified in terms of their permissions, their institutional power to effect changes and bring about certain states of affairs, and their rights and obligations to one another. We employ an action formalism to specify the social constraints governing the behaviour of the members and then use a computational framework to animate the specification and investigate its properties. For the action formalism, we have employed the Event Calculus [29], the action language $\mathcal{C}+$ [22], and an extended form of $\mathcal{C}+$ specifically designed for modelling the institutional aspects of agent systems [57–59].

In this paper we demonstrate how the theoretical and computational frameworks can be used with the language $\mathcal{C}+$ to specify and execute an argumentation protocol based on Brewka's reconstruction [8], in the Situation Calculus [52], of a theory of formal disputation originally proposed by Rescher [53]. We presented a preliminary formulation in an earlier paper [4]. This present paper is a refined and much extended version.

We are focusing here on the procedural aspects of the protocol rather than on the underlying logic of disputation employed by Brewka or on the adequacy of this particular protocol for the conduct of debate and the resolution of disputes. The features of Brewka's protocol are representative of a wide range of other more complex argumentation and dispute resolution procedures that have been proposed in the literature, and to which the methods of this paper can be similarly applied.

The specification of the argumentation protocol is structured into three separate levels, covering:

(i) the physical capabilities of the participant agents (in the present context, the messages/utterances each agent is actually capable of transmitting);
(ii) the rules defining the protocol itself, specifying which actions are 'proper' and 'timely' according to the protocol and their effects on the protocol state;
(iii) the permissions, prohibitions and obligations of the agents, and the sanctions and enforcement strategies that deal with non-compliance.

In any given implementation of the protocol, it may or may not be permitted for an agent to perform an action that is not proper or timely; conversely, there may be protocol actions that are proper and timely but that are nevertheless not permitted under certain circumstances, because, for instance, they lead to protocol runs with undesirable properties. The rules comprising level (ii) of the specification correspond to *constitutive norms* that define the meaning of the protocol actions. Levels (i) and (iii), respectively, can be seen as representing the *physical* and *normative* environment within which the protocol is executed. We have also been concerned with the concept of social role. Briefly, a role is associated with a set of (role) preconditions that agents must satisfy in order to be eligible to occupy that role and a set of (role) constraints that govern the behaviour of the agents once they occupy that role. We will not discuss role assignment in this paper. For the example in this paper, we will assume for simplicity that the participant agents are already assigned to certain roles, and that these roles do not change during an execution of the protocol.

A note on terminology. In the earlier version of this paper [4], and in the treatment of other examples, we defined a protocol by specifying the conditions under which an action was said to be 'valid' according to the protocol. Here, we have employed a finer structure, further classifying 'valid' actions as proper or timely, in line with suggestions that have also been made by Prakken et al. [45,49]. A 'valid' action in our earlier terminology is one that is both proper and timely. Other terminology in common use employs the term 'successful' where we say 'valid': one then distinguishes between an action, such as an utterance or the transmission of a message of a certain form, which is an 'attempt' to make a claim, say, and the conditions under which the attempt to claim is 'successful' (sometimes, 'effective'). We prefer to avoid the term 'successful' since even an unsuccessful 'attempt' can have effects on the protocol state. We also avoid use of the term 'legal' for 'valid' or 'successful' since it is ambiguous as to whether it refers to the constitutive element of the protocol (level (ii) of our specification) or the normative environment in which the protocol is executed (level (iii)). Also related is the concept of institutional (or 'institutionalised') power (sometimes, 'competence' or 'capacity'). This refers to the characteristic feature of institutions—legal systems, formal organisations, or informal groupings—whereby designated agents, often when acting in specific roles, are empowered to create or modify facts of special significance in that institution—*institutional facts* in the terminology of Searle [56]. (See e.g. [27,35] for further discussion and references to the literature.) Thus in the present example it is natural to say that, under certain circumstances, an agent acting in a certain role has power (competence, capacity) to declare the dispute resolved in favour of one or other of the protagonists; or that in certain circumstances an agent has power to object to an action by one of the other participants; or more generally, that the argumentation protocol defines the conditions under which an agent has the power to perform one of the argumentation actions. We will not refer explicitly to power in the specification of the argumentation protocol presented here. The classification of actions into proper and timely already provides a more detailed specification.

In this paper we use the language $\mathcal{C}+$ to formulate the specification. An advantage of $\mathcal{C}+$, compared with other action formalisms, is that it can be given an explicit semantics in terms of transition systems. This enables us to analyse and prove properties of the protocol. The concluding sections of the paper present some illustrative examples.

The paper is structured as follows. First, we briefly describe the $\mathcal{C}+$ language. Second, we present the 'Causal Calculator' software implementation, a computational framework for executing specifications formalised in $\mathcal{C}+$. Third, we summarise Brewka's reconstruction of Rescher's theory of formal disputation. Fourth, we specify, prove properties of, and execute (a form of) Brewka's argumentation protocol with the use of $\mathcal{C}+$ and the Causal Calculator. Finally, we discuss related research, summarise the presented work, and point out directions for further investigations.

2. The $\mathcal{C}+$ language

$\mathcal{C}+$, as mentioned above, is an action language with an explicit transition systems semantics. We describe here the version of $\mathcal{C}+$ presented in [22].

2.1. Basic definitions

A *multi-valued propositional signature* is a set σ of symbols called *constants*, and for each constant $c \in \sigma$, a non-empty finite set *dom*(c) of symbols, disjoint from σ, called the *domain* of c. For simplicity, in this presentation we will assume that every domain contains at least two elements.

An *atom* of signature σ is an expression of the form $c = u$ where $c \in \sigma$ and $u \in dom(c)$. A Boolean constant is one whose domain is the set of truth values $\{\mathrm{t}, \mathrm{f}\}$. When c is a Boolean constant we often write c for $c = \mathrm{t}$ and $\neg c$ for $c = \mathrm{f}$. A *formula* φ of signature σ is any propositional combination of atoms of σ. An *interpretation* I of σ is a function that maps every constant in σ to an element of its domain. An interpretation I *satisfies* an atom $c = u$ if $I(c) = u$. The satisfaction relation is extended from atoms to formulas according to the standard truth tables for the propositional connectives. A *model* of a set X of formulas of signature σ is an interpretation of σ that satisfies all formulas in X. If every model of a set X of formulas satisfies a formula φ then X *entails* φ, written $X \models \varphi$.

2.2. Syntax

The representation of an action domain in $\mathcal{C}+$ consists of *fluent* constants and *action* constants.

- Fluent constants are symbols characterising a state. They are divided into two categories: simple fluent constants and statically determined fluent constants. Simple fluent constants are related to actions by *dynamic laws* (that is, laws describing a transition $(s_i, \varepsilon_i, s_{i+1})$ from a state s_i to its successor state s_{i+1}). Statically determined fluent constants are characterised by *static laws* (that is, laws describing an individual state) relating them to other fluent constants. Static laws can also be used to express constraints between simple fluent constants. Static and dynamic laws are defined below.
- Action constants are symbols characterising state transitions. In a transition $(s_i, \varepsilon_i, s_{i+1})$, the transition label ε_i, also called an 'event', represents the occurrence of actions performed concurrently by one or more agents or occurring in the environment. Transitions may be non-deterministic. Action constants are used to name actions, attributes of actions, or properties of transitions as a whole.

An *action signature* $(\sigma^{\mathrm{f}}, \sigma^{\mathrm{a}})$ is a non-empty set σ^{f} of fluent constants and a non-empty set σ^{a} of action constants. An *action description* D in $\mathcal{C}+$ is a non-empty set of *causal laws* that define a transition system of a particular type. A causal law can be either a *static law* or a *dynamic law*. A static law is an expression

$$\textsf{caused } F \textsf{ if } G \tag{1}$$

where F and G are formulas of fluent constants. In a static law, constants in F and G are evaluated on the same state. A dynamic law is an expression

$$\text{caused } F \text{ if } G \text{ after } H \tag{2}$$

where F, G and H are formulas such that every constant occurring in F is a simple fluent constant, every constant occurring in G is a fluent constant, and H is any combination of fluent constants and action constants. In a transition from state s_i to state s_{i+1}, constants in F and in G are evaluated on s_{i+1}, fluent constants in H are evaluated on s_i and action constants in H are evaluated on the transition itself. F is called the *head* of the static law (1) and the dynamic law (2).

The full $\mathcal{C}+$ language also provides *action dynamic laws*, which are expressions of the form

$$\text{caused } \alpha \text{ if } H$$

where α is a formula containing action constants only and H is a formula of action and fluent constants. We will not use action dynamic laws in this paper and so omit the details in the interests of brevity.

The $\mathcal{C}+$ language provides various abbreviations for common forms of causal laws. For example, a dynamic law of the form

$$\text{caused } F \text{ if } \top \text{ after } H \wedge \alpha$$

where α is a formula of action constants is often abbreviated as

$$\alpha \text{ causes } F \text{ if } H$$

In the case where H is $\top$ the above is usually written as α causes F.

When presenting the argumentation protocol specification, we will often employ the causes abbreviation to express the effects of the agents' actions. We will also employ the $\mathcal{C}+$ abbreviation

$$\text{default } F$$

which is shorthand for the static law

$$\text{caused } F \text{ if } F$$

expressing that F holds in the absence of information to the contrary.

When it aids readability, we will write

$$F \text{ iff } G$$

as a shorthand for the pair of static laws

$$\text{caused } F \text{ if } G$$

and

$$\text{default } \neg F$$

Two further abbreviations that we will employ are nonexecutable and inertial; a dynamic law of the form

$$\text{caused } \bot \text{ if } \top \text{ after } \alpha \wedge H$$

where α is a formula containing only action constants and H is a formula containing only fluent constants is abbreviated as:

$$\text{nonexecutable } \alpha \text{ if } H$$

In the case where H is $\top$ the above can be written as nonexecutable α.

The inertia of a fluent constant c over time is represented as:

$$\text{inertial } c$$

This is an abbreviation for the *set* of dynamic laws of the form (for all values $u \in dom(c)$):

$$\text{caused } c = u \text{ if } c = u \text{ after } c = u$$

A $\mathcal{C}+$ action description is a non-empty set of causal laws. Of particular interest is the sub-class of *definite* action descriptions. A $\mathcal{C}+$ action description D is *definite* if:

- the head of every causal law of D is an atom or $\bot$, and
- no atom is the head of infinitely many causal laws of D.

The $\mathcal{C}+$ action description in this paper will be definite.

2.3. Semantics

It is not possible in the space available here to give a full account of the $\mathcal{C}+$ language and its semantics. We trust that the $\mathcal{C}+$ language, and especially its abbreviations, are sufficiently natural that readers can follow the presentation of the case study in later sections. Interested readers are referred to [22,23] for further technical details. For completeness, we summarise here the semantics of *definite* action descriptions ignoring (as we are) the presence of action dynamic laws (and assuming that the domain of every constant contains at least two elements). We emphasise the transition system semantics, as in [57].

Every action description D of $\mathcal{C}+$ defines a labelled transition system, as follows:

- States of the transition system are interpretations of the fluent constants σ^{f}. It is convenient to identify a state s with the set of fluent atoms satisfied by s (in other words, $s \models f = v$ if and only if $f = v \in s$ for every fluent constant f).
  Let $T_{\text{static}}(s)$ denote the heads of all static laws in D whose conditions are satisfied by s:

  $$T_{\text{static}}(s) =_{\text{def}} \{F \mid \text{static law (1) is in } D, s \models G\}$$

  For a definite action description D, an interpretation s of σ^{f} is a *state* of the transition system defined by D (or simply, a state of D for short) when

  $$s = T_{\text{static}}(s) \cup \mathit{Simple}(s)$$

  where $\mathit{Simple}(s)$ denotes the set of simple fluent atoms satisfied by s. (So $s - \mathit{Simple}(s)$ is the set of statically determined fluent atoms satisfied by s.)
- Transition labels of the transition system defined by D (also referred to as *events*) are the interpretations of the action constants σ^{a}.
  A *transition* is a triple (s, ε, s') in which s is the initial state, s' is the resulting state, and ε is the transition label (or event). Since transition labels are interpretations of σ^{a}, it is meaningful to say that a transition label ε satisfies a formula α of σ^{a}: when $\varepsilon \models \alpha$ we sometimes say that the transition (s, ε, s') is of type α.
- Let $E(s, \varepsilon, s')$ denote the heads of all dynamic laws of D whose conditions are satisfied by the transition (s, ε, s'):

  $$E(s, \varepsilon, s') =_{\text{def}} \{F \mid \text{dynamic law (2) is in } D, s' \models G, s \cup \varepsilon \models H\}$$

  For a definite action description D, (s, ε, s') is a *transition of* D (or in full, a transition of the transition system defined by D) when s and s' are interpretations (set of atoms) of σ^{f} and ε is an interpretation of σ^{a} such that:
  – $s = T_{\text{static}}(s) \cup \mathit{Simple}(s)$ (s is a state of D)
  – $s' = T_{\text{static}}(s') \cup E(s, \varepsilon, s')$

For any non-negative integer m, a *path* or *history* of D of length m is a sequence

$$s_0\, \varepsilon_0\, s_1\, \ldots\, s_{m-1}\, \varepsilon_{m-1}\, s_m$$

where $(s_0, \varepsilon_0, s_1), \ldots, (s_{m-1}, \varepsilon_{m-1}, s_m)$ are transitions of D.

3. The causal calculator

The Causal Calculator (CCALC) is a computational framework designed and implemented by the Action Group of the University of Texas for representing action and change in the $\mathcal{C}+$ language and performing a range of computational tasks on the resulting formalisations. CCALC has been applied to several 'challenge problems' (see, for example, [1,33,34]).

