# Probabilistic Strategies in Dialogical Argumentation

Anthony Hunter[1]

Department of Computer Science, University College London,
Gower Street, London WC1E 6BT, UK

**Abstract.** In dialogical argumentation, a participant is often unsure what moves the other participant(s) might make. If the dialogue is proceeding according to some accepted protocol, then a participant might be able to determine what are the possible moves that the other might make, but the participant might be unsure as to which move will be chosen by the other agent. In this paper, propositional executable logic is augmented with probabilities that reflect the probability that any given move will be chosen by the agent. This provides a simple and lucid language that can be executed to generate a dialogue. Furthermore, a set of such rules for each agent can be represented by a probabilistic finite state machine (PFSM). For modelling dialogical argumentation, a PFSM can be used by one agent to model how the other agent may react to any dialogical move. An agent can then analyze the PFSM to determine the most likely outcomes of a dialogue given any choices it makes. This can be used by the agent to determine its choice of moves in order to optimize its outcomes from the dialogue.

## 1 Introduction

Dialogical argumentation involves agents exchanging arguments in activities such as discussion, debate, persuasion, and negotiation. Dialogue games are now a common approach to characterizing argumentation-based agent dialogues (e.g. [1–11]). In order to compare and evaluate dialogical argumentation systems, we proposed in a previous paper that first-order executable logic could be used as common theoretical framework to specify and analyze dialogical argumentation systems [12]. Then in [13], propositional executable logic was presented as a special case, and for which a finite state machine (FSM) can be generated. An FSM is a useful structure for investigating various properties of the dialogue, including conformance to protocols, and application of the minimax strategy.

We can improve on analyzing argumentation strategies by harnessing probabilistic information about an opponent to offer better decision making. In this paper, we address this need by introducing a probability assignment to the argumentation moves. For this, we introduce probabilistic executable logic and show how a specification for the protocols for a pair of agents in probabilistic executable logic can be represented and analyzed in the form of a probabilistic finite state machine (PFSM). By using Markov chains, we can determine the expected utility of such a specification, and we can optimize the expected utility for one of the agents by changing its specification to be deterministic.

## 2 Probabilistic executable logic

We assume a set of atoms $\mathcal{A}$ which we use to form propositional formulae in the usual way using 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 $\alpha$ is a literal, then each of $\oplus\alpha$, $\ominus\alpha$, $\boxplus\alpha$, and $\boxminus\alpha$ is an **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 $\alpha$ to its next private state; $\ominus\alpha$ means that the action by an agent is to delete the literal $\alpha$ from its next private state; $\boxplus\alpha$ means that the action by an agent is to add the literal $\alpha$ to the next public state; and $\boxminus\alpha$ means that the action by an agent is to delete the literal $\alpha$ from the next public state.

We use the action units to form **action formulae** as follows using the disjunction and conjunction connectives. If $\alpha_1, \ldots, \alpha_n$ are action units, and $v \in [0, 1]$, then $(v : \alpha_1 \wedge \ldots \wedge \alpha_n)$ is an **action option**. As we will see later, $v$ denotes the probability that an agent will undertake the actions $\alpha_1, \ldots, \alpha_n$. We compose the action options into action rules as follows. If $\beta$ is a classical formula, and $\gamma_1 = (v^1 : \alpha^1_1 \wedge \ldots \wedge \alpha^1_n)$, …, $\gamma_m = (v^m : \alpha^m_1 \wedge \ldots \wedge \alpha^m_n)$ are action options, where $v^1 + \ldots + v^m = 1$, then $\beta \Rightarrow \gamma_1 \vee \ldots \vee \gamma_m$ is an **action rule**. So an action rule has a consequent in "disjunctive normal form". Each disjunct is action option, and the sum of the probabilities is 1.

*Example 1.* Let $\mathcal{A} = \{\mathtt{b}(\mathtt{a}_1), \mathtt{b}(\mathtt{a}_2), \mathtt{c}(\mathtt{a}_1), \mathtt{c}(\mathtt{a}_2)\}$. So the following is an action rule (which we might use in an example where $\mathtt{b}$ denotes belief, and $\mathtt{c}$ denotes claim, and $\mathtt{a}_1$ and $\mathtt{a}_2$ are some items of information).

$$\mathtt{b}(\mathtt{a}_1) \wedge \mathtt{b}(\mathtt{a}_2) \Rightarrow (0.2 : \boxplus\mathtt{c}(\mathtt{a}_1) \wedge \boxminus\mathtt{c}(\mathtt{a}_2)) \vee (0.5 : \boxplus\mathtt{c}(\mathtt{a}_1)) \vee (0.3 : \boxplus\mathtt{c}(\mathtt{a}_2))$$

Implicit in the definitions for the language is the fact that we can use it as a meta-language [14]. For this, the object-language will be represented by terms in this meta-language. For instance, the object-level formula $\mathtt{p}(\mathtt{a}, \mathtt{b}) \rightarrow \mathtt{q}(\mathtt{a}, \mathtt{b})$ 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 atom $\mathtt{belief}(\mathtt{p}(\mathtt{a}, \mathtt{b}) \rightarrow \mathtt{q}(\mathtt{a}, \mathtt{b}))$ where $\mathtt{belief}$ is a predicate symbol. Note, in general, no special meaning is ascribed to the predicate symbols or terms. They are used as in classical logic. Also, the terms and predicates are all ground, and so it is essentially a propositional language.

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 $e = (s_1, a_1, p, a_2, s_2, t)$, 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 $e(n) = (s_1(n), a_1(n), p(n), a_2(n), s_2(n))$ where $e(0)$ is the* **initial state**. *We assume $a_1(0) = a_2(0) = \emptyset$. 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 $n$.*

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.

