Labelling-based Algorithms for SETAFs

Wolfgang Dvořák, Anna Rapberger and Johannes P. Wallner
Institute of Logic and Computation, TU Wien, Austria

Abstract. In this work we consider labelling-based algorithms for evaluating SETAFs. We propose labellings that label both arguments and attacks for labelling-based algorithms and systematically investigate label propagation rules for stable and complete-based semantics, i.e., under which circumstances we can determine the label of an argument or attack by the already fixed labels of the neighbours.

Keywords. labelling-based algorithms, collective attacks

1. Introduction

Formal models within the field of abstract argumentation provide the key machinery underlying many state-of-the-art approaches in argumentation in AI [1]. Central to such models are argumentation frameworks (AFs) initially proposed and developed by Dung [7], which nowadays not only find a multitude of applications in formal argumentation, but also were extended in several ways to incorporate further useful features [2]. Among the generalizations, an appealing way to extend AFs is to generalize binary attacks within AFs to collective attacks, resulting in argumentation frameworks with collective attacks (SETAFs) [14,10]. SETAFs are both close to AFs, in that only attacks between arguments are representable, and provide the representational advantage of allowing to directly model conflicts arising from a joint collection of arguments, a scenario that can be modelled via AFs only through auxiliary structures.

Development and usability of formalizations of abstract arguments, such as SETAFs, relies on studying computational approaches, witnessed, e.g., by the argumentation competition [19,11], that establish the basic research needed for development of mature tools for argumentative decision support. For AFs, in the recent years we experienced a steady gain of interest and research detailing many computational properties and algorithms [5,3]. Algorithms for AFs can be roughly categorized into two classes: (i) reduction-based approaches and (ii) direct approaches. In the former we find algorithms making use of delegating certain tasks to highly-engineered solvers, such as solvers for the Boolean satisfiability (SAT) problem. On the other hand, direct approaches aim to develop an algorithm dedicated to the reasoning tasks at hand, without relying on translations to other (constraint) languages. While reduction-based approaches generally fared better in recent performance comparisons (see competitions [19,11]), direct approaches showed good performance on certain families of instances [4]. Furthermore, direct approaches focus (more) on task-specific and AF-specific particularities to boost

performance and provide shortcuts. The most prominent approach for direct algorithms are labelling-based algorithms (see, e.g., [6,16,15,20,18]), which base their computation on assigning labels to arguments and a *propagation* procedure, which, on a high-level, shares similarities to procedures for solving the SAT problem. Labelling-based algorithms in the current state of the art are not as competitive as approaches using SAT solvers for AFs, however, labelling-based algorithms detail ways in which assignment of labels imply (non-)labelling of other arguments. In this way, labelling-based algorithms open up opportunities to employ such techniques also in reduction-based approaches, in addition to provide a dedicated approach to solve reasoning tasks on AFs.

Recently, reduction-based systems for SETAFs based on ASP [8] and SAT-solvers[1] have been presented, however direct algorithms for SETAFs have not received much attention, with the exception of [13] which presents certain pruning strategies when computing preferred extensions. We take up this opportunity and develop labelling-based algorithms for SETAFs. Based on a recent proposal of defining semantics of SETAFs via three-valued argument labellings [10], we

- propose to extend assignment of label also to attacks, which reflects the richer structure of collective attacks than the binary attacks in AFs;
- provide a labelling-based algorithm for computing the grounded extension/labelling of a SETAF in linear time, and we
- provide a systematic analysis of possible label propagations for stable and complete-based semantics.

This paper is organized as follows. We start with a brief recap of SETAFs and its semantics in Section 2 and then introduce our argument-attack labellings in Section 3. In Section 4 we then present a linear time labelling-based algorithm for grounded semantics and investigate possible label propagations for stable and complete-based semantics in Sections 5 & 6. We conclude the paper with pointers to future work.

2. Argumentation Frameworks with Collective Attacks

We introduce formal definitions of argumentation frameworks with collective attacks (SETAFs) following [14,10].

Definition 1. A SETAF is a pair $F = (A,R)$ where A is finite, and $R \subseteq (2^A \setminus \emptyset) \times A$ is the attack relation.

