Towards Dialogue Games for the Down-Admissible and Up-Complete Procedures

Martin Caminada*and Richard Booth†

Abstract

Recently, two operators - the *down-admissible* and *up-complete* operators - have been proposed, in the context of *aggregating argument labellings*, for *repairing* a given initial labelling so that it becomes rational, in the sense that it respects the attack relation between the arguments. The purpose of this research note is to give some initial thoughts on defining *dialogue games* for each of them, as well as for their combination.

1 Introduction

In the field of *abstract argumentation* [7], dialogue games provide a useful tool to understand the various different argumentation semantics that have been proposed, and provide a firm link between such semantics and natural forms of discussion [8]. Recently a new research strand on *aggregation of argument labellings* has emerged [1, 5], in which the evaluations of several agents over a set of arguments are aggregated into a group evaluation. Typically, simple-minded aggregation procedures such as the *credulous* or *sceptical* operators [5] suffer from the problem that their output fails to be *collectively rational*, i.e., the output labelling fails to pay sufficient respect to the attack relation between the arguments. In [5], one suggested remedy is to perform a *repair* to the output labelling by a combination of two operators called the *down-admissible* and *up-complete* operators. These concepts can be defined in two alternative ways: either in terms of minimising, respectively maximising according to a natural "committedness" ordering between labellings, or as the result of certain *transition sequences* from one labelling to another. So far what is missing is a dialogue-style characterisation of these concepts. In particular we would like to be able to define a dialogue procedure for establishing whether a given argument is accepted in the down-admissible or up-complete labellings of a given initial labelling. The purpose of this note is to give some initial thought on how to define dialogue games for each individually, as well as their combination.

The plan of this note is as follows. In the next section we give necessary background concepts from abstract argumentation and define the down-admissible and up-complete

*University of Aberdeen, UK
†University of Luxembourg

labellings. Then we informally describe the down-admissible and up-complete dialogue games in sections 3 and 4 respectively. Finally we describe a dialogue game for the combined game in Section 5.

2 Abstract argumentation

We start by assuming a countably infinite set U of argument names, from which all possible argumentation frameworks are built. We restrict ourselves to *finite* argumentation frameworks.

Definition 1 *An* argumentation framework *(AF for short)* $\mathcal{A} = (Args, \rightharpoonup)$ *is a pair consisting of a finite set* $Args \subseteq U$ *of arguments and an* attack *relation* $\rightharpoonup \subseteq Args \times Args$.

An AF can be visualised as a directed graph, with nodes and edges representing arguments and attacks respectively.

Example 1 *The AF* $(\{a_1, a_2, a_3\}, \{(a_1, a_2), (a_2, a_1), (a_2, a_3)\}$ *can be pictured as follows.*

An AF is evaluated by assigning one of the labels `in`, `out` or `undec` to each argument in *Args*, standing for *accepted*, *rejected* and *undecided* respectively [2, 4]. Given an AF $\mathcal{A}$, an $\mathcal{A}$*-labelling* is a function $L : Args \to \{\texttt{in}, \texttt{out}, \texttt{undec}\}$. We denote the set of all possible $\mathcal{A}$-labellings by $Labs(\mathcal{A})$. For each $\texttt{x} \in \{\texttt{in}, \texttt{out}, \texttt{undec}\}$ we denote by $\texttt{x}(L)$ the inverse image of $\texttt{x}$ under L, and we let $\texttt{dec}(L) \stackrel{\text{def}}{=} \texttt{in}(L) \cup \texttt{out}(L)$, i.e., $\texttt{dec}(L)$ denotes the set of arguments *decided* (either accepted or rejected) by L. Finally given $A \subseteq Args$ we denote by $L[A]$ the restriction of L to A.

