Argument & Computation -1 (2023) 1–14
DOI 10.3233/AAC-220020
IOS Press

Decidability in argumentation semantics

Paul E. Dunne
University of Liverpool, Ashton Building, Liverpool L69 7ZF, UK

Abstract. Much of the formal study of algorithmic concerns with respect to semantics for abstract argumentation frameworks has focused on the issue of computational complexity. In contrast matters regarding *computability* have been largely neglected. Recent trends in semantics have, however, started to concentrate not so much on the formulation of novel semantics but more on identifying common properties: for example, from basic ideas such as conflict-freeness through to quite sophisticated ideas such as serializability.

The aim of this paper is to look at the implications these more recent studies have for *computability* rather than computational complexity.

Keywords: Extension-based semantics, semantic property, computability

1. Introduction

Computational complexity, the focus of most formal work on algorithmic matters as these affect traditional semantics within abstract argumentation frameworks, see e.g. Dvorak and Dunne [3], addresses questions of limits on *efficient* solution methods. In this focus there is, usually unstated, an assumption that some form of solution method, i.e. algorithm, actually is possible. A far more fundamental concern, however, is that of *computability*: whether a given problem is capable of algorithmic solutions *at all*. Recently there has been an increasing interest in treatments of semantics through analysis of generic principle. Such work has its origins in Baroni and Giacomin [1]. In this, the central interest is not so much what makes a novel semantics distinct (and arguably useful) from previously proposed forms but rather on identifying features that all reasonable semantics ought to have in common. This approach raises a very general question: suppose we have some effective mechanism by which a semantics can be described, what is it possible in this case to say about *recognising* such properties *computationally*? The aim of the current paper is first formally to define this problem and then to establish some basic computational properties.

In some respects considering, in the context of abstract argumentation frameworks, questions such as "Is it *possible* to solve this problem (by an algorithm)?" rather than "Is it possible to solve this problem *efficiently*?" may seem strange. There are, however, good reasons for doing so. We can first note the historical significance of such questions: computability (or, as it is also referred to, recursive function theory) predates the first programmable computers by several decades: the question "Can this be solved algorithmically?" has a much longer history than its counterpart "Can this be solved by an efficient algorithm?". Even if one accepts its historical importance, the possibility of unsolvable questions arising in the specific instance of argumentation frameworks seems unlikely and examples almost surely highly contrived and artificial: defined not so much as having a "real context" but more simply to demonstrate a point. Argumentation frameworks are directed graphs and treated, in all but a small body of work, as *finite structures*; in practical settings this finite aspect is always present. So if we are only concerned

with the computational features of *single* finite objects then unsolvability is not a consideration: given a finite graph, an interpretable question about that graph's properties (e.g. from "How many edges?" through to "What is the longest simple path between two nodes?") we can always answer the question by an algorithmic method (albeit sometimes a rather lengthy one).

In summary, if questions of *effective* (as opposed to *efficient*) methods are going to arise, these will not be at the level of individual frameworks. Argumentation, however, has moved on from the realm of issues which can be related to a purely finite framework context. From its initial proposal, we have the concept of *argument semantics* as subsets of finite sets satisfying particular criteria. Here, again, we are still in a finite environment: semantics are sets of subsets, some subsets will meet the criteria and others will fail to do so. The core question becomes whether a given set meets the criteria with respect to a given framework. If this, the *verification problem* is computable then general issues of algorithmic *solution* do not arise since every question can be resolved, at worst, by explicit enumeration and piecemeal review of every subset. It is, of course, conceivable that "semantics" may be formulated in which such "list and check" methods would fail since the verification problem is itself undecidable. Yet, while imaginable as a mathematical conceit, it is hard to think of any such "semantics" arising in practice. Frameworks do not yield unsolvable questions, (practical) semantics (properties of frameworks), also fail to yield such problems. It is when we move one level higher, as is demonstrated in the present paper, and look at properties of *semantics* that computability becomes a non-trivial issue. Properties of classes of semantics (from the "principle based" proposal of Baroni and Giacomin [1] onwards) have proven to be an important simplification in shifting interest from a justification of novel approaches through "other proposals can't do this" to "this approach has all of the benefits of earlier ideas *and* it can do this". The principle-based analysis offers a subtle but significant direction in which novel semantics are treated. It is in this move from frameworks to properties of frameworks to properties of properties of frameworks that non-computability arises. One might ask, being aware of this issue, why computability should matter: mathematicians, after all, continue their efforts despite the implications of Gödel's work. To be aware of the *possibility* of non-computability may avoid efforts to automate that which provably cannot be automated: for example, we cannot develop an algorithm that given a description of a semantics will determine whether that semantics respects the conflict-freeness principle.

The shifting from frameworks and semantics to *properties* of semantics does, however, raise one issue: how do we *represent* a property e.g. conflict-freeness in *computational* terms. Frameworks are directed graphs and their semantics, according to the criteria used, simply subsets of their arguments. Now it is certainly possible to enumerate all frameworks and their corresponding, say conflict-free sets. In studying computational properties of principles we need to capture an infinite range of possibilities by a finite form. The mechanisms that are needed to do so are, ultimately, those that lead to questions such as "Does this semantics σ respect this principle π?" being incapable of algorithmic solution.

The consequences of viewing properties of semantics from a computational perspective go further than basic uncomputability. If we consider the classical undecidable computational problem, *The Halting Problem* of Turing [7], which considers if encodings of programs will come to a halt on a given input, although undecidable in the sense that no algorithm exists which is guaranteed always to give a correct response in a finite time, there is at least a "partial decision method": one which will recognise all positive instances. This is simply to simulate the program on its given input and report accept should this simulation end. For the classification of semantics we do not even have such partial decision methods: there is no effective algorithm guaranteed to recognise all positive (or all negative) instances. In formal language while cases such as The Halting Problem are *recursively enumerable* (although not *recursive*) natural semantic properties such as conflict-freeness fail even to be recursively enumerable.

We give preliminary background in Section 2. We assume the reader has some acquaintance with the concept of Model of Computation as captured by, among others, Turing Machines, Post Machines etc. Excellent introductory background concerning these may be found in Hopcroft, Motwani and Ullman [4] with more advanced discussion in Hopcroft and Ullman [5].

In Section 3 we present one method of encoding an argument framework and collection of subsets of its argument as a finite length binary sequence. Results are given in Section 5 and Section 6 with conclusions in Section 7.