Notice that SETAFs that only allow for binary attacks are equivalent to Dung argumentation frameworks (AFs) as introduced in [7]. We write $S \mapsto_R b$ if there is a set $S' \subseteq S$ with $(S',b) \in R$. Moreover, we write $S' \mapsto_R S$ if $S' \mapsto_R b$ for some $b \in S$. We drop subscript R in $\mapsto_R$ if there is no ambiguity.

Definition 2. Given a SETAF $F = (A,R)$, an argument $a \in A$ is *defended* (in F) by a set $S \subseteq A$ if for each $B \subseteq A$, such that $B \mapsto_R a$, also $S \mapsto_R B$. A set T of arguments is defended (in F) by S if each $a \in T$ is defended by S (in F).

[1] `https://bitbucket.org/andreasniskanen/joukko`

Next, we introduce the semantics we study in this work. These are the stable, preferred, complete, and grounded semantics, which we will abbreviate by *stb*, *pref*, *com*, and *grd*, respectively. For a given semantics σ, $\sigma(F)$ denotes the set of extensions of F under σ.

Definition 3. Given a SETAF $F = (A,R)$, a set $S \subseteq A$ is *conflict-free* (in F), if $S' \cup \{a\} \not\subseteq S$ for each $(S',a) \in R$. We denote the set of all conflict-free sets in F as $cf(F)$. $S \in cf(F)$ is called *admissible* (in F) if S defends itself. We denote the set of admissible sets in F as $adm(F)$. For a conflict-free set $S \in cf(F)$, we say that

- $S \in stb(F)$, if $S \mapsto a$ for all $a \in A \setminus S$,
- $S \in pref(F)$, if $S \in adm(F)$ and there is no $T \in adm(F)$ such that $T \supset S$,
- $S \in com(F)$, if $S \in adm(F)$ and $a \in S$ for all $a \in A$ defended by S, and
- $S \in grd(F)$, if $S = \bigcap_{T \in com(F)} T$.

As shown in [14], most of the fundamental properties of Dung AFs extend to SETAFs. We have the same relations between the semantics, i.e., $stb(F) \subseteq pref(F) \subseteq com(F)$. Moreover, the grounded extension is the unique minimal complete extension for any SETAF $F = (A,R)$ and can be alternatively defined as the least fixed point of the *characteristic function* $\Gamma_F : 2^A \to 2^A$, $\Gamma_F(S) = \{a \in A \mid a \text{ is defended by } S \text{ in } F\}$.

Semantics for SETAFs can be alternatively defined through argument labellings, as proposed in [10]. An argument labelling is formally defined as a function from arguments to a set of labels $\{in, out, undec\}$. Intuitively, an argument a belongs to a extension if it is labelled *in*; a is attacked by an extension if it is labelled *out*; and a is neither accepted nor attacked by accepted arguments if it is labelled *undec*.

Definition 4. An argument labelling for a SETAF $F = (A,R)$ is a total function $\mathscr{L} : A \to \{in, out, undec\}$. We define $l(\mathscr{L}) = \{a \in A \mid \mathscr{L}(a) = l\}$ for $l \in \{in, out, undec\}$.

We state the labelling-based formulations of the considered semantics.

Definition 5. Let $F = (A,R)$ be a SETAF. An argument labelling $\mathscr{L}$ of F is called

- *complete* if for all $a \in A$,

$$\mathscr{L}(a) = in \;\Leftrightarrow\; \text{for all } (S,a) \in R, \text{ there is } b \in S \text{ with } \mathscr{L}(b) = out,$$
$$\mathscr{L}(a) = out \;\Leftrightarrow\; \text{there is } (S,a) \in R \text{ such that for all } b \in S,\ \mathscr{L}(b) = in;$$

- *preferred* if $\mathscr{L}$ is complete and $in(\mathscr{L})$ is maximal with respect to set-inclusion among all complete argument labellings of F;
- *stable* if $\mathscr{L}$ is complete and $undec(\mathscr{L}) = \emptyset$;
- *grounded* if $\mathscr{L}$ is complete and $in(\mathscr{L})$ is minimal with respect to set-inclusion among all complete argument labellings of F.

In [10], it has been shown that the classes of argument labellings we introduced in Definition 5 correspond to the respective extensions and vice versa.