We can compare different labellings according to how *committed* they are [5]. Given any two $\mathcal{A}$-labellings L_1, L_2 we write $L_1 \sqsubseteq L_2$ iff both $\texttt{in}(L_1) \subseteq \texttt{in}(L_2)$ and $\texttt{out}(L_1) \subseteq \texttt{out}(L_2)$. In other words every argument labelled `in` (resp. `out`) by L_1 is also labelled `in` (resp. `out`) by L_2. It is easy to see $\sqsubseteq$ forms a partial order over $Labs(\mathcal{A})$.

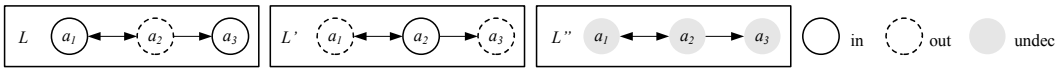
A major research question in abstract argumentation theory has been that of establishing when a given $\mathcal{A}$-labelling can be said to represent a *rational* evaluation of the arguments in *Args*. Of course such an evaluation should somehow respect the attack relation. Several definitions, or so-called *argumentation semantics*, have been proposed. A fundamental concept is that of a *complete* labelling.

Definition 2 *Let* $\mathcal{A} = (Args, \rightharpoonup)$ *be an AF and* L *be an* $\mathcal{A}$*-labelling. We say* L *is a* complete $\mathcal{A}$*-labelling iff, for all* $a \in Args$*:*

- *If* $L(a) = \texttt{in}$ *then* $L(b) = \texttt{out}$ *for all* $b \in Args$ *s.t.* $b \rightharpoonup a$.
- *If* $L(a) = \texttt{out}$ *then* $L(b) = \texttt{in}$ *for some* $b \in Args$ *s.t.* $b \rightharpoonup a$.
- *If* $L(a) = \texttt{undec}$ *then* $L(b) \neq \texttt{in}$ *for all* $b \in Args$ *s.t.* $b \rightharpoonup a$ *and* $L(c) = \texttt{undec}$ *for some* $c \in Args$ *s.t.* $c \rightharpoonup a$.

We denote the set of complete $\mathcal{A}$*-labellings by* $Comp(\mathcal{A})$.

Example 2 *Consider the AF from Example 1. Then there are three possible complete labellings for this framework, which can be pictured as follows.*



Complete labellings form the basis of several other semantics such as *grounded*, *preferred*, *stable* [7]. For instance the grounded $\mathcal{A}$-labelling $Gr(\mathcal{A})$ is defined to be the (unique) complete $\mathcal{A}$-labelling L such that $\texttt{in}(L)$ is $\subseteq$-minimal among all labellings in $Comp(\mathcal{A})$. Due to their fundamental nature and intuitiveness, we focus mainly on complete labellings in this paper. We will, however, also use the concept of *admissible* $\mathcal{A}$-labelling, which is an $\mathcal{A}$-labelling that satisfies the first two restrictions of Definition 2, but not necessarily the third. An example of an admissible $\mathcal{A}$-labelling in Example 2 would be L such that $L(a_1) = \texttt{in}$, $L(a_2) = \texttt{out}$ and $L(a_3) = \texttt{undec}$.

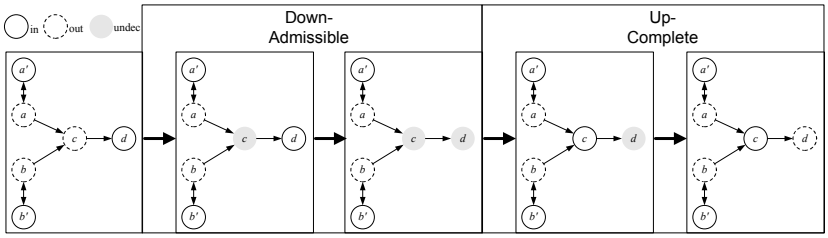
2.1 Down-admissible and up-complete

We begin with the down-admissible construction, which uses the definition of the 'committedness' relation $\sqsubseteq$ between $\mathcal{A}$-labellings.