2. Preliminaries

Definition 1 (Dung [2])**.** An *argumentation framework* (AF) is a pair $(\mathcal{X}, \mathcal{A})$ where $\mathcal{X}$ is a finite set of entities, called *arguments*, and $\mathcal{A}$ a binary relation on $\mathcal{X}$. For any $p, q \in \mathcal{X}$ we say that p *attacks* q if $\langle p, q\rangle \in \mathcal{A}$.

We use the shorthand $\mathcal{H}$ for an arbitrary framework $(\mathcal{X}, \mathcal{A})$ where the argument and attack set are implicit.

Definition 2. Let $\mathcal{H}$ be an argumentation framework, $x \in \mathcal{X}$ and $S \subseteq \mathcal{X}$. We define $\{x\}^+$ as $\{y \in \mathcal{X} | x \text{ attacks } y\}$, $\{x\}^-$ as $\{y \in \mathcal{X} | y \text{ attacks } x\}$, S^+ as $\cup\{\{x\}^+ | x \in \mathcal{X}\}$, and S^- as $\cup\{\{x\}^- | x \in \mathcal{X}\}$. The set S is said to be *conflict-free* if $S \cap S^+ = \emptyset$. A set S is said to *defend* x iff $\{x\}^- \subseteq S^+$. The characteristic function $\mathcal{F} : 2^{\mathcal{X}} \to 2^{\mathcal{X}}$ is defined as $\mathcal{F}(S) = \{x | S \text{ defends } x\}$.

Definition 3. Let $\mathcal{H}$ be an argumentation framework. A subset S of $\mathcal{X}$ is said to be:

- an admissible set if S is conflict-free and $S \subseteq \mathcal{F}(S)$
- a complete extension if S is conflict-free and $S = \mathcal{F}(S)$
- a grounded extension if S is the smallest (w.r.t. $\subseteq$) complete extension
- a preferred extension if S is a maximal (w.r.t. $\subseteq$) complete extension

For σ an arbitrary semantics (e.g. conflict-free, preferred, complete) the notation $\sigma(\mathcal{H})$ describes

$$\sigma(\mathcal{H}) = \{S \subseteq \mathcal{X} : S \text{ satisfies the constraints specified by } \sigma\}$$

It will be helpful to use the shorthand $E_\sigma(\mathcal{H}, \mathbb{S})$ with $\mathbb{S} \subseteq 2^{\mathcal{X}}$ for the predicate which takes the value $\top$ whenever $\mathbb{S} = \sigma(\mathcal{H})$. It is well known (as shown in Dung's original paper [2]) that for any AF, $\mathcal{H}$,

$$\text{pr}(\mathcal{H}) \subseteq \text{adm}(\mathcal{H}) \subseteq \text{cf}(\mathcal{H})$$

A number of general principles for argumentation semantics have been presented in Baroni and Giacomin [1]. Among these are

a. A semantics σ *respects* τ if

$$\forall \mathcal{H}\ S \in \sigma(\mathcal{H}) \quad \Rightarrow \quad S \in \tau(\mathcal{H})$$

b. We also have (in addition to the concept "a semantics σ respects a semantics τ") that of a semantics σ respecting a *principle* π. For example, σ respects *reinstatement* if

$$\forall \mathcal{H}\ S \in \sigma(\mathcal{H}) \text{ if } \forall y \in \{x\}^- \text{ it holds that } y \in S^+ \text{ then } x \in S$$

Definition 4. An *alphabet*, Σ is a finite set of symbols, Σ^* denotes the set of all finite length sequences of symbols (*words*) from Σ (including ε the *empty* word). A *language*, L over Σ is a subset of Σ^*. The *complement* of a language L, denoted coL, is the language $\Sigma^* \setminus L$. A language is *recursively enumerable* (r.e.) (also called *partially decidable*) if there is a program, M, taking as an input $w \in \Sigma^*$ and which halts and accepts w whenever $w \in L$. A language is *recursive* (or *decidable*) if both L and coL are r.e.

Since we may encode Turing Machine programs, M, as words $\chi(M) \in \{0,1\}^*$ (see for example, Hopcroft and Ullman [5, pp. 181–2]) we can define the set of all r.e. languages by

$$\text{RE} = \big\{\chi(M) : M \text{ is a Turing machine program}\big\}$$

For a given program, M taking inputs from $\{0,1\}^*$, $L(M)$ is the subset of $\{0,1\}^*$ on which M will halt and report accept.

We can introduce the concept of *property* of a language simply via the set of Turing machine programs that accept languages with the property. That is if Π is some property (e.g. empty, infinite, contains only odd length words, etc) then $\Pi \subseteq \text{RE}$.

3. Encodings in binary and their consequences

A device which we make extensive use of can be summarised in the following terms: we have a set of objects, $\mathbb{O}$; we wish, uniquely, to describe these as words over $\{0,1\}$. Since we can order the words in $\{0,1\}^*$ (e.g. use standard lexicographic ordering and the convention $0 \prec 1$) we can use such an ordering precisely to define "the first object in $\mathbb{O}$", "the second object in $\mathbb{O}$" and generally the k'th object in $\mathbb{O}$: go through the words in $\{0,1\}^*$ in order, the first such word that describes a valid encoding will define the "first" object. In general the k'th word discovered that defines a valid encoding will be the k'th object. We thus have total functions over $\mathbb{N}$ and $\mathbb{O}$:

$$\text{enc}^{\mathbb{O}} : \mathbb{O} \to \mathbb{N}; \qquad \text{dec}^{\mathbb{O}} : \mathbb{N} \to \mathbb{O}$$

$$\text{enc}^{\mathbb{O}}(\omega) = k \quad \text{with} \quad \text{dec}^{\mathbb{O}}(k) = \omega$$

$$\text{dec}^{\mathbb{O}}(k) = \omega_k \in \mathbb{O} \quad \text{the encoding of } \omega_k \text{ in } \{0,1\}^* \text{ is the } k\text{'th valid coding word in the ordering}$$

Notice that,

$$\text{dec}^{\mathbb{O}}\big(\text{enc}^{\mathbb{O}}(\omega)\big) = \omega; \qquad \text{enc}^{\mathbb{O}}\big(\text{dec}^{\mathbb{O}}(k)\big) = k$$

Such schemes allow us to treat properties of structures as functions over $\mathbb{N}$.