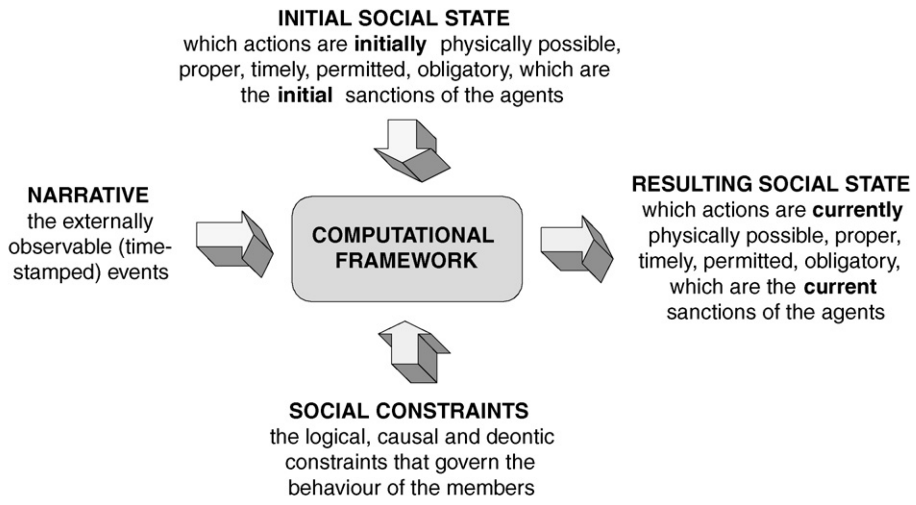
INITIAL SOCIAL STATE
which actions are initially physically possible, proper, timely, permitted, obligatory, which are the initial sanctions of the agents
NARRATIVE
the externally observable (time-stamped) events
COMPUTATIONAL FRAMEWORK
RESULTING SOCIAL STATE
which actions are currently physically possible, proper, timely, permitted, obligatory, which are the current sanctions of the agents
SOCIAL CONSTRAINTS
the logical, causal and deontic constraints that govern the behaviour of the members

Fig. 1. Executing the specification of an open agent system: Prediction queries.

A detailed account of CCALC's operation and functionality may be found in [22]. This section describes the way we use *prediction*, *planning* and *postdiction* queries to execute the specification of the social constraints (or protocol rules) of an open agent system. In each type of query, CCALC has as input a definite $\mathcal{C}+$ action description D^{soc} expressing the specification of social constraints. We refer to the states of the transition system defined by D^{soc} as *social states* (or protocol states). In other words, a social state is an interpretation (with some further properties) of the fluent constants of D^{soc}. These constants express, amongst other things, which actions are physically possible, proper, timely, permitted or obligatory, and whether an agent has been sanctioned for performing forbidden actions or not complying with its obligations.

- Prediction queries. The computation of an answer to this type of query involves: (i) an initial social state expressing, amongst other things, which actions are initially physically possible, proper, timely, permitted or obligatory, and the initial sanctions of the agents (an initial social state may be partial or complete), and (ii) a *narrative*, that is, a description of temporally-sorted externally observable events of the system (a narrative is expressed as a sequence of transitions). The outcome of a prediction query (if any) is the current social state, that is, the state resulting from the events described in the narrative, expressing, amongst other things, which actions are currently physically possible, proper, timely, permitted or obligatory, and the current sanctions of the agents (see Fig. 1). Computing answers to prediction queries may be performed at run-time to inform the members of a system, at any time during the execution of the system, of their permissions, obligations, sanctions and so on.
- Planning queries. Agents may issue planning queries to CCALC: (i) at design-time in order to generate plans that will facilitate them in avoiding run-time conflicts (say), and (ii) at run-time in order to update their plans.
- Postdiction queries. New members of a system may seek to determine the past states of that system. Similar information may be requested by agents that have 'crashed' and resumed their operation. Such information can be produced via the computation of answers to postdiction queries.

The computation of answers to queries may be additionally used to prove properties of the social constraints' specification; in Section 8 we employ CCALC to prove properties of the argumentation protocol specification.

4. Rescher's theory of formal disputation

We describe, specify, prove properties of, and animate an argumentation protocol—a procedure for the resolution of a dispute—based on Brewka's reconstruction [8] of Rescher's Theory of Formal Disputation (RTFD) [53]. We have picked this example because (i) in defining a set repertoire of possible moves for each participant, and their effects, it is typical of the kind of protocols that are encountered in the multi-agent systems (MAS) field, (ii) it provides a concrete and comparatively simple example of a formal procedure for the resolution of disputes, and (iii) Brewka's formalisation in the Situation Calculus provides a natural starting point and basis for comparison. This section briefly presents RTFD. Sections 5 and 6 present Brewka's reconstruction and our variation of RTFD respectively.

According to RTFD, argumentation may be viewed as a three-player game: the *proponent* claims a particular thesis and the *opponent* may question this thesis. The *determiner* decides whether the proponent's thesis was successfully defended or not. The main actions that the participants may perform are the following (p, q below are logical formulas):

- Categorical assertions. These are assertions of the form 'p is the case' and are performed by the proponent.
- Cautious assertions. These are assertions of the form 'p is the case for all you have shown' and are performed by the opponent.
- Provisoed assertions. These assertions are performed by either the proponent or the opponent and are expressed as follows: 'p generally obtains provided that q'. A provisoed assertion of this form is accompanied with either a categorical assertion or a cautious assertion about q.

The argumentation commences with a categorical assertion by the proponent regarding the topic of the argumentation, say p. The opponent may question the topic by either a challenge of the form '$\neg p$ is the case for all you have shown' or by a provisoed denial of the form '$\neg p$ generally obtains provided that q' and 'q is the case for all you have shown'. The argumentation continues in this manner until the topic has been 'accepted' by both the proponent and opponent (the precise meaning of 'accepting' a formula will be presented in the following section), in which case the determiner declares the proponent the winner, or the proponent itself does not accept the topic any more, in which case the determiner declares the opponent the winner. If neither of these alternatives take place and the participants cannot perform any additional reasonable actions, or if a deadline occurs, then the determiner decides about the winner "based on the plausibility of the proponent's claims that were not conceded by the opponent" [8, p. 271]. Rescher's theory exhibits the *silence implies consent* principle. According to this principle, an agent that does not explicitly challenge a claim performed by the other is assumed to concede to the claim.

5. Brewka's reconstruction of RTFD

An argumentation system, according to [8, Definition 4.9], includes as core components a *logic of disputation* and an *argumentation context*. In Brewka's reconstruction, the logic of disputation is preferential default logic [9] and the argumentation context is formalised with the use of a Situation Calculus dialect [52]. The main actions of the protocol are the following: *claiming*, *conceding to*, *retracting*, and *denying* propositions and default rules, *declaring* the winner of the argumentation, and *objecting* to actions performed by the other participants.

The semantics of the protocol actions are given in terms of the *premises*[1] held by the proponent and opponent. The $premise(ag, q, s)$ fluent expresses the formulas q that ag holds explicitly. The related fluent $accepts(ag, q, s)$ is used to represent the formulas that ag holds implicitly: $accepts(ag, q, s)$ expresses that q follows in the logic of disputation L from the premises explicitly held by agent ag in argumentation record s:

$$accepts(ag, q, s) \text{ iff } \{p \mid premise(ag, p, s)\} \vdash_L q \tag{3}$$

An argumentation record is a situation (in the terminology of the Situation Calculus) and so includes the history of the protocol.

The semantics of a *claim* action, for example, of a proposition or a default rule, are given by the following Situation Calculus *effect axiom*:

$$premise(ag, q, do(claim(ag, q), s)) \tag{4}$$

Expression (4) states that the successor situation following the performance of a claim action by ag includes a premise about the claimed proposition (or default). In expression (4) q represents either a proposition or a default rule of the form $n :: a : b/c$ where n is a label associated with the default rule, a is the prerequisite, b is the justification and c is the consequent of the rule [8].

[1] In work on argumentation protocols and dialogue games the term 'commitment' is often used where we say 'premise'. We will not use the term 'commitment' here partly to keep the link with Brewka's account, but also because 'commitment' has another meaning related to obligation which might cause confusion in later sections of the paper.

Brewka distinguishes between 'possible' and 'legal' actions. Possible actions are specified by means of Situation Calculus *possibility axioms*. Consider the following possibility axiom of the retract action:

$$poss(retract(ag, q), s) \leftrightarrow premise(ag, q, s) \tag{5}$$

The above axiom states that it is 'possible' for an agent to retract a proposition (or a default) q if and only if that agent has a premise about that proposition (or default). The conditions that determine whether an action is possible or not are specified in a protocol-independent manner.

'Legal' actions, in contrast to possible actions, are specified in a protocol-dependent manner. Consider the following example of a legal action:

$$legal(declare(det, pro), s) \rightarrow accepts(pro, topic, s) \tag{6}$$

The above expression states that declaring the proponent 'pro' as the winner is legal only if the proponent accepts the topic of the argumentation.

A point of departure of Brewka's reconstruction from Rescher's theory is the introduction of the 'object' action. The participants of an argumentation protocol may perform illegal actions; the effects of an illegal action are the same as if the action were a legal one—provided that no other participant objects to the illegal action. If some participant objects immediately (that is, no other action takes place between the illegal action and the objection), then the effects of the illegal action are 'cancelled'.

The 'object' mechanism is not a new idea in the field of argumentation protocols. Prakken points out that an object mechanism of this type is part of Robert's Rules of Order (RRO): "[t]he general rule is that anything goes until any member objects, after which RRO must be strictly applied" [45, p. 10]. One can find similar mechanisms in most procedures for the conduct of formal debates and disputes.

Enabling agents to object to other agents' actions can lead to a more flexible argumentation protocol. In Brewka's modification of RTFD, for example, the proponent might choose not to object to an illegal action performed by the determiner because it (the proponent) calculates that the illegal action will serve its benefit better than having the illegal action ruled out. However, it can be argued that Brewka's object mechanism is too simplistic, if it is a model of how argumentation processes are actually conducted, and too rigid, if it is a model of how argumentation processes ought to be conducted. Consider for example the case where an agent, say the determiner, repeatedly performs illegal actions. The proponent and opponent have to object to every illegal action performed by the determiner because if they do not object, they implicitly accept the illegal actions as legal ones. In the formalisation to be presented in the following sections, we propose a way to address this issue.

Note that according to Brewka's treatment, an objection will 'undo' the effects of an illegal action if and only if the objection takes place immediately after the illegal action. If for some reason an agent fails to object immediately to an illegal action (say, another action took place between the illegal action and the objection) then it will be considered that this agent does not object and so implicitly agrees to the treatment of the illegal action as a legal one. One way of overcoming this limitation is by specifying that one may effectively object to *Act* at the latest n time-points after *Act*'s performance. Such a specification, however, would raise several complications, such as the following. Assume that *Act* takes place at time t and an agent objects to this action at t' $(t < t' < t + n)$. The effects of the objection may include: (i) undoing the effects of all actions that took place between t and t' (therefore, it would be necessary to keep track of a protocol history fragment), (ii) undoing the effects of *Act*, and (iii) (possibly) applying the effects of all actions that took place between t and t'. We expect that such an object mechanism would be practical only when n is small—in any setting, computerised or not. For this reason we will follow Brewka and assume that objections can only be made to the immediately preceding action. We will allow, however, for the possibility that more than one participant objects to the last action. Further discussion of alternative object mechanisms is beyond the scope of the paper.

Finally, Brewka's reconstruction formalises Rescher's 'silence implies consent' principle as follows: an agent, say the proponent, is assumed to have an explicit premise about a proposition, if the opponent has an explicit premise about this proposition, and as long as it (the proponent) does not deny or retract the proposition.

6. A variation of Brewka's reconstruction of RTFD

Although it is our general aim in this paper to present a reconstruction of Brewka's account of RTFD we make the following adjustments to Brewka's version. We have in mind a setting where autonomous software agents in a MAS engage in the argumentation as part of a negotiation or dispute resolution process. In this setting, the protocol actions would be transmissions of messages. During the argumentation, the proponent, opponent and determiner must perform their chosen actions by specified *deadlines* (timeouts). Without this feature there is no practical way of controlling the exchanges, of determining whether a participant has 'spoken', because otherwise one might have to wait indefinitely for messages to arrive over the communication channels. For similar reasons, it is also necessary to impose a limit on the number of exchanges, or on the total elapsed time for the argumentation process. (Although Brewka states that the argumentation regarding a proposition may terminate due to a deadline, he provides no further details about how this would work and how such deadlines would affect the argumentation process.) Having introduced deadlines, it is necessary to express the conditions in which an action can be said to be timely, that is, whether a participant has 'spoken' by the specified deadline, and so on.

We further refine Brewka's distinction between 'possible' and 'legal' actions. We consider not only what kinds of actions are proper (one possible interpretation of the Situation Calculus predicate *poss*—see, for example, axiom (5)) but also which of these actions each agent will be *practically able* to perform at each stage of a given implementation. Moreover, we express the conditions in which an action can be said to be permitted (one possible interpretation of Brewka's 'legal' actions) or even obligatory. We specify enforcement strategies to deal with the performance of forbidden actions and non-compliance with obligations.

Finally, even though we depart from Brewka's reconstruction of RTFD in the aforementioned points, we maintain (a form of) the object mechanism.

Here then is our variant of Brewka's reconstruction of RTFD. We will refer to it as RTFD*. The argumentation commences when the proponent claims the topic of the argumentation—any other action does not count as the commencement of the protocol. The protagonists (proponent and opponent) then take it in turn to perform actions. Each turn lasts for a specified time period during which the protagonist may perform several actions (send several messages) up to some specified limit. After each such action the other participants are given an opportunity to object within another specified time period. In other words, *Ag*'s action *Act* is followed by a time period during which *Ag* may not perform any actions and the other participants may object to *Act*. The determiner may declare the winner only at the end of the argumentation, that is, when the specified period for the argumentation elapses. (Other specifications of the protocol's terminating conditions are possible—for instance, the argumentation may end earlier than the specified time period if a protagonist concedes to the other's arguments.) If at the end of the argumentation both the proponent and opponent have accepted the topic of the argumentation, then the determiner may only declare the proponent the winner. If, however, the proponent does not accept the topic then the determiner may only declare the opponent the winner. Finally, if the proponent accepts the topic and the opponent does not, the determiner has *discretion* to declare either of them the winner. It may also have an *obligation* to decide one way or the other, depending on which version of the protocol we choose to adopt.

