ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-15

Executable Logic for Dialogical Argumentation

Elizabeth Black[1] and **Anthony Hunter**[2]

Abstract. Argumentation between agents through dialogue is an important cognitive activity. There have been a number of proposals for formalizing dialogical argumentation. However, each proposal involves a number of quite complex definitions, and there is significant diversity in the way different proposals define similar features. This complexity and diversity has hindered analysis and comparison of the space of proposals. To address this, we present a general approach to defining a wide variety of systems for dialogical argumentation. Our solution is to use an executable logic to specify individual systems for dialogical argumentation. This means we have a common language for specifying a wide range of systems, we can compare systems in terms of a range of standard properties, we can identify interesting classes of system, and we can execute the specification of each system to analyse it empirically.

1 INTRODUCTION

Dialogical argumentation involves agents exchanging arguments in activities such as discussion, debate, persuasion, and negotiation [3]. Dialogue games are now a common approach to characterizing argumentation-based agent dialogues (e.g. [2, 4, 7, 10, 13, 14, 15, 17, 18, 19, 20, 21, 22, 25, 26]). Dialogue games are normally made up of a set of communicative acts called moves, and sets of rules stating: which moves it is legal to make at any point in a dialogue (the *protocol*); the effect of making a move; and when a dialogue terminates. One attraction of dialogue games is that it is possible to embed games within games, allowing complex conversations made up of nested dialogues of more than one type.

In the current state of the art, describing a system for dialogical argumentation involves complex definitions with no standard way of presenting them. Hence, it is difficult to ensure the definitions are correct, it is difficult to show that they are well-behaved, and it is difficult to compare different proposals. There is a lack of consideration of general properties of argumentation dialogues, and as a result, there is a lack of formal criteria to delineate types of system. Furthermore, there is a lack of theoretical tools for designing systems and a lack of prototyping tools for evaluating systems empirically.

To address these issues, this paper introduces a simple and general framework for defining dialogical argumentation systems, called the *Framework for Dialogical Argumentation* (FDA). Each state of the dialogue comprises a private state for each agent, and a public state that all agents see. Each of these components of a dialogue state is represented by a set of literals. A dialogical argumentation system is specified in an executable logic by a set of logical rules. The rules specify how the components of a dialogue state are changed (by adding and/or deleting literals) to create the next dialogue state.

[1] Department of Informatics, King's College London, London, WC2R 2LS, UK, email: elizabeth.black@kcl.ac.uk
[2] Department of Computer Science, University College London, London, WCE1 6BT, UK, email: a.hunter@cs.ucl.ac.uk

2 LANGUAGE

We assume a set of variable and function symbols, from which we can construct terms and ground terms in the usual way. We also assume a set of predicate symbols, and we use these with terms, to construct literals and ground literals in the usual way. We use the literals to form classical formulae (respectively ground classical formulae) in the usual way using the disjunction, conjunction, and negation connectives. We construct modal formulae using the $\boxplus$, $\boxminus$, $\oplus$, and $\ominus$ modal operators. We only allow literals to be in the scope of a modal operator. If ϕ is a literal (respectively ground literal), then each of $\oplus\alpha$, $\ominus\alpha$, $\boxplus\alpha$, and $\boxminus\alpha$ is an **action unit** (respectively **ground action unit**). Informally, we describe the meaning of action units as follows.

- $\oplus\alpha$ means that the action by an agent is to add the literal α to its next private state.
- $\ominus\alpha$ means that the action by an agent is to delete the literal α from its next private state.
- $\boxplus\alpha$ means that the action by an agent is to add the literal α to the next public state.
- $\boxminus\alpha$ means that the action by an agent is to delete the literal α from the next public state.

We use the action units to form **action formulae** (respectively **ground action formulae**) as follows using the disjunction and conjunction connectives: (1) If ϕ is an action unit (respectively ground action unit), then ϕ is an action formula (respectively ground action formula); And (2) If α and β are action formulae, then $\alpha \vee \beta$ and $\alpha \wedge \beta$ are action formulae (respectively ground action formulae).

We define the action rules as follows, where $\mathsf{Variables}(\alpha)$ returns the set of variables occurring in α. Since the classical formulae and action formulae, as defined above, do not involve quantifiers, all variables in these formulae are free. For the action rules, we assume all free variables are in the scope of implicit universal quantifiers given outermost.

- If ϕ is a classical formula and ψ is an action formula such that $\mathsf{Variables}(\psi) \subseteq \mathsf{Variables}(\phi)$, then $\phi \rightarrow \psi$ is an **action rule**.
- If $\phi \rightarrow \psi$ is an action rule and $\mathsf{Variables}(\phi) = \emptyset$, then $\phi \rightarrow \psi$ is a **ground action rule**.

Example 1. *Consider the action rule* $\mathtt{b}(\mathtt{X}) \Rightarrow \boxplus\mathtt{c}(\mathtt{X})$ *where the predicates* `b` *denotes belief, and* `c` *denotes claim, and* `X` *is a variable. So the rule says that if an agent has a belief that can instantiate* `X`*, then the action is to claim it. Hence if* $\mathtt{b}(\mathtt{p})$ *is a literal in the agent's private state, or in the public state, then we will see later that we can obtain* $\mathtt{b}(\mathtt{p}) \Rightarrow \boxplus\mathtt{c}(\mathtt{p})$ *as a ground action rule.*

Implicit in the definitions for the language is the fact that we can use it as a meta-language [27]. For this, the object-language will be represented by terms in this meta-language. For instance, the object-level form $\mathtt{p}(\mathtt{a},\mathtt{b}) \rightarrow \mathtt{q}(\mathtt{a},\mathtt{b})$ can be represented by a term where the object-level literals $\mathtt{p}(\mathtt{a},\mathtt{b})$ and $\mathtt{q}(\mathtt{a},\mathtt{b})$ are represented by constant symbols, and $\rightarrow$ is represented by a function symbol. Then we can form the literal $\mathtt{belief}(\mathtt{p}(\mathtt{a},\mathtt{b}) \rightarrow \mathtt{q}(\mathtt{a},\mathtt{b}))$ where belief is a predicate symbol.