As a simple example, consider the set

$$\text{EP} = \big\{(p,q) : \text{Both } p \text{ and } q \text{ are even Natural numbers}\big\}$$

So that,

$$\text{EP} = \big\{(2,2), (2,4), (4,2), (2,6), (4,4), (6,2), (2,8), (4,6), (6,4), (8,2), \ldots\big\}$$

We can write $(p, q) \in \text{EP}$ as $0^p 10^q$ and hence $\text{enc}^{\text{EP}}((2, 2)) = 1$ since 00100 is the first word (in lexicographic order) that encodes an element of EP. Similarly $\text{dec}^{\text{EP}}(6) = 000000100$, i.e. the pair $(6, 2) \in \text{EP}$

3.1. Argument frameworks $\mathcal{H}$

Since the names of arguments are unimportant it is only needed to describe the number of these. If $|\mathcal{X}| = n$, then $\mathcal{X}$ is described by the word 1^n (i.e. a sequence of n 1s). This is followed by 00 to indicate that the next section of the encoding describes $\mathcal{A}$. An attack $\langle x_i, x_j \rangle$ is an ordered pair with $1 \leqslant i, j \leqslant n$, hence we can encode such as $1^i 01^j$ separating distinct attacks with the sequence 00 and using the sequence 000 to indicate the end.

For example the AF,

$$\left(\{x_1, x_2, x_3\}, \left\{\langle x_1, x_2\rangle, \langle x_2, x_3\rangle, \langle x_2, x_1\rangle, \langle x_3, x_1\rangle\right\}\right)$$

would be coded as 11100101100110111001101001110100 0.

In order to ensure uniqueness attacks are described with increasing order of source and increasing order of target for attacks with the same source.

3.2. Frameworks and sets of subsets of arguments, $(\mathcal{H}, \mathbb{S})$ with $\mathbb{S} \subseteq 2^{\mathcal{X}}$

Let $\mathbb{A}$ denote the set of all objects $(\mathcal{H}, \mathbb{S})$ with $\mathbb{S} \subseteq 2^{\mathcal{X}}$.

We have the scheme used for AFs above. Noting that $\mathbb{S}$ is a list of some set of subsets of $\mathcal{X}$. Following the 000 indicating the final attack in $\mathcal{A}$ for $\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ an arbitrary set in $\mathbb{S}$ with the convention that the indices are given in increasing order. We use $1^{i_1}01^{i_2} \cdots 1^{i_k}$ to code it. Sets are presented in increasing order of size and increasing value of argument index (sets of the same size being ordered lexicographically using the indices of members, e.g. $\{x_2\}$ before $\{x_5\}$, $\{x_1, x_4\}$ before $\{x_2, x_3\}$ etc.)

Distinct sets in $\mathbb{S}$ are separated by 00 and the end of the list by 000. For example to describe the system $\mathbb{S} = \{\emptyset, \{x_1\}, \{x_3\}, \{x_2, x_3\}\}$ we would have 0010011100110111000. With the example we would encode the system as

11100101100110111001101001110100000100110111000

Note that the empty set in $\mathbb{S}$ is expressed as 000 (end of attack list token) immediately followed by 00 (next set in $\mathbb{S}$ token). We also note the distinction $S = \{\emptyset\}$ encoded as $1^{|\mathcal{X}|}00\,\text{code}(\mathcal{A})00000000$ (that is end of $\mathcal{A}$ code 000; content of empty set 00; end of $\mathbb{S}$ code 000) and $\mathbb{S} = \emptyset$ coded as $1^{|\mathcal{X}|}00\,\text{code}(\mathcal{A})000000$.

For the pair $(\mathcal{H}, \mathbb{S})$ we use $\mu(\mathcal{H}, \mathbb{S}) \in \{0, 1\}^*$ to denote its (unique) encoding as a binary word.

3.3. Power set constructions

Define the *power set iteration operation*, $\mathbb{D}^{(k)}$ for $k \geqslant 0$ applied to a (finite) set, S, as

$$\mathbb{D}^{(k)}(S) = \begin{cases} S & \text{if } k = 0 \\ 2^{\mathbb{D}^{(k-1)}(S)} & \text{if } k \geqslant 1 \end{cases}$$

Hence $\mathbb{D}^{(0)}(S), \mathbb{D}^{(1)}(S), \mathbb{D}^{(2)}(S), \ldots$ are successively S itself, the set of all subsets of S, the set of all subsets of the set of all subsets of S, and so on (more tersely S, 2^S, 2^{2^S}).

Lemma 1. *For any finite S and $k \geqslant 1$ there is a total computable mapping*

$$\eta^{(k)} : \mathbb{D}^{(k)}(S) \quad \leftrightarrow \quad [0, 1, 2, \ldots, |\mathbb{D}^{(k)}(S)| - 1]$$

Proof. First note that $|\mathbb{D}^{(k)}(S)| = 2^{|\mathbb{D}^{(k-1)}(S)|}$. Order the sets in $\mathbb{D}^{(k-1)}(S)$ by increasing size (and "lexicographically" for sets of the same size). Consider the binary encoding of any whole number, w, with $0 \leqslant w \leqslant |\mathbb{D}^{(k)}(S)| - 1$. We can use the value (0 or 1) of the ith bit to describe the presence (1) or absence (0) of the ith set in $\mathbb{D}^{(k-1)}(S)$ when describing a member of $\mathbb{D}^{(k)}(S)$. Thus the whole number 0 describes the empty set (binary encoding as a string of $|\mathbb{D}^{(k-1)}(S)|$ 0s) while the whole number $|\mathbb{D}^{(k)}(S)| - 1$ describes the set containing all members of $\mathbb{D}^{(k-1)}(S)$ (binary encoding as a string of $|\mathbb{D}^{(k-1)}(S)|$ 1s).

It should be clear that given any S, $k \geqslant 1$ and $U \in \mathbb{D}^{(k)}(S)$ we form $\eta^{(k)}(U)$ and from any S, $k \geqslant 1$ and $0 \leqslant u \leqslant |\mathbb{D}^{(k)(S)}| - 1$ recover $\eta^{(k)^{-1}}(u)$ the corresponding member of $\mathbb{D}^{(k)}(S)$. □

Now it is not hard to see that we can also think in terms of numerical properties via functions, $f : \mathbb{N} \to \mathbb{N}$. For example $\sharp\text{cf} : \mathbb{N} \to \mathbb{N}$ describes "the number of conflict-free sets in $\text{dec}^{(\mathbb{A})}(n)$".