Definition 3 ([5]) *Given an $\mathcal{A}$-labelling L, the* down-admissible *labelling of L, denoted by $\lfloor L$, is the (unique) greatest element (under $\sqsubseteq$) of the set of all admissible $\mathcal{A}$-labellings M such that $M \sqsubseteq L$.*

A constructive definition of $\lfloor L$ is given in [5]. It can be arrived at by just iteratively relabelling every argument that is illegally `in` or illegally `out` with `undec` until no illegal `in` or `out` labels remain. We will refer to such a sequence of changes as a *transition sequence*. The end result is a labelling that is admissible.

Example 3 Consider the sequence of the 3 leftmost labellings of the AF $\mathcal{A}$ shown below.[1] (In this, as with all our pictures, a node with solid boundary denotes an `in`-labelled argument, a dotted boundary denotes `out`, and a solid grey node indicates `undec`).



The initial labelling L is on the far left. The only argument illegally labelled `in` or `out` in L is c (because it is `out` but none of its attackers is `in`), so its label is changed to `undec` (2nd labelling). This change in turn causes d to become illegally `in`, so then d's label is also changed to `undec`. At this point there are no illegally `in` or `out` arguments left and so the procedure stops with $\lfloor L$ as the 3rd labelling.

[1] Figure courtesy of Edmond Awad.

As the example illustrates, $\downarrow L$ might not be a complete labelling. To ensure a complete labelling we need an additional step which applies the up-complete operator.

Definition 4 ([5]) *Given an admissible $\mathcal{A}$-labelling L, the* up-complete *labelling of L, denoted by $\uparrow L$, is the (unique) smallest element (under $\sqsubseteq$) of the set of all complete $\mathcal{A}$-labellings M such that $L \sqsubseteq M$.*

There is also a constructive definition of $\uparrow L$ in [5]. Just iteratively change every illegally `undec` argument to `in` or `out` as appropriate, until no illegally labelled arguments remain. As with the down-admissible procedure above, we shall also use the term "transition sequence" to refer to such sequences of changes in this context.

Example 4 Consider the sequence of the 3 rightmost labellings in Example 3, which starts with $L' = \downarrow L$ from the previous example. There is one illegally `undec` argument, namely c. Since both attackers a, b are labelled `out`, we change c's label to `in`. At this point d becomes illegally `undec` because it now has an attacker c which is `in`. Hence we change d's label to `out`. Now there are no illegally `undec` arguments and so the process stops and returns the rightmost labelling as $\uparrow L'$.

We denote by $\Downarrow L$ the composite operation of performing the down-admissible followed by the up-complete procedures on L.

3 The down-admissible game

In what follows we let L_{init} denote some given initial $\mathcal{A}$-labelling. What we are looking for is some dialogue game that, for any given $a \in Args$, establishes whether $[\downarrow L_{init}](a) =$ `in`, i.e., whether a is labelled `in` by the down-admissible labelling of L_{init}.

The idea behind the down-admissible game is to re-apply the *admissibility (Socratic) game* [3] in the specific context of down-admissible. The goal of the admissibility game is to establish, for some given $a \in Args$, whether there exists some admissible $\mathcal{A}$-labelling L such that $L(a) =$ `in`. The game has two players **M** (whose aim is to establish the claim) and **S** (whose aim is to refute the claim). **M** begins the discussion by stating "`in`(a)", and then each player takes turns to put forward arguments, with **M** making moves always of the form "`in`(b')" and **S** making moves always of the form "`out`(c')", according to the following informal protocol. (We refer to [3] for the formal details.)

(1) Each move of **M** (except the first) contains an attacker of the argument in the directly preceding move of **S**.

(2) Each move of **S** contains an attacker of an argument contained in some (not necessarily the directly preceding) move of **M**.

(3) **S** is not allowed to repeat his moves.

(4) **M** is allowed to repeat his moves.

(5) If **S** uses an argument previously used by **M**, then **S** wins the game.