3 STATES

We use a state-based model of dialogical argumentation with the following definition of an execution state. To simplify the presentation, we restrict consideration in this paper to two agents. An execution represents a finite or infinite sequence of execution states. If the sequence is finite, then t denotes the terminal state, otherwise $t = \infty$.

Definition 1. *An* **execution** e *is a tuple* $e = (s_1, a_1, p, s_2, a_2, t)$*, where for each* $n \in \mathbb{N}$ *where* $0 \leq n \leq t$*,* $s_1(n)$ *is a set of ground literals,* $a_1(n)$ *is a set of ground action units,* $p(n)$ *is a set of ground literals,* $a_2(n)$ *is a set of ground action units,* $s_2(n)$ *is a set of ground literals, and* $t \in \mathbb{N} \cup \{\infty\}$*. For each* $n \in \mathbb{N}$*, if* $0 \leq n \leq t$*, then an* **execution state** *is* $e(n) = (s_1(n), a_1(n), p(n), a_2(n), s_2(n))$*. We call* $s_1(n)$ *the private state of agent 1 at time* n*,* $a_1(n)$ *the action state of agent 1 at time* n*,* $p(n)$ *the public state at time* n*,* $a_2(n)$ *the action state of agent 2 at time* n*,* $s_2(n)$ *the private state of agent 2 at time* n*. We call* $e(0)$ *the* **starting state***.*

Example 2. *The first 5 steps of an infinite execution where each row in the table is an execution state.*

n	$s_1(n)$	$a_1(n)$	$p(n)$	$a_2(n)$	$s_2(n)$
0	$\mathtt{b}(\mathtt{p})$		$\mathtt{t}(\mathtt{ann})$		$\mathtt{b}(\neg\mathtt{p})$
1	$\mathtt{b}(\mathtt{p})$	$\boxplus\mathtt{c}(\mathtt{p})$ $\boxminus\mathtt{t}(\mathtt{ann})$ $\boxplus\mathtt{t}(\mathtt{bob})$	$\mathtt{t}(\mathtt{ann})$		$\mathtt{b}(\neg\mathtt{p})$
2	$\mathtt{b}(\mathtt{p})$		$\mathtt{c}(\mathtt{p})$ $\mathtt{t}(\mathtt{bob})$	$\boxplus\mathtt{c}(\neg\mathtt{p})$ $\boxminus\mathtt{t}(\mathtt{bob})$ $\boxplus\mathtt{t}(\mathtt{ann})$	$\mathtt{b}(\neg\mathtt{p})$
3	$\mathtt{b}(\mathtt{p})$	$\boxplus\mathtt{c}(\mathtt{p})$ $\boxminus\mathtt{t}(\mathtt{ann})$ $\boxplus\mathtt{t}(\mathtt{bob})$	$\mathtt{c}(\mathtt{p})$ $\mathtt{c}(\neg\mathtt{p})$ $\mathtt{t}(\mathtt{ann})$		$\mathtt{b}(\neg\mathtt{p})$
4	$\mathtt{b}(\mathtt{p})$		$\mathtt{c}(\mathtt{p})$ $\mathtt{c}(\neg\mathtt{p})$ $\mathtt{t}(\mathtt{bob})$	$\boxplus\mathtt{c}(\neg\mathtt{p})$ $\boxminus\mathtt{t}(\mathtt{bob})$ $\boxplus\mathtt{t}(\mathtt{ann})$	$\mathtt{b}(\neg\mathtt{p})$
5	...	...	...	...	...

Later we will see how we can assign each agent one of the following action rules to generate the execution where the predicates b *denotes belief,* c *denotes claim, and* t *denotes turn, and* X *is a variable.*

- $\mathtt{t}(\mathtt{ann}) \wedge \mathtt{b}(\mathtt{X}) \Rightarrow \boxplus\mathtt{c}(\mathtt{X}) \wedge \boxminus\mathtt{t}(\mathtt{ann}) \wedge \boxplus\mathtt{t}(\mathtt{bob})$
- $\mathtt{t}(\mathtt{bob}) \wedge \mathtt{b}(\mathtt{X}) \Rightarrow \boxplus\mathtt{c}(\mathtt{X}) \wedge \boxminus\mathtt{t}(\mathtt{bob}) \wedge \boxplus\mathtt{t}(\mathtt{ann})$

In general, there is no restriction on the literals that can appear in the private and public state. The choice depends on the specific dialogical argumentation we want to specify. This flexibility means we can capture diverse kinds of information in the private state about agents by assuming predicate symbols for their own beliefs, objectives, preferences, arguments, etc, and for what they know about other agents. The flexibility also means we can capture diverse information in the public state about moves made, commitments made, etc. Furthermore, we can augment the literals in a private or public state using builtin predicates as explained next.

4 BUILTIN PREDICATES

Builtin predicates are literals that can be inferred from the literals in a private state plus the public state. For example, the builtin predicate $\mathtt{member}(\mathtt{a},\{\mathtt{b},\mathtt{a},\mathtt{c}\})$ holds in any state (assuming the usual definition). A convenient way to define builtin predicates is to use Prolog, but we could define and implement them in other languages.