7. Specifying RTFD*

We present a $\mathcal{C}+$ action description D^{RTFD^*} that expresses the specification of RTFD*. Table 1 shows a subset of the action signature $(\sigma^{\text{f}}, \sigma^{\text{a}})$ of D^{RTFD^*}. Variables are written with an upper-case first letter and constants start with a lower-case letter. The intended reading of the constants of $(\sigma^{\text{f}}, \sigma^{\text{a}})$ will be explained during the presentation of the RTFD* specification.

There are three roles in the argumentation protocol: proponent, opponent, and determiner. Although our specification does not rely on the assumption that there is at most one agent occupying a role in any given execution of the protocol, in the concrete example presented we will deal with the usual case where there are three agents, one in each role. In the protocol presented agents do not change role. We will call *pro* the agent that occupies the role of the proponent, *opp* the agent that occupies the role of the opponent and *det* the agent that occupies the role of the determiner. Accordingly, the example action description contains the following three static laws:[2]

$$role\_of(pro) = proponent \text{ if } \top$$
$$role\_of(opp) = opponent \text{ if } \top$$
$$role\_of(det) = determiner \text{ if } \top$$

[2] For brevity, we will omit in the remaining of the paper the keyword caused which appears at the beginning of static and dynamic laws in the original presentation of $\mathcal{C}+$ [22].

Table 1
A subset of the action signature of D^{RTFD^*}

Variables:	Domain:
Ag, Ag'	$\{pro, opp, det\}$
$Protag, Protag'$	$\{pro, opp\}$
Det	$\{det\}$
P, Q	a finite set of propositions/default rules
Act	$\{claim(Protag, Q), concede(Protag, Q), retract(Protag, Q),$ $deny(Protag, Q), declare(Det, Protag)\}$
Rigid Constants:	**Domain:**
$role\_of(Ag)$	$\{proponent, opponent, determiner\}$
$topic$	a finite set of propositions
$implies(P, Q)$	Boolean
Simple Fluent Constants:	**Domain:**
$turn$	$\{proponent, opponent, determiner\}$
$premise(Protag, Q)$	{t, f, u}
$initialState, sanctioned(Ag)$	Boolean
$winner$	$\{pro, opp, none\}$
Statically Determined Fluent Constants:	**Domain:**
$proper(Act), timely(Act), per(Act), obl(Act),$ $objectionable(Act), accepts(Protag, Q), fair$	Boolean
$winning$	$\{pro, opp, none\}$
Action Constants σ^{act} (Boolean):	
$claim(Protag, Q), concede(Protag, Q), retract(Protag, Q),$ $deny(Protag, Q), declare(Det, Protag)$	
Action Constants $\sigma^{\text{a}_{\text{obj}}}$ (Boolean):	
$objected(Ag)$	

The *role_of*(Ag) fluent constants are thus 'rigid', in the sense that their values are the same in all states (the Ag variable ranges over the participants *pro*, *opp* and *det*). In other versions of the protocol, we might introduce other roles, for instance that of an 'observer' who does not participate in the argumentation proper but could be allowed to object to actions made by the protagonists. We do not show that variation here.

Other rigid constants are *topic*, whose value is a proposition expressing the topic of the argumentation, and Boolean fluent constants $implies(P, Q)$, used to represent the underlying logic of disputation, as described in a following section.

Each simple fluent constant of D^{RTFD^*} is inertial, that is to say, its value persists by default from one state to the next. The constraint that a fluent constant f is inertial is expressed in $\mathcal{C}+$ by means of the causal law abbreviation

$$\textsf{inertial}\ f \tag{7}$$

$\mathcal{C}+$ abbreviations were presented in Section 2.

The action constants σ^{a} of D^{RTFD^*} are partitioned into two sets, σ^{act} and $\sigma^{\text{a}_{\text{obj}}}$. The set σ^{act} includes the main actions (*claim*, *concede*, *retract*, *deny*, *declare*) of the argumentation, as summarised in Table 1. For convenience, we specify that exactly one action from σ^{act} takes place at each state transition. This is done by means of the following $\mathcal{C}+$ laws:

$$\textsf{nonexecutable}\ \alpha_i \wedge \alpha_j, \quad \text{for all } \alpha_i, \alpha_j \in \sigma^{\text{act}},\ \alpha_i \neq \alpha_j \tag{8}$$

together with

$$\textsf{nonexecutable}\ \neg\alpha_1 \wedge \neg\alpha_2 \wedge \cdots \wedge \neg\alpha_n \tag{9}$$

where $\alpha_1, \alpha_2, \ldots, \alpha_n$ are the action constants of σ^{act}. The restriction expressed by laws (8) and (9) simplifies the RTFD* specification, and is also convenient when analysing executions of the protocol (see Sections 8 and 9).

The action constants of $\sigma^{a_{obj}}$ are Boolean constants of the form $objected(Ag)$ used to represent that an objection has been made by agent Ag. In the specification presented in this paper we have abstracted away details of how an objection is transmitted within the specified deadline (recall that every action Act is followed by a time period during which no action may take place apart from an objection to Act). Instead, every transition of the transition system defined by D^{RTFD^*} corresponds to a *claim*, *concede*, *retract*, *deny*, or *declare* action by one of the participants together with an indication of whether that action was objected to by one or more of the others. For example, a transition satisfying the action formula

$$claim(pro, Q) \wedge objected(opp) \wedge objected(det)$$

represents a transition in which a claim that Q by *pro* is followed by objections by both *opp* and *det*. Similarly, a transition satisfying

$$claim(pro, Q) \wedge \neg objected(opp) \wedge \neg objected(det)$$

expresses a claim that has not been objected to. We could have produced a more elaborate $\mathcal{C}+$ action description expressing the objection mechanism and its associated deadlines but the details are rather fiddly and we do not present them here so as not to distract attention from the main points of the paper. Notice that because of laws (8) and (9), the $objected(Ag)$ action constants do not need to specify which action has been objected to since there is exactly one such action at each state transition. We do find it useful to specify the agent who objected, and to allow for the possibility that more than one agent objected within the deadline. Other versions of the specification may be constructed that do not include laws (8) and (9) but the representation of objections is then more intricate. We omit the details.

We begin by specifying the well-formed actions of the RTFD* protocol. A $claim(Ag, Q)$ action, for example, is well-formed only if Ag is an agent occupying the role of the proponent or that of the opponent. There are two ways of specifying the well-formed actions. The simpler method, and the one we choose here, is to specify this in the action signature. For instance, the action signature summarised in Table 1 contains action constants of the form $claim(Protag, Q)$ where *Protag* ranges over the two protagonists, *pro* and *opp*. The alternative method is to include ill-formed actions in the action signature and specify explicitly that they are 'non-executable', as in the following example:

$$\textsf{nonexecutable } claim(Ag, Q) \textsf{ if } role\_of(Ag) = determiner$$

According to the above law, there is no transition in the transition system defined by D^{RTFD^*} corresponding to a claim made by an agent occupying the role of determiner. This latter way of dealing with ill-formed actions results in a more generic and flexible specification that would require fewer changes to accommodate participants' changing roles (assuming protocol versions in which this was possible), new agents participating in the protocol, and so on. The former way (which we choose here) results in a specification with simpler laws (they have fewer conditions) and is thus easier to follow.

An objection is not well-formed if an agent objects to its own actions. Accordingly, we include laws of the following form

$$\textsf{nonexecutable } claim(Protag, Q) \wedge objected(Protag) \tag{10}$$

and similarly for the other action constants in σ^{act}. We do not support objections to objections since they are of little practical or theoretical interest.

We now present a number of causal laws expressing when a well-formed protocol action of RTFD* is physically (practically) possible, proper, timely, permitted or obligatory, and what the effects of an action are.

7.1. Physical capability

The *system events* of the RTFD* specification are the timeouts—these are issued by a global clock. (To avoid clutter, several constants of the action signature of D^{RTFD^*}, including those representing timeouts, are not presented in Table 1.) A type of timeout event is used to denote the turn of each participant. When RTFD* commences (this happens when the proponent claims the topic of the argumentation) a global clock starts 'ticking'. The first timeout signals the end of the proponent's turn and the beginning of the opponent's turn to 'speak', by setting $turn = opponent$. The next timeout signals the end of the opponent's turn and the beginning of the proponent's turn, by setting $turn = proponent$, and so on.

The remaining actions of the RTFD* specification are those performed by the protocol participants. We have chosen to specify that any protagonist is always capable of signalling a claim, concede, retract, deny, and object action, and the determiner is always capable of signalling a declare and object action (except that no agent is capable of objecting to its own actions). The effects of these actions are presented next.

At the initial state of the protocol the protagonists have no premises, that is, the value of every $premise(Protag, Q)$ fluent constant is f. The variable Q ranges over the formulas (propositions and default rules) that the two protagonists may claim, concede to, retract, and deny (see Table 1). We assume that a finite number of such formulas can be identified and specified at the outset. This is necessary for implementation in $\mathcal{C}+$ and the CCALC system, though not necessarily in other formalisms.

The protocol commences with the proponent's claim of the topic. The effects of a claim action are expressed as follows:

$$\begin{aligned}&claim(Protag, Q) \text{ causes } premise(Protag, Q) = \mathsf{t} \text{ if}\\&\quad premise(Protag, Q) = \mathsf{f} \wedge\\&\quad \neg objected(pro) \wedge \neg objected(opp) \wedge \neg objected(det)\end{aligned} \tag{11}$$

The above expression is an abbreviation for the $\mathcal{C}+$ fluent dynamic law:

$$\begin{aligned}&premise(Protag, Q) = \mathsf{t} \text{ if } \top \text{ after}\\&\quad claim(Protag, Q) \wedge premise(Protag, Q) = \mathsf{f} \wedge\\&\quad \neg objected(pro) \wedge \neg objected(opp) \wedge \neg objected(det)\end{aligned}$$

Law (11) expresses that *Protag*'s claim of Q leads from a state in which *Protag* has no explicit premise that Q (that is, $premise(Protag, Q) = \mathsf{f}$) to a state in which it does have an explicit premise that Q (that is, $premise(Protag, Q) = \mathsf{t}$), on condition that no (other) agent, *pro*, *opp*, or *det*, objects to the claim. An objection is only effective in blocking the effects of the claim action if it is well-founded (in a sense to be specified below). If the objection is not well-founded then it does not block the effects of the claim action (though it might have other effects, such as exposing the objecting agent to sanctions). We therefore add to law (11) the further constraint that:

$$\begin{aligned}&claim(Protag, Q) \text{ causes } premise(Protag, Q) = \mathsf{t} \text{ if}\\&\quad premise(Protag, Q) = \mathsf{f} \wedge\\&\quad \neg objectionable(claim(Protag, Q))\end{aligned} \tag{12}$$

Boolean fluent constants $objectionable(Act)$ are used to represent that an objection to *Act* is well-founded.

When is an objection to an action *Act* 'well-founded', that is to say, when is $objectionable(Act)$ true? In general, whenever *Act* is not a proper and timely action of the protocol. However, we find it adds flexibility to define *objectionable* separately. Not all improper or untimely actions need to be objectionable. As we explain in the next section, sometimes improper or untimely actions have no effect on the state of the argumentation, and in those circumstances the protocol can be simplified by ignoring objections to them. Specification of the proper, timely, and objectionable actions in RTFD* will be given in the next section.

For convenience we introduce a special abbreviation, writing

$$Act \text{ p\_causes } F \text{ if } G$$

(for 'provisionally causes') as shorthand for the pair of causal laws of the form

$$Act \text{ causes } F \text{ if } G \wedge \neg objected(Ag_1) \wedge \cdots \wedge \neg objected(Ag_n)$$

$$Act \text{ causes } F \text{ if } G \wedge \neg objectionable(Act)$$

where $Ag_1, \ldots, Ag_n$ are the participants in the argumentation (*pro*, *opp*, and *det* in the present example). When G is $\top$ we write

$$Act \text{ p\_causes } F$$

The laws (11) and (12) stating the effects of a claim can thus be written succinctly as follows:

$$\begin{array}{l} \mathit{claim}(\mathit{Protag}, Q)\ \mathsf{p\_causes}\ \mathit{premise}(\mathit{Protag}, Q) = \mathsf{t}\ \text{ if} \\ \qquad \mathit{premise}(\mathit{Protag}, Q) = \mathsf{f} \end{array} \tag{13}$$

Suppose that protagonist *Protag* claims a proposition Q. Opponent *Protag′* may respond to *Protag*'s claim by conceding to, or denying the claim. If *Protag′* does neither then we say that *Protag′* has an 'unconfirmed' premise that Q, denoted by $\mathit{premise}(\mathit{Protag}', Q) = \mathsf{u}$. The value of a *premise* fluent constant is set to 'unconfirmed' as follows, for every pair of distinct protagonists *Protag* and *Protag′*:

$$\begin{array}{l} \mathit{claim}(\mathit{Protag}, Q)\ \mathsf{p\_causes}\ \mathit{premise}(\mathit{Protag}', Q) = \mathsf{u}\ \text{if} \\ \qquad \mathit{premise}(\mathit{Protag}, Q) = \mathsf{f} \wedge \\ \qquad \mathit{premise}(\mathit{Protag}', Q) = \mathsf{f} \end{array} \tag{14}$$

In other words, *Protag*'s claim of Q leads (subject to possible objections) to a state in which *Protag′* has an unconfirmed premise that Q, provided that *Protag′* does not already have a premise that Q (that is, provided that $\mathit{premise}(\mathit{Protag}', Q) = \mathsf{f}$). If *Protag′* already has a premise that Q (that is, $\mathit{premise}(\mathit{Protag}', Q) = \mathsf{t}$) then its premise does not become unconfirmed, and it does not need to respond to *Protag*'s claim.