We develop this notion more precisely in the next section.

4. Languages, predicates, and argument semantics

We have described a language L as a set of words over a finite alphabet (often $\Sigma = \{0, 1\}$).

We have predicates, p, whose domain we consider to be the Natural numbers, $\mathbb{N}$.

An argument semantics, σ, we treat as a set of pairs $(\mathcal{H}, \mathbb{S})$ with $\mathcal{H}$ an AF and $\mathbb{S} \subseteq 2^{\mathcal{X}}$ so that $\sigma(\mathcal{H}) = \mathbb{S}$, or equivalently that $E_\sigma(\mathcal{H}, \mathbb{S})$ holds.

Superficially these may seem to be three quite distinct formalisms. One of the gains of the encoding mechanisms described in the previous section is in demonstrating that all are *equivalent*: we can treat a language, L, as a predicate p_L or argument semantics σ_L; we can go in the opposite direction and formulate from a predicate, p, languages over single character alphabets, $L_p^{(\mathbb{N})}$ and argument semantics $\sigma_p^{(\mathbb{N})}$. Finally we may move from an argument semantics, σ, to languages L_σ or predicates p_σ. The fact that encoding schemes over some collection of objects, $\mathbb{O}$ can be *ordered* is a key tool here.

The purpose of this section is to describe these translations and introduce the notational conventions used in Section 5.

4.1. Languages to predicates and semantics

Let $\text{lex} : \{0, 1\}^* \to \mathbb{N}$ be the usual lexicographic ordering (with $0 \prec 1$)

Suppose $L \subseteq \{0, 1\}^*$. The predicate $p_L : \mathbb{N} \to \{\top, \bot\}$ has

$$p_L(n) = \top \quad \Leftrightarrow \quad \text{lex}^{-1}(n) \in L$$

The argument semantics, σ_L is described by

$$w \in L \quad \Leftrightarrow \quad E_{\sigma_L}\big(\text{dec}^{(\mathbb{A})}\big(\text{lex}(w)\big)\big) = \top$$

4.2. Predicates to languages and semantics

Given $p : \mathbb{N} \to \{\top, \bot\}$ the language, $L_p^{(\mathbb{N})} \subseteq \{1\}^*$ corresponding to it has

$$L_p^{(\mathbb{N})} = \left\{1^n : p(n) = \top\right\}$$

The argument semantics $\sigma_p^{(\mathbb{N})}$ has

$$E_{\sigma_p^{(\mathbb{N})}}(\mathcal{H}, \mathbb{S}) = \top \quad \Leftrightarrow \quad p\left(\text{enc}^{(\mathbb{A})}(\mathcal{H}, \mathbb{S})\right) = \top$$

4.3. Semantics to languages and predicates

If σ is an argument semantics the language $L_\sigma \subseteq \{0, 1\}^*$ is

$$L_\sigma = \left\{\mu(\mathcal{H}, \mathbb{S}) : E_\sigma(\mathcal{H}, \mathbb{S}) = \top\right\}$$

The predicate p_σ has

$$p_\sigma(n) = \top \quad \Leftrightarrow \quad E_\sigma\left(\text{dec}^{(\mathbb{A})}(n)\right) = \top$$

4.4. Summary

A semantics, σ, has an associated predicate $E_\sigma : \mathbb{A} \to \{\top, \bot\}$ so that $E_\sigma(\mathcal{H}, \mathbb{S}) \Leftrightarrow \mathbb{S} = \sigma(\mathcal{H})$.
A semantics, σ, is described by a predicate, $\sigma^{(\mathbb{N})} : \mathbb{N} \to \{\top, \bot\}$ for which $\sigma^{(\mathbb{N})}(n) \Leftrightarrow E_\sigma(\text{dec}^{(\mathbb{A})}(n))$.
A predicate $p : \mathbb{N} \to \{\top, \bot\}$ has an associated semantics σ_p for which $p(n) \Leftrightarrow E_{\sigma_p}(\text{dec}^{(\mathbb{A})}(n))$.
Predicates, $q : \mathbb{N} \to \{\top, \bot\}$ (regardless of their source) are languages, L_q, of finite length *single* character words so that $L_q = \{1^n : q(n) = \top\}$.

5. Decidability in argumentation semantics

In looking at properties of semantics from a computational perspective we would like these to have some desirable features, One such is given by the concept of *verifiability*.

A semantics, σ, is *verifiable* if there is a program, M, that given $\mu(\mathcal{H}, \{S\})$ as input behaves as follows:

1. M checks that its input is valid, i.e. describes an AF $\mathcal{H}$ and a subset S of $\mathcal{X}$, rejecting if this is not the case.
2. M checks $S \in \sigma(\mathcal{H})$ and rejects if $S \notin \sigma(\mathcal{H})$.

Theorem 1. *Let σ be a verifiable semantics. The language*

$$L_\sigma = \left\{\mu(\mathcal{H}, \mathbb{S}) : E_\sigma(\mathcal{H}, \mathbb{S}) = \top\right\}$$

is recursive.

Proof. Consider the program, M_σ that does the following on input $w \in \{0, 1\}^*$.

1. M_σ checks $w = \mu(\mathcal{H}, \mathbb{S})$ with $\mathbb{S} \in \mathbb{D}^{(2)}(\mathcal{X})$ rejecting if this is not the case.
2. For each $S \in \mathbb{S}$, M_σ verifies $S \in \sigma(\mathcal{H})$ rejecting if any $S \notin \sigma(\mathcal{H})$ is found.
3. For each $S \in \mathbb{D}^{(1)}(\mathcal{X}) \setminus \mathbb{S}$, M_σ verifies $S \notin \sigma(\mathcal{H})$ rejecting if any $S \in \sigma(\mathcal{H})$ is found.
4. If all of the tests in (1)–(3) succeed then M_σ accepts.

The program M_σ will accept any $(\mathcal{H}, \mathbb{S})$ with $E_\sigma(\mathcal{H}, \mathbb{S}) = \top$ and reject any $(\mathcal{H}, \mathbb{S})$ with $E_\sigma(\mathcal{H}, \mathbb{S}) = \bot$ hence L_σ is recursive. □

The notion of "semantic properties", as promoted in the principle-based concepts of Baroni and Giacomin [1], becomes rather more complicated. Suppose we have some property, π. We wish to formulate the question "does a semantics σ respect property π?" in computational terms, i.e. as a language of finite words.