Example 3. *Suppose we have predicates of the form* $\mathtt{belief}(\mathtt{Y})$ *in the private state of an agent, and* Y *is a formula. We can define builtin predicates* bels*,* argument*,* entails*, and* literal *as follows. For this, we use some builtin predicates that normally occur in Prolog software. These are* member*,* subset*,* atom*, and* $\mathtt{findall}(\mathtt{X},\mathtt{A},\mathtt{L})$*. The latter returns a list* L *of all the groundings for the variable* X *in atom* A *for which that instantiated atom is true. So for example, if we have the program* $\mathtt{p}(\mathtt{a},\mathtt{b}), \mathtt{p}(\mathtt{e},\mathtt{c}), \mathtt{p}(\mathtt{d},\mathtt{b}), \mathtt{p}(\mathtt{f},\mathtt{b})$*, and we have the call* $\mathtt{findall}(\mathtt{X},\mathtt{p}(\mathtt{X},\mathtt{b}),\mathtt{L})$ *then* L *is* $[\mathtt{a},\mathtt{d},\mathtt{f}]$*.*

$\mathtt{bels}(\mathtt{B}) :- \mathtt{findall}(\mathtt{X},\mathtt{belief}(\mathtt{X}),\mathtt{B}).$
$\mathtt{argument}(\mathtt{S},\mathtt{C}) :- \mathtt{bels}(\mathtt{B}),\mathtt{subset}(\mathtt{S},\mathtt{B}),\mathtt{entails}(\mathtt{S},\mathtt{C}).$
$\mathtt{entails}(\mathtt{S},\mathtt{C}) :- \mathtt{literal}(\mathtt{C}),\mathtt{member}(\mathtt{C},\mathtt{S}).$
$\mathtt{entails}(\mathtt{S},\mathtt{C}) :- \mathtt{member}(\mathtt{X} \rightarrow \mathtt{C},\mathtt{S}),\mathtt{entails}(\mathtt{S},\mathtt{X}).$
$\mathtt{entails}(\mathtt{S},\mathtt{X} \wedge \mathtt{Y}) :- \mathtt{entails}(\mathtt{S},\mathtt{X}),\mathtt{entails}(\mathtt{S},\mathtt{Y}).$
$\mathtt{literal}(\mathtt{X}) :- \mathtt{atom}(\mathtt{X}).$
$\mathtt{literal}(\neg\mathtt{X}) :- \mathtt{atom}(\mathtt{X}).$

The above is an example, and so in general, we do not assume any fixed definition for say argument or entails. For instance, here argument is defined so that there is no condition to ensure that the support S is minimal or consistent. If we require those conditions, then we revise this definition for the application.

For $e(n) = (s_1(n), a_1(n), p(n), a_2(n), s_2(n))$, the **reasoning state** for an agent x is $s_x(n) \cup p(n)$. This denotes the literals that agent x has available at time n in its private state and the public state. An agent has access to the definitions of the builtin predicates via a **base function**, denoted $Base$, that returns the closure of the literals that can be inferred from the reasoning state and the definitions of the builtin predicates.

Example 4. *Let* $Prog$ *be the Prolog program given in Ex 3. For an agent* x *with reasoning state* $s_x(n) \cup p(n)$*, let*

$$Base(s_x(n), p(n)) = \{\phi \mid Prog \cup s_x(n) \cup p(n) \vdash_{Prolog} \phi\}$$

where $Prog \cup s_x(n) \cup p(n) \vdash_{Prolog} \phi$ *denotes that the ground atom* ϕ *follows from the program* $Prog$ *and the literals in* $s_x(n) \cup p(n)$*. Suppose* $s_1(1)$ *contains* $\mathtt{belief}(\mathtt{p})$ *and* $\mathtt{belief}(\mathtt{p} \rightarrow \mathtt{q})$*, then* $Base(s_1(1), p(1))$ *contains* $\mathtt{argument}(\{\mathtt{p},\mathtt{p} \rightarrow \mathtt{q}\},\mathtt{q})$*. In this example, we skip the straightforward details of translating between literals and Prolog syntax (e.g. representing sets as lists).*

We could define builtin predicates to capture a range of proposals for argumentation, such as for ASPIC+ [24], DeLP [12], ABA [9], classical logic [3], or abstract argumentation [6, 8]. Since $Base$ is the closure of the reasoning state, it is straightforward to define it without using Prolog (e.g. declaratively using classical logic, or imperatively using pseudocode, or a programming language).

5 SYSTEMS

We define each FDA system in terms of a set of agents, where each agent is defined by a set of action rules. The action rules for an agent specify what moves the agent can potentially make based on the current state of the dialogue, and a selection function picks a subset of these to act upon.

Definition 2. *A* **system** *is a tuple* $(Base, Rules_x, Select_x, Start)$ *where* $\{1,2\}$ *is the set of agents,* $Base$ *is a base function,* $Rules_x$ *is the set of action rules for agent* x*,* $Select_x$ *is the selection function for agent* x*, and* $Start$ *is the set of starting states.*

Given the current state of an execution, the following definition captures which rules are fired. For agent x these are the ground rules that have the condition literals satisfied by the current private state $s_x(n)$ and public state $p(n)$, together with any implied builtin predicates. In this paper, we use classical entailment, denoted $\models$, for the satisfaction relation, but other entailment relations such as for Belnap's four logic could be used.

Definition 3. *For a system* $(Base, Rules_x, Select_x, Start)$ *and an execution* $e = (s_1, a_1, p, a_2, s_2, t)$*, the* **fired action formulae***, denoted* $\mathsf{Fired}_x(n)$*, is defined as follows where* $x \in \{1,2\}$*,* $n \in \{1,\ldots,t\}$*, and* $\mathsf{Grd}(Rules_x) = \{\phi' \Rightarrow \psi' \mid \phi \Rightarrow \psi \in Rules_x$ *and* $\phi' \Rightarrow \psi'$ *is a ground version of* $\phi \Rightarrow \psi\}$*.*

$$\{\psi' \mid \phi' \Rightarrow \psi' \in \mathsf{Grd}(Rules_x) \text{ and } Base(s_x(n), p(n)) \models \phi'\}$$

The **selection function** $Select_x$ picks a subset of the heads of the fired grounded action rules for an agent x and $n \in \mathbb{N}$, thereby specifying how the current state is changed into the next state of the execution. In general, we want simple definitions for the selection function. We illustrate some options below. Note, the second option below is an alternative to encoding turn-taking in action rules (c.f. Ex. 2). For the fourth option below, we assume each agent has a ranking over its ground action rules reflecting its preferences over the actions.