*Example 2.* The first 5 steps of an infinite execution where each row in the table is an execution state where `b` denotes belief, and `c` denotes claim.

| $n$ | $s_1(n)$ | $a_1(n)$ | $p(n)$ | $a_2(n)$ | $s_2(n)$ |
|---|---|---|---|---|---|
| 0 | $\mathtt{b(a)}$ | | | | $\mathtt{b(\neg a)}$ |
| 1 | $\mathtt{b(a)}$ | $\boxplus\mathtt{c(a)}, \boxminus\mathtt{c(\neg a)}$ | | | $\mathtt{b(\neg a)}$ |
| 2 | $\mathtt{b(a)}$ | | $\mathtt{c(a)}$ | $\boxplus\mathtt{c(\neg a)}, \boxminus\mathtt{c(a)}$ | $\mathtt{b(\neg a)}$ |
| 3 | $\mathtt{b(a)}$ | $\boxplus\mathtt{c(a)}, \boxminus\mathtt{c(\neg a)}$ | $\mathtt{c(\neg a)}$ | | $\mathtt{b(\neg a)}$ |
| 4 | $\mathtt{b(a)}$ | | $\mathtt{c(a)}$ | $\boxplus\mathtt{c(\neg a)}, \boxminus\mathtt{c(a)}$ | $\mathtt{b(\neg a)}$ |
| 5 | ... | ... | ... | ... | ... |

We define a system in terms of the action rules for each agent, which specify what moves the agent can potentially make based on the current state of the dialogue. In this paper, we assume agents take turns, and at each time point the actions are from the head of just one rule (as defined in the rest of this section). We also assume in this paper that at most one rule can have an antecedent satisfiable at any time.

**Definition 2.** *A* **system** *is a tuple $(Rules_x, Initials)$ where $Rules_x$ is the set of action rules for agent $x \in \{1, 2\}$, and $Initials$ is the set of initial states.*

For an agent $x$, the information it has available at any point $n$ in the dialogue is determined by the private state $s_x(n)$ and public state $p(n)$. We augment this set of atoms by the closed world assumption as follows.

**Definition 3.** *Let $s_x(n)$ be the private state of agent $x$ at time $n$, and let $p(n)$ be the public state of agent $x$ at time $n$. The* **knowledge of agent $x$ at time $n$**, *denoted $k_x(n)$, is defined as follows.*

$$k_x(n) = s_x(n) \cup p(n) \cup \{\neg\alpha \mid \alpha \in \mathcal{A} \text{ and } \alpha \notin s_x(n) \cup p(n)\}$$

*Example 3.* Let $\mathcal{A} = \{\mathtt{b(a)}, \mathtt{b(\neg a)}, \mathtt{b(b)}, \mathtt{b(\neg b)}, \mathtt{c(a)}, \mathtt{c(\neg a)}, \mathtt{c(b)}, \mathtt{c(\neg b)}\}$. Also let $s_1(2) = \{\mathtt{b(a)}\}$ and $p(2) = \{\mathtt{c(a)}\}$. Therefore, $k_1(2) = \{\mathtt{b(a)}, \neg\mathtt{b(\neg a)}, \neg\mathtt{b(b)}, \neg\mathtt{b(\neg b)}, \mathtt{c(a)}, \neg\mathtt{c(\neg a)}, \neg\mathtt{c(b)}, \neg\mathtt{c(\neg b)}\}$.

We give two constraints on an execution to ensure that they are well-behaved. The first (propagated) ensures that 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. The second (engaged) ensures that an execution does not have one state with no actions followed immediately by another state with no actions (otherwise the dialogue can lapse) except at the end of the dialogue where neither agent has further actions.

**Definition 4.** *An execution $(s_1, a_1, p, a_2, s_2, t)$ is* **propagated** *iff for all $x \in \{1, 2\}$, for all $n \in \{0, \ldots, t-1\}$, where $a(n) = a_1(n) \cup a_2(n)$*

1. $s_x(n+1) = (s_x(n) \setminus \{\phi \mid \ominus\phi \in a_x(n)\}) \cup \{\phi \mid \oplus\phi \in a_x(n)\}$
2. $p(n+1) = (p(n) \setminus \{\phi \mid \boxminus\phi \in a(n)\}) \cup \{\phi \mid \boxplus\phi \in a(n)\}$

**Definition 5.** *Let $e = (s_1, a_1, p, a_2, s_2, t)$ be an execution and $a(n) = a_1(n) \cup a_2(n)$. $e$ is* **finitely engaged** *iff (1) $t \neq \infty$; (2) for all $n \in \{1, \ldots, t-2\}$, if $a(n) = \emptyset$, then $a(n+1) \neq \emptyset$; (3) $a(t-1) = \emptyset$; and (4) $a(t) = \emptyset$. $e$ is* **infinitely engaged** *iff (1) $t = \infty$; and (2) for all $n \in \mathbb{N}$, if $a(n) = \emptyset$, then $a(n+1) \neq \emptyset$.*

The next definition shows how a system provides the initial state of an execution and the actions that can appear in an execution. It also ensures turn taking by the two agents. Given the current state of an execution $n$, the following definition captures which rules are fired. For agent $x$, these are the rules that have the condition literals satisfied by the current knowledge of the agent (i.e. $k_x(n)$). We use classical entailment, denoted $\models$, for satisfaction, but other relations could be used (e.g. Belnap's four valued logic). Also recall that in this paper we assume that the antecedents of the rules for an agent are such that at most one rule can fire for any point of the execution. In general, this is not essential, but it makes the definitions simpler.