(6) If **M** uses an argument previously used by **S**, then **S** wins the game.

(7) If **M** cannot make a move any more, then **S** wins the game.

(8) If **S** cannot make a move any more, then **M** wins the game.

In essence, what the admissibility game does is to try to construct an admissible labelling around a (via the `in`-arguments stated by **M**). However, what we are interested in is an admissible labelling that is actually *contained* ($\sqsubseteq$) in the initial labelling. After all, an argument is labelled `in` by the biggest admissible labelling that is $\sqsubseteq$-smaller than the initial labelling (that is, by the down-admisisble) iff it is labelled `in` by *at least one* admissible labelling that is $\sqsubseteq$-smaller than the initial labelling. That is, it is sufficient to find an admissible labelling (that labels our argument `in`) *within the boundaries* of the initial labelling.

To make sure the game stays within the boundaries of the initial labelling we have to add additional rules to the 8 above:

(9) If **M** makes a move $\texttt{in}(a)$ and a is not labelled `in` by the initial labelling then **M** loses the game (that is, **S** wins).

(10) If **S** makes a move $\texttt{out}(a)$ and a is not labelled `out` by the initial labelling then **S** wins the game (that is, **M** loses).

We call this revised admissibility game the *down-admissible game*.

Conjecture 1 *Let $a \in Args$ and $L_{init} \in Labs(\mathcal{A})$. There exists an admissible labelling $L' \sqsubseteq L_{init}$ such that $L'(a) = \texttt{in}$ iff player* **M** *can win the down-admissible game for argument a.*

Since, as mentioned above, we have $[\downarrow L_{init}](a) = \texttt{in}$ iff there exists an admissible labelling $L' \sqsubseteq L_{init}$ such that $L'(a) = \texttt{in}$, the above conjecture would be enough to give us our desired procedure for establishing whether $[\downarrow L_{init}](a) = \texttt{in}$. To prove this conjecture it is likely to be helpful to realise that:

Correctness - the down-admissible game constructs an admissible labelling within ($\sqsubseteq$) the initial labelling.

Completeness - If there exists an admissible labelling within ($\sqsubseteq$) the initial labelling then this admissible labelling serves as some kind of "roadmap" (see [3]) for playing the down-admissible game and winning it (the trick is that as long as **M** is only playing moves that are `in` in the admissible labelling, he's going to win).

4 The up-complete game

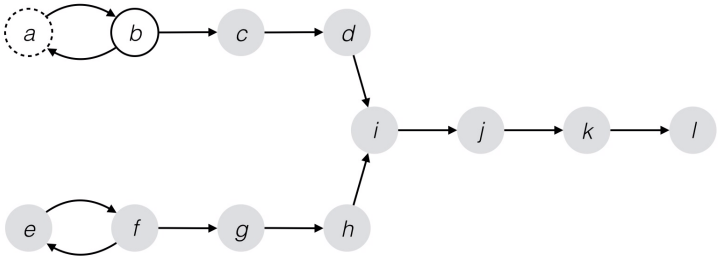
We are now interested in establishing whether $[\uparrow L_{init}](a) = \texttt{in}$. For this, we will base our procedure on the *grounded persuasion game* [6], which was introduced to establish whether a given argument was labelled `in` by the grounded $\mathcal{A}$-labelling $Gr(\mathcal{A})$. In that game there are 2 players **P** (the proponent), whose aim is to establish the claim for a given argument a, and **O** (the opponent), whose aim is to refute the claim. There are four kinds of move `claim`, `why`, `because` and `concede`, and the game is played according to the following rules (again, we refer the reader to [6] for the precise formal definition).