- $Select_x$ is an **exhaustive selection function** iff $Select_x(n) = \mathsf{Fired}_x(n)$.
- $Select_x$ is a **turn-taking selection function** iff
  - $Select_x(n) = \mathsf{Fired}_x(n)$ when x is 1 and n is odd
  - $Select_x(n) = \mathsf{Fired}_x(n)$ when x is 2 and n is even
  - $Select_x(n) = \emptyset$ when x is 2 and n is odd
  - $Select_x(n) = \emptyset$ when x is 1 and n is even
- $Select_x$ is a **non-deterministic selection function** iff $Select_x(n) = \{\phi\}$ where ϕ is the head of a randomly selected fired rule for agent x.
- $Select_x$ is a **ranked selection function** iff $Select_x(n) = \{\phi\}$ where ϕ is the head of the fired rule of highest rank for agent x.

In order to relate an action state in an execution with an action formula, we require the following definition of satisfaction.

Definition 4. *For an action state* $a_x(n)$*, and an action formula* ϕ*,* $a_x(n)$ **satisfies** ϕ*, denoted* $a_x(n)|\sim \phi$*, as follows.*

1. $a_x(n)|\sim \alpha$ *iff* $\alpha \in a_x(n)$ *when* α *is an action unit*
2. $a_x(n)|\sim \alpha \wedge \beta$ *iff* $a_x(n)|\sim \alpha$ *and* $a_x(n)|\sim \beta$
3. $a_x(n)|\sim \alpha \vee \beta$ *iff* $a_x(n)|\sim \alpha$ *or* $a_x(n)|\sim \beta$

For an action state $a_x(n)$*, and an action formula* ϕ*,* $a_x(n)$ **minimally satisfies** ϕ *denoted* $a_x(n) \Vdash \phi$*, iff* $a_x(n)|\sim \phi$ *and for all* $\{\psi_1, ..., \psi_i\} \subset a_x(n)$*,* $\{\psi_1, ..., \psi_i\}|\not\sim \phi$*.*

Example 5. *Consider the execution in Example 2. For agent 1 at n = 1, we have* $a_1(1) \Vdash \boxplus \mathtt{c(p)} \wedge \boxminus \mathtt{t(ann)} \wedge \boxplus \mathtt{t(bob)}$*.*

A system generates an execution when the first state $e(0)$ is an allowed starting state according to the system, and each action state minimally satisfies the selected actions for each agent, and each subsequent private state (respectively each subsequent public state) is the current private state (respectively current public state) for the agent updated by the actions given in the action state, as defined next.

Definition 5. *A system* $(Base, Rules_x, Select_x, Start)$ **generates** *an execution* $(s_1, a_1, p, s_2, a_2, t)$ *iff for all* $x \in \{1,2\}$ *and for all* $n \in \{0,\ldots,t-1\}$ *and where* $a(n) = a_1(n) \cup a_2(n)$

1. $e(0) \in Start$
2. $s_x(n+1) = (s_x(n) \setminus \{\phi \mid \ominus\phi \in a_x(n)\}) \cup \{\phi \mid \oplus\phi \in a_x(n)\}$
3. $p(n+1) = (p(n) \setminus \{\phi \mid \boxminus\phi \in a(n)\}) \cup \{\phi \mid \boxplus\phi \in a(n)\}$
4. $a_x(m) \Vdash \bigwedge(Select_x(m))$ *for* $m \in \{1,\ldots,t\}$
5. $a_1(m) \neq \emptyset$ *or* $a_2(m) \neq \emptyset$ *for* $m \in \{1,\ldots,t-1\}$
6. $a_x(t) = \emptyset$

Given the starting state, the subsequent states then depend on which action rules are fired and which actions are selected: Condition 1 ensures that the execution starts from an allowed starting point; Condition 2 ensures that the next private state for an agent is the current private state minus those literals that need to be removed, plus those literals that need to be added; Condition 3 ensures that the next public state is the current public state minus those literals that need to be removed, plus those literals that need to be added; Condition 4 ensures that after the starting state, the actions for each agent minimally satisfy those that are selected actions for the agent; and Conditions 5 and 6 ensure that if either agent has actions, then the execution continues, otherwise the execution terminates.

Example 6. *Consider the system where there are no builtin predicates,* $Select_x$ *is the exhaustive selection function, and the starting state is* $(\{\alpha, \delta\}, \{\}, \{\beta\}, \{\}, \{\beta\})$*.*

- $Rules_1 = \{\alpha \wedge \delta \Rightarrow \boxplus\alpha \wedge \ominus\delta)\}$*;*
- $Rules_2 = \{\alpha \wedge \beta \Rightarrow \oplus\alpha \wedge \boxminus\beta \wedge \ominus\beta\}$*;*

For this, there is one execution. It is a simplistic dialogue in which agent 1 has a literal in its private state that it makes public, and this causes agent 2 to change its private state to containing that literal.

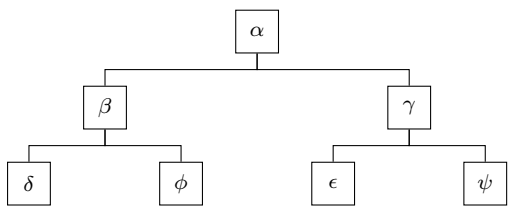
n	$s_1(n)$	$a_1(n)$	$p(n)$	$a_2(n)$	$s_2(n)$
0	α, δ		β		β
1	α, δ	$\boxplus\alpha, \ominus\delta$	β		β
2	α		α, β	$\oplus\alpha, \boxminus\beta, \ominus\beta$	β
3	α		α		α

Given a system, all the executions generated by the system with the same starting state are collected into an execution tree. So given the starting state at the root, each path is an execution.