**Definition 6.** *Let $S = (Rules_x, Initials)$ be a system and $e = (s_1, a_1, p, a_2, s_2, t)$ be an execution. $S$* **generates** *$e$ iff (1) $e$ is propagated; (2) $e$ is finitely engaged or infinitely engaged; (3) $e(0) \in Initials$; and (4) for all $m \in \{1, \ldots, t-1\}$*

1. *If $m$ is odd, then $a_2(m) = \emptyset$ and either $a_1(m) = \emptyset$ or there is an $\phi \Rightarrow (v^1 : \psi^1) \vee \ldots \vee (v^j : \psi^j) \in Rules_1$ where $k_1(m) \models \phi$ and there is an $i \in \{1, \ldots, j\}$ where $\psi^i = \alpha_1 \wedge \ldots \wedge \alpha_p$ and $a_1(m) = \{\alpha_1, \ldots, \alpha_p\}$*
2. *If $m$ is even, then $a_1(m) = \emptyset$ and either $a_2(m) = \emptyset$ or there is an $\phi \Rightarrow (v^1 : \psi^1) \vee \ldots \vee (v^j : \psi^j) \in Rules_2$ where $k_2(m) \models \phi$ and there is an $i \in \{1, \ldots, j\}$ where $\psi^i = \alpha_1 \wedge \ldots \wedge \alpha_p$ and $a_2(m) = \{\alpha_1, \ldots, \alpha_p\}$*

The rules for each agent constitute the protocol for each agent. So the set of executions generated by a system is the set of executions allowed by the protocols of the two agents.

*Example 4.* We can obtain the execution in Example 2 with the following rules where the first is for agent 1, and the second is for agent 2. The first action option of the first rule is used in steps 1 and 3, and the first action option of the second rule is used in steps 2 and 4.

- $\texttt{b(a)} \Rightarrow (\texttt{0.6} : \boxplus\texttt{c(a)} \wedge \boxminus\texttt{c}(\neg\texttt{a})) \vee (\texttt{0.4} : \boxplus\top)$
- $\texttt{b}(\neg\texttt{a}) \Rightarrow (\texttt{0.8} : \boxplus\texttt{c}(\neg\texttt{a}) \wedge \boxminus\texttt{c(a)}) \vee (\texttt{0.2} : \boxplus\top)$

Note, for the second action option, we have $\boxplus\top$ as the only action. This denotes an "empty action", or a "skip action", since there is no material effect on the public or private states by this choice of action.

So far we have not considered the probabilities. Given a system and an execution state $e$, for each execution state $e(n)$, we can obtain the probability that this state will result in execution state $e(n+1)$. We obtain this from the action rule, by taking the probability of the action option, as follows.

**Definition 7.** *Let $S = (Rules_x, Initials)$ be a system and $e = (s_1, a_1, p, a_2, s_2, t)$ be an execution. A* **probability function for execution** *$e$, denoted $pr$, is defined as follows. For all $m \in \{1, \ldots, t-1\}$*

1. *If $m$ is odd, and $a_1(m) = \emptyset$, then $pr(e(m), e(m+1)) = 1$.*
2. *If $m$ is odd, and there is an $\phi \Rightarrow (v^1 : \psi^1) \vee \ldots \vee (v^j : \psi^j) \in Rules_1$ s.t. $k_1(m) \models \phi^i$, then $pr(e(m), e(m+1)) = v^i$, where $i \in \{1, \ldots, j\}$.*
3. *If $m$ is even, and $a_2(m) = \emptyset$, then $pr(e(m), e(m+1)) = 1$.*
4. *If $m$ is even, and there is an $\phi \Rightarrow (v^1 : \psi^1) \vee \ldots \vee (v^j : \psi^j) \in Rules_2$ s.t. $k_2(m) \models \phi^i$, then $pr(e(m), e(m+1)) = v^i$, where $i \in \{1, \ldots, j\}$.*

*Example 5.* Using the action rules in Example 4 to generate the execution in Example 2, we obtain the following for the steps 1 to 5: $pr(e(1), e(2)) = 0.6$, $pr(e(2), e(3)) = 0.8$, $pr(e(3), e(4)) = 0.6$, and $pr(e(4), e(5)) = 0.8$.

Executable logic can be used to formalize a diverse range of dialogical argumentation where agents exchange arguments and attacks [12, 13]. It is straightforward to formalize the exchange of a wide range of moves and content for both abstract and logical argumentation, and richer notions, such as value-based argumentation [15]), can be captured.

## 3 Probabilistic finite state machines

Probabilistic finite state machines (PFSMs) are an important approach in computer science for modelling behaviours of interacting modules when there is uncertainty in the choices made by those modules. The formalization augments finite state machines with a probability distribution over the transitions coming out of each state [16]. In this section, we show how a executable logic system, together with a choice of initial state, can be used to generate a PFSM.

**Definition 8.** *A tuple $M = (States, Arcs, Start, Prob)$ is a* **probabilistic finite state machine** *(PFSM) where $States$ is a set of states, $Start \in States$, $Arc \subseteq States \times States$ is a set of arcs, $Prob : Arcs \to [0, 1]$ is a probability function such that $\sum_{s' \in States \text{ s.t. } (s,s') \in Arcs} Prob(s, s') = 1$ for each $s \in States$.*

The $Prob$ assignment is the probability that that arc is chosen. The transitions out of a state sum to 1.

**Definition 9.** *Let $M$ be a PFSM, and let $\rho = \sigma_1, \ldots, \sigma_k \subseteq States$ be a sequence of states. $\rho$ is a* **walk** *in $M$ iff (1) if $k = 1$, then $\sigma_1$ is the node reached immediately from the $Start$ node and (2) if $k > 1$, then $\sigma_1, \ldots, \sigma_{k-1}$ is a walk in $M$ and $(\sigma_{k-1}, \sigma_k) \in Arcs$.*

*Example 6.* The following is an PFSM where the probability of each transition labels the arc. For this, $\sigma_1, \sigma_2, \sigma_3, \sigma_2, \sigma_3, \sigma_4$ is a walk.

$$\begin{tikzcd}
 & & & & \sigma_5 \arrow[r, "1"] & \sigma_7 \arrow[r, "1"] & \sigma_9 \\
\text{start} \arrow[r] & \sigma_1 \arrow[r, "1"] & \sigma_2 \arrow[r, bend right, "1"'] & \sigma_3 \arrow[l, bend right, "0.2"'] \arrow[ur, bend left, "0.5"] \arrow[r, "0.3"] & \sigma_4 \arrow[r, "1"] & \sigma_6 \arrow[r, "1"] & \sigma_8
\end{tikzcd}$$

We can construct an FSM that represents the set of executions for an initial state for a system. For this, each state is a tuple $(y, s_1(n), p(n), s_2(n))$, where $n$ is an execution step and $y$ is the agent holding the turn when $n < t$ and $r$ is 0 when $n = t$. The probability that any given actions are executed is given by the probability assignment to the rule. So each execution is a Markov chain.