Responding to a claim, that is, conceding to or denying a claim, can be expressed as follows:

$$\begin{array}{l} \mathit{concede}(\mathit{Protag}, Q)\ \mathsf{p\_causes}\ \mathit{premise}(\mathit{Protag}, Q) = \mathsf{t}\ \text{if} \\ \qquad \mathit{premise}(\mathit{Protag}, Q) = \mathsf{u} \end{array} \tag{15}$$

$$\begin{array}{l} \mathit{deny}(\mathit{Protag}, Q)\ \mathsf{p\_causes}\ \mathit{premise}(\mathit{Protag}, Q) = \mathsf{f}\ \text{if} \\ \qquad \mathit{premise}(\mathit{Protag}, Q) = \mathsf{u} \end{array} \tag{16}$$

A claim may be retracted. The effects of a retraction are twofold:

$$\begin{array}{l} \mathit{retract}(\mathit{Protag}, Q)\ \mathsf{p\_causes}\ \mathit{premise}(\mathit{Protag}, Q) = \mathsf{f}\ \text{if} \\ \qquad \mathit{premise}(\mathit{Protag}, Q) = \mathsf{t} \end{array} \tag{17}$$

$$\begin{array}{l} \mathit{retract}(\mathit{Protag}, Q)\ \mathsf{p\_causes}\ \mathit{premise}(\mathit{Protag}', Q) = \mathsf{f}\ \text{if} \\ \qquad \mathit{premise}(\mathit{Protag}, Q) = \mathsf{t} \wedge \\ \qquad \mathit{premise}(\mathit{Protag}', Q) = \mathsf{u} \end{array} \tag{18}$$

for every pair of distinct protagonists *Protag* and *Protag′*. According to law (17), *Protag*'s retraction of its claim that Q results in $\mathit{premise}(\mathit{Protag}, Q) = \mathsf{f}$; according to law (18) this retraction also removes the unconfirmed premise held by the other protagonist, *Protag′*. If *Protag′* responded to *Protag*'s claim before *Protag* retracted it then the value of $\mathit{premise}(\mathit{Protag}', Q)$ would not be unconfirmed, and in that case *Protag*'s retraction will not affect the premise held by *Protag′*.

The effects of laws (13)–(18) for a single proposition Q, and assuming no objections, are summarised in Fig. 2. Notice that, because the *premise* fluent constants are inertial, *claim*, *concede*, *deny*, and *retract* actions have no effect on the argumentation state if the pre-conditions in laws (13)–(18) are not satisfied. Transitions corresponding to such actions are omitted from the diagram for clarity.

At the end of the argumentation the determiner may declare the winner—the effects of this action can be expressed as follows:

$$\mathit{declare}(\mathit{Det}, \mathit{Protag})\ \mathsf{p\_causes}\ \mathit{winner} = \mathit{Protag} \tag{19}$$

Det represents the agent occupying the role of the determiner. The fluent constant $\mathit{winner} = \mathit{Protag}$ expresses that the winner of the argumentation is *Protag*. The use of p_causes here again deals with the possibility that some agent other than *Det* objects to the declaration.

Laws (13)–(19) express the effects of the main protocol actions. Suppose that a protagonist claims a proposition Q when it already has a premise that Q, or that it sends a $\mathit{retract}(\mathit{Protag}, Q)$ message when it has no premise that Q. Such actions are not proper, according to the protocol. Moreover, an action may be untimely, as when, for instance, a protagonist speaks out of turn. Some improper or untimely actions have no effects on the state of the argumentation. Others do have effects—however, their effects may be blocked by objections. In the next section we specify when a protocol action is proper and timely, and we specify when an action is objectionable.

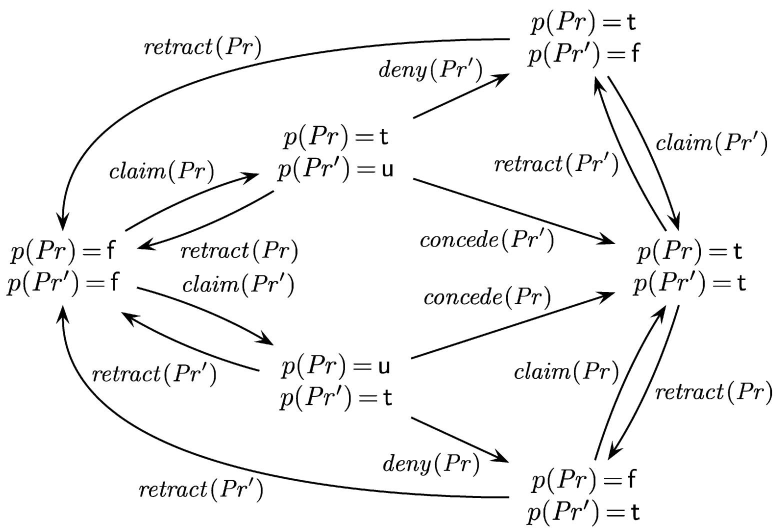

Fig. 2. Effects of *claim*, *concede*, *deny*, *retract* actions. The proposition Q is omitted for clarity. Pr, Pr' are shorthands for $Protag$, $Protag'$ respectively. Similarly, $p(Pr)$, $p(Pr')$ are shorthands for $premise(Protag, Q)$ and $premise(Protag', Q)$. Actions that do not change the protocol state are not shown in the diagram.

Table 2
Proper and timely actions in D^{RTFD^*}

Action	proper	timely
$claim(Protag, Q)$	$premise(Protag, Q) = \mathsf{f}$	$(\neg initialState \vee topic = Q) \wedge turn = role\_of(Protag)$
$concede(Protag, Q)$	$premise(Protag, Q) = \mathsf{u}$	$\neg initialState \wedge turn = role\_of(Protag)$
$retract(Protag, Q)$	$premise(Protag, Q) = \mathsf{t}$	$\neg initialState \wedge turn = role\_of(Protag)$
$deny(Protag, Q)$	$premise(Protag, Q) = \mathsf{u}$	$\neg initialState \wedge turn = role\_of(Protag)$
$declare(Det, Protag)$	$winner \neq Protag \wedge (winning = Protag \vee winning = none)$	$turn = role\_of(Det)$

7.2. Proper and timely actions

The second column of Table 2 shows the conditions in which the main protocol actions are said to be proper. A $claim(Protag, Q)$ action, for example, is proper if and only if $Protag$ does not have a premise that Q, that is, $premise(Protag, Q) = \mathsf{f}$:

$$proper(claim(Protag, Q)) \text{ iff } premise(Protag, Q) = \mathsf{f} \tag{20}$$

In other words, a $claim(Protag, Q)$ action is improper when:

- $premise(Protag, Q) = \mathsf{t}$; it is improper to make repeated claims, that is, claim something one has a premise about.
- $premise(Protag, Q) = \mathsf{u}$; this signifies that the other protagonist $Protag'$ has claimed Q (see law (14)). A proper response to the other's claim is either a concession or a denial (see Table 2).

A $retract(Protag, Q)$ action is proper if and only if $premise(Protag, Q) = \mathsf{t}$; otherwise there is nothing to retract.

The reader may have noticed that the conditions for proper *claim*, *concede*, *deny*, and *retract* actions coincide with the *premise*(*Protag*, *Q*) pre-conditions in the laws (13)–(18). This is a feature of Brewka's version of RTFD (or rather, our chosen formulation of some of its details) and will not necessarily be the case in other examples. In other argumentation and dialogue protocols it is common, for example, to say that a claim by *Protag* is not proper if it is inconsistent, in the underlying logic of disputation, with the premises currently held by *Protag*. This is easily added in our formulation but is not a feature of Brewka's RTFD. Other formulations of proper actions will be discussed in Section 10.

We now turn to declarations. A declaration is proper when the following conditions are satisfied:

$$\begin{aligned}&\mathit{proper}(\mathit{declare}(\mathit{Det},\mathit{Protag}))\ \textsf{iff}\\&\qquad \mathit{winner}\neq\mathit{Protag}\ \wedge\\&\qquad(\mathit{winning}=\mathit{Protag}\ \vee\ \mathit{winning}=\mathit{none})\end{aligned}\tag{21}$$

In words: declaring *Protag* the winner of the argumentation is proper if and only if:

- *Protag* has not already been declared winner, and
- *Protag* is currently 'winning', that is, the conflict is resolved in favour of *Protag*, or no protagonist is 'winning', that is, the conflict is unresolved.

The statically defined fluent constant *winning* is defined as follows:

$$\begin{aligned}&\mathit{winning}=\mathit{Protag}\ \textsf{if}\\&\qquad \mathit{role\_of}(\mathit{Protag})=\mathit{proponent}\ \wedge\\&\qquad \mathit{role\_of}(\mathit{Protag}')=\mathit{opponent}\ \wedge\\&\qquad \mathit{topic}=Q\ \wedge\\&\qquad \mathit{accepts}(\mathit{Protag},Q)\ \wedge\\&\qquad \mathit{accepts}(\mathit{Protag}',Q)\end{aligned}\tag{22}$$

$$\begin{aligned}&\mathit{winning}=\mathit{Protag}'\ \textsf{if}\\&\qquad \mathit{role\_of}(\mathit{Protag})=\mathit{proponent}\ \wedge\\&\qquad \mathit{role\_of}(\mathit{Protag}')=\mathit{opponent}\ \wedge\\&\qquad \mathit{topic}=Q\ \wedge\\&\qquad \neg\mathit{accepts}(\mathit{Protag},Q)\end{aligned}\tag{23}$$

$$\textsf{default}\ \mathit{winning}=\mathit{none}\tag{24}$$

Laws (22) and (23) express a resolved dispute. An *accepts*(*Protag*, *Q*) fluent constant states that *Q* follows, in the logic of disputation, from *Protag*'s explicit premises. (A discussion of the logic of disputation is presented in Section 7.5.) Law (22) deals with the case that both protagonists accept the topic *Q* of the dispute; law (23) deals with the case where the proponent no longer accepts the topic *Q*. Law (24) corresponds to an unresolved dispute; in the absence of information to the contrary (that is, laws (22) and (23)), the dispute is unresolved.

Even if proper, an action may not be in compliance with the protocol specification—for example, it may be untimely. The third column of Table 2 shows the conditions in which the main protocol actions are said to be timely. At the initial state of the protocol, expressed by the simple fluent constant *initialState*, only the proponent's claim of the argumentation topic is timely (at the initial protocol state *turn* = *proponent*). At the other protocol states a protagonist's claim, concession, retraction, or denial is timely if it is the protagonist's turn to 'speak'. A timely claim, for instance, is expressed in $\mathcal{C}+$ as follows:

$$\begin{aligned}&\mathit{timely}(\mathit{claim}(\mathit{Protag},Q))\ \textsf{iff}\\&\qquad(\neg\mathit{initialState}\vee\mathit{topic}=\ Q)\ \wedge\\&\qquad \mathit{turn}=\mathit{role\_of}(\mathit{Protag})\end{aligned}\tag{25}$$

Declarations are defined to be timely as follows:

$$\mathit{timely}(\mathit{declare}(\mathit{Det},\mathit{Protag}))\ \textsf{iff}\ \mathit{turn}=\mathit{role\_of}(\mathit{Det})\tag{26}$$

A declaration is timely if and only if it is the determiner's turn to 'speak'—this happens when the argumentation time elapses.

Table 3
Objectionable actions in D^{RTFD^*}

Action	objectionable
$claim(Protag, Q)$	$proper(claim(Protag, Q)) \wedge \neg timely(claim(Protag, Q))$
$concede(Protag, Q)$	$proper(concede(Protag, Q)) \wedge \neg timely(concede(Protag, Q))$
$retract(Protag, Q)$	$proper(retract(Protag, Q)) \wedge \neg timely(retract(Protag, Q))$
$deny(Protag, Q)$	$proper(deny(Protag, Q)) \wedge \neg timely(deny(Protag, Q))$
$declare(Det, Protag)$	$\neg(proper(declare(Det, Protag)) \wedge timely(declare(Det, Protag))) \wedge winner \neq Protag$

In this version we do not allow for early declarations, in the case where both protagonists accept the argumentation topic, or the proponent does not accept it, before the argumentation time elapses; this can be easily adjusted.

What about improper declarations? Or untimely actions? Some of these actions do have effects on the protocol state—however, these effects may be blocked by a well-founded objection. Table 3 presents the conditions in which an objection to *Act* is well-founded, that is, when *Act* is objectionable. In the present example a claim is objectionable if and only if it is untimely and proper:

$$\begin{aligned}&objectionable(claim(Protag, Q)) \text{ iff}\\&\qquad proper(claim(Protag, Q)) \wedge \neg timely(claim(Protag, Q))\end{aligned} \tag{27}$$

According to the protocol specified above, improper claims do not have effects on the protocol state. It is therefore not necessary to object to them. Similarly, a concession, retraction or denial is objectionable if and only if it is untimely and proper.

A $declare(Det, Protag)$ action is objectionable if and only if: (i) it is not proper and timely, and (ii) $winner \neq Protag$ (see Table 3). Notice that if $winner = Protag$ at the time of a $declare(Det, Protag)$ action then this action will not have any effects on the value of the *winner* fluent constant. Accordingly, we choose to say that an improper or untimely $declare(Det, Protag)$ action is objectionable only when $winner \neq Protag$ at the time of the declaration.

We presented the conditions in which an action is considered proper or timely, and the consequences of the performance a proper/improper, timely/untimely action. In the following section we focus on a separate issue: the circumstances in which a protocol participant is permitted or even obliged to perform a particular action. In Section 7.4 we discuss potential consequences of the performance of forbidden actions and non-compliance with obligations.

7.3. Permitted actions

In Section 5 we identified a side-effect of the object mechanism: *Ag*'s repeated performance of objectionable actions requires repeated objections from some other participant Ag' otherwise the effects of *Ag*'s objectionable actions are not blocked. One way of addressing this issue is to say that the performance of repeated objectionable actions is not *permitted*. Consider the following example specification of permission:

$$\begin{aligned}&per(concede(Protag, Q)) \text{ iff}\\&\qquad \neg objectionable(concede(Protag, Q)) \vee\\&\qquad objActions(Protag) < 10\end{aligned} \tag{28}$$

Here, there is an upper limit (10 in the example) on the total number of objectionable actions a protagonist *Protag* is permitted to make. The *per* fluent constants express permission; the simple fluent constants $objActions(Ag)$ record the number of objectionable actions issued by *Ag*. The formulation of $\mathcal{C}+$ laws to maintain the $objActions(Ag)$ fluent constants is very straightforward and we omit the details. Laws similar to (28) express the permission to claim, retract, deny and declare (we may impose a lower threshold for the permitted objectionable declarations). In the present example, an objection is never objectionable; therefore, an objection is always permitted.