Example 7. *Consider the system where there are no builtin predicates,* $Select_x$ *is the exhaustive selection function, and the starting state is* $(\{\}, \{\}, \{\alpha\}, \{\}, \{\})$*.*

- $Rules_1 = \{\alpha \Rightarrow \boxminus\alpha \wedge (\boxplus\beta \vee \boxplus\gamma)\}$
- $Rules_2 = \{\beta \Rightarrow \boxminus\beta \wedge (\boxplus\delta \vee \boxplus\phi), \gamma \Rightarrow \boxminus\gamma \wedge (\boxplus\epsilon \vee \boxplus\psi)\}$

Each branch refers to an execution (i.e. a dialogue). Each node in this tree is a public state.



We can also get branching in the execution tree by using the non-deterministic selection function. For instance, we can generate the above execution tree if we have $Rules_1$ containing $\alpha \Rightarrow \boxminus\alpha \wedge \boxplus\beta$ and $\alpha \Rightarrow \boxminus\alpha \wedge \boxplus\gamma$ and $Rules_2$ containing $\beta \Rightarrow \boxminus\beta \wedge \boxplus\delta$, $\beta \Rightarrow \boxminus\beta \wedge \boxplus\phi$, $\gamma \Rightarrow \boxminus\gamma \wedge \boxplus\epsilon$, and $\gamma \Rightarrow \boxminus\gamma \wedge \boxplus\psi$.

6 CASE STUDY

Here we consider a simple persuasion dialogue system adapted from the system by Prakken [22, 23]. It supports a range of dialogical moves including assertion of claims, querying of claims, retraction of claims, assertion of arguments for claims, and assertion of counterarguments to arguments. This means that counterarguments can be presented to arguments by recursion. For this, we assume the following variables and function: X is an agent and op(X) is the other agent; And B and C are formulae, and S is a set of formulae.

- bel(C) is a literal in the private state of an agent to denote that the agent believes formula C.
- claim(X, C) is a literal in the public state that represents that agent X has claimed formula C.
- why(X, C) is a literal in the public state that represents that agent X has queried the other agent about C.
- concede(X, C) is a literal in the public state that represents that agent X has conceded to the other agent that it believes C.
- posit(X, S, C) is a literal in the public state that represents that agent X has presented S as support for an argument with claim C.

In addition, we require the following builtin predicates that help agents to identify what moves to make.

- unqueried(C) holds when there is a claim for C, and there is no why move concerning C, and no argument for C has been posited.
- unconceded(X, B) holds when agent X has not conceded the formula B.
- hasarg(X, S, C) holds when agent X believes each of the items in S and S is the support for an argument with claim C.
- lackarg(X, C) holds when agent X lacks some beliefs for the support for an argument with claim C
- new(S, C) holds when no argument with support S and claim C has been posited.

Using these builtin predicates, we can define the following action rules that are in both R_1 and R_2, where X, C, S, T, and B are variables.

- claim(X, C) ∧ unqueried(C) ⇒ ⊞why(op(X), C)
- claim(X, C) ∧ why(op(X), C) ∧ lacksarg(X, C) ⇒ ⊟claim(X, C) ∧ ⊟why(op(X), C)
- why(op(X), C) ∧ hasarg(X, S, C) ∧ new(S, C) ⇒ ⊞posit(X, S, C) ∧ ⊟why(op(X), C)
- posit(X, S, C) ∧ in(B, S) ∧ bel(B) ∧ unconceded(op(X), B) ⇒ ⊞concede(op(X), B)
- posit(X, S, C) ∧ (in(B, S) ∨ B = C) ∧ hasarg(op(X), T, ¬B) ∧ new(T, ¬B) ⇒ ⊞posit(op(X), T, ¬B)

The following is the sequence of actions in the execution of the system with the turn-taking selection function. The dialogue is between agent 1 (Olga) and agent 2 (Paul) concerning the claim by Paul that a particular car is safe. Olga believes that the airbags can explode without an accident, and Paul believes that the report on this comes from unreliable newspaper reports. Olga also believes that the car is too fast. The starting state includes an action which in effect has initiated the dialogue. For the starting state, $s_1(0)$ contains bel(airbag), bel(expld), bel(expld → ¬safe), bel(fast), and bel(fast → ¬safe), and $p(0) = \emptyset$, and $s_2(0)$ contains bel(airbag), bel(airbag → safe), bel(unreliablenews), and bel(unreliablenews → ¬expld).

n	x	$a_x(n)$
0	2	⊞claim(Paul, safe)
1	1	⊞why(Olga, safe)
2	2	⊟why(Olga, safe) ⊞posit(Paul, {airbag, airbag → safe}, safe)
3	1	⊞concede(Olga, airbag) ⊞posit(Olga, {expld, expld → ¬safe}, ¬safe) ⊞posit(Olga, {fast, fast → ¬safe}, ¬safe)
4	2	⊞posit(Paul, {unreliablenews, unreliablenews → ¬expld}, ¬expld)

In the above execution, we can see how the actions on the public state can capture information about moves and commitments made by each agent x. With similar builtin predicates, and action rules, we can capture a range of existing proposals for dialogical argumentation in this formalism (e.g. [2, 4, 10]). Moreover, the approach allows any literals to be used in the execution state so allowing richer modeling of the information an agent has about the world and/or about the other agent, for instance taking into account uncertainty or goals, thereby allowing for more sophisticated behaviours to be captured via appropriate action rules.

7 PROPERTIES

We have presented FDA systems as a way to capture a wide range of interesting and useful systems for dialogical argumentation. So a natural question is how general is this approach? In this section, we consider some properties that hold for finite FDA systems (i.e. a system where for each action rule, there is a finite number of groundings of the rule). We show: (1) For any finite state machine (FSM), there is an FDA system and starting state that generates exactly the execution sequences consumed by the FSM; and (2) For any finite FDA system, and a starting state, there is an FSM that consumes exactly the finite execution sequences of the FDA system for that starting state.

A tuple $(States, Trans, Initial, Ends, Alphabet)$ is a **finite state machine** (FSM) where $States$ is a set of states such that $Initial \in States$ is the initial state and $Ends \subseteq States$ are the end states, $Alphabet$ is a set of letters, and $Trans : States \times Alphabet \mapsto States$ is the transition function that given a state and a letter returns the next state.