**Definition 10.** *A PFSM $M = (States, Arcs, Start, Prob)$* **represents** *a system $S = (Rules_x, Initials)$ for an initial state $I \in Initials$ iff*

*(1)* $$\begin{aligned} States = \{(y, s_1(n), p(n), s_2(n)) \mid\ & S \text{ generates } e = (s_1, a_1, p, a_2, s_2, t) \\ & \text{and } I = (s_1(0), a_1(0), p(0), a_2(0), s_2(0)) \\ & \text{and } y = 0 \text{ when } n = t \\ & \text{and } y = 1 \text{ when } n < t \text{ and } n \text{ is odd} \\ & \text{and } y = 2 \text{ when } n < t \text{ and } n \text{ is even } \} \end{aligned}$$

*(2) $Start = (1, s_1(0), p(0), s_2(0))$ where $I = (s_1(0), a_1(0), p(0), a_2(0), s_2(0))$*

*(3) $Arcs$ is the smallest subset of $States \times States$ s.t. for all executions $e$ and for all $n < t$ there is a transition $\tau \in Arcs$ such that*

$$\tau = ((x, s_1(n), p(n), s_2(n)), (y, s_1(n + 1), p(n + 1), s_2(n + 1)))$$

*where $x$ is 1 when $n$ is odd, $x$ is 2 when $n$ is even, $y$ is 1 when $n + 1 < t$ and $n$ is odd, $y$ is 2 when $n + 1 < t$ and $n$ is even, and $y$ is 0 when $n + 1 = t$.*

(4) *$Prob$ is the smallest subset of $Arcs \times [0, 1]$ s.t. for all $\tau \in Trans$,*

*if $\tau = ((x, s_1(n), p(n), s_2(n)), (y, s_1(n + 1), p(n + 1), s_2(n + 1)))$*
*then $Prob(\tau) = pr(e(n), e(n + 1))$*

*where* $e(n+1) = (s_1(n+1), a_1(n+1), p(n+1), a_2(n+1), s_2(n+1), q(n+1))$, $e(n) = (s_1(n), a_1(n), p(n), a_2(n), s_2(n), q(n))$, *and pr is the execution probability function.*

*Example 7.* Consider the PFSM in Example 6 where the states are defined as follows.

$$\begin{array}{lll}
\sigma_1 = (\mathtt{1}, \emptyset, \{\mathtt{a}\}, \emptyset) & \sigma_4 = (\mathtt{2}, \emptyset, \{\mathtt{d}\}, \emptyset) & \sigma_7 = (\mathtt{1}, \emptyset, \{\mathtt{e}\}, \emptyset) \\
\sigma_2 = (\mathtt{2}, \emptyset, \{\mathtt{b}\}, \emptyset) & \sigma_5 = (\mathtt{2}, \emptyset, \{\mathtt{e}\}, \emptyset) & \sigma_8 = (\mathtt{0}, \emptyset, \{\mathtt{d}\}, \emptyset) \\
\sigma_3 = (\mathtt{1}, \emptyset, \{\mathtt{c}\}, \emptyset) & \sigma_6 = (\mathtt{1}, \emptyset, \{\mathtt{d}\}, \emptyset) & \sigma_9 = (\mathtt{0}, \emptyset, \{\mathtt{e}\}, \emptyset)
\end{array}$$

This can be obtained from the following set of rules for each agent so that the PFSM represents the system: (1) $\mathtt{a} \Rightarrow (\mathtt{1} : \boxplus\mathtt{b} \wedge \boxminus\mathtt{a})$; (2) $\mathtt{b} \Rightarrow (\mathtt{1} : \boxplus\mathtt{c} \wedge \boxminus\mathtt{b})$; and (3) $\mathtt{c} \Rightarrow (\mathtt{0.2} : \boxplus\mathtt{b} \wedge \boxminus\mathtt{c}) \vee (\mathtt{0.3} : \boxplus\mathtt{d} \wedge \boxminus\mathtt{c}) \vee (\mathtt{0.5} : \boxplus\mathtt{e} \wedge \boxminus\mathtt{c})$.

*Example 8.* Consider the following set of action rules. For this, let $\mathtt{h(a)}$ denote that an agent holds an argument $\mathtt{a}$ in its private state, let $\mathtt{a(a)}$ denote that argument $\mathtt{a}$ has been posited in the public state, and let $\mathtt{e(a,b)}$ denote that argument $\mathtt{a}$ attacks argument $\mathtt{b}$.

– $\mathtt{h(a)} \wedge \mathtt{h(b)} \Rightarrow (\mathtt{0.5} : \boxplus\mathtt{a(a)}) \vee (\mathtt{0.5} : \boxplus\mathtt{a(b)})$
– $\mathtt{h(a)} \wedge \neg\mathtt{a(a)} \wedge \mathtt{a(b)} \wedge \mathtt{e(a,b)} \Rightarrow (\mathtt{1} : \boxplus\mathtt{a(a)} \wedge \boxplus\mathtt{e(a,b)})$
– $\mathtt{h(b)} \wedge \neg\mathtt{a(b)} \wedge \mathtt{a(a)} \wedge \mathtt{e(a,b)} \Rightarrow (\mathtt{1} : \boxplus\mathtt{a(b)} \wedge \boxplus\mathtt{e(a,b)})$

The first action rule adds the argument $\mathtt{a}$ to the public state or adds the argument $\mathtt{b}$ to the public state. The second action rule adds $\mathtt{a}$ to the public state if it has not already been added, and it adds the attack by $\mathtt{a}$ on $\mathtt{b}$ to the public state. The third action rule adds $\mathtt{b}$ to the public state if it has not already been added, and it adds the attack by $\mathtt{a}$ on $\mathtt{b}$ to the public state.

