Towards an Argumentation Protocol in the Event Calculus

Periklis Mantenoglou[1,*], Alexander Artikis[2,3]

[1]Örebro University, Sweden
[2]University of Piraeus, Greece
[3]NCSR Demokritos, Greece

Abstract

We present a formalisation of RTFD*, a multi-agent argumentation protocol, in the Run-Time Event Calculus (RTEC$^{\rightarrow}$), a stream reasoning engine with native support for events with delayed effects. Compared to the original specification of RTFD* in the action language C+, our formulation supports time windows for objections and efficient reasoning over high-velocity streams of agent messages, allowing us to handle more realistic argumentation scenarios. We report empirical results demonstrating low-latency monitoring of argumentation procedures at scale.

Keywords

argumentation, Event Calculus, stream reasoning, multi-agent systems

1. Introduction

Contemporary applications are manifested as agentic systems [1]. At the same time, the literature on multi-agent systems (MAS) spans across several decades [2]. A key task in the formal specification and analysis of MAS concerns 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. Consider e.g. Brewka's reconstruction [3] of Rescher's Theory of Formal Disputation (RTFD) [4], or the recent work on argumentative Large Language Models [5]. 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. Moreover, Artikis et al. [6] presented a variant of Brewka's reconstruction of RTFD, i.e. RTFD*, structuring the procedural aspects of the protocol in the following levels: (a) the physical capabilities of the participant agents; (b) 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 (c) the permissions, prohibitions and obligations of the agents.

There are three roles in RTFD*: proponent, opponent and determiner. The proponent claims a thesis, the opponent questions this thesis, and the determiner decides whether the proponent's thesis was successfully defended or not. The argumentation commences when the proponent claims the topic of the argumentation. The protagonists — the proponent and the opponent — then take it in turn to perform actions, i.e. claim, concede to, retract, or deny a proposition. Each turn lasts for a specified time period during which the protagonist may perform several actions. After each such action the other protagonist is given an opportunity to object. The determiner may declare the winner only at the end of the argumentation, i.e. when the specified period for the argumentation elapses. For example, 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. RTFD* was formalised in the action language $\mathcal{C}+$ [7]. This way, RTFD* is directly executable — e.g. the 'Causal Calculator' may be used to execute temporal projection queries over RTFD* [8].

8th International Workshop on Artificial Intelligence and fOrmal VERification, Logic, Automata, and sYnthesis. OVERLAY 2026 @ FLoC, July 18–19, Lisbon, Portugal

*Corresponding author.
periklis.mantenoglou@oru.se (P. Mantenoglou); a.artikis@unipi.gr (A. Artikis)
https://periklismant.github.io/ (P. Mantenoglou); https://users.iit.demokritos.gr/~a.artikis/ (A. Artikis)
0009-0002-3275-1522 (P. Mantenoglou); 0000-0001-6899-4599 (A. Artikis)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-09

CEUR-WS.org/Vol-4324/paper12.pdf

On the other hand, the approach of [6] has the following shortcomings. First, an 'object mechanism' dictates that after each action of an agent the other participants are given an opportunity to object within another specified time period. However, Artikis et al. presented a simplified specification of the object mechanism, according to which the objection to an action, if any, is said to take place at the same time as the action itself. Second, the approach of [6] fails to scale to argumentations with large volumes of agent messages, such as those anticipated by argumentations between LLM-based agents.

We have been developing a formalisation of RTFD* that addresses the aforementioned issues. We employ RTEC$^{\rightarrow}$ [9], i.e. an implementation of the Event Calculus [10] designed for stream reasoning. RTEC$^{\rightarrow}$ has built-in support for events with delayed effects, and thus allows us to express deadlines as well as formulate an elaborate account of the 'object' mechanism. Furthermore, RTEC$^{\rightarrow}$ includes various optimisation techniques, supporting high-velocity streams of agent messages.

2. Run-Time Event Calculus

The Run-Time Event Calculus (RTEC) [11] is a logic programming implementation of the Event Calculus [10]. We focus on its latest version, i.e. RTEC$^{\rightarrow}$, which supports events with delayed effects [12].

RTEC$^{\rightarrow}$ employs a linear timeline with non-negative integer time-points. Variables start with an upper-case letter, while predicates and constants start with a lower-case letter. The specification of a domain in RTEC$^{\rightarrow}$ includes domain objects, 'events', leading to changes in the state of the world, and 'fluents', which are domain properties that may have different values at different points in time.