Theorem 1 ([10])**.** *For SETAF $F = (A,R)$, for $\sigma \in \{com, grd, pref, stb\}$, there is a one-to-one correspondence between extensions and labellings such that for each $S \in \sigma(F)$ there is a σ argument labelling $\mathscr{L}$ with $in(\mathscr{L}) = S$, $out(\mathscr{L}) = S^+$ and $undec(\mathscr{L}) = A \setminus (S \cup S^+)$.*

For computational purposes we will refer to the size of the SETAFs and use the following notations. For a set AF $F = (A,R)$, we call $|A|$ the number of arguments, $|R|$ the number of attacks, and $\|R\| = |R| + \sum_{(S,a)\in R} |S|$ the size of the attack relation. The input size $\|F\|$ of an SETAF F is given by the number of arguments plus the size of the attack relation, i.e., $\|F\| = |A| + \|R\|$.

3. Labellings on Arguments and Collective Attacks

In this section, we introduce argument-attack labellings for SETAFs which generalize the idea of labellings introduced in [10] by assigning labels to both arguments and attacks. Note that labels on attacks have been previously applied to a different generalization of Dung AFs which allows for attacks on attacks [15].

In contrast to binary attacks (a,b), where one can directly infer the acceptance status of b from the acceptance status of the single argument a, collective attacks (S,a) require to check the acceptance status of each argument $b \in S$ in order to derive conclusions about a. We will therefore introduce argument-attack labellings in order to store the status of attacks in SETAFs. Intuitively, an attack (S,a) is labeled *in* if S belongs to an extension; (S,a) is labeled *out* if S is attacked by an extension; and (S,a) is labeled *undec* if S does neither belong to an extension nor is attacked by an extension.

Definition 6. An *argument-attack labelling* of a SETAF $F = (A,R)$ is a total function $\lambda : A \cup R \to \{in, out, undec\}$; we write $l(\lambda) = \{a \in A \mid \lambda(a) = l\}$ for $l \in \{in, out, undec\}$.

We define complete argument-attack labellings for SETAFs.

Definition 7. Let (A,R) be a SETAF. A labelling is a *complete argument-attack labelling* if it satisfies the following constraints:

- For every attack $(S,a) \in R$,

$$\lambda((S,a)) = in \Leftrightarrow \text{ for all } b \in S,\ \lambda(b) = in, \tag{1}$$

$$\lambda((S,a)) = out \Leftrightarrow \text{ there is } b \in S \text{ with } \lambda(b) = out. \tag{2}$$

- For every argument $a \in A$,

$$\lambda(a) = in \Leftrightarrow \text{ for all } (S,a) \in R, \lambda((S,a)) = out, \tag{3}$$

$$\lambda(a) = out \Leftrightarrow \text{ there is } (S,a) \in R \text{ with } \lambda((S,a)) = in. \tag{4}$$

The following constraints for *undec*-labels follow from (1) and (2) (or (3) and (4), respectively): For an attack $(S,a) \in R$, $\lambda((S,a)) = undec$ iff there is $b \in S$ which is labelled *undec* and no other argument $b' \in S$ is labelled *out*. For an argument $a \in A$, $\lambda(a) = undec$ iff there is an attack $(S,a) \in R$ which is labelled *undec* and a is not attacked by an *in*-labelled attack.

Complete argument labellings and complete argument-attack labellings are closely related; in fact, it can be shown that each complete argument-attack labelling induces a complete argument labelling and vice versa.

Proposition 1. *For any SETAF $F = (A,R)$, the following holds:*

(a) If λ is a complete argument-attack labelling of F then $\mathscr{L} = \lambda|_A$ is a complete argument labelling of F.
(b) If $\mathscr{L}$ is a complete argument labelling of F then λ is a complete argument-attack labelling of F where λ is defined as follows:

(i) $\lambda(a) = \mathscr{L}(a)$ for $a \in A$;
(ii) $\lambda((S,a)) = in$ if for all $b \in S$, $\mathscr{L}(b) = in$;
(iii) $\lambda((S,a)) = out$ if there is $b \in S$ with $\mathscr{L}(b) = out$;
(iv) $\lambda((S,a)) = undec$ else.