With this set of action rules, and the initial state $(\{\mathtt{h(a)}, \mathtt{h(b)}\}, \{\}, \{\}, \{\}, \{\})$, we obtain the following PFSM, with the states defined below.

$$\begin{tikzcd}
 & & \sigma_2 \arrow[r, "1"] & \sigma_4 \arrow[dr, "1"] & & & \\
\text{start} \arrow[r] & \sigma_1 \arrow[ur, "0.5"] \arrow[dr, "0.5"'] & & & \sigma_6 \arrow[r, "1"] & \sigma_7 \arrow[r, "1"] & \sigma_8 \\
 & & \sigma_3 \arrow[r, "1"'] & \sigma_5 \arrow[ur, "1"'] & & &
\end{tikzcd}$$

$$\begin{array}{ll}
\sigma_1 = (\mathtt{1}, \{\mathtt{h(a)}, \mathtt{h(b)}\}, \{\}, \{\}) & \sigma_5 = (\mathtt{1}, \{\mathtt{h(a)}, \mathtt{h(b)}\}, \{\mathtt{a(b)}\}, \{\}) \\
\sigma_2 = (\mathtt{2}, \{\mathtt{h(a)}, \mathtt{h(b)}\}, \{\mathtt{a(a)}\}, \{\}) & \sigma_6 = (\mathtt{2}, \{\mathtt{h(a)}, \mathtt{h(b)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{e(a,b)}\}, \{\}) \\
\sigma_3 = (\mathtt{2}, \{\mathtt{h(a)}, \mathtt{h(b)}\}, \{\mathtt{a(b)}\}, \{\}) & \sigma_7 = (\mathtt{1}, \{\mathtt{h(a)}, \mathtt{h(b)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{e(a,b)}\}, \{\}) \\
\sigma_4 = (\mathtt{1}, \{\mathtt{h(a)}, \mathtt{h(b)}\}, \{\mathtt{a(a)}\}, \{\}) & \sigma_8 = (\mathtt{0}, \{\mathtt{h(a)}, \mathtt{h(b)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{e(a,b)}\}, \{\})
\end{array}$$

So this set of action rules implicitly constructs an abstract argument graph in the public state. The end state is $\sigma_8$ and it contains the specification of the abstract argument graph $\mathtt{a} \rightarrow \mathtt{b}$.

**Proposition 1.** *For each $S = (Rules_x, Initials)$, there is a PFSM $M$ such that $M$ represents $S$ for an initial state $I \in Initials$.*

In the next definition, we provide the conditions under which a walk over $k$ states is equivalent to the $k$ steps of an execution after the initial state.

**Definition 11.** *A sequence of states $\rho = \sigma_1, \ldots, \sigma_k$* **reflects** *an execution $e = (s_1, a_1, p, a_2, s_2, t)$ iff for each $i \in \{1, \ldots, k-1\}$,*

1. *$\sigma_i = (s_1(i), p(i), s_2(i))$*
2. *$Prob((\sigma_i, \sigma_{i+1})) = pr(e(i), e(i+1))$*

So for each initial state for a system, we can obtain a PFSM that is a concise representation of the executions of the system following that initial state.

**Proposition 2.** *Let $S = (Rules_x, Initials)$ be a system, and let $M$ be a PFSM that represents $S$ for $I \in Initials$:*

1. *For all $\rho$ s.t. $\rho$ is a walk in $M$, there is an execution $e$ s.t. $S$ generates $e$ and $e(0) = I$ and $\rho$ reflects $e$.*
2. *For all finite executions $e$ s.t. $S$ generates $e$ and $e(0) = I$, then there is a $\rho$ such that $\rho$ is a walk in $M$ and $\rho$ reflects $e$.*

A PFSM provides a more efficient representation of all the possible executions than the set of executions for an initial state. For instance, if there is a set of states that appear in some permutation of each of the executions then this can be more compactly represented by an PFSM.

Furthermore, we can ask simple questions such as is termination possible, is termination guaranteed, and is one system subsumed by another? Then by analyzing the Markov chains, we can answer questions such as what is the probability that we leave a state and never return, or the probability that we visit a state infinitely often? So by translating a system into a PFSM, we can harness substantial theory and tools for analyzing PFSMs.

There are various options we have for dealing with cycles in PFSMs. Here, we consider dropping arcs so as to remove cycles. For this, when there is a walk that is looping back to a state that has already been visited on the walk, that arc is dropped. Then the probability of the arcs that are dropped are redistributed to any remaining arcs.

**Definition 12.** *Let $M$ be a PFSM. $M' = (States, Arcs', Start, Prob)$ is* **derived** *from $M = (Starts, Arcs, Start, Prob)$ iff*

1. *$Arcs'$ is a maximal subset of $Arcs$ s.t. for every walk $\sigma_1, \ldots, \sigma_n$ in $M$, if $\sigma_1 = \sigma_n$, then $(\sigma_{n-1}, \sigma_n) \notin Arcs'$.*
2. *For each $\sigma_i \in States$, if $(\sigma_i, \sigma_j) \in Arcs'$, let $Prob'((\sigma_i, \sigma_j)) =$*

$$\frac{Prob((\sigma_i, \sigma_j))}{\sum_{(\sigma_i, \sigma_k) \in Arcs'} Prob((\sigma_i, \sigma_k))}$$

*Example 9.* Consider the PFSM in Example 6. The following is derived from it.

$$\begin{tikzcd}
 & & & & \sigma_5 \arrow[r, "1"] & \sigma_7 \arrow[r, "1"] & \sigma_9 \\
\text{start} \arrow[r] & \sigma_1 \arrow[r, "1"] & \sigma_2 \arrow[r, "1"] & \sigma_3 \arrow[r, "0.375"] \arrow[ur, bend left, "0.625"] & \sigma_4 \arrow[r, "1"] & \sigma_6 \arrow[r, "1"] & \sigma_8
\end{tikzcd}$$