Definition 1 (RTEC$^{\rightarrow}$ Predicates). *RTEC$^{\rightarrow}$ employs the following predicates:*

- happensAt(E, T)*: event* E *occurs at time-point* T*.*
- initiatedAt$(F = V, T)$*: a time period during which fluent* F *has value* V *is initiated at* T*.*
- terminatedAt$(F = V, T)$*: a time period during which fluent* F *has value* V *is terminated at* T*.*
- holdsAt$(F = V, T)$*: fluent* F *has value* V *at time-point* T*.*
- holdsFor$(F = V, I)$*: fluent* F *has value* V *continuously during the maximal interval in list* I*.*
- fi$(F = V, F = V', D)$*: an initiation of* $F = V$ *leads to a* ***f****uture* ***i****nitiation of* $F = V'$ *after* D *time-points, unless* $F = V$ *is terminated in the meantime.*

In RTEC$^{\rightarrow}$, RTFD* actions are formulated as events (e.g. $claim(Protag, Q)$) and their effects on the protocol state as fluent-value pairs (FVPs). $premise(Protag, Q) = explicit$, e.g., is an FVP stating that $Protag$ holds an explicit premise about proposition Q.

The RTFD* protocol may be expressed as an RTEC$^{\rightarrow}$ *event description*, i.e. a set of rules defining 'simple' and 'statically determined' FVPs. A simple FVP is defined using a set of initiatedAt and terminatedAt rules, and is subject to the common-sense law of inertia, i.e. an FVP $F{=}V$ holds at a time-point T, if $F{=}V$ has been 'initiated' by an event at a time-point earlier than T, and not 'terminated' by another event in the meantime.

Example 1 (Concession). *These rules specify the effects of a* $concede$ *action on simple fluent* $premise$*.*

$$\begin{aligned}&\text{initiatedAt}(premise(Protag, Q) = explicit, T) \leftarrow \\ &\quad \text{happensAt}(concede(Protag, Q), T), \text{holdsAt}(premise(Protag, Q) = unconfirmed, T), \\ &\quad \text{not holdsAt}(objectionable(Protag, concede, Q) = \text{true}, T).\end{aligned} \tag{1}$$

$$\begin{aligned}&\text{initiatedAt}(premise(Protag, Q) = explicit, T) \leftarrow \\ &\quad \text{happensAt}(concede(Protag, Q), T), \text{holdsAt}(premise(Protag, Q) = unconfirmed, T), \\ &\quad \text{not happensAt}(object(concede, Q), T).\end{aligned} \tag{2}$$

'not' *expresses negation-by-failure,* $objectionable(Protag, A, Q)$ *expresses whether action* A *about proposition* Q*, performed by* $Protag$*, is said to be 'objectionable', and* $object(concede, Q)$ *is an event denoting that at least one agent has expressed an objection. According to rules* (1) *and* (2)*, a protagonist* $Protag$ *is said to adopt an explicit premise about* Q *when* $Protag$ *concedes to* Q*, provided that* $Protag$ *has an unconfirmed premise about* Q*, and the concession is not objectionable or no agent objects to it.* ◊

The definition of a statically determined FVP $F = V$ is a rule with head $\mathsf{holdsFor}(F = V, I)$; it computes the temporal intervals of $F = V$ by applying operations on the intervals of other FVPs.

Example 2 (Objectionable concession). *In RTFD*, a concession is said to be objectionable when it is 'proper' but not 'timely':*

$$\begin{aligned}&\mathsf{holdsFor}(objectionable(Protag, concede, Q) = \mathsf{true}, I) \leftarrow \\ &\quad \mathsf{holdsFor}(proper(Protag, concede, Q) = \mathsf{true}, I_1), \\ &\quad \mathsf{holdsFor}(timely = Protag, I_2), \mathsf{relative_complement_all}(I_1, [I_2], I).\end{aligned} \tag{3}$$

$objectionable$ is a statically determined fluent, defined in terms of the $proper$ and $timely$ fluents. The list of maximal intervals I during which a concession by a protagonist $Protag$ to proposition Q is said to be objectionable is computed by the relative complement of the list of maximal intervals I_1 during which the concession to Q from $Protag$ is said to be proper, and the list of maximal intervals I_2 during which it is timely for $Protag$ to speak. See [6] for the specification of proper and timely actions. ◊