We have shown in the previous section that any argumentation semantics describes a predicate over $\mathbb{N}$ and such predicates define languages of words from a single character alphabet. In computational terms, we can therefore focus on the computation of languages, $L \subseteq \{1\}^*$. Each such language offers an (admittedly usually highly artificial) argumentation semantics; every argumentation semantics (in the sense we have defined these) describes such a language. In computational terms we would only be interested in verifiable semantics, that is to say those for which the associated single character language, $L_{\sigma^{(\mathbb{N})}}$ has a Turing Machine program, M_σ which recognises

$$L_{\sigma^{(\mathbb{N})}} = \left\{1^n : E_\sigma\left(\mathrm{dec}^{(\mathbb{A})}(n)\right) = \top\right\}$$

We can code these Turing machine programs in binary and order these codes. In summary, a semantics property, π, is

$$L_\Pi = \left\{\chi(M) : L(M) \subseteq \{1\}^* \text{ and } \sigma_L \text{ respects } \pi\right\}$$

There is, of course, a well-known issue with this approach, as exemplified in the next result.

Theorem 2. *There are semantics, σ, for which the language $L_{\sigma^{(\mathbb{N})}}$ is not recursive.*

Proof. Suppose the contrary and that for every σ we have a Turing machine program $M_{\sigma^{(\mathbb{N})}}$ that on input 1^n halts and reports $\top$ whenever $E_\sigma(\mathrm{dec}^{(\mathbb{A})}(n)) = \top$ and halts and reports $\bot$ with all other instances. Define the predicate diagsem : $\mathbb{N} \to \{\top, \bot\}$ as

$$\mathrm{diagsem}(k) = \begin{cases} \bot & \text{if } M_k(1^k) = \top \\ \top & \text{if } M_k(1^k) = \bot \end{cases}$$

Here M_k is the k'th Turing machine as given by the standard lexicographic ordering of TM codes with single symbol input alphabets. The predicate diagsem defines a semantics, $\sigma_{\mathrm{diagsem}}$ as described in Section 4.2. From the starting assumption, the language $L_{\sigma^{(\mathbb{N})}_{\mathrm{diagsem}}}$ is recursive and so there is a TM, M_{diagsem} that halts and reports $\top$ on instances 1^n for which $E_{\sigma_{\mathrm{diagsem}}}(\mathrm{dec}^{(\mathbb{A})}(n)) = \top$ and which halts and reports $\bot$ on all other instances. Since M_{diagsem} is a TM program with a single symbol input alphabet there is some $t \in \mathbb{N}$ such that $\chi(M_{\mathrm{diagsem}})$ is the t'th codeword. What, however, is the value diagsem(t)? It cannot be $\top$ since by definition diagsem(t) $= \top$ if $M_t(1^t) = \bot$; similarly it cannot be $\bot$ since this would only happen should $M_t(1^t) = \top$. It follows that $L_{\sigma^{(\mathbb{N})}_{\mathrm{diagsem}}}$ is not recursive. □

The proof technique, "diagonalization", of Theorem 2 is one of the classical tools used in the study of properties of infinite structures and dates back to the late 19th century.

One consequence is,

Corollary 1. *There are semantics, σ, for which the language $L_{\sigma(\mathbb{N})}$ is not recursively enumerable.*

Proof. Let σ be a semantics. Define its *complementary* semantics, coσ through the behaviour $E_{\text{co}\sigma}(\mathcal{H}, \mathbb{S}) = \neg E_\sigma(\mathcal{H}, \mathbb{S})$. The language $L_{\text{co}\sigma(\mathbb{N})} = \text{coL}_{\sigma(\mathbb{N})}$, i.e $L_{\text{co}\sigma(\mathbb{N})} = \{1\}^* \setminus L_{\sigma(\mathbb{N})}$. It is a well-known result from computability that if a language, L is *not* recursive then its complement coL is not r.e. This suffices to prove the claim. □

We could deal with each of the various semantic principles that have been proposed on a case-by-case basis. We can, however, proceed in a much more general style.

Definition 5. Let σ be any semantics. A *blocking set* for σ is a specific $T \subseteq \mathcal{X}$ for which there are choices $\mathcal{A}_{\text{in}}$ and $\mathcal{A}_{\text{out}}$ with $T \in \sigma(\mathcal{H}_{\text{in}})$ but $T \notin \sigma(\mathcal{H}_{\text{out}})$, $\mathcal{H}_{\text{in}}$, $\mathcal{H}_{\text{out}}$ being the AFs with attack sets $\mathcal{A}_{\text{in}}$ respectively $\mathcal{A}_{\text{out}}$.

The *augmentation* of σ by a blocking set $T_\sigma^{\mathcal{X}}$ is the semantics, denoted $\sigma_{+,T^{\mathcal{X}}}$, defined as

$$\left\{(\mathcal{H}, \mathbb{S}) : \mathbb{S} = \sigma(\mathcal{H}) \text{ or } T_\sigma^{\mathcal{X}} \in \mathbb{S}\right\}$$

For example for $\sigma = \text{cf}$ we could choose $T_{\text{cf}}^{\mathcal{X}} = \mathcal{X}$; for $\sigma = \text{com}$, $T_{\text{com}}^{\mathcal{X}} = \{x_1\}$.

Theorem 3. *Let σ be a semantics with a blocking set $T_\sigma^{\mathcal{X}}$. Suppose that π is a semantics property that σ respects but $\sigma_{+,T^{\mathcal{X}}}$ does not respect (for example conflict-freeness is respected by* cf *but not by* $\text{cf}_{+,\mathcal{X}}$*). Furthermore assume that L_σ is infinite (i.e. there are infinitely instances $(\mathcal{H}, \mathbb{S})$ such that $E_\sigma(\mathcal{H}, \mathbb{S}) = \top$).*

$$L_\Pi = \left\{\chi(M) : L(M) \subseteq \{1\}^* \textit{ and } \sigma_{L(M)} \textit{ respects } \pi\right\}$$

is not recursively enumerable.

Proof. The proof uses a technique adapted from the proof of Rice's Theorem for r.e. properties, see Hopcroft and Ullman [5, Theorem 8.7, pp. 191–2].