**Proposition 3.** *If $M$ is a PFSM, and $M'$ is derived from $M$, then $M'$ is an acyclic PFSM.*

We can regard each walk in $M$ as a Markov chain. Hence, if the graph is acyclic, then we can calculate the probability of walk $\sigma_1, \ldots, \sigma_k$ as follows.

$$\prod_{i \in \{1,\ldots,k-1\}} Prob((\sigma_i, \sigma_{i+1}))$$

For acyclic graphs, the probability that a dialogue is in any particular end state (i.e. state with no arcs out of the state) is the sum of the probabilities of the walks that terminate in that end state. Let $\mathsf{Walks}(M, \sigma) = \{\rho \mid \rho = \sigma_1, \ldots, \sigma_k$ is a walk in $M$ and $\sigma_k = \sigma\}$ be the set of walks with $\sigma$ as end state.

**Proposition 4.** *Let $M$ be an PFSM, and let $\mathsf{End}(M)$ be the set of end states in $M$. If $M$ is acyclic, then $\sum_{\sigma \in \mathsf{End}(M)} \sum_{\rho \in \mathsf{Walks}(M,\sigma)} Prob(\rho) = 1$*

*Example 10.* For the PFSM in Example 9, $Prob(\sigma_1, \sigma_2, \sigma_3, \sigma_5, \sigma_7, \sigma_9) = 0.625$, and $Prob(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_6, \sigma_8) = 0.375$.

In the rest of the paper, we assume that we will have an acyclic PFSM. So if a system gives a cycle, we use the above method to get an acyclic PFSM.

## 4 Analyzing outcomes through search

A PFSM generated by a system and an initial state represents the uncertainty that an agent will make any move allowed by the protocol at each point in the dialogue. By exhaustively constructing a search tree, we can analyze an acyclic PFSM. A **search tree** $T$ for an acyclic PFSM $M$ is the smallest tree where every walk $\rho$ in $M$ that terminates in an end state in $M$ (i.e. a state with no transitions out of the state) corresponds to a branch in $T$. So for each walk $\rho = \tau_1 \ldots \tau_{t-1}$, the source node in $\tau_1$ is the root of the tree, and the destination node in $\tau_{t-1}$ is a leaf node. The probability of a leaf is the product of the probability values on the arcs from root to leaf.

We use a **utility function** to measure the value of each leaf in a search tree. For argumentation, there is a wide range of utility functions. In the example below, we have used grounded semantics to determine whether a specified argument (i.e. a goal argument) is in the grounded extension of the argument graph in the public state of the leaf node. A refinement is the **weighted utility function** which weights the utility by $1/d$ where $d$ is the depth of the leaf. This favours shorter dialogues. Further definitions arise from using other semantics and richer formalisms such as valued-based argumentation [15].

**Definition 13.** *For an PFSM $M$, where $Prob$ is the probability function, let the search tree be $T$, let $U$ be a utility function, and let $E$ be a function defined as follows.*

- *If $\sigma$ is a leaf node in $T$, then $E(\sigma)$ is $U(\sigma)$.*
- *If $\sigma$ is a non-leaf node in $T$, $\sigma$ has children $\sigma_1, \ldots, \sigma_k$, where for each $i \in \{1, \ldots, k\}$, the transition from $\sigma$ to $\sigma_i$, is $\tau_i$, and the probability of the transition is $Prob(\tau_i)$, then $E(\sigma) = \sum_{i \in \{1,\ldots,k\}} Prob(\tau_i) \times E(\sigma_i)$.*

*If $\sigma$ is the root of the search tree $T$, then the* **expected tree utility** *of $T$, denoted $E(T, Prob, U)$, is $E(\sigma)$.*

*Example 11.* In this example, we assume that agent 1 has a goal of making an argument $\mathtt{c}$ hold in the grounded extension of the abstract argument graph that exists in the end state of the execution. This goal is represented by the predicate $\mathtt{g(c)}$ in the private state of agent 1. In its private state, each agent has zero or more arguments that it holds represented by the predicate $\mathtt{h(c)}$, where $\mathtt{c}$ is an argument, and zero or more attacks $\mathtt{e(d,c)}$ from argument $\mathtt{d}$ to argument $\mathtt{c}$. In the public state, each argument $\mathtt{c}$ is represented by the predicate $\mathtt{a(c)}$. Each agent can add attacks $\mathtt{e(d,c)}$ to the public state, if the attacked argument is already in the public state (i.e. $\mathtt{a(c)}$ is in the public state), and the agent also has the attacker in its private state (i.e. $\mathtt{h(d)}$ is in the private state). Note, $\mathtt{r(c)}$ is used to denote that the agent has used its right to present argument $\mathtt{c}$.

$\neg\mathtt{a(a)} \land \mathtt{h(a)} \Rightarrow (\mathtt{1} : \boxplus\mathtt{a(a)})$
$\neg\mathtt{r(b)} \land \neg\mathtt{a(b)} \land \mathtt{h(b)} \land \mathtt{a(a)} \land \mathtt{e(b,a)} \Rightarrow (\mathtt{0.5} : \boxplus\mathtt{a(b)} \land \boxplus\mathtt{e(b,a)}) \lor (\mathtt{0.5} : \oplus\mathtt{r(b)})$

Let M be the following PFSM representing the system defined by the above action rules.

$$\begin{tikzcd}
 & \text{start} \arrow[r] & \sigma_1 \arrow[r, "1"] & \sigma_2 \arrow[d, "1"] & & & \\
\sigma_6 & \sigma_5 \arrow[l, "1"'] & \sigma_4 \arrow[l, "1"'] & \sigma_3 \arrow[l, "0.5"'] \arrow[r, "0.5"] & \sigma_7 \arrow[r, "1"] & \sigma_8 \arrow[r, "1"] & \sigma_9
\end{tikzcd}$$