- **P** and **O** take turns. Each turn of **P** contains a single move `claim` or `because`. In each turn **O** plays one or more moves. **O**'s turn starts with an optional sequence of `concede` moves and finishes (when possible) with a single `why` move.
- **P** gets *committed* to arguments used in `claim` and `because` moves. **O** gets committed to arguments used in `concede` moves.
- **P** starts with `claim` `in`(a) where a is the main argument of the discussion. `claim` cannot be repeated later in the game.
- In consecutive turns **P** provides reasons for the directly preceding `why` L move of **O** by moving `because` L' where L' is a *reason* for L (and where L', L here are *partial* labellings).
- **P** can play `because` only if the reason given does not contain any arguments already mentioned (in **P**'s commitment store) but not yet accepted (not in **O**'s commitment store). We call such arguments *open issues*.
- **O** addresses the most recent open issue L (`in`(a') or `out`(a')) in the discussion. If **O** is committed to reasons for L it must `concede` L otherwise **O** questions all reasons that **O** is not committed to with `why`.
- **O** can question with `why` just one argument.
- The moves `claim`, `concede` and `because` can be played only if new commitments do not contradict a previous one.
- The discussion terminates when no more moves are possible. If **O** conceded the main argument then **P** wins, otherwise **O** wins.

Now, the trick is that with just a small modification, we can re-apply the grounded persuasion game to establish whether $[\uparrow L_{init}](a) = \texttt{in}$. The key concept here is that of the *commitment store*. Each player (**P** and **O**) has his own commitment store, which is essentially a **partial** function from *Args* to $\{\texttt{in}, \texttt{out}\}$. In the grounded persuasion game both players start with the empty commitment store, and they gradually build up commitments during the course of the game.

If we want to use the grounded persuasion game to compute $\uparrow L_{init}$ then the only thing we have to do is to start with the initial labelling L_{init} instead of with the empty labelling. That is, at the start of the game each player has a commitment store equal to L_{init} (that is, as for the `in`- and `out`-labelled arguments. The `undec`-labelled arguments are simply omitted from the partial labelling.) We call the resulting game the *up-complete game (for a) with initial commitment store* L_{init}.

Example 5 Suppose the AF and initial admissible labelling L_{init} shown below.



We are interested in whether argument j is labelled `in` by $\uparrow L_{init}$. However, we're not interested in arguments k or l, and we're not interested in the *entire* up-complete labelling. Here's how the discussion would go:

1. **P** : `claim` `in`(j)
2. **O** : `why` `in`(j)
3. **P** : `because` `out`(i)
4. **O** : `why` `out`(i)
5. **P** : `because` `in`(d)
6. **O** : `why` `in`(d)
7. **P** : `because` `out`(c)
8. **O** : `concede` `out`(c)
9. **O** : `concede` `in`(d)
10. **O** : `concede` `out`(i)
11. **O** : `concede` `in`(j)

Note that the `concede` move by **O** on line 8 is made because **O** already knows the reason why c is `out`, because he is committed (initial labelling) that b is `in`. The winner of this game is **P**.

The fact that **P** wins this game has to do with making the right choice (choosing d instead of h). Let's see what happens when **P** makes the wrong choice.

1. **P** : `claim` `in`(j)
2. **O** : `why` `in`(j)
3. **P** : `because` `out`(i)
4. **O** : `why` `out`(i)
5. **P** : `because` `in`(h)
6. **O** : `why` `in`(h)
7. **P** : `because` `out`(g)
8. **O** : `why` `out`(g)
9. **P** : `because` `in`(f)
10. **O** : `why` `in`(f)
11. **P** : `because` `out`(e)
12. **O** : `why` `out`(e)

Now **P** cannot move anymore. **P** cannot move because `out`(e) is already an open issue.

Hence we see that what matters is not whether the claim is successfully defended in a particular game (as a game can go wrong due to wrong decisions of **P**). What matters is that there exists *at least one* game that is won by the proponent **P**.