3. Objection Time Windows

In the implementation of RTFD* of [6], an action can only be objected to at the time it is performed, as denoted in rule (2), which evaluates both the happensAt condition for $concede$ and the happensAt condition for $object$ at the same time-point T. This is infeasible in a real-world argumentation game. We address this issue by updating the definition of the $premise$ fluent in order to allow $object$ actions to be valid within a time window after the action that may be objected to. For the $concede$ action, e.g., we replace rule (2) with the following rules:

$$\begin{aligned}&\mathsf{initiatedAt}(premise(Protag, Q) = explicit, T) \leftarrow \\ &\quad \mathsf{happensAt}(concede(Protag, Q), T), \\ &\quad \mathsf{holdsAt}(premise(Protag, Q) = unconfirmed, T), \\ &\quad \mathsf{not\ holdsAt}(objectionable(Protag, concede, Q) = \mathsf{true}, T).\end{aligned} \tag{4}$$

$$\begin{aligned}&\mathsf{initiatedAt}(premise(Protag, Q) = concession_attempt, T) \leftarrow \\ &\quad \mathsf{happensAt}(concede(Protag, Q), T), \\ &\quad \mathsf{holdsAt}(premise(Protag, Q) = unconfirmed, T), \\ &\quad \mathsf{holdsAt}(objectionable(Protag, concede, Q) = \mathsf{true}, T).\end{aligned} \tag{5}$$

$$\begin{aligned}&\mathsf{fi}(premise(Protag, Q) = concession_attempt, premise(Protag, Q) = explicit, D) \leftarrow \\ &\quad objectionDeadline(D).\end{aligned} \tag{6}$$

$$\begin{aligned}&\mathsf{initiatedAt}(premise(Protag, Q) = unconfirmed, T) \leftarrow \\ &\quad \mathsf{happensAt}(object(concede, Q), T), \\ &\quad \mathsf{holdsAt}(premise(Protag, Q) = concession_attempt, T).\end{aligned} \tag{7}$$

Rule (4) expresses that, when $concede(Protag, Q)$ is not objectionable, $Protag$ holds Q explicitly right after their concession. On the other hand, rules (5)–(7) are used when $concede(Protag, Q)$ is objectionable. In that case, rule (5) initiates a time period during which $premise(Protag, Q)$ is "frozen" in an intermediate state, indicated by value $concession_attempt$, expressing that $Protag$ has performed a concession action about Q and this action can still be objected to. As long as $premise(Protag, Q) = concession_attempt$, the only actions about Q that are considered proper are objections, and the value of $premise(Protag, Q)$ only changes when (i) the objection time window expires, or (ii) there is an objection to the concession action. Case (i) is handled by rule (6), expressing that, an initiation of $premise(Protag, Q) = concession_attempt$ leads to an initiation of $premise(Protag, Q) = explicit$ after D time-points—where D is defined via background knowledge predicate $objectionDeadline$—unless $premise(Protag, Q) = concession_attempt$ is terminated in the meantime. Such a termination may only be brought about by rule (7), which changes the value of $premise(Protag, Q)$ to $unconfirmed$ when an objection takes place, thus handling case (ii). We introduce similar values for the premise fluent to support objection time windows for the remaining possible agent actions. For the $deny$ action, e.g., we employ value $denial_attempt$ for $premise$.

4. Experimental Evaluation

We evaluated RTEC$^{\rightarrow}$ on the task of computing the temporal intervals of all FVPs in our argumentation event description over streams of agent actions. We used synthetic streams that emulate realistic argumentation scenarios. In each experiment, we either monitor: (i) a single dispute, i.e., two protagonists contending about a claim, possibly employing numerous relevant propositions to back their position, or (ii) several such disputes taking place in parallel. We experimented with all nine combinations of 1, 5, and 25 parallel disputes, each involving 5, 25, or 125 disputable propositions.