Below we give states for the PFSM. Note, in the end state $\sigma_6$ the public state contains the abstract argument graph containing $\mathtt{a(a)}$, $\mathtt{a(b)}$, and $\mathtt{e(b,a)}$ (i.e. the graph is $\mathtt{a} \leftarrow \mathtt{b}$). and in the end state $\sigma_9$ the public state contains the abstract argument graph $\mathtt{a(a)}$ (i.e. the graph is $\mathtt{a}$).

$$\begin{aligned}
\sigma_1 &= (1, \{\mathtt{h(a)}, \mathtt{h(b)}, \mathtt{a(b,a)}\}, \{\}, \{\}) \\
\sigma_2 &= (2, \{\mathtt{h(a)}, \mathtt{h(b)}, \mathtt{a(b,a)}\}, \{\mathtt{a(a)}\}, \{\}) \\
\sigma_3 &= (1, \{\mathtt{h(a)}, \mathtt{h(b)}, \mathtt{a(b,a)}\}, \{\mathtt{a(a)}\}, \{\}) \\
\sigma_4 &= (2, \{\mathtt{h(a)}, \mathtt{h(b)}, \mathtt{a(b,a)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{e(b,a)}\}, \{\}) \\
\sigma_5 &= (1, \{\mathtt{h(a)}, \mathtt{h(b)}, \mathtt{a(b,a)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{e(b,a)}\}, \{\}) \\
\sigma_6 &= (0, \{\mathtt{h(a)}, \mathtt{h(b)}, \mathtt{a(b,a)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{e(b,a)}\}, \{\}) \\
\sigma_7 &= (2, \{\mathtt{h(a)}, \mathtt{h(b)}, \mathtt{a(b,a)}, \mathtt{r(b)}\}, \{\mathtt{a(a)}\}, \{\}) \\
\sigma_8 &= (1, \{\mathtt{h(a)}, \mathtt{h(b)}, \mathtt{a(b,a)}, \mathtt{r(b)}\}, \{\mathtt{a(a)}\}, \{\}) \\
\sigma_9 &= (0, \{\mathtt{h(a)}, \mathtt{h(b)}, \mathtt{a(b,a)}, \mathtt{r(b)}\}, \{\mathtt{a(a)}\}, \{\})
\end{aligned}$$

Suppose the utility of a being in the grounded extension of the graph in the end state is 10, and the utility of a not being in the grounded extension of the graph in the end state is -5. There are two leaves of the search tree to consider: $E(\sigma_6) = -5$ and $E(\sigma_9) = 10$. Hence, the expected tree utility is 2.5.

We can characterize expected tree utility in terms of expected utility of a lottery. We start by briefly reviewing the notion of a lottery. A **lottery** is a probability distribution over a set of possible outcomes. A lottery with possible outcomes $\pi_1$,..,$\pi_n$, that occur with probabilities $p_1, .., p_n$ respectively, is written as $[p_1, \pi_1; ....; p_n, \pi_n]$. Note, each outcome can be a lottery. For a utility function $U$, the **expected utility of a lottery** $L$, denoted $E(L, U)$, is $E(L, U) = \sum_{i=1}^{n} p_i \times U(\pi_i)$.

**Definition 14.** *Let $M$ be a PFSM, with probability function $Prob$, and let $T$ be a search tree for $M$. We define the coding function $L$ as follows:*

- *If $\sigma$ is a leaf node, then $L(\sigma) = \sigma$;*
- *If $\sigma$ is a non-leaf node with children $\sigma_1, \ldots, \sigma_k$, and transitions $\tau_1, \ldots, \tau_k$ respectively, then $L(\sigma) = [Prob(\tau_1) : L(\sigma_1), \ldots, Prob(\tau_k) : L(\sigma_k)]$.*

*If $\sigma$ is the root of the search tree $T$, then the* **tree lottery** *for $T$, denoted $L(T, Prob)$, is $L(\sigma)$.*

*Example 12.* Let $T$ be a search tree for the PFSM in Example 11. So $L(T, Prob)$ $= [1, [1, [0.5, [1, [1, \sigma_6]; 0.5, [1, [1, \sigma_9]]]]]]$.

The following result shows that evaluating the expected tree utility (i.e. Definition 13) corresponds to evaluating the expected utility of a tree lottery (i.e. Definition 14).

**Proposition 5.** *Let $M$ be a PFSM, with probability function $Prob$, and let $T$ be a search tree for $M$, Let $U$ be a utility function. If $E(T, Prob, U)$ is the expected tree utility of $T$, and $E(L(T, Prob), U)$ is the expected utility of tree lottery $L(T, Prob)$, then $E(T, Prob, U) = E(L(T, Prob), U)$.*

An agent can then analyze the PFSM to determine the most likely outcomes of a dialogue given any choices it makes. This can be used by the agent to determine its choice of moves in order to optimize its outcomes from the dialogue. It does this by changing its probabilities on its action rules. So that for each action rule, only one action option has probability 1, and the other action options have probability 0 (i.e. we make it deterministic).

*Example 13.* Consider Example 11. The following set of deterministic rules is the optimal set of rules for agent 1.

$\neg\mathtt{a}(\mathtt{a}) \wedge \mathtt{h}(\mathtt{a}) \Rightarrow (\mathtt{1} : \boxplus\mathtt{a}(\mathtt{a}))$
$\neg\mathtt{r}(\mathtt{b}) \wedge \neg\mathtt{a}(\mathtt{b}) \wedge \mathtt{h}(\mathtt{b}) \wedge \mathtt{a}(\mathtt{a}) \wedge \mathtt{e}(\mathtt{b},\mathtt{a}) \Rightarrow (\mathtt{0} : \boxplus\mathtt{a}(\mathtt{b}) \wedge \boxplus\mathtt{e}(\mathtt{b},\mathtt{a})) \vee (\mathtt{1} : \oplus\mathtt{r}(\mathtt{b}))$

In general for a set of action rules $Rules_1$, let $Rules_1^i$ contain a deterministic version of each rule in $Rules_1$. For each set $Rules_1^i$, together with the set $Rules_2$, and starting state, we obtain the PFSM $M^i$, and calculate expected tree utility of the search tree $T^i$. A set $Rules_1^i$ that gives the maximum expected tree utility is an optimal set of deterministic rules for agent 1, and thereby specifies the actions that agent 1 should do to maximize its utility from the dialogue with agent 2 (assuming that $Prob$ is a good estimate of the moves that agent 2 would make).