Conjecture 2 *Let L_{init} be an admissible $\mathcal{A}$-labelling and $a \in Args$. Then $[\uparrow L_{init}](a) =$* `in` *iff it is possible for* **P** *to win the up-complete game for a with initial commitment store $L[\mathtt{dec}(L)]$ (that is, iff there exists at least one up-complete game for a with initial store $L[\mathtt{dec}(L)]$ that is won by* **P***).*

Why is this game actually correct? The trick is that in a successful game, the sequence of `concede` moves of the opponent actually constitutes a transition sequence

(each `concede` move being a transition step) of the procedure for computing the up-complete. That is, in order to prove completeness and correctness it suffices to show that:

1. If there exists an up-complete game for argument a that is won by the proponent then its sequence of `concede` moves coincides with a transition sequence that ultimately labels a `in`, which then implies that a is labelled `in` by the up-complete labelling.
2. If a is labelled `in` by the up-complete labelling then there exists an up-complete game whose sequence of `concede` moves constitutes a transition sequence that ultimately labels a `in`.

There might be a small issue that still needs to be taken care of regarding the well-definedness of a game that is lost. Suppose that in the above example e would be labelled `in` and f would be labelled `out` (instead of `undec`). Now consider the following game:

1. **P** : `claim` `in`(j)
2. **O** : `why` `in`(j)
3. **P** : `because` `out`(i)
4. **O** : `why` `out`(i)
5. **P** : `because` `in`(h)
6. **O** : `why` `in`(h)
7. **P** : `because` `out`(g)
8. **O** : `why` `out`(g)

Now, at this moment **P** should *not* be allowed to reply with

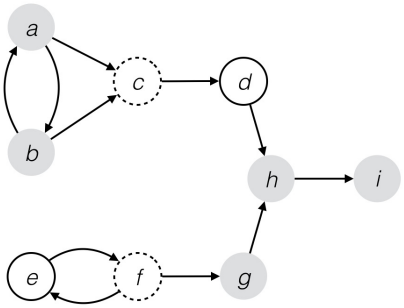
9. **P** : `because` `in`(f)

as **P** is already committed to `out`(f), since this is in the initial labelling (and therefore in the initial commitment store of **P**).

5 The combined game

We now know what the two individual games (the up-complete and the down-admissible games) look like. The next question is whether there's any way of *combining* them. After all, what we are interested in in aggregation setting such as [1, 5] is, given an initial labelling L_{init}, whether an argument a is `in` in $\Uparrow(L_{init})$. We do this as follows. First, run the up-complete game based directly on L_{init} (instead of on $\downarrow(L_{init})$). If the proponent can win this game, then comes the second phase (the second challenge). Now we run the down-admissible game on the arguments in L_{init} that were actually used for forcing the `concede`s in the up-complete game. Perhaps an example will make things clear.

Example 6 Suppose the AF and the initial labelling L_{init} are as indicated below.



Suppose i is the argument we are interested in. The proponent could try to win the up-complete game in the following way:

1. **P** : `claim` `in`(i)
2. **O** : `why` `in`(i)
3. **P** : `because` `out`(h)
4. **O** : `why` `out`(h)
5. **P** : `because` `in`(d)
6. **O** : `concede` `in`(d)
7. **O** : `concede` `out`(h)
8. **O** : `concede` `in`(i)

Note we need a small alteration of the up-complete game such that moves 7,8 become necessary, because we need to explicitly identify which arguments from L_{init} are actually used (in this case `in`(d)).

Then comes the second game, the down-admissible game based on the arguments of L_{init} used in the up-complete game. That is, the down-admissible game will be played on `in`(d).

1. **M** : `in`(d) *"I have an admissible labelling in which d is* `in`*"*
2. **S** : `out`(c) *"Then in your labelling c must be* `out`*, based on what grounds?*

Now **M** will lose the game as **M** can only move `in`(a) or `in`(b), both of which are not supported by L_{init}, so rule (9) of the down-admissible game applies. So although the proponent (**P** in the up-complete game and **M** in the down-admissible game) wins the first game (the up-complete) he loses the second game (the down-admissible). Therefore, proponent loses the combined game.