A language $Lang$ is a set of strings where each string is a sequence of letters. An FSM **accepts** a string $\tau_1 ... \tau_k$ in $Lang$ iff there is a sequence of states $\sigma_1, ..., \sigma_k$ such that σ_1 is the initial state, σ_k is an end state, and for each $1 \leq i < k$, $Trans(\sigma_i, \tau_i) = \sigma_{i+1}$.

Definition 6. *Let Lang be the set of strings formed from the letters in Alphabet. An execution $e = (s_1, a_1, p, a_2, s_2, t)$* **mimics** *a string $\rho \in Lang$ iff (1) ρ is a sequence of $t - 2$ letters; (2) for all n such that $1 < n \leq t - 1$, $|p(n) \cap Alphabet| = 1$; and (3) if τ is the nth letter in ρ, then $\tau \in p(n + 1)$.*

We explain the conditions in Def. 6 as follows: (1) the execution terminates at t, and the string has $t - 2$ letters; (2) each public state from $n = 2$ to $n = t - 1$ contains one letter; and (3) the nth letter of the string occurs as a positive literal in the $(n + 1)$th public state.

Example 8. *For the string $\rho = \tau_a\tau_b\tau_b\tau_c$, the following execution mimics it.*

n	$s_1(n)$	$a_1(n)$	$p(n)$	$a_2(n)$	$s_2(n)$
0			$start$		
1		$\boxplus\sigma_a$, $\boxplus\tau_a$, $\boxminus start$	$start$		
2		$\boxminus\sigma_a$, $\boxminus\tau_a$, $\boxplus\sigma_b$, $\boxplus\tau_b$	σ_a, τ_a		
3		$\boxminus\sigma_b$, $\boxminus\tau_b$, $\boxplus\sigma_b$, $\boxplus\tau_b$	σ_b, τ_b		
4		$\boxminus\sigma_b$, $\boxminus\tau_b$, $\boxplus\sigma_b$, $\boxplus\tau_c$	σ_b, τ_b		
5		$\boxminus\sigma_b$, $\boxminus\tau_c$, $\boxplus\sigma_c$	σ_b, τ_c		
6			σ_c		

We can generate this execution from the following set of action rules in R_1, and $R_2 = \emptyset$.

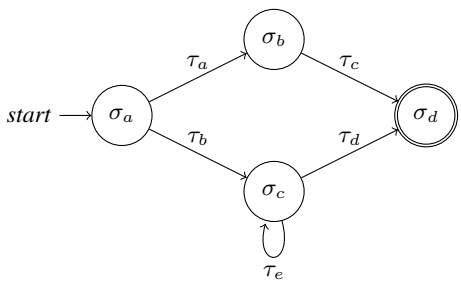
$$start \Rightarrow \boxminus start \wedge \boxplus\sigma_a \wedge \boxplus\tau_a$$
$$\sigma_a \wedge \tau_a \Rightarrow \boxminus\sigma_a \wedge \boxminus\tau_a \wedge \boxplus\sigma_b \wedge (\boxplus\tau_b \vee \boxplus\tau_c)$$
$$\sigma_b \wedge \tau_b \Rightarrow \boxminus\sigma_b \wedge \boxminus\tau_b \wedge \boxplus\sigma_b \wedge (\boxplus\tau_b \vee \boxplus\tau_c)$$
$$\sigma_b \wedge \tau_c \Rightarrow \boxminus\sigma_b \wedge \boxminus\tau_c \wedge \boxplus\sigma_c$$

Definition 7. *An FDA system S* **simulates** *an FSM M iff*

- *for all ρ such that M accepts ρ, there is an e such that S generates e and e mimics ρ.*
- *for all finite e such that S generates e, then there is a ρ such that M accepts ρ and e mimics ρ.*

Example 9. *The following FDA system S simulates the FSM M below where the starting state is $(\emptyset, \emptyset, \{start\}, \emptyset, \emptyset)$ and each agent has the following action rules and the exhaustive selection function.*

$$start \Rightarrow \boxminus start \wedge \boxplus\sigma_a \wedge (\boxplus\tau_a \vee \boxplus\tau_b)$$
$$\sigma_a \wedge \tau_a \Rightarrow \boxminus\sigma_a \wedge \boxminus\tau_a \wedge \boxplus\sigma_b \wedge \boxplus\tau_c$$
$$\sigma_a \wedge \tau_b \Rightarrow \boxminus\sigma_a \wedge \boxminus\tau_b \wedge \boxplus\sigma_c \wedge (\boxplus\tau_d \vee \boxplus\tau_e)$$
$$\sigma_b \wedge \tau_c \Rightarrow \boxminus\sigma_b \wedge \boxminus\tau_c \wedge \boxplus\sigma_d$$
$$\sigma_c \wedge \tau_d \Rightarrow \boxminus\sigma_c \wedge \boxminus\tau_d \wedge \boxplus\sigma_d$$
$$\sigma_c \wedge \tau_e \Rightarrow \boxminus\sigma_c \wedge \boxminus\tau_e \wedge \boxplus\sigma_c \wedge (\boxplus\tau_d \vee \boxplus\tau_e)$$



In the above example, we introduce an atom σ_i in the language of the action rules for each state σ_i in the FSM, and we introduce an action rule for each transition that effectively creates the transition from state σ_i and letter τ_j to state σ_k by deleting σ_i and τ_j and adding σ_k in the next public state in the execution. Next, we generalize this to give the result that any FSM can be simulated by an FDA system.

Theorem 1. *For each FSM M, there is an FDA system S such that S simulates M.*

Now we turn to showing that each execution generated by a finite FDA system and a starting state can be modelled by an FSM. For this, we require the following definition which says that a string reflects an execution when each letter in the sequence is a tuple $(a_1(n), a_2(n))$ where the first item is the actions of agent 1 at time n, and the second item is the actions of agent 2 at time n.