It is known that the language

$$L_{\text{out}} = \left\{\left\langle \chi(M), w\right\rangle : w \notin L(M)\right\}$$

is not r.e. (see Hopcroft and Ullman [5, Theorem 8.4, p. 184])

Suppose the contrary and that for some such property within the conditions stated, π say, there is some program, M_Π accepting L_Π: that is M_Π, given $\chi(M)$, M_Π halts and accepts all instances for which $\sigma_{L(M)}$ respects the property π.

Let σ have property π but be such that $\sigma_{+,T^{\mathcal{X}}}$ does not.

$$L_\sigma = \left\{1^k : E_\sigma\left(\text{dec}^{(\mathbb{A})}(k)\right) = \top\right\}$$

$$L_{\sigma_{+,T^{\mathcal{X}}}} = \left\{1^k : E_{\sigma_{+,T^{\mathcal{X}}}}\left(\text{dec}^{(\mathbb{A})}(k)\right) = \top\right\}$$

With these $L_\sigma \subset L_{\sigma_{+,T^{\mathcal{X}}}}$.

Suppose we have an instance $\langle \chi(M), w\rangle$ of L_{out} and consider the semantics defined as follows. Given 1^k with $\text{dec}^{(\mathbb{A})}(k) = (\mathcal{H}, \mathbb{S})$ we run a "pre-compilation" stage using a program Q.

1. Q simulates M running with input w. If M halts and accepts w then Q starts $M_{\sigma_{+,T^{\mathcal{X}}}}$ with $1^{\text{enc}^{(\mathbb{A})}(\mathcal{H},\mathbb{S})}$, the encoding of a program accepting the language $L_{\sigma_{+,T^{\mathcal{X}}}}$.
2. Regardless of what happens in (1) (e.g. this step could be performed in parallel) Q will run M_σ with input $1^{\text{enc}^{(\mathbb{A})}(\mathcal{H},\mathbb{S})}$.

Now the semantics described by Q will correspond to $\sigma_{+,T^{\mathcal{X}}}$ in the event that $w \in L(M)$ and to $\sigma \subset \sigma_{+,T^{\mathcal{X}}}$ should $w \notin L(M)$. Thus given an instance $\langle \chi(M), w\rangle$ of L_{out} we can using Q construct the code of program, $M_{\tau,M,w}$ taking as its inputs $1^{\text{enc}^{(\mathbb{A})}(\mathcal{H},\mathbb{S})}$.

Suppose now that the property defined by L_Π were r.e.,

$$L_\Pi = \left\{\chi(M_\sigma) : \sigma \text{ respects } \pi\right\}$$

Then we build a program to recognise L_{out} by

a. Given $\langle \chi(M), w\rangle$ use Q to compile the code $\chi(M_{\tau,M,w})$ whose inputs are of the form $1^{\text{enc}^{(\mathbb{A})}(\mathcal{H},\mathbb{S})}$.
b. Run M_Π with input $\chi(M_{\tau,M,w})$.

Then M_Π will accept $\chi(M_{\tau,M,w})$ only if the semantics it describes corresponds to L_σ that is $w \notin L(M)$ so contradicting the fact that L_{out} is not r.e. □

Corollary 2. *The languages describing the following properties for a semantics, σ, are not recursively enumerable.*

a. *Conflict-freeness.*
b. *Reinstatement.*
c. *Admissibility.*
d. *Strong admissibility, i.e. $\sigma(\mathcal{H}) \subseteq \text{adm}(\mathcal{H})$ and $S \in \sigma(\mathcal{H}) \Rightarrow (\forall p \in S, q \in p^- \; \exists r \in S \setminus p : \langle r, q\rangle \in \mathcal{A})$.*
e. *Naivety, i.e. $S \in \sigma(\mathcal{H}) \Rightarrow (S \in \text{cf}(\mathcal{H}))$ and is a maximal (w.r.t. $\subseteq$) set in $\text{cf}(\mathcal{H})$.*
f. *Abstention, i.e $\forall x \in \mathcal{X}$ if $S, T \in \sigma(\mathcal{H})$ with $x \in S$, $x \in T^+$ then there is some $Q \in \sigma(\mathcal{H})$ with $x \notin Q \cup Q^+$.*
g. *I-maximality, i.e. $\forall S, T \in \sigma(\mathcal{H})\; S \subseteq T \rightarrow S = T$.*

Proof. From Theorem 3, it suffices to identify σ, and an augmentation of σ by a blocking set $T^{\mathcal{X}}$ in such a way that σ has the property of interest but $\sigma_{+,T^{\mathcal{X}}}$ does not.

a. $\sigma = \text{cf}$, $T^{\mathcal{X}} = \mathcal{X}$.
b. $\sigma = \text{com}$, $T^{\mathcal{X}} = \{x_1\}$.
c. $\sigma = \text{adm}$, $T^{\mathcal{X}} = \mathcal{X}$.
d. $\sigma = \text{gnd}$, $T^{\mathcal{X}} = \mathcal{X}$.
e. $\sigma = \text{naive}$, $T^{\mathcal{X}} = \{x_1, x_2\}$.
f. $\sigma = \text{com}$, $T^{\mathcal{X}} = \mathcal{X}$.
g. $\sigma = \text{pr}$, $T^{\mathcal{X}} = \{x_1\}$ □

The case of "abstention" is of interest since, of the standard argumentation semantics, only the complete semantics has this property (conventionally unique status semantics such as the grounded only "satisfy" abstention in a rather vacuous sense).

The results above are a consequence of the languages concerned i.e. "property of a semantics σ", failing to satisfy one of the three necessary and sufficient conditions prescribed by Rice's Theorem for r.e. properties. Namely, let L_Π be

$$L_\Pi = \{\chi(M) : L(M) \text{ has property } \Pi\}$$

RE1. If L_Π is r.e. and $\chi(M) \in L_\Pi$ is such that $L(M)$ is infinite then every r.e. L' with $L' \supset L(M)$ is in L_Π, i.e. the code $\chi(M')$ with $L(M') = L'$ is in L_Π.

RE2. If L_Π is r.e. and $\chi(M) \in L_\Pi$ is such that $L(M)$ is infinite then there is a finite $L' \subset L$ which is in L_Π, i.e. the code $\chi(M')$ with $L(M') = L'$ is in L_Π.