## 5 Discussion

In this paper, we have provided a simple, yet expressive language, for specifying the argumentation protocols of agents, and for representing the likelihood that an agent will make any specific move. We have shown how a specification in executable logic can be represented by a PFSM. This can be analyzed to determine expected utility for an agent, and it can be adjusted to optimize the performance of one agent over the other with respect to expected utility.

Some general frameworks for dialogue games have been proposed [17, 7], but they lack sufficient detail to formally analyze or implement specific systems. A more detailed framework, that is based on situation calculus, has been proposed by Brewka [18], though the emphasis is on protocols, and not on the likelihood of moves, or of strategies. Probability theory (for example [19–21]) and utility theory (for example [22–25, 20]) have been considered in other frameworks for multi-agent argumentation though none of these offers a general logical language for specifying diverse protocols for dialogical argumentation with abstract and logic-based arguments, and for representing the uncertainty of moves made by each agent in argumentation. We believe this is the first paper that provides a general framework for specifying probabilistic protocols for dialogical argumentation, and for using them for optimizing the probabilistic strategies in terms of the expected utility of the Markov chains that can be obtained by the protocols. Potentially, this is a powerful tool for assessing agents in dialogical argumentation, and optimizing their outcomes from the argumentation.

## References

1. Amgoud, L., Maudet, N., Parsons, S.: Arguments, dialogue and negotiation. In: European Conf. on Artificial Intelligence (ECAI 2000), IOS Press (2000) 338–342
2. Black, E., Hunter, A.: An inquiry dialogue system. Autonomous Agents and Multi-Agent Systems **19**(2) (2009) 173–209
3. Dignum, F., Dunin-Keplicz, B., Verbrugge, R.: Dialogue in team formation. In: Issues in Agent Communication. Springer (2000) 264–280
4. Fan, X., Toni, F.: Assumption-based argumentation dialogues. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'11). (2011) 198–203
5. Hamblin, C.: Mathematical models of dialogue. Theoria **37** (1971) 567–583
6. Mackenzie, J.: Question begging in non-cumulative systems. Journal of Philosophical Logic **8** (1979) 117–133

7. McBurney, P., Parsons, S.: Games that agents play: A formal framework for dialogues between autonomous agents. Journal of Logic, Language and Information **11** (2002) 315–334
8. McBurney, P., van Eijk, R., Parsons, S., Amgoud, L.: A dialogue-game protocol for agent purchase negotiations. Journal of Autonomous Agents and Multi-Agent Systems **7** (2003) 235–273
9. Parsons, S., Wooldridge, M., Amgoud, L.: Properties and complexity of some formal inter-agent dialogues. J. of Logic and Comp. **13**(3) (2003) 347–376
10. Prakken, H.: Coherence and flexibility in dialogue games for argumentation. J. of Logic and Comp. **15**(6) (2005) 1009–1040
11. Walton, D., Krabbe, E.: Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning. SUNY Press (1995)
12. Black, E., Hunter, A.: Executable logic for dialogical argumentation. In: European Conf. on Artificial Intelligence (ECAI'12), IOS Press (2012) 15–20
13. Hunter, A.: Analysis of dialogical argumentation via finite state machines. In: Scalable Uncertainty Methods. Volume 8078 of LNCS., Springer (2013) 1–14
14. Wooldridge, M., McBurney, P., Parsons, S.: On the meta-logic of arguments. In: Argumentation in Multi-agent Systems. Volume 4049 of LNCS., Springer (2005) 42–56
15. Bench-Capon, T.: Persuasion in practical argument using value based argumentation frameworks. Journal of Logic and Computation **13**(3) (2003) 429–448
16. Rabin, M.: Probabilistic automata. Information and Control **6** (1963) 230245
17. Maudet, N., Evrard, F.: A generic framework for dialogue game implementation. In: Proc. 2nd Workshop on Formal Semantics & Pragmatics of Dialogue, University of Twente (1998) 185 – 198
18. Brewka, G.: Dynamic argument systems: A formal model of argumentation processes based on situation calculus. J. Logic & Comp. **11**(2) (2001) 257–282
19. Hunter, A.: Modelling uncertainty in persuasion. In: Proceedings of the International Conference on Scalable Uncertainty Management. Volume 8078 of LNCS., Springer (2013) 57–70
20. Rienstra, T., Thimm, M., Oren, N.: Opponent models with uncertainty for strategic argumentation. In: Proceedings of IJCAI'13, IJCAI/AAAI (2013)
21. Hadjinikolis, C., Siantos, Y., Modgil, S., Black, E., McBurney, P.: Opponent modelling in persuasion dialogues. In: Proceedings of IJCAI. (2013)
22. Rahwan, I., Larson, K.: Pareto optimality in abstract argumentation. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008), AAAI Press (2008)
23. Riveret, R., Prakken, H., Rotolo, A., Sartor, G.: Heuristics in argumentation: A game theory investigation. In: Computational Models of Argument (COMMA 2008). Volume 172 of Frontiers in Artificial Intelligence and Applications., IOS Press (2008) 324–335
24. Matt, P., Toni, F.: A game-theoretic measure of argument strength for abstract argumentation. In: Logics in A.I. Volume 5293 of LNCS. (2008) 285–297
25. Oren, N., Norman, T.: Arguing using opponent models. In: Argumentation in Multi-agent Systems. Volume 6057 of LNCS. (2009) 160–174