Proponent would have done better if he chose a different way of winning the up-complete game:

1. **P** : `claim` `in`(i)
2. **O** : `why` `in`(i)
3. **P** : `because` `out`(h)
4. **O** : `why` `out`(h)
5. **P** : `because` `in`(g)
6. **O** : `why` `in`(g)
7. **P** : `because` `out`(f)

(These last two steps are not required in the up-complete games defined earlier, but we might still need them.) The down-admissible game would then need to be about an attacker of f that can be claimed `in`.

1. **M** : `in`(e)
2. **S** : `out`(f)
3. **M** : `in`(e)

So here **M** wins the discussion (the down-admissible game). Notice that the game stays within the boundaries of L_{init}, as required. So the proponent wins the combined game if he wins both the up-complete game and the down-admissible game.

Conjecture 3 *Let L_{init} be an $\mathcal{A}$-labelling and $a \in Args$. $[\Uparrow\Downarrow(L_{init})](a) =$* `in` *iff the proponent can win the combined game (that is, if the proponent has a way of winning the up-complete game in a way that also allows him to win the subsequent down-admissible game.)*

In Example 6 the up-complete game only uses one single argument from L_{init} (`in`(d) and `out`(f) respectively). What happens if the up-complete uses *more* than one argument? Then, the opponent can choose which argument the proponent should defend in the subsequent down-admissible game. This is because *all* arguments used by the up-complete game should be in the down-admissible labelling.

One of the challenges is how to redefine the up-complete game such that it explicitly identifies the arguments of L_{init} that are actually used. Let's first examine what goes wrong otherwise, using Example 6.

1. **P** : `claim` `in`(i)
2. **O** : `why` `in`(i)
3. **P** : `because` `out`(h)
4. **O** : `concede` `out`(h)
5. **O** : `concede` `out`(i)

(Note that **O** `concede`s in step 4 because **O** knows why h is out, because **O** is already committed to the fact that d is `in`, since this is in the initial labelling.)

But in that case, the game will not even be *able* to reach argument f, which would seem vital for winning the subsequent down-admissible game.

Our impression is that this can be fixed by giving L_{init} a special position in the game. Instead of loading it as the initial commitment store at the beginning of the game, we give it a special status as the "background labelling" of the game. The rule is that whenever the proponent (**P**) does a `claim` move or a `because` move of something that is in the background labelling, the opponent (**O**) has to `concede` immediately. Our guess is that this would fix the problem. Moreover, it can also be applied to the up-complete game on itself (and not just to the up-complete game in the particular context of $\Uparrow\Downarrow(L_{init})$).

Acknowledgements

Richard Booth is supported by the Fonds National de la Recherche, Luxembourg (DYN-GBaT project).

References

[1] Richard Booth, Edmond Awad, and Iyad Rahwan. Interval methods for judgment aggregation in abstract argumentation. In *Proc. KR 2014*, 2014.

[2] Martin Caminada. On the issue of reinstatement in argumentation. In *Logics in artificial intelligence*, pages 111–123. Springer, 2006.

[3] Martin Caminada, Wolfgang Dvořák, and Srdjan Vesic. Preferred semantics as Socratic discussion. *Journal of Logic and Computation*, 2014.

[4] Martin Caminada and Dov Gabbay. A logical account of formal argumentation. *Studia Logica*, 93(2-3):109–145, 2009.

[5] Martin Caminada and Gabriella Pigozzi. On judgment aggregation in abstract argumentation. *Autonomous Agents and Multi-Agent Systems*, 22(1):64–102, 2011.

[6] Martin Caminada and Mikołaj Podlaszewski. Grounded semantics as persuasion dialogue. *COMMA*, 245:478–485, 2012.

[7] Phan Minh 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.

[8] Sanjay Modgil and Martin Caminada. Proof theories and algorithms for abstract argumentation frameworks. In *Argumentation in artificial intelligence*, pages 105–129. Springer, 2009.