In another formulation, $objActions(Ag)$ could express the number of objectionable actions issued by *Ag* to which well-founded (successful) objections have been made by other participants. In this case, *Ag* would be permitted to perform objectionable actions until the number of successful objections issued by the other participants to *Ag*'s actions has reached the specified limit.

We could formulate further permission laws to classify as not permitted other kinds of exchanges, such as repeated claims and retractions by the protagonists ('yes it is'; 'no it isn't'; 'yes it is'; 'no it isn't'; ...). Although such an exchange could be both proper and timely according to the protocol, it may nevertheless be undesirable—it does not progress the resolution of the dispute.

In some cases it is meaningful to say that the determiner is not only permitted, but obliged to declare a protagonist the winner of the dispute. Such an obligation will arise when the following conditions hold:

$$\begin{array}{l} \mathit{obl}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \textsf{ iff} \\ \quad \mathit{turn} = \mathit{role\_of}(\mathit{Det}) \wedge \\ \quad \mathit{winner} \neq \mathit{Protag} \wedge \\ \quad \mathit{winning} = \mathit{Protag} \end{array} \tag{29}$$

(The *obl* fluent constants express obligation.) According to law (29), it is obligatory to declare *Protag* the winner of the dispute if and only if: (i) it is the determiner's turn to 'speak', (ii) *Protag* has not already been declared the winner, and (iii) *Protag* is currently 'winning' the dispute, that is, the dispute is resolved in favour of *Protag*.

Given expression (29), it is necessary to represent explicitly the relationship between the permission and obligation to declare the winner. For instance, the protocol should never reach a state in which both $\mathit{obl}(\mathit{declare}(\mathit{Det}, \mathit{Protag}))$ and $\mathit{per}(\mathit{declare}(\mathit{Det}, \mathit{Protag}'))$ are true for $\mathit{Protag} \neq \mathit{Protag}'$. One way is to add the following laws, for every pair of distinct protagonists *Protag* and *Protag′*:

$$\begin{array}{l} \mathit{per}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \textsf{ if} \\ \quad \mathit{obl}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \end{array} \tag{30}$$

$$\begin{array}{l} \mathit{per}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \textsf{ if} \\ \quad \neg \mathit{obl}(\mathit{declare}(\mathit{Det}, \mathit{Protag}')) \wedge \\ \quad \neg \mathit{objectionable}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \end{array} \tag{31}$$

$$\begin{array}{l} \mathit{per}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \textsf{ if} \\ \quad \neg \mathit{obl}(\mathit{declare}(\mathit{Det}, \mathit{Protag}')) \wedge \\ \quad \mathit{objActions}(\mathit{Det}) < 10 \end{array} \tag{32}$$

$$\textsf{default } \neg \mathit{per}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \tag{33}$$

According to laws (30)–(33), if the determiner is obliged to declare *Protag* the winner then is it is also permitted to declare *Protag*, but not *Protag′*. When there is no obligation on the determiner, at most ten objectionable declarations are permitted (that is, the permission to declare is expressed as the permission to perform any other protocol action).

It is not meaningful to associate obligations with the remaining protocol actions; therefore, we do not need to update the specification of permitted claims, concessions, retractions, denials or objections. Table 4 presents the conditions in which a main protocol action is permitted or obligatory. Clearly, different specifications are possible. One

Table 4
Permission and obligation in $D^{\mathrm{RTFD}*}$

Action	per	obl
$\mathit{claim}(\mathit{Protag}, Q)$	$\neg \mathit{objectionable}(\mathit{claim}(\mathit{Protag}, Q)) \vee \mathit{objActions}(\mathit{Protag}) < 10$	$\bot$
$\mathit{concede}(\mathit{Protag}, Q)$	$\neg \mathit{objectionable}(\mathit{concede}(\mathit{Protag}, Q)) \vee \mathit{objActions}(\mathit{Protag}) < 10$	$\bot$
$\mathit{retract}(\mathit{Protag}, Q)$	$\neg \mathit{objectionable}(\mathit{retract}(\mathit{Protag}, Q)) \vee \mathit{objActions}(\mathit{Protag}) < 10$	$\bot$
$\mathit{deny}(\mathit{Protag}, Q)$	$\neg \mathit{objectionable}(\mathit{deny}(\mathit{Protag}, Q)) \vee \mathit{objActions}(\mathit{Protag}) < 10$	$\bot$
$\mathit{declare}(\mathit{Det}, \mathit{Protag})$	$\mathit{obl}(\mathit{declare}(\mathit{Det}, \mathit{Protag}) \vee (\neg \mathit{obl}(\mathit{declare}(\mathit{Det}, \mathit{Protag}')) \wedge (\neg \mathit{objectionable}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \vee \mathit{objActions}(\mathit{Det}) < 10))$	$\mathit{turn} = \mathit{role\_of}(\mathit{Det}) \wedge \mathit{winner} \neq \mathit{Protag} \wedge \mathit{winning} = \mathit{Protag}$

may forbid, for instance, repetitive improper claims, concessions, retractions, and denials, and ill-founded objections (objections to *Act* when *Act* is not objectionable). Although these actions have no effects on the protocol state and thus are not objectionable, they may nevertheless be forbidden (not permitted) in order to deter participants from performing 'meaningless' actions, to decrease network traffic. In [4] we presented an RTFD* specification in which all retractions were forbidden, whether proper/improper, timely/untimely, objectionable/non-objectionable. Furthermore, the obligation to declare the winner could arise even in the case of an unresolved dispute.

Note that it is practically possible for an agent to perform forbidden actions and not comply with its obligations. We introduce enforcement strategies as a way of dealing with this type of behaviour. Such strategies are presented next.

7.4. Enforcement strategies

We want to reduce or eliminate:

- The protagonists' performance of forbidden actions, repeated objectionable actions in this example.
- The determiner's non-compliance with the obligation to declare the winner of the dispute. (To simplify the presentation, we will not address the determiner's forbidden declarations.)

We employ the simple fluent constants *sanctioned* to identify the aforementioned types of behaviour. *Protag*'s forbidden claim, for example, results in initiating *sanctioned*(*Protag*):

$$\begin{array}{l} \mathit{claim}(\mathit{Protag}, Q) \text{ causes } \mathit{sanctioned}(\mathit{Protag}) \text{ if} \\ \quad \neg \mathit{per}(\mathit{claim}(\mathit{Protag}, Q)) \end{array} \tag{34}$$

One way of reducing the performance of forbidden actions and non-compliance with obligations is by penalising such behaviour. For example: if at the close of the argumentation the dispute is unresolved, we could say that declaring *Protag* the winner is objectionable if *Protag* performed some forbidden actions but *Protag*$'$ did not. In this case, the penalty *Protag* pays for its forbidden actions is that it will not win an unresolved dispute if *Protag*$'$: (i) does not perform forbidden actions, and (ii) objects to a potential *declare*(*Det*, *Protag*) action. (This does not necessarily imply that *Protag*$'$ will win the dispute.) The specification of this example sanction may expressed by updating the definition of objectionable declarations as follows (for every pair of distinct protagonists *Protag* and *Protag*$'$):

$$\begin{array}{l} \mathit{objectionable}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \text{ if} \\ \quad \neg \mathit{proper}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \wedge \\ \quad \mathit{winner} \neq \mathit{Protag} \end{array} \tag{35}$$

$$\begin{array}{l} \mathit{objectionable}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \text{ if} \\ \quad \neg \mathit{timely}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \wedge \\ \quad \mathit{winner} \neq \mathit{Protag} \end{array} \tag{36}$$

$$\begin{array}{l} \mathit{objectionable}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \text{ if} \\ \quad \mathit{winning} = \mathit{none} \wedge \\ \quad \mathit{winner} \neq \mathit{Protag} \wedge \\ \quad \mathit{sanctioned}(\mathit{Protag}) \wedge \\ \quad \neg \mathit{sanctioned}(\mathit{Protag}') \end{array} \tag{37}$$

$$\text{default } \neg \mathit{objectionable}(\mathit{declare}(\mathit{Det}, \mathit{Protag})) \tag{38}$$

Laws (35), (36) and (38) express the definition of objectionable declarations presented in Table 3; law (37) expresses the protagonists' sanctions. When *turn* = *determiner* (the argumentation has ended), *winning* = *none* (the dispute is unresolved) and *winner* = *none* (no protagonist is declared winner), declaring *Protag* would be proper, timely and non-objectionable. If, however, *Protag* is sanctioned and *Protag*$'$ is not, declaring *Protag* would be proper, timely but objectionable. (If both protagonists are sanctioned then objectionable declarations are defined only by laws (35), (36) and (38), that is, proper, timely declarations are non-objectionable.)

Similarly, we could have expressed the protagonists' sanctions in terms of objectionable claims, concessions, retractions, and so on. For example, we could have specified that the retractions of a sanctioned protagonist are objectionable, even if proper and timely, and therefore, their effects could be blocked by objections.

In addition to dealing with the protagonists' forbidden actions, we want to reduce the possibility that the determiner will not comply with its obligation. Assuming that the determiner is obliged to declare *Protag* the winner of the dispute (see law (29)), the determiner may:

- declare *Protag*′ the winner (more precisely, the last declaration before the last protocol timeout concerns *Protag*′),
- make no declaration, or
- declare *Protag* the winner.

In the third case the obligation is (temporarily) discharged. New evidence, however, may arise after the declaration (and before the last protocol timeout), such as an untimely retraction by *Protag*, obliging the determiner to declare *Protag*′ the winner (by setting $winning = Protag'$) and this obligation may not be discharged.

Notice that the employed object mechanism is inadequate for ensuring a 'fair' result in the three aforementioned scenarios. (A 'fair' result implies that *Protag* is declared winner if the dispute is resolved in its favour.) In the first case, a well-founded objection to the action $declare(Det, Protag')$ will block the effects of the declaration but will not force $winner = Protag$. In the second case there is no declaration to object to, and in the third case the declaration was not objectionable at the time.

What sanctions should be enforced in these cases? There are a number of possibilities. We may specify, for instance, that a sanctioned determiner is disqualified from acting as a determiner in future argumentations, for a specified time period perhaps. If the argumentation takes place in the context of a computational system including several protocols, such as a system of negotiation or deliberative assemblies, a sanctioned determiner may have restricted permissions or its actions may be objectionable when participating in the remaining protocols, possibly occupying other roles. In addition, we may specify that it is proper and timely to initiate proceedings against a sanctioned determiner (assuming the existence of an adjudicating authority) in order to enforce compliance with the determiner's obligation, not as a 'punishment' to the determiner but as a way of discouraging it to avoid complying with the protocol rules and ensuring 'fairness'.

Sanctions are one means by which the performance of forbidden actions and non-compliance with obligations may be addressed. Another possible strategy is to devise physical controls that will force agents to comply with the protocol rules. For instance, repetitive objectionable actions may be physically blocked (since, in the present example, they are forbidden). The general strategy of designing mechanisms to force compliance and eliminate non-permitted behaviour is what Jones and Sergot [26] referred to as *regimentation*. Regimentation devices have often been employed in order to eliminate 'anti-social' behaviour in computational systems (see, for instance, [28,37,54]). It has been argued [26], however, that regimentation is rarely desirable (it results in a rigid system that may discourage agents from entering it [45]), and not always practical. In any case, violations may still occur even when regimenting a computational system (consider, for instance, a faulty regimentation device). For all of these reasons, we have to allow for sanctioning and not rely exclusively on regimentation mechanisms.

7.5. Additional considerations

We presented a formalisation of the RTFD* protocol rules. In order to perform computational experiments (such as those presented in Section 9), we need to code up (fragments of) the logic of disputation. The following law, for example, states that a protagonist accepts all (classical) logical implications of each of its premises:

$$\begin{aligned} &accepts(Protag, Q) \text{ iff} \\ &\qquad premise(Protag, P) = \mathsf{t} \wedge implies(P, Q) \end{aligned} \tag{39}$$

The *implies* here are simply suitably chosen rigid constants. In a similar manner, we may specify the acceptance of propositions as a result of the conjunctions of an agent's premises and of premises regarding default rules.

We have formalised only a small fragment of the logic of disputation, enough to conduct simple experiments. The focus of this paper lies on the protocol rules rather than the logic of disputation (that, we assume, does not necessarily have to be prioritised default logic as in Brewka's version). Moreover, a complete formalisation of the logic of disputation (on top of a formalisation of the protocol rules) would substantially increase the number of D^{RTFD^*} laws, thus significantly increasing CCALC's time of computing answers to queries (regarding D^{RTFD^*}). These complications could be addressed by employing alternative action languages. We discuss this issue in Section 11.

Brewka, in his reconstruction of RTFD, places emphasis on the formalisation of the 'silence implies consent' principle: a protagonist that does not explicitly challenge a claim by the other protagonist is assumed to concede to the claim. We may incorporate this principle in RTFD* by modifying the formalisation of the logic of disputation as follows:

$$\begin{aligned}&\mathit{accepts}(\mathit{Protag}, Q) \text{ iff}\\ &\quad(\mathit{premise}(\mathit{Protag}, P) = \mathsf{t} \vee \mathit{premise}(\mathit{Protag}, P) = \mathsf{u}) \wedge\\ &\quad\mathit{implies}(P, Q)\end{aligned} \tag{40}$$

Recall that $\mathit{premise}(\mathit{Protag}, P) = \mathsf{u}$ expresses that *Protag* has an unconfirmed premise that P, that is, *Protag* has not responded to a claim that P made by the other protagonist. In a protocol adopting the 'silence implies consent' principle, a protagonist accepts all logical implications of each of its explicit *and unconfirmed* premises.

Note that if we do not want to incorporate the 'silence implies consent' principle then we disregard unconfirmed premises in the logic of disputation, by using law (39) instead of (40).