RE3. The set of *finite* languages, L, such that $L \in L_\Pi$ can be enumerated. That is to say, there is a program which will produce the output $L_1\#L_2\#\cdots L_k\#\cdots$ in which L_i is a list of all of the members of the i'th finite language (ordering is unimportant) and in which *every* finite language $L \in L_\Pi$ (eventually) will appear.

The languages L_{CF}, and L_{reinst} fail to satisfy (RE1) (the so-called "*containment property*"). It is, however, possible to show these satisfy (RE2).

Theorem 4. *Let σ be a semantics that satisfies*

$$\exists k \in \mathbb{N}\ E_\sigma(\mathcal{H}_k, \mathbb{T})$$

where $|\mathcal{X}_k| \leqslant k$. Let τ be the semantics with corresponding language

$$L_\tau = \{1^m : \text{dec}^{(\mathbb{A})}(m) = (\mathcal{H}, \mathbb{S}), E_\sigma(\mathcal{H}, \mathbb{S}) \text{ and } |\mathcal{X}| \leqslant k\}$$

If π is a semantic property that σ respecting π implies τ respecting π then L_Π satisfies (RE2), i.e. for every infinite language with the property there is a finite subset with the property.

Proof. If $\chi(M_\sigma) \in L_\Pi$ then the condition indicates that $\chi(M_\tau) \in L_\Pi$. The language L_τ is finite by virtue of its corresponding AFs having only finitely many arguments. □

6. A positive result

Although many of the semantic properties of interest yield non r.e. languages and hence fail even to be partially decidable, we can conclude by presenting what is in some ways, a rather surprising positive result. This arises in the technically sophisticated concept of *serializability* from Thimm [6].

We first need the concept of *initial sets*, which Xu and Cayrol [8] introduced as the non-empty S in $\mathcal{H}$ such that

$$S \in \text{adm}(\mathcal{H}) \quad \text{and} \quad \forall : \emptyset \subset T \subset S : T \notin \text{adm}(\mathcal{H})$$

Letting $\text{IS}(\mathcal{H})$ denote the initial sets of $\mathcal{H}$, Thimm [6] groups initial sets into three classes.

1. *unattacked* initial sets S: those with $S^- = \emptyset$.
2. *unchallenged* initial sets S: those for which $S^- \neq \emptyset$ and for all $T \in \text{IS}(\mathcal{H})$ $T^+ \cap S = \emptyset$.
3. *challenged* initial sets S: those with $S^- \neq \emptyset$ and for which some $T \in \text{IS}(\mathcal{H})$ has $T^+ \cap S \neq \emptyset$

For $\mathcal{H} = (\mathcal{X}, \mathcal{A})$:
$\text{IS}^{\not\leftarrow}(\mathcal{H})$ are the unattacked initial sets of $\mathcal{H}$.
$\text{IS}^{\not\leftrightarrow}(\mathcal{H})$ the unchallenged initial sets of $\mathcal{H}$.
$\text{IS}^{\leftrightarrow}(\mathcal{H})$ the challenged initial sets of $\mathcal{H}$.
The *reduct* of $\mathcal{H} = (\mathcal{X}, \mathcal{A})$ with respect to $S \subseteq \mathcal{X}$ is the AF, $\mathcal{H}_S$ with arguments $\mathcal{X} \setminus (S \cup S^+)$ and attacks $\mathcal{A} \setminus \{\langle p, q\rangle : p \in S \cup S^+ \text{ or } q \in S \cup S^+\}$.
Finally the concept of a semantics being *serializable* is presented in Thimm [6].
Let σ be a semantics, i.e. a mapping from any $\mathcal{H}$ to sets of subsets of $\mathcal{X}$.
A *state* is a pair $(\mathcal{H}, S)$ with $S \subseteq \mathcal{X}$.
A *selector* is a function $\alpha : 2^{2^{\mathcal{X}}} \times 2^{2^{\mathcal{X}}} \times 2^{2^{\mathcal{X}}} \rightarrow 2^{2^{\mathcal{X}}}$ satisfying $\alpha(R, S, T) \subseteq R \cup S \cup T$ for all $R, S, T \subseteq 2^{\mathcal{X}}$.
A *terminator* is any Boolean function β mapping states to $\{0, 1\}$.
A *transition* takes $(\mathcal{H}, S)$ and $T \in \alpha(\text{IS}^{\not\leftarrow}(\mathcal{H}), \text{IS}^{\not\leftrightarrow}(\mathcal{H}), \text{IS}^{\leftrightarrow}(\mathcal{H}))$ and returns the state $(\mathcal{H}_T, S \cup T)$.
We write $(\mathcal{H}, S) \rightsquigarrow^{(\alpha)} (\mathcal{H}', T)$ if the state $(\mathcal{H}', T)$ results after a finite number of applications of α. If $\beta(\mathcal{H}', T) = 1$ the state is terminal in which case we write $(\mathcal{H}, S) \rightarrow^{(\alpha,\beta)} (\mathcal{H}', T)$.
Finally let $\mathcal{G}^{(\alpha,\beta)}(\mathcal{H})$ be

$$\mathcal{G}^{(\alpha,\beta)}(\mathcal{H}) = \left\{S \subset \mathcal{X} : (\mathcal{H}, \emptyset) \rightsquigarrow^{(\alpha,\beta)} \left(\mathcal{H}', S\right)\right\}$$

(for some AF, $\mathcal{H}'$)

Definition 6 (Thimm [6])**.** A semantics σ is *serializable* if there are a selector α and terminator β with which $\sigma(\mathcal{H}) = \mathcal{G}^{(\alpha,\beta)}(\mathcal{H})$.