The following characterization of preferred, grounded and stable semantics in terms of argument-attack labellings is immediate by Proposition 1 and Theorem 1.

Proposition 2. *For any SETAF $F = (A,R)$, for $\sigma \in \{com, grd, pref, stb\}$, there is a one-to-one correspondence between extensions and argument-attack labellings such that for each $S \in \sigma(F)$ there is a σ argument-attack labelling λ with $in(\lambda) = S$, $out(\lambda) = S^+$ and $undec(\lambda) = A \setminus (S \cup S^+)$.*

In the following, we will only consider argument-attack labellings and thus refer to them simply as labellings. In our algorithm we typically do not immediately label all the arguments but will consider partial labellings that will be completed during the algorithm. To this end we will sometimes also use additional labels to encode an intermediate state of an argument label.

Definition 8. A *partial labelling* is a partial function $\mu : A \cup R \to \{in, out, undec\} \cup \mathscr{E}$, where $\mathscr{E}$ is a possibly empty set of additional labels.

Remark 1. An alternative approach towards labellings for SETAFs would be to label subsets of argument instead of attacks. Labelling all subsets of arguments can lead to labellings of exponential size and is thus not well suited for computational purposes. However, when inspecting the conditions for the labels of attacks in Definition 7 we observe that the label of an attack (S,a) is determined by the set S. Thus, for an arbitrary complete labelling and two attacks (S,a) and (S,b), i.e, the attacks have the same set of arguments but attack different arguments, we have that both attacks have the same label. That is, our argument-attack labellings can be equivalently interpreted as labellings of arguments and certain sets of arguments, i.e., those sets S such that there is an attack (S,a). If the representation of SETAFs in an implementation allows to identify attacks with the same set of arguments efficiently one can exploit this observation for more succinct representations of labellings.

4. A Labelling-based Algorithm for Grounded Semantics

The basic idea for a grounded algorithm is to start with the unattacked arguments and iteratively compute the defended arguments and add them to the extension until a fixed point is reached. When done in a naive way, in each iteration one would check for each argument not in the (partial) extension whether it is defended against all incoming at-

tacks, via computing the status of such an attack. We avoid this repetitive computations by using our labellings that also label attacks and by that store the status of an attack.

In terms of our labellings the basic idea of our algorithm is to assign the label *in* to all unattacked arguments and then propagate labels to other arguments as follows.
For an unlabelled argument $a \in A$,

1. we set $\mu(a) = in$ if for all attacks $(S,a) \in R$, we have $\mu((S,a)) = out$, and
2. we set $\mu(a) = out$ if there is $(S,a) \in R$ with $\mu((S,a)) = in$;

and for an unlabelled attack $(S,a) \in R$,

1. we set $\mu((S,a)) = in$ if we have $\mu(b) = in$ for all $b \in S$, and
2. we set $\mu((S,a)) = out$ if there is $b \in S$ with $\mu(b) = out$.

Finally, all arguments and attacks that remain unlabelled when the fixed-point of these propagations is reached are labelled *undec*.

In Algorithm 1 we efficiently implement these labelling propagation idea by: a) (re)considering arguments (attacks) only if the label of an incident attack (argument) has changed; b) use counters for arguments (attacks) to efficiently store the progress on incident attacks (arguments) in order to avoid rechecking all the neighbors if the label of one neighbor changes (similar counters have been used in linear time algorithms for computing the grounded extension of a Dung AFs [12,17]).

As a data structure we will use a partial labelling μ that maps arguments and attacks to the labels $\{in, out, undec\}$ and a function χ that maps each argument and attack to an integer between 0 and $\max(|A|,|R|)$. For an argument a the value $\chi(a)$ represents the number of attacks (S,a) that are not labeled *out* by μ, while for an attack (S,b) the value $\chi((S,b))$ represents the number of arguments $c \in S$ that are not labeled *in* by μ. That is, if the counter turns to 0 we know that we can label the argument, or attack respectively, *in*. The algorithm, whenever setting the label of an argument or attack, updates the χ-values of all the neighbors and if a counter turns to 0 updates the label of the corresponding argument or attack.