8. Proving properties of RTFD*

The explicit transition systems semantics of the $\mathcal{C}+$ language enables us to prove various properties of the presented RTFD* specification (which is expressed by means of the definite action description D^{RTFD^*}). We may prove, for example, that as long as the determiner complies with its obligations the protocol result will be 'fair'. Suppose we specify 'fairness' as follows:

$$\mathit{fair} \text{ iff } \mathit{winning} = \mathit{none} \ \vee\ \mathit{winner} = \mathit{winning} \tag{41}$$

The statically determined fluent constant *fair* holds in a protocol state if and only if the dispute is unresolved ($\mathit{winning} = \mathit{none}$) or the dispute is resolved in favour of the declared winner ($\mathit{winner} = \mathit{winning}$). We want to examine whether or not a protocol result is 'fair', that is, whether or not the fluent constant *fair* is true in the final state of a protocol execution.

Proposition 1. *The protocol result will be 'fair' if and only if in the final protocol state there is no obligation on the determiner to declare.*

Proof. First we will prove that if in the final protocol state there is no obligation on the determiner then the protocol result will be 'fair'. Assume a final protocol state s in which there is no obligation on the determiner and the result is not 'fair':

$$s \not\models \mathit{obl}(\mathit{declare}(\mathit{det}, \mathit{Protag})), \quad \text{for any } \mathit{Protag}$$

$$s \models \mathit{winning} \neq \mathit{none} \wedge \mathit{winner} \neq \mathit{winning} \wedge \mathit{turn} = \mathit{role\_of}(\mathit{det})$$

(Recall that $\mathit{role\_of}(\mathit{det}) = \mathit{determiner}$ is 'rigid'. Moreover, in a final protocol state $\mathit{turn} = \mathit{determiner}$.)

Now, $s \models \mathit{winning} \neq \mathit{none}$ iff $s \models \mathit{winning} = \mathit{Protag}$ for some protagonist *Protag*, and so $s \models \mathit{winning} = \mathit{Protag} \wedge \mathit{winner} \neq \mathit{Protag}$. Since s is a state of D^{RTFD^*}, it is an interpretation of σ^{f} such that $s = T_{\mathrm{static}}(s) \cup \mathit{Simple}(s)$, where $T_{\mathrm{static}}(s) =_{\mathrm{def}} \{F \mid \text{static law '}F \text{ if } G\text{' is in } D^{\mathrm{RTFD}^*}, s \models G\}$ and $\mathit{Simple}(s)$ denotes the set of simple fluent atoms satisfied by s (see Section 2.3). From law (29) and the fact that

$$s \models \mathit{turn} = \mathit{role\_of}(\mathit{det}) \wedge \mathit{winning} = \mathit{Protag} \wedge \mathit{winner} \neq \mathit{Protag}$$

we have that $\mathit{obl}(\mathit{declare}(\mathit{det}, \mathit{Protag})) \in T_{\mathrm{static}}(s)$.

According to our initial assumption, however, there is no obligation on the determiner in s, which implies that $s \neq T_{\mathrm{static}}(s) \cup \mathit{Simple}(s)$. Therefore, s is not a state of D^{RTFD^*}.

Second we will prove that if the protocol result is 'fair' then in the final state there is no obligation on the determiner to declare. Assume a final protocol state s in which there is an obligation on the determiner to declare and the result is 'fair'. If *fair* is true in s then either $\mathit{winning} = \mathit{none}$ is in s or $\mathit{winner} = \mathit{winning}$ is in s. Consider first the case $\mathit{winning} = \mathit{none}$:

$$s \models \mathit{obl}(\mathit{declare}(\mathit{det}, \mathit{Protag})) \wedge \mathit{winning} = \mathit{none} \wedge \mathit{turn} = \mathit{role\_of}(\mathit{det})$$

for some *Protag*. Given law (29), we have that $obl(declare(det, Protag)) \notin T_{\text{static}}(s)$. Moreover, since *obl* are statically determined fluent constants, we also have that $obl(declare(det, Protag)) \notin Simple(s)$. Therefore, $s \neq T_{\text{static}}(s) \cup Simple(s)$ and s is not a state of D^{RTFD^*}.

The proof that s is not a state of D^{RTFD^*} for the case in which $winner = winning$ is similar. □

The obligation to declare might never arise during a protocol execution. Proposition 1 shows that in this case the result will be 'fair'. If an obligation to declare does arise, however, the protocol result is also guaranteed to be 'fair', as long as the determiner complies with the obligation.

CCALC provides an automated means for proving properties of a protocol specification. We express the $\mathcal{C}+$ action description D^{RTFD^*} in CCALC's input language and then query CCALC about D^{RTFD^*} to prove properties of the RTFD* specification. (The types of query that CCALC computes were presented in Section 3. Details of CCALC's input language may be found in [1,32].) Consider the following example:

Proposition 2. *A proper, timely concede(Protag, Q) action always leads to a state in which Protag has an explicit premise that Q.*

We instruct CCALC to compute all states s' such that

- (s, ε, s') is a transition of D^{RTFD^*},
- $s \models proper(concede(Protag, Q)) \wedge timely(concede(Protag, Q))$, and
- $\varepsilon \models concede(Protag, Q)$.

For every state s' computed by CCALC we obtain

$$s' \models premise(Protag, Q) = \mathsf{t}$$

This is because, briefly, a proper and timely concession is not objectionable in the presented RTFD* (see Table 3). Therefore, the effects of such an action, expressed by law (15), cannot be blocked. In other words, even when

$$\varepsilon \models concede(Protag, Q) \wedge objected(Ag)$$

the resulting state s' always includes $premise(Protag, Q) = \mathsf{t}$.

More details about the computational experiments performed with CCALC on D^{RTFD^*} are presented in the following section.

We may prove further properties of the RTFD* specification, in the manner shown above, such as that the effects of any non-objectionable action cannot be blocked, an objection always blocks the effects of an objectionable action, there is always a permitted action for a participant when it is its turn to 'speak', performing more objectionable actions than what is specified is always forbidden and sanctioned, the determiner is never obliged and forbidden to declare the winner, and so on (the last two properties may be viewed as 'discouragement of disruption' and 'rule-consistency' [36]).

In Section 11 we discuss alternative techniques for proving properties of a protocol specification.

9. Executing RTFD*

Proving properties of a specification can be seen as a design-time activity. For instance, protocol designers may wish to prove properties of a protocol specification in order to determine whether or not this specification meets their requirements. Additionally, agents (or their designers) may wish to prove various properties of a protocol specification when deciding whether to enter (deploy their agents in) that protocol.

At run-time, we may execute a protocol specification to provide, amongst other things, information about the protocol state current at each time. Computation of such information is a special case of a prediction query (see Section 3). A protocol state—which actions are proper, timely, objectionable, permitted, and so on—may be publicised to (a subset of) the protocol participants, or their designers. (Such run-time services may be provided by a central server or in various distributed configurations. Further discussion of these architectural issues is outside the scope of this paper.) Other run-time services include the calculation of plans, by means of computing answers to planning queries, and retrieval of past protocol states, by means of computing answers to postdiction queries (as already mentioned in Section 3, plans may be additionally computed at design-time). We show in this section example prediction and planning queries on the RTFD* specification, and the results obtained. To save space, details of postdiction query examples are omitted.

To execute the presented RTFD* protocol, one has to choose specific values for the following parameters:

- Duration of timeouts; this can be expressed as the maximum number of messages that can be exchanged in the given communication channel in the interval defined by any two consecutive timeout events. In our formalisation a timeout duration is expressed as the maximum number of transitions that may take place between any two consecutive timeout events (in this specification a transition is labelled with either a main protocol action, objected or not, or a timeout event).
- Number of turns for each protagonist.
- Number of permitted objectionable actions (see Section 7.3).

To conduct computational experiments, arbitrary numerical values were chosen for these parameters. For a concrete illustration we will present here experiments in which at most two main protocol actions are physically possible between two consecutive timeout events (that is, at most two transitions may take place between two consecutive timeout events), each protagonist has three turns to 'speak', and one objectionable action is permitted. As already mentioned, the specified logic of disputation is very simple (only classical implication is allowed). Moreover, for the experiments presented 'silence implies consent' *is* incorporated in the protocol specification.

We mentioned earlier that the propositions that the two protagonists may claim, concede to, retract, and deny should also be specified at the outset. (As will be discussed later, this restriction may be lifted by re-compiling D^{RTFD^*} each time a protagonist claims a new proposition.) In order to keep sample runs small we will present queries and the computed results concerning only two propositions: the topic q, and p.

The first column of Table 5 shows a sample run of RTFD*; the information displayed on the remaining columns, that is, which actions are proper, timely, objectionable, permitted and obligatory at each state, is produced by computing answers to a number of prediction queries. To save space, in Table 5 *claim* is written as *cl*, *concede* as *cn*, *retract* as *rtr*, *deny* as *dn* and *declare* as *dcl*. Terms containing variables *Protag* and *Q* stand for all their instances.

Consider the following prediction query: at the initial protocol state the proponent *pro* claims the topic of the argumentation q; which actions are proper, timely, objectionable, permitted or obligatory at the resulting state? The answer to this query is displayed in the six rows below the $claim(pro, q)$ action labelled $cl(pro, q)$.

At the state resulting from $claim(pro, q)$, proponent *pro* has a premise that q (see law (13)). Therefore, at that state, it is improper for *pro* to claim q (see law (20)). It is proper for *pro*, however, to claim p because, at that state, *pro* does not have a premise that p. The remaining information displayed in Table 5 is computed in a similar manner.

Note that at the state resulting from $claim(pro, q)$ it is timely only for *pro* to perform an action. This is so because it is the proponent's turn to speak (see Table 2 for the specification of timely actions).

$claim(opp, p)$, $concede(opp, q)$ and $deny(opp, q)$ are objectionable in the state reached by the performance of $claim(pro, q)$. This is due to the fact that these actions are untimely and proper. At the same state declaring either *pro* or *opp* is objectionable because in either case the declaration is untimely and $winner = none$ (see Table 3 for the specification of objectionable actions).

All actions are permitted at the resulting state of the query presented above. Recall that an action is permitted if it is not objectionable or the specified limit of objectionable actions has not been reached; in this sample run one objectionable action per participant is permitted. In the same state, no action is obligatory; obligations are associated with declarations and arise when it is the determiner's turn to speak. (The specifications of permissions and obligations were presented in Table 4.)

The next action in the protocol run presented in Table 5 is a timeout that sets $turn = opponent$. Consequently, in the following state it is timely only for *opp* to perform an action. After the timeout *opp* denies q while *pro* objects to *opp*'s action. The denial is not objectionable and, therefore, *pro*'s objection does not block the effects of *opp*'s action. The next action in the protocol run, however, $claim(pro, p)$, is objectionable and thus *opp*'s objection blocks the effects of *pro*'s action.

Table 5
A sample run of RTFD*

act	proper	timely	obj	per	obl
$cl(pro, q)$					
	$cl(Protag, p)$	$cl(pro, Q)$	$cl(opp, p)$	all	none
	$cn(opp, q)$	$cn(pro, Q)$	$cn(opp, q)$		
	$rtr(pro, q)$	$rtr(pro, Q)$	$dn(opp, q)$		
	$dn(opp, q)$	$dn(pro, Q)$	$dcl(det, Protag)$		
	$dcl(det, pro)$				
timeout					
	$cl(Protag, p)$	$cl(opp, Q)$	$cl(pro, p)$	all	none
	$cn(opp, q)$	$cn(opp, Q)$	$rtr(pro, q)$		
	$rtr(pro, q)$	$rtr(opp, Q)$	$dcl(det, Protag)$		
	$dn(opp, q)$	$dn(opp, Q)$			
	$dcl(det, pro)$				
$dn(opp, q)$, $objected(pro)$					
	$cl(pro, p)$	$cl(opp, Q)$	$cl(pro, p)$	all	none
	$cl(opp, Q)$	$cn(opp, Q)$	$rtr(pro, q)$		
	$rtr(pro, q)$	$cn(opp, Q)$	$dcl(det, Protag)$		
	$dcl(det, Protag)$	$dn(opp, Q)$			
$cl(pro, p)$, $objected(opp)$					
	$cl(pro, p)$	$cl(opp, Q)$	$cl(pro, p)$	$cl(pro, q)$	none
	$cl(opp, Q)$	$cn(opp, Q)$	$rtr(pro, q)$	$cl(opp, Q)$	
	$rtr(pro, q)$	$cn(opp, Q)$	$dcl(det, Protag)$	$cn(Protag, Q)$	
	$dcl(det, Protag)$	$dn(opp, Q)$		$rtr(pro, p)$	
				$rtr(opp, Q)$	
				$dn(Protag, Q)$	
				$dcl(det, Protag)$	

After the performance of *pro*'s objectionable claim, *pro* is no longer permitted to perform an objectionable action because it reached the specified limit of permitted objectionable actions. Non-conformance with this prohibition will sanction *pro*.

A final remark on the protocol run presented in Table 5 concerns the 'silence implies consent' principle. Due to this principle, at the first two states displayed in Table 5 it is proper to declare *pro*, but improper to declare *opp*, the argumentation winner. In these states *pro* is 'winning' the dispute because $premise(pro, q) = \mathsf{t}$ and $premise(opp, q) = \mathsf{u}$ which imply, due to 'silence implies consent' (see law (40)), that both protagonists accept q, the topic of argumentation (the *winning* fluent constant is defined by laws (22)–(24)). If 'silence implies consent' were not incorporated in the protocol then $premise(opp, q) = \mathsf{u}$ would not imply that *opp* accepts the topic (see law (39)), and thus no protagonist would be 'winning'. In this case, it would be proper to declare either *pro* or *opp* the argumentation winner.

Table 6 presents another sample protocol run of RTFD* and the associated information produced by computations of query answers. The timeout sets $turn = determiner$. The proponent *pro* does not accept the argumentation topic and thus the determiner is obliged to declare the opponent *opp* the winner. Clearly, the next action of the presented run, $declare(det, pro)$, does not discharge this obligation. The obligation is discharged when $declare(det, opp)$, although *pro* objects to this declaration; *pro*'s objection does not block the effects of the declaration as the latter action is not objectionable.