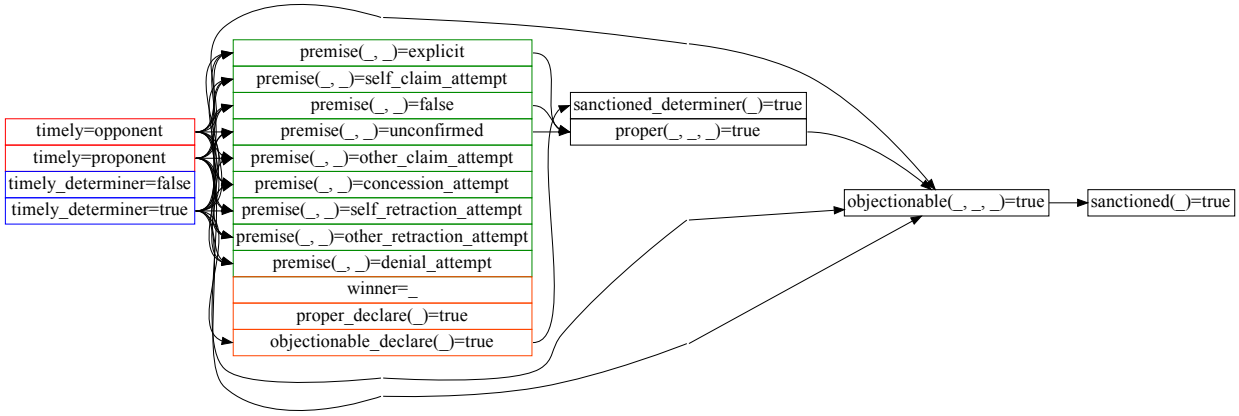


Figure 1: The FVP dependency graph of our RTEC$^{\rightarrow}$ event description for RTFD*.

The target FVPs of our experiments denote the premises held by agents about propositions, whether actions are proper, timely and objectionable, as well as the sanctions that should be imposed on agents. Figure 1 presents the "FVP dependency graph" of our argumentation event description; there is one vertex for each FVP in the event description while edges denote dependencies among FVPs. Vertices with the same colour denote that the corresponding FVPs depend circularly on one another. In RTEC$^{\rightarrow}$, such cyclic dependencies are handled in a way that minimises re-computations [12].

Number of Disputes	Number of Propositions per Dispute					
	5		25		125	
	Time	Intervals	Time	Intervals	Time	Intervals
1	0.005	55	0.01	93	0.02	109
5	0.161	1K	0.77	3K	1.5	3K
25	13	33K	105	62K	260	70K

Table 1
RTEC$^{\rightarrow}$ reasoning times (seconds) and the number of computed intervals per argumentation round (20 time-points) for all parallel disputes. In all cases, the input narrative included 125 events per dispute.

Table 1 presents our results. Reasoning times scale gracefully with both the number of parallel disputes and the number of propositions per dispute. For a single dispute, even with 125 propositions, RTEC$^{\rightarrow}$ produces results in 20 milliseconds. The largest configuration—25 parallel disputes with 125 propositions each, producing $70K$ intervals—is processed within minutes, demonstrating that our formulation can be used to monitor realistic large-scale argumentation scenarios.

In the future, we would like to extend our formulation in order to impose deadlines on agents' turns, avoiding undesired scenarios where agents take arbitrarily long to speak. We would also like to enable coalitions of agents—a party of proponents and a party of opponents—participating in each dispute.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] E. L. Malfa, G. L. Malfa, S. Marro, J. M. Zhang, E. Black, M. Luck, P. Torr, M. J. Wooldridge, Large language models miss the multi-agent mark, in: NeurIPS, Position Paper Track, 2025.

[2] M. J. Wooldridge, The logical modelling of computational multi-agent systems, Ph.D. thesis, University of Manchester, UK, 1992.

[3] G. Brewka, Dynamic argument systems: A formal model of argumentation processes based on situation calculus, J. Log. Comput. 11 (2001) 257–282.

[4] N. Rescher, Dialectics: A Controversy-Oriented Approach to the Theory of Knowledge, State University of New York Press, 1977.

[5] G. Freedman, A. Dejl, D. Gorur, X. Yin, A. Rago, F. Toni, Argumentative large language models for explainable and contestable claim verification, in: T. Walsh, J. Shah, Z. Kolter (Eds.), AAAI, AAAI Press, 2025, pp. 14930–14939.

[6] A. Artikis, M. J. Sergot, J. Pitt, An executable specification of a formal argumentation protocol, Artif. Intell. 171 (2007) 776–804.

[7] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, H. Turner, Nonmonotonic causal theories, Artif. Intell. 153 (2004) 49–104.

[8] V. Akman, S. T. Erdogan, J. Lee, V. Lifschitz, H. Turner, Representing the zoo world and the traffic world in the language of the causal calculator, Artif. Intell. 153 (2004) 105–140.

[9] P. Mantenoglou, A. Artikis, Temporal specification optimisation for the event calculus, in: AAAI-25, 2025, pp. 15075–15082.

[10] R. Kowalski, M. Sergot, A logic-based calculus of events, New Gen. Computing 4 (1986) 67–96.

[11] A. Artikis, M. J. Sergot, G. Paliouras, An event calculus for event recognition, IEEE Trans. Knowl. Data Eng. 27 (2015) 895–908.

[12] P. Mantenoglou, M. Pitsikalis, A. Artikis, Reasoning over streams of events with delayed effects, J. Artif. Intell. Res. 84 (2025).