Definition 8. *A string ρ* **reflects** *an execution $e = (s_1, a_1, p, a_2, s_2, t)$ iff ρ is the string $\tau_0 \ldots \tau_{t-1}$ and for each $0 \leq n < t$, τ_n is the tuple $(a_1(n), a_2(n))$.*

Example 10. *The string $\rho = \tau_a\tau_b\tau_c\tau_d\tau_e$ reflects the following execution where $\tau_a = (\{\}, \{\})$, τ_b is $(\{\boxplus\alpha, \boxminus\delta\}, \{\boxplus\beta, \boxminus\delta\})$, τ_c is $(\{\boxminus\alpha\}, \{\boxplus\gamma\})$, τ_d is $(\{\boxplus\delta, \boxminus\gamma\}, \{\})$, and τ_e is $(\{\}, \{\boxplus\epsilon\})$.*

n	$s_1(n)$	$a_1(n)$	$p(n)$	$a_2(n)$	$s_2(n)$
0			δ		
1		$\boxplus\alpha$, $\boxminus\delta$	δ	$\boxplus\beta$, $\boxminus\delta$	
2		$\boxminus\alpha$	α, β	$\boxplus\gamma$	
3		$\boxplus\delta$, $\boxminus\gamma$	β, γ		
4			β, δ	$\boxplus\epsilon$	
5			β, δ, ϵ		

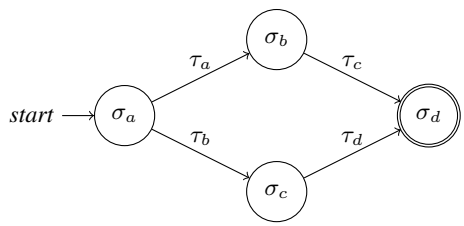
Definition 9. *Let $S = (Base, Rules_x, Select_x, Start)$ be an FDA system. An FSM M* **fabricates** *S with respect to $c \in Start$ iff*

- *for all ρ such that M accepts ρ, there is an e such that S generates e and $e(0) = c$ and ρ reflects e.*
- *for all finite e such that S generates e and $e(0) = c$, then there is a ρ such that M accepts ρ and ρ reflects e.*

Example 11. *Let S be an FDA system where each agent has the following action rules, and the exhaustive selection function, and let the starting state be $c = (\{\alpha\}, \{\}, \{\}, \{\}, \{\})$.*

$$\alpha \Rightarrow (\boxplus\beta \vee \boxplus\gamma) \wedge \ominus\alpha$$
$$\beta \Rightarrow \boxplus\delta \wedge \boxminus\beta$$
$$\gamma \Rightarrow \boxplus\delta \wedge \boxminus\gamma$$

The FSM M below fabricates the FDA system S.



σ_a *is* $(\{\alpha\}, \{\}, \{\})$ $\quad$ τ_a *is* $(\{\boxplus\beta, \ominus\alpha\}, \{\})$
σ_b *is* $(\{\}, \{\beta\}, \{\})$ $\quad$ τ_b *is* $(\{\boxplus\gamma, \ominus\alpha\}, \{\})$
σ_c *is* $(\{\}, \{\gamma\}, \{\})$ $\quad$ τ_c *is* $(\{\boxminus\beta, \boxplus\delta\}, \{\boxminus\beta, \boxplus\delta\})$
σ_d *is* $(\{\}, \{\delta\}, \{\})$ $\quad$ τ_d *is* $(\{\boxminus\gamma, \boxplus\delta\}, \{\boxminus\gamma, \boxplus\delta\})$

So the way we show that there is a way to have an FSM that fabricates a system is to build an FSM where each state is a tuple $(s_1(n), p(n), s_2(n))$, and each letter in the alphabet is a tuple $(a_1(n), a_2(n))$, for some n in an execution. Then the transitions in the FSM are defined by the action rules in the system. For this theorem, we are drawing on the fact that the ground action rules are essentially propositional (i.e. there is only a finite number of terms that can be used to ground the action rules).

Theorem 2. *For each $S = (Base, Rules_x, Select_x, Start)$, if S is a finite FDA system, and $c \in Start$, then there is an FSM M such that M fabricates S w.r.t. c.*

By using FSMs, we can also consider questions about specific systems, such as: Is termination possible; Is termination guaranteed; Are all states possible (i.e. reachable); And is a system minimal (i.e. are

some states redundant)? So by translating a system into an FSM, we can harness substantial theory and tools for analysing FSMs.

Whilst, we have shown that the FDA approach subsumes FSMs, and how a finite FDA system, with a particular starting state, can be modelled as an FSM, we need more than FSMs to better model infinite executions and to capture non-finite FDA systems. For this, we will turn to ω-automata and to temporal logics (including Gabbay's executable temporal logic [11]).

8 DISCUSSION

In this paper, we have presented a uniform way of presenting dialogical argumentation systems. The approach is based on a simple executable logic. Each action rule has an antecedent that refers to the current state of the dialogue, and a head that specifies the possible actions that can be undertaken on the next state of the dialogue. This uniform representation means that different dialogical argumentation systems can be compared more easily than before. Furthermore, properties of them (such as termination, consistency, fairness, deadlock, etc) can be identified and used to classify different approaches.

In dialogue systems, a protocol specifies the moves that are allowed by the participants. We can represent a protocol by an FDA system S^P. An FDA system S is compliant with a protocol S^P when each of the rules ϕ in $Rules_x$ is more constrained than a rule ϕ^P in $Rules_x^P$ (i.e. the condition of the ϕ entails the condition of ϕ^P and every set of action units satisfying the head of ϕ satisfies the head of ϕ^P). Furthermore, the strategy of an agent x is encoded in the action rules in $Rules_x$ since the moves an agent wants to make, and under what conditions, are specified by the conditions and heads of the rules. Whilst the choice of selection function also affects the strategy of an agent, we believe that only simple standard selection functions should be used, and that using action rules for capturing the strategy will be easier and more flexible to specify and better to analyse.