In the Algorithm we use a data structure *Args* that stores a collection of the arguments that have been labelled but whose label has not yet been propagated to its neighbors. To implement *Args* we can use any data structure that implements the following operations in $O(1)$ time: (i) test whether *Args* is empty, (ii) add one element to *Args*, and (iii) extract one element, i.e., return an element from *Args* and remove that element from *Args*. Standard implementations of queues and stacks provide these properties. Thus we can assume we have a method $ExtractArg(Args)$ that, if *Args* is non-empty, returns an element and removes that element from *Args*. Also notice that our algorithm does not add duplicates to *Args*.

Using a standard queue data structure we obtain that the grounded extension can be computed in linear time w.r.t. size of the input SETAF.

Theorem 2. *Algorithm 1 can be implemented to run in time $O(\|F\|)$ for any SETAF F.*

Proof. Let $F = (A,R)$ and assume *Args* is implemented via a standard queue data structure. That is, the initialization of *Args* and the operations $Args \neq \emptyset$, $extractArg(Args)$ and $Args \cup \{b\}$ are in constant time. First consider the initialization phase of the algorithm. To set the counter $\chi(a)$ for argument a we iterate over all incoming attacks. Hence, we can set all the counters for the argument in $O(|A| + |R|)$. To set the counter $\chi((S,a))$ for

Algorithm 1 Grounded-labelling(F)

Require: SETAF $F = (A,R)$
Ensure: μ corresponds to the grounded labelling
// Initialize Data Structure
1: $\chi(a) \leftarrow |\{(S,a) \in R\}|$ for $a \in A$
2: $\chi((S,a)) \leftarrow |S|$ for $(S,a) \in R$
3: $Args \leftarrow \{a \in A \mid \chi(a) = 0\}$
4: $\mu(a) \leftarrow in$ for $a \in A$ with $\chi(a) = 0$
// Algorithm
5: **while** $Args \neq \emptyset$ **do**
6: $a \leftarrow ExtractArg(Args)$
7: **if** $\mu(a) = in$ **then**
8: **for** $(S,b) \in R$ with $a \in S$ **do**
9: $\chi((S,b)) \leftarrow \chi((S,b)) - 1$
10: **if** $\chi((S,b)) = 0$ **then** $\mu((S,b)) \leftarrow in$; $\mu(b) \leftarrow out$; $Args \leftarrow Args \cup \{b\}$
11: **end for**
12: **else**
13: **for** $(S,b) \in R$ with $a \in S$, $\mu((S,b)) \neq out$ **do**
14: $\mu((S,b)) \leftarrow out$
15: $\chi(b) \leftarrow \chi(b) - 1$
16: **if** $\chi(b) = 0$ **then** $\mu(b) \leftarrow in$; $Args \leftarrow Args \cup \{b\}$
17: **end for**
18: **end if**
19: **end while**
20: $\mu(a) \leftarrow undec$ for $a \in A$ with $\mu(a) \notin \{in, out\}$
21: **return** μ

attack (S,a) we iterate over all arguments in S and thus we can set all the counters for attacks in $O(\|R\|)$. Finally, for each argument $a \in A$ it only requires constant time to check whether $\chi(a) = 0$, set $\mu(a) = in$, and add a to the queue in $O(|A|)$. Hence, the initialization phase is in $O(|A| + \|R\|)$. Now consider the while loop. Notice that each argument is only added once to the queue and thus processed only once in the loop and for each argument we process all the outgoing attacks. All the other operations are in constant time. Thus it follows that each attack $(S,a) \in R$ is processed $O(|S|)$ many times and the overall running time of the while loop and the algorithm is in $O(|A| + \|R\|) = O(\|F\|)$. □