In addition to producing the protocol state current at each time, we may compute plans in order to facilitate agents to achieve their goals. Consider the following planning query: we are in a state in which: (i) it is the determiner's turn to speak, (ii) the dispute is unresolved, that is, no protagonist is 'winning', and (iii) there is no declared winner. Find all paths to a final protocol state, that is, a state reached after the final timeout, in which the proponent is declared the winner.

CCALC finds several solutions to this query. All solutions include the action $declare(det, pro)$ and the final timeout. Moreover, the declaration is either not objected, or the proponent is not the only protagonist sanctioned. If the propo-

Table 6
A sample run of RTFD*

act	proper	timely	obj	per	obl
timeout					
	cl(*Protag*, *Q*) *dcl*(*det*, *opp*)	*dcl*(*det*, *Protag*)	*cl*(*Protag*, *Q*) *dcl*(*det*, *pro*)	*cl*(*Protag*, *Q*) *dcl*(*det*, *opp*)	*dcl*(*det*, *opp*)
dcl(*det*, *pro*)					
	cl(*Protag*, *Q*) *dcl*(*det*, *opp*)	*dcl*(*det*, *Protag*)	*cl*(*Protag*, *Q*)	*cl*(*Protag*, *Q*) *dcl*(*det*, *opp*)	*dcl*(*det*, *opp*)
dcl(*det*, *opp*), *objected*(*pro*)					
	cl(*Protag*, *Q*)	*dcl*(*det*, *Protag*)	*cl*(*Protag*, *Q*) *dcl*(*det*, *pro*)	*cl*(*Protag*, *Q*) *dcl*(*det*, *Protag*)	none

nent was sanctioned and the opponent was not, the *declare*(*det*, *pro*) action would be objectionable (see law (37)) and thus an objection would block its effects.

Here is another planning query: given the initial state of the protocol, is it possible to reach a state, within the maximum number of transitions of D^{RTFD^*}, in which an agent: (i) has exceeded the limit of permitted objectionable actions and, (ii) is not sanctioned?

CCALC finds no solution within the maximum number of transitions of D^{RTFD^*}. Given the chosen values for the number of turns of each protagonist (three), and the maximum number of transitions that may take place between any two consecutive timeouts (two), it can be calculated that the maximum number of transitions of D^{RTFD^*}, that is, the number of transitions of the longest path of D^{RTFD^*}, is twenty-one. (Intuitively, in the longest path of D^{RTFD^*} the proponent performs two actions and then a timeout takes place, signalling the opponent's turn; the opponent then performs two actions, followed by a timeout signalling the proponent's turn. The aforementioned sequence of actions is repeated three times, that is, the number of turns of each protagonist. Finally, the determiner performs two actions and the last timeout takes place.) Since there is no solution within the maximum number of transitions, starting from the initial protocol state, we will never reach a state in which an agent has exceeded the limit of permitted objectionable actions and is not sanctioned.

10. Related work

In Section 6 we discussed the points of departure of our formalisation from Brewka's account. Briefly:

- We employed the $\mathcal{C}+$ language to specify the argumentation protocol, instead of the Situation Calculus. Moreover, we *executed* the protocol specification with the use of CCALC, thus providing several design-time and run-time services to protocol designers, agent designers and agents themselves.
- We introduced deadlines to cater for realistic multi-agent protocols.
- We refined Brewka's distinction of possible and legal actions. In our formalisation an action can be classified as physically possible, proper, timely, objectionable, permitted or obligatory.
- We maintained the object mechanism; however, we forbid multiple objectionable actions to avoid having to object to each such action.
- We introduced sanctions in order to discourage participants from performing forbidden actions and not complying with obligations.
- We retained the 'silence implies consent' principle, as an *optional* feature of the protocol.

Apart from 'static' argument systems, Brewka [8, Section 6] formalises 'dynamic' argument systems, that is, argument systems in which participants can start a meta-level debate, arguing about the protocol rules. 'Dynamic' protocol specifications are out of the scope if this paper.

Argumentation protocols have long been studied in the fields of philosophy and computer science—see [48] for a recent review. The focus of this paper was on the procedural part of argumentation, that is, we focused on the specification of the protocol rules rather than the logic of disputation. This is in contrast to research in non-monotonic and uncertain reasoning argument systems that, as Brewka [8] mentions, have been used to define inference systems for existing non-monotonic logics, or a non-standard consequence relation for logics based on a notion of argument. (See, for example, [7,10,15,31,44,50]; Chesñevar et al. [11], and Prakken and Vreeswijk [51] provide two surveys.) Different logics of disputation are suited to different types of argument. Our protocol formalisation was not bound to a specific logic of disputation.

Like [8,24,25,46,47], we adopted a 'public protocol semantics' [48, Section 6], that is, we made no assumptions about the participants' internal architectures. This is contrast to approaches that allow for protocol rules referring to a participant's internal belief base (for instance, [2,30,40–43]).

A line of research that is closely related to our work is that of Bodenstaff et al. [6]. These researchers employ Shanahan's [60] 'full Event Calculus' to formalise Prakken's [47] dialogue system for argumentation, and Parsons, Wooldridge and Amgoud's [43] persuasion dialogue. Bodenstaff et al. formalise the procedural aspect of argumentation, expressing the 'legal' protocol moves in terms of a 'reply structure', distinguishing between 'attacking' and 'surrendering' replies. In this paper we presented a finer classification of protocol actions than Bodenstaff and colleagues' classification of 'legal'/'illegal' actions. Concerning the reply structure, Prakken [48] notes that it is not a standard feature of all dialogue systems (see, for example [21]). In any case, it is possible to adjust our formalisation in order to express a reply structure (in this paper our aim was to reconstruct Brewka's account of RTFD that did not explicitly include such a structure). This can be done by adjusting our specification of proper actions.

Apart from research on argumentation, work that has similar objectives to ours comes from the distributed artificial intelligence literature on norm-governed systems specification. A few examples are the approaches on 'artificial social systems' [20,38,39,62,63,65], 'law-governed interaction' [37], and 'electronic institutions' [16–19,55]. Close to our work is Yolum and Singh's [66,67] work on 'commitment protocols'. These researchers formalise, in Shanahan's 'full Event Calculus', a set of operations on commitments such as create, discharge, cancel, release, and so on. Moreover, they employ an Event Calculus planner [61] to facilitate the planning of commitment protocol participants. It is important to note that in Yolum and Singh's work the term 'commitment' refers to a form of (directed) obligation between agents, and is *not* used as an alternative term for 'premise'. It is difficult to see how an argumentation protocol, or many other interaction protocols for multi-agent systems (for instance, protocols for negotiation, voting, performing transactions in electronic marketplaces, and so on), can be specified simply in terms of commitments in this sense. At the very least, a specification of a protocol's constitutive norms is also required.

11. Conclusion

We have focused in this paper on the formal representation of the procedural aspects of an argumentation protocol, using Brewka's reconstruction of Rescher's theory of formal disputation as a concrete example. We distinguished in the specification between the constitutive rules defining the protocol itself—the protocol actions and their effects on the protocol state—and the physical environment within which the protocol is executed, and the normative environment which may place further constraints on what protocol actions are permitted or obligatory. Although in the simplest cases a protocol action is permitted if and only if it is both proper and timely, there are many reasons why this need not always be so. We presented some simple examples by way of illustration, but it should be clear that there are many other possibilities that we did not discuss. There is also much choice in deciding where to place the boundary between the constitutive and normative components. For instance, we chose in the example presented here to say that the claim of Q by a protagonist who already holds Q as a premise is not proper, has no effect on the protocol state, but is permitted. We could have chosen to say instead that such a claim is proper but never permitted. Or that it is not proper, does have an effect on the protocol state (in that it requires the other protagonist to concede or deny), but is not permitted. One objective of the present work is to allow such variations to be specified and examined, and to help evaluate the effectiveness of proposed sanctioning and enforcement mechanisms.

In an earlier formalisation of Brewka's protocol [4] we specified the constitutive elements of the protocol by defining the conditions under which an agent has institutional power (competence, capacity) to perform a particular protocol action; we then said that an action is 'valid' when it is performed by an empowered agent. In this paper we have constructed a more structured and detailed specification by defining separately the conditions under which an action is proper and timely, as has also been suggested by Prakken et al. [45,49]. One aim of the paper was to see how this additional structure would be reflected in the specification of a concrete example. A more detailed specification still would define what exact forms of message or utterance count as expressing a claim, concede, deny, retract, declare, and object action. We have not defined that level of detail here. It would be an important component in a run-time mechanism but is not so important if we are primarily interested, as here, in investigating properties of the specified protocol. We have also found that it adds flexibility to specify separately which actions are objectionable, though that is perhaps difficult to demonstrate convincingly in a comparatively simple protocol such as Brewka's. Finally, we have shown how the 'silence implies consent' principle can be included straightforwardly as an optional component by marking as unconfirmed those premises that have not been explicitly conceded or retracted by the opponent.

In principle, the kind of specifications presented in this paper could also be expressed in other temporal reasoning formalisms. The $\mathcal{C}+$ language, however, has a number of important features that have led us to choose it as the basis for further developments. First, it is a comparatively expressive formalism with fine control for specifying default persistence of fluent constants. The availability of static laws moreover means, amongst other things, that complex specifications can be given structure. For example, most of the rules defining *proper*(...) and *timely*(...) constants in this paper can be expressed as static laws with simple conjunctions as their conditions. Additional structure can be provided by introducing suitably chosen intermediate concepts, such as the fluent *winning* used in our example, themselves defined by means of static laws. This ability is important if large specifications are to be undertaken.

Second, besides its semantics through translation to the formalism of 'non-monotonic causal theories' [22], a $\mathcal{C}+$ action description has an explicit semantics in terms of transition systems. This is important because it provides a link to a wide range of other formalisms and tools based on transition systems. We have been able to devise, for example, an extended form of $\mathcal{C}+$ specifically designed for representing norms and institutions [57–59], including direct support for (a version of) the 'counts as' relation for actions [27] and a treatment of permitted/forbidden states, transitions and runs. The relationship between permitted and obligatory actions presented in this paper, for instance, which had to be formulated explicitly in the $\mathcal{C}+$ specification (see Section 7.3), is built into the semantics of the language in the extended version. The development of the extended form of $\mathcal{C}+$ took place in parallel with the methods presented in this paper, and, therefore, we leave its presentation to a separate paper. Moreover, the relationship between obligations, non-compliant behaviour and sanctions in protocol specifications (see, for instance, Section 7.4), and the notion of permission built in to the extended $\mathcal{C}+$ language needs to be explored more fully.

The language $\mathcal{C}+$ (and its derivatives) also has some important limitations. Most obviously, from a representational point of view, the language inherits the limitations of transition systems, in particular that the executable actions (transitions) in any given state s of the system, and their effects, can depend only on the state s and not on the path or history by which state s was reached (unless of course we encode the entire history in every state s). The $\mathcal{C}+$ language itself (though not the underlying formalism of causal theories) can only express causes relationships between successive states; delayed effects cannot be expressed directly. This is what makes a detailed formulation of the object mechanism awkward and fiddly to express in $\mathcal{C}+$, and why we chose to omit the details from this paper. (But see [14] for a modified form of $\mathcal{C}+$ which can express delayed effects.) In other protocols where the moves available to a participant might depend on the entire history of the protocol so far (a participant is restricted on the number of repeated claims it can make, for instance), these limitations are not so easily overcome. Extensions to $\mathcal{C}+$ capable of expressing such constraints in a concise manner are one direction of our current research.

From the point of view of implementation, CCALC provides an immediate and convenient means of implementing $\mathcal{C}+$ action descriptions. The execution of the RTFD* specification, however, confirmed our previous experience regarding CCALC's efficiency, and in particular that it does not provide a practical means for supporting run-time activities (see Section 9). A major limitation, of course, is the need to encode the underlying logic of disputation, which here we have done by means of explicit instances of the *implies* fluent constants. This clearly works only for small examples. It also requires that we are able to specify at the outset the complete set of propositions and default rules that could be claimed during the argumentation. Whilst it is possible to improve the efficiency of the CCALC implementation by pre-compiling this part of the action description (the *implies* constants are 'rigid'), and even in principle re-compiling the action description every time a protagonist claims a new, unanticipated, proposition, this is clearly not a practical way of supporting run-time activities. It is only adequate for conducting computational experiments with the specification of the kind discussed in Section 9. (The 'logic of disputation' for the examples in that section is trivial but that was chosen deliberately to keep the sample runs small enough to be presented.)

CCALC is not the only means by which $\mathcal{C}+$ action descriptions could be executed. We have also used versions of the Event Calculus to specify and execute (an earlier version of) the RTFD* protocol [3, Sections 6.10–6.12]. Given an instance of the specification and a narrative—a record of what actions have been performed so far—this (Prolog) implementation allows all protocol states, including what is permitted and obligatory at each state, to be queried and computed efficiently. It is not necessary to specify all propositions in advance, and it is comparatively easy to implement the underlying logic of disputation as another Prolog module to be added to the protocol specification. The Event Calculus implementation, however, is not well suited to planning and postdiction tasks. More importantly, we also lose the explicit transition system semantics which we see as the single most important advantage of the $\mathcal{C}+$ formulation. A discussion comparing the use of $\mathcal{C}+$ and the Event Calculus for developing executable specifications of multi-agent protocols, including argumentation protocols, can be found in [5].

Recent work by Craven [13] has investigated the relationships between Event Calculus and $\mathcal{C}+$. His $\mathcal{EC}+$ implementation provides an efficient Event Calculus style of computation of narratives with (a restricted form of) the $\mathcal{C}+$ language, providing a promising means of supporting run-time activities. He has also exploited the transition system semantics to connect $\mathcal{C}+$ to model checking software (specifically NuSMV [12]). This allows protocol properties, expressed in Liner Temporal Logic (LTL) and Computation Tree Logic (CTL), to be verified by means of standard model checking techniques on protocol specifications expressed in the $\mathcal{C}+$ language. An area of future work is to investigate what types of property, especially types of property identified for argumentation protocols (see, for example, [48, Section 8] and [36]), can be expressed by means of LTL and CTL, and thus proven on protocols formalised in $\mathcal{C}+$.