The FDA approach presented in this paper is the first proposal that uses a simple logical formalism for specifying and comparing diverse systems for dialogical argumentation. There have been few other proposals for general frameworks. Situation calculus has been used by Brewka [5] for a general framework. Situation calculus is based on second-order logic which is a complex logic to use and to reason with. The specifications are not based on the simple action rules used in the FDA approach. Rather, the specifications are based on second-order formulae that delineate the possible and necessary dialogue acts. The situation calculus framework only considers the public state, and so there is no consideration of private states. Finally, given the form of the second-order specifications, it is unlikely that it would be practical to execute the specifications.

General frameworks for dialogue games have been proposed by Maudet and Evrard [16] and by Parsons and McBurney [18]. They have both private and public aspects to the dialogue state, and diverse kinds of moves. They offer insights on issues concerning the formalisation of specific dialogical argumentation systems. However, they do not provide a formal definition of what constitutes a system for dialogical argumentation. It is therefore unclear what counts as a system and what does not. This means that it is difficult to identify general properties of the framework, and it is difficult to consider properties of specific classes of system.

In future work, we will systematically classify dialogical argumentation systems in the literature, develop a richer understanding of the role of protocols (extending for instance the proposals by Amgoud et al [1]), identify classes of FDA system with good properties, and generalize the FDA approach by considering uncertainty in states and conditions for action rules that consider previous states.

REFERENCES

[1] L. Amgoud, S. Belabbés, and H. Prade, 'A formal general setting for dialogue protocols', in *Artificial Intelligence: Methodology, Systems, and Applications*, volume 4183 of *LNCS*, pp. 13–23. Springer, (2006).

[2] L. Amgoud, N. Maudet, and S. Parsons, 'Arguments, dialogue and negotiation', in *Fourteenth European Conference on Artifcial Intelligence (ECAI 2000)*, pp. 338–342. IOS Press, (2000).

[3] Ph. Besnard and A. Hunter, *Elements of Argumentation*, MIT Press, 2008.

[4] E. Black and A. Hunter, 'An inquiry dialogue system', *Autonomous Agents and Multi-Agent Systems*, **19**(2), 173–209, (2009).

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

[6] M. Caminada and D. Gabbay, 'A logical account of formal argumentation', *Studia Logica*, **93**, 109–145, (2009).

[7] F. Dignum, B. Dunin-Keplicz, and R. Verbrugge, 'Dialogue in team formation', in *Issues in Agent Communication*, 264–280, Springer, (2000).

[8] P. Dung, 'On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games', *Artificial Intelligence*, **77**(2), 321–357, (1995).

[9] P. Dung, R. Kowalski, and F. Toni, 'Dialectical proof procedures for assumption-based admissible argumentation', *Artificial Intelligence*, **170**, 114–159, (2006).

[10] X Fan and F Toni, 'Assumption-based argumentation dialogues', in *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'11)*, pp. 198–203, (2011).

[11] D. Gabbay, 'The declarative past and imperative future: Executable temporal logic', in *Temporal logic in specification*, volume 398 of *LNCS*, pp. 409–448. Springer, (1989).

[12] A. García and G. Simari, 'Defeasible logic programming: An argumentative approach', *Theory and Practice of Logic Programming*, **4**(1), 95–138, (2004).

[13] C. Hamblin, 'Mathematical models of dialogue', *Theoria*, **37**, 567–583, (1971).

[14] D. Hitchcock, P. McBurney, and S. Parsons, 'A framework for deliberation dialogues', in *Fourth Biennial Conference of the Ontario Society for the Study of Argumentation (OSSA 2001)*, (2001).

[15] J. Mackenzie, 'Question begging in non-cumulative systems', *Journal of Philosophical Logic*, **8**, 117–133, (1979).

[16] N. Maudet and F. Evrard, 'A generic framework for dialogue game implementation', in *Proc. 2nd Workshop on Formal Semantics & Pragmatics of Dialogue*, p. 185198. University of Twente, (1998).

[17] P. McBurney and S. Parsons, 'Dialogue games in multi-agent systems', *Informal Logic*, **22**, 257–274, (2002).

[18] P. McBurney and S. Parsons, 'Games that agents play: A formal framework for dialogues between autonomous agents', *Journal of Logic, Language and Information*, **11**, 315–334, (2002).

[19] P. McBurney, R. van Eijk, S. Parsons, and L. Amgoud, 'A dialogue-game protocol for agent purchase negotiations', *Journal of Autonomous Agents and Multi-Agent Systems*, **7**, 235–273, (2003).

[20] S. Parsons, M. Wooldridge, and L. Amgoud, 'On the outcomes of formal inter-agent dialogues', in *2nd Int. Conf. on Autonomous Agents and Mutli-Agent Systems*, pp. 616–623, (2003).

[21] S. Parsons, M. Wooldridge, and L. Amgoud, 'Properties and complexity of some formal inter-agent dialogues', *J. of Logic and Comp.*, **13**(3), 347–376, (2003).

[22] H. Prakken, 'Coherence and flexibility in dialogue games for argumentation', *J. of Logic and Comp.*, **15**(6), 1009–1040, (2005).

[23] H. Prakken, 'Formal sytems for persuasion dialogue', *Knowledge Engineering Review*, **21**(2), 163–188, (2006).

[24] H. Prakken, 'An abstract framework for argumentation with structural arguments', *Argument and Computation*, **1**, 93–124, (2010).

[25] F. Sadri, F. Toni, and P. Torroni, 'Dialogues for negotiation: Agent varieties and dialogue sequences', in *8th Int. Workshop on Agent Theories, Architectures, and Languages*, pp. 69–84, (2001).

[26] D. Walton and E. Krabbe, *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*, SUNY Press, 1995.

[27] M. Wooldridge, P. McBurney, and S. Parsons, 'On the meta-logic of arguments', in *Argumentatoin in Multi-agent Systems*, volume 4049 of *LNCS*, pp. 42–56. Springer, (2005).