The following example is taken from Thimm [6].
Choose as selector function

$$\alpha_{\text{adm}}(R, S, T) = R \cup S \cup T \quad \text{subject to } R \in \text{IS}^{\not\leftarrow}, S \in \text{IS}^{\not\leftrightarrow}, T \in \text{IS}^{\leftrightarrow}$$

Let the termination function be $\beta(\mathcal{H}, S) = 1$ if and only if $\text{IS}(\mathcal{H}) = \emptyset$: hence the termination condition is such that the process of "select (using α_{adm}) – apply transition (form the reduct)" leads to a framework with no initial sets. Using these Thimm [6] has shown that the *preferred semantics are serializable*. Changing the termination condition from "$\beta(\mathcal{H}, S) = 1$ if and only if $\text{IS}(\mathcal{H}) = \emptyset$" to "$\beta(\mathcal{H}, S) = 1$", that is irrespective of further conditions on the state $(\mathcal{H}, S)$, establishes that the *admissibility semantics are serializable*. In fact all of the cases in Definition 3 are serializable as also *stable semantics* (S is admissible with $S^+ = \mathcal{X} \setminus S$). In contrast, however, neither semi-stable (S is admissible with $S \cup S^+$ maximal) nor ideal (S is admissible with every argument of S in every preferred extension) are serializable.
It is clear that

$$\sigma(\mathcal{H}) = \{S \subseteq \mathcal{X} : S \text{ is an initial set of } \mathcal{H}\}$$

is verifiable, as also the associated semantics arising from unattacked, unchallenged and challenged initial sets. Recall that a *state* is a pair $\langle \mathcal{H}, S \rangle$ mapped by a selector function α whose arguments come from $\mathrm{IS}^{\nleftarrow}(\mathcal{H})$, $\mathrm{IS}^{\nrightarrow}(\mathcal{H})$ and $\mathrm{IS}^{\leftrightarrow}(\mathcal{H})$ to a new state. We thus have

$$L_{\mathrm{trans}} = \left\{ (q_{\mathrm{init}}, q_{\mathrm{dest}}) : \exists \alpha \text{ with } q_{\mathrm{init}} \rightsquigarrow^{(\alpha)} q_{\mathrm{dest}} \right\}$$

Theorem 5. *L_{trans} is recursive.*

Proof. Recall that instances are pairs of states $\langle (\mathcal{H}, S), (\mathcal{H}', T) \rangle$. Furthermore selectors, α, may be viewed as 4-tuples of whole numbers (u, v, w, t) with $0 \leqslant u, v, w, t \leqslant 2^{2^{|\mathcal{X}|}} - 1$. From Lemma 1 we can move between sets of subsets of $\mathcal{X}$ and whole numbers in this range. Finally we note that starting from the 4-tuple $(0, 0, 0, 0)$ we can consider each 4-tuple in turn ending at $(2^{2^{|\mathcal{X}|}} - 1, 2^{2^{|\mathcal{X}|}} - 1, 2^{2^{|\mathcal{X}|}} - 1, 2^{2^{|\mathcal{X}|}} - 1)$. It is thus possible systematically to review *every* selector function.

We do not, of course, propose the following algorithm as a practical method: it is solely to demonstrate the claim of the Theorem.

Given $\langle (\mathcal{H}, S), (\mathcal{H}', T) \rangle$ proceed as follows

a. Set the current state $(\mathcal{G}, R)$ to be $(\mathcal{H}, S)$
b. Choose the next selector (u, v, w, t).
c. If the subsets corresponding to (u, v, w) are not from $(\mathrm{IS}^{\nleftarrow}(\mathcal{G}), \mathrm{IS}^{\nrightarrow}(\mathcal{G}), \mathrm{IS}^{\leftrightarrow}(\mathcal{G}))$ go back to (b).
d. If the subset corresponding to t is not a subset of the union of those given by u, v, w go back to (b)
e. Let $B = \eta^{-1}(t)$. Update the current state to $(\mathcal{G}_B, S \cup B)$.
f. If $B = \emptyset$ return to (b).
g. Recursively call the procedure with input $\langle (\mathcal{G}_B, S \cup B), (\mathcal{H}', T) \rangle$.
h. If the result of (f) is accept then report accept otherwise return to (b)
i. If all selectors considered then reject.

Notice that the reduct operation reduces the number of arguments while the second state argument increases. Thus the recursive step at (g) will eventually terminate. □

Although Theorem 5 establishes the decidability of whether a selector function exists that moves from one state to another it is a little bit unsatisfactory in one regard: it can be shown that the preferred semantics is serializable via the selector $\alpha(X, Y, Z) = X \cup Y \cup Z$ a choice which is clearly effective. A selector function discovered by the procedure of Theorem 5 only identifies a sequence of transitions relevant to one framework.

7. Conclusions

Our main aim in this paper has been to consider a different aspect of algorithmics and argumentation semantics: that of decidability rather than computational complexity. The issue of whether algorithmic processes exist is not one that arises in typical environments: these will involve finite systems and deal with questions concerning specific semantics. Such semantics are matters that concern properties of finite sets of subsets of arguments. When, however, we start to consider argumentation semantics in the principle based terms promoted by, among others, Baroni and Giacomin [1] we face a new set of problems. In principle, it is always possible to analyze new proposals on an *ad hoc* basis in determining which desirable properties such have. Were it possible, however, to demonstrate adherence to given

principle *algorithmically* such *ad hoc* approaches become redundant: codify the semantics in a form open to computational processing, present the codified form to a suitable program for analysis. We have shown that the main obstacle to this process is not the first aspect (finite codification of a semantics) but the second. The scale of unsolvability going beyond merely non-recursive (as in classical examples such as the Halting Problem of Turing [7]): in these when an instance is positive then it will, eventually, be accepted. Instead for semantic properties we face non-recursive enumerability: neither positive nor negative instances are guaranteed eventually to be recognized. The interplay between computation and argument semantics as described in Section 4 provides an insight into how powerful the notion of argument semantics is. Whether there are significant gains from this viewpoint is the object of study in future work.

References

[1] P. Baroni and M. Giacomin, On principle-based evaluation of extension-based argumentation semantics, *Artificial Intelligence* **171**(10–15) (2007), 675–700.
[2] P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, *Artificial Intelligence* **77** (1995), 321–357. doi:10.1016/0004-3702(94)00041-X.
[3] W. Dvorak and P.E. Dunne, Computational problems in formal argumentation and their complexity, in: *Handbook of Formal Argumentation*, College Publications, 2018, pp. 631–687.
[4] J.E. Hopcroft, R. Motwani and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison–Wesley, 2001.
[5] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison–Wesley, 1979.
[6] M. Thimm, Revisiting initial sets in abstract argumentation, *Argument & Computation* **13**(3) (2022), 325–360.
[7] A.M. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of the London mathematical society* **2**(1) (1937), 230–265. doi:10.1112/plms/s2-42.1.230.
[8] Y. Xu and C. Cayrol, Initial sets in abstract argumentation frameworks, *Journal of Applied Non-Classical Logics* **28**(2–3) (2018), 260–279. doi:10.1080/11663081.2018.1457252.