Acknowledgements

This article is an updated and extended version of [4]. We are grateful to the reviewers and the participants of ICAIL'03, who gave us useful feedback. We should also like to thank Joohyung Lee and Vladimir Lifschitz for their suggestions regarding the $\mathcal{C}+$ language and CCALC. The authors themselves, however, are solely responsible for any misunderstanding about the use of these technologies.

This work was supported partly by the EU ALFEBIITE Project (IST-1999-10298) and partly by the EPSRC project 'Theory and Technology of Norm-Governed Self-Organising Networks' (GR S74911 01).

References

[1] V. Akman, S. Erdogan, J. Lee, V. Lifschitz, H. Turner, Representing the zoo world and the traffic world in the language of the Causal Calculator, Artificial Intelligence 153 (1–2) (2004) 105–140.

[2] L. Amgoud, N. Maudet, S. Parsons, Modelling dialogues using argumentation, in: Proceedings of the International Conference on Multiagent Systems (ICMAS), IEEE Computer Society, 2000, pp. 31–38.

[3] A. Artikis, Executable specification of open norm-governed computational systems, PhD thesis, University of London, November 2003. Retrieved March 6, 2006, from http://www.iit.demokritos.gr/~a.artikis/publications/artikis-phd.pdf, also available from the author.

[4] A. Artikis, M. Sergot, J. Pitt, An executable specification of an argumentation protocol, in: Proceedings of Conference on Artificial Intelligence and Law (ICAIL), ACM Press, 2003, pp. 1–11.

[5] A. Artikis, M. Sergot, J. Pitt, Specifying norm-governed computational societies, Technical Report 2006/5, Imperial College London, Department of Computing, 2006. Retrieved March 6, 2006, from http://www.doc.ic.ac.uk/research/technicalreports/2006/DTR06-5.pdf.

[6] L. Bodenstaff, H. Prakken, G. Vreeswijk, On formalising dialogue systems for argumentation in the event calculus, in: Proceedings of Workshop on Non-Monotonic Reasoning, 2006.

[7] A. Bondarenko, P.M. Dung, R. Kowalski, F. Toni, An abstract, argumentation-theoretic approach to default reasoning, Artificial Intelligence 93 (1997) 63–101.

[8] G. Brewka, Dynamic argument systems: a formal model of argumentation processes based on situation calculus, Journal of Logic and Computation 11 (2) (2001) 257–282.

[9] G. Brewka, T. Eiter, Prioritizing default logic, in: Festschrift 60th Anniversary of W. Bibel, Kluwer, 1998.

[10] C. Cayrol, On the relation between argumentation and coherence based entailment, in: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1995, pp. 1443–1448.

[11] C. Chesñevar, A. Maguitman, R. Loui, Logical models of argument, ACM Computing Surveys 32 (4) (2000) 337–383.

[12] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV version 2: An opensource tool for symbolic model checking, in: Proc. International Conference on Computer-Aided Verification (CAV 2002), Copenhagen, July 2002, in: Lecture Notes in Computer Science, vol. 2404, Springer, 2002, see http://nusmv.irst.itc.it.

[13] R. Craven, Execution mechanisms for the action language $\mathcal{C}+$, PhD thesis, University of London, September 2006.

[14] R. Craven, M. Sergot, Distant causation in $\mathcal{C}+$, Studia Logica 79 (1) (2005) 73–96.

[15] P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Artificial Intelligence 77 (2) (1995) 321–358.

[16] M. Esteva, D. de la Cruz, C. Sierra, ISLANDER: an electronic institutions editor, in: C. Castelfranchi, L. Johnson (Eds.), Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), ACM Press, 2002, pp. 1045–1052.

[17] M. Esteva, J. Padget, C. Sierra, Formalizing a language for institutions and norms, in: J.-J. Meyer, M. Tambe (Eds.), Intelligent Agents VIII: Agent Theories, Architectures, and Languages, in: Lecture Notes in Artificial Intelligence, vol. 2333, Springer, 2002, pp. 348–366.

[18] M. Esteva, J. Rodriguez-Aguilar, J. Arcos, C. Sierra, P. Garcia, Institutionalising open multi-agent systems, in: E. Durfee (Ed.), Proceedings of the International Conference on Multi-agent Systems (ICMAS), IEEE Press, 2000, pp. 381–382.
[19] M. Esteva, J. Rodriguez-Aguilar, C. Sierra, P. Garcia, J. Arcos, On the formal specifications of electronic institutions, in: F. Dignum, C. Sierra (Eds.), Agent Mediated Electronic Commerce, in: Lecture Notes in Artificial Intelligence, vol. 1991, Springer, 2001, pp. 126–147.
[20] D. Fitoussi, M. Tennenholtz, Choosing social laws for multi-agent systems: minimality and simplicity, Artificial Intelligence 119 (1–2) (2000) 61–101.
[21] J. Fulda, The logic of 'improper cross', Artificial Intelligence and Law 8 (4) (2001) 337–341.
[22] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, H. Turner, Nonmonotonic causal theories, Artificial Intelligence 153 (1–2) (2004) 49–104.
[23] E. Giunchiglia, J. Lee, V. Lifschitz, H. Turner, Causal laws and multi-valued fluents, in: Proceedings of Workshop on Nonmonotonic Reasoning, Action and Change (NRAC), 2001.
[24] T. Gordon, The pleadings game: an exercise in computational dialectics, Artificial Intelligence and Law 2 (1994) 239–292.
[25] T. Gordon, The Pleadings Game: An Artificial Intelligence Model of Procedural Justice, Kluwer Academic Publishers, 1995.
[26] A. Jones, M. Sergot, On the characterisation of law and computer systems: the normative systems perspective, in: Deontic Logic in Computer Science: Normative System Specification, J. Wiley and Sons, 1993, pp. 275–307.
[27] A. Jones, M. Sergot, A formal characterisation of institutionalised power, Journal of the IGPL 4 (3) (1996) 429–445.
[28] M. Klein, J. Rodriguez-Aguilar, C. Dellarocas, Using domain-independent exception handling services to enable robust open multi-agent systems: the case of agent death, Journal of Autonomous Agents and Multi-Agent Systems 7 (1–2) (2003) 179–189.
[29] R. Kowalski, M. Sergot, A logic-based calculus of events, New Generation Computing 4 (1) (1986) 67–96.
[30] S. Kraus, K. Sycara, A. Evenchik, Reaching agreements through argumentation: a logical model and implementation, Artificial Intelligence 104 (1–2) (1998) 1–69.
[31] P. Krause, S. Ambler, J. Fox, A logic of argumentation for uncertain reasoning, Computational Intelligence 11 (1) (1995) 113–131.
[32] J. Lee, V. Lifschitz, H. Turner, A representation of the zoo world in the language of the Causal Calculator, in: Proceedings of Symposium on Formalizations of Commonsense Knowledge, 2001.
[33] V. Lifschitz, Missionaries and cannibals in the Causal Calculator, in: A. Cohn, F. Giunchiglia, B. Selman (Eds.), Proceedings of Conference on Principles of Knowledge Representation and Reasoning (KR), Morgan Kaufmann, 2000, pp. 85–96.
[34] V. Lifschitz, N. Mccain, E. Remolina, A. Tacchella, Getting to the airport: the oldest planning problem in AI, in: J. Minker (Ed.), Logic-Based Artificial Intelligence, Kluwer, 2000, pp. 147–168.
[35] D. Makinson, On the formal representation of rights relations, Journal of Philosophical Logic 15 (1986) 403–425.
[36] P. McBurney, S. Parsons, M. Wooldridge, Desiderata for agent argumentation protocols, in: C. Castelfranchi, L. Johnson (Eds.), Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), ACM Press, 2002, pp. 402–409.
[37] N. Minsky, V. Ungureanu, Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems, ACM Transactions on Software Engineering and Methodology (TOSEM) 9 (3) (2000) 273–305.
[38] Y. Moses, M. Tennenholtz, On computational aspects of artificial social systems, in: Proceedings of Workshop on Distributed Artificial Intelligence (DAI), 1992, pp. 267–284.
[39] Y. Moses, M. Tennenholtz, Artificial social systems, Computers and Artificial Intelligence 14 (6) (1995) 533–562.
[40] S. Parsons, P. McBurney, Argumentation-based communication between agents, in: Communication in Multiagent Systems, in: Lecture Notes in Computer Science, vol. 2650, Springer, 2003, pp. 164–178.
[41] S. Parsons, M. Wooldridge, L. Amgoud, An analysis of formal inter-agent dialogues, in: Proceedings of Conference on Autonomous Agents and Multi-Agent Systems, ACM Press, 2002, pp. 394–401.
[42] S. Parsons, M. Wooldridge, L. Amgoud, On the outcomes of formal inter-agent dialogues, in: Proceedings of Conference on Autonomous Agents and Multi-Agent Systems, ACM Press, 2003, pp. 616–623.
[43] S. Parsons, M. Wooldridge, L. Amgoud, Properties and complexity of some formal inter-agent dialogues, Journal of Logic and Computation 13 (3) (2003) 347–376.
[44] J. Pollock, Oscar—a general purpose defeasible reasoner, Journal of Applied Non-Classical Logics 6 (1) (1996) 89–113.
[45] H. Prakken, Formalising Robert's rules of order, Technical Report 12, GMD—German National Research Center for Information Technology, 1998.
[46] H. Prakken, On dialogue systems with speech acts, arguments, and counterarguments, in: Proceedings of Workshop on Logics in Artificial Intelligence, in: Lecture Notes in Artificial Intelligence, vol. 1919, Springer, 2000, pp. 224–238.
[47] H. Prakken, Coherence and flexibility in dialogue games for argumentation, Journal of Logic and Computation 15 (2005) 1009–1040.
[48] H. Prakken, Formal systems for persuasion dialogue, Knowledge Engineering Review 21 (2) (2006) 163–188.
[49] H. Prakken, T. Gordon, Rules of order for electronic group decision making—a formalization methodology, in: J. Padget (Ed.), Collaboration between Human and Artificial Societies, in: Lecture Notes in Computer Science, vol. 1624, Springer, 1999, pp. 246–263.
[50] H. Prakken, G. Sartor, Argument-based extended logic programming with defeasible priorities, Journal of Applied Non-Classical Logics 7 (1) (1997) 25–75.
[51] H. Prakken, G. Vreeswijk, Logics for defeasible argumentation, in: D. Gabbay, F. Guenthner (Eds.), Handbook of Philosophical Logic, vol. 4, Kluwer Academic Publishers, 2002, pp. 218–319.
[52] R. Reiter, Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems, The MIT Press, 2001.
[53] N. Rescher, Dialectics: A Controversy-Oriented Approach to the Theory of Knowledge, State University of New York Press, 1977.
[54] J. Rodriguez-Aguilar, F. Martin, P. Noriega, P. Garcia, C. Sierra, Towards a test-bed for trading agents in electronic auction markets, AI Communications 11 (1) (1998) 5–19.
[55] J. Rodriguez-Aguilar, C. Sierra, Enabling open agent institutions, in: K. Dautenhahn, A. Bond, L. Canamero, B. Edmonds (Eds.), Socially Intelligent Agents: Creating Relationships with Computers and Robots, Kluwer Academic Publishers, 2002, pp. 259–266.

[56] J. Searle, Speech Acts, Cambridge University Press, 1969.

[57] M. Sergot, $(\mathcal{C}+)^{++}$: An action language for modelling norms and institutions, Technical Report 2004/8, Department of Computing, Imperial College London, 2004. Retrieved March 6, 2006, from http://www.doc.ic.ac.uk/research/technicalreports/2004/DTR04-8.pdf.

[58] M. Sergot, Modelling unreliable and untrustworthy agent behaviour, in: B. Dunin-Keplicz, A. Jankowski, A. Skowron, M. Szczuka (Eds.), Proceedings of Workshop on Monitoring, Security, and Rescue Techniques in Multiagent Systems (MSRAS), in: Advances in Soft Computing, Springer, 2004, pp. 161–178.

[59] M. Sergot, R. Craven, The deontic component of action language nC+, in: L. Goble, J.-J.C. Meyer (Eds.), Deontic Logic and Artificial Normative Systems. Proc. 8th International Workshop on Deontic Logic in Computer Science (DEON'06), Utrecht, July 2006, in: Lecture Notes in Artificial Intelligence, vol. 4048, Springer, 2006, pp. 222–237.

[60] M. Shanahan, The event calculus explained, in: M. Wooldridge, M. Veloso (Eds.), Artificial Intelligence Today, in: Lecture Notes in Artificial Intelligence, vol. 1600, Springer, 1999, pp. 409–430.

[61] M. Shanahan, An abductive event calculus planner, Journal of Logic Programming 44 (2000) 207–239.

[62] Y. Shoham, M. Tennenholtz, On the synthesis of useful social laws for artificial agent societies, in: W. Swartout (Ed.), Proceedings of Conference on Artificial Intelligence (AAAI), The AAAI Press/The MIT Press, 1992, pp. 276–281.

[63] Y. Shoham, M. Tennenholtz, On social laws for artificial agent societies: off-line design, Artificial Intelligence 73 (1–2) (1995) 231–252.

[64] M. Singh, A social semantics for agent communication languages, in: F. Dignum, M. Greaves (Eds.), Issues in Agent Communication, in: Lecture Notes in Computer Science, vol. 1916, Springer, 2000, pp. 31–45.

[65] M. Tennenholtz, On computational social laws for dynamic non-homogeneous social structures, Journal of Experimental and Theoretical Artificial Intelligence 7 (1995) 379–390.

[66] P. Yolum, M. Singh, Flexible protocol specification and execution: applying event calculus planning using commitments, in: C. Castelfranchi, L. Johnson (Eds.), Proceedings of Conference on Autonomous Agents and Multiagent Systems (AAMAS), ACM Press, 2002, pp. 527–535.

[67] P. Yolum, M. Singh, Reasoning about commitments in the event calculus: An approach for specifying and executing protocols, Annals of Mathematics and Artificial Intelligence 42 (1–3) (2004) 227–253.