5. A Labelling-based Algorithm for Stable Semantics

Next we consider an algorithm to compute stable labellings. In contrast to grounded semantics we now deal with potentially many extensions and thus follow the standard scheme of labelling based algorithms. The rough idea of labelling-based algorithms is to start with all arguments (and in our case also attacks) unlabelled, in each step pick an argument and then consider two branches: one where we add the argument to the extension, i.e., labelled the argument *in*; and one where we decide that the argument is excluded from the extension, i.e., the argument must be labelled *out* for stable semantics. When all arguments are labelled, one tests whether the labelling is valid w.r.t. the considered semantics and, if so, it is added to the set of extensions. By that procedure we would consider all possible candidates for valid labellings and thus also obtain all the

extensions. In order to design an efficient algorithm one aims to cut off branches that do not lead to valid labellings as soon as possible. One approach are label propagation rules, i.e., one uses the already fixed labels of the arguments and attacks to determine labels of their neighboring arguments and attacks. By that, one avoids unnecessary branching in the algorithm. The general structure is given in Algorithm 2; key component is the function update(.) that a) sets an argument to a specific label, b) applies label propagation rules to determine labels of other arguments and attacks, and c) identifies local inconsistencies in the labelling which already determine that a partial labelling cannot be extended to a valid labelling. The update function returns the updated labelling and a Boolean variable that indicates whether it has identified any local inconsistencies.

Let us next discuss how we can propagate labels for stable semantics. We will first consider the case where we have a partial labelling μ and an argument $a \in A$. By systematically inspecting the definition of a stable labellings (i.e., complete labellings that have no *undec* label) we extract the following rules to obtain a label for a when considering its incident attacks.

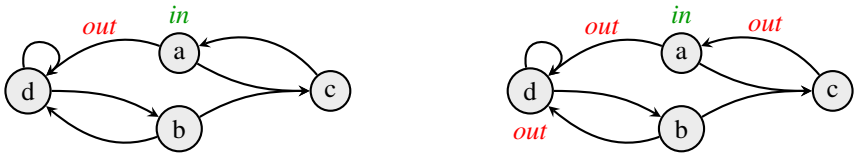
1. $a \in A$ must be labelled *in* if either (a) for all attacks $(S,a) \in R$, we have $\mu((S,a)) = out$; or (b) there is $(S,b) \in R$, $a \in S$, with $\mu((S,b)) = in$.
2. $a \in A$ must be labelled *out* if either (a) there is $(S,a) \in R$ with $\mu((S,a)) = in$; or (b) there is $(S,b) \in R$, $a \in S$, we have $\mu((S,b)) = out$ and for all $c \in S \setminus \{a\}$, $\mu(c) = in$.

One way to determine the acceptance status of an argument a is by considering attackers of a; e.g., rule 2.(a) states that if there is an attack (S,a) which is labelled *in* we have a is labelled *out* (otherwise the generated extension is not conflict-free). On the other hand, also outgoing attacks, i.e., attacks (S,b) where $a \in S$, can be used to determine the acceptance status of a in case certain requirements are met. Rule 2.(b) states that a must be labelled *out* if there is some attack (S,b) where $a \in S$ is the last unlabelled argument and every other argument is labelled *in*; otherwise, the attack (S,b) would be labelled *out* and thus the labelling would be invalid.

Algorithm 2 Stable-labellings(F)

Require: SETAF $F = (A,R)$, $\mu_\emptyset$ initial/empty partial labelling
Ensure: *Labs* corresponds to the set of stable labellings

```
1:  Labs ← ∅
2:  PartialLabs ← {μ_∅}
3:  while PartialLabs ≠ ∅ do
4:      μ ← PartialLabs.pull()
5:      while μ has unlabelled arguments do
6:          a ← pick an unlabelled argument
7:          μ', valid ← update(μ, a, out)
8:          if valid then PartialLabs.push(μ')
9:          μ, valid ← update(μ, a, in)
10:         if not valid then GOTO Line 3
11:     end while
12:     Labs ← Labs ∪ {μ}
13: end while
14: return Labs
```



(a) Partial labelling μ. (b) Updated labelling μ'.

Figure 1. SETAF F with partial labelling μ and updated labelling μ' (cf. Example 1).

Next consider an attack $(S,a) \in R$. From the definition of a stable labelling we can extract the following rules to obtain a label for (S,a) when just considering the arguments involved in the attack.

1. $(S,a) \in R$ must be labelled *in* if either (a) $\mu(a) = out$ and for all $(S',a) \in R$, $S' \neq S$, we have $\mu((S',a)) = out$; or (b) for all $b \in S$, we have $\mu(b) = in$.
2. $(S,a) \in R$ must be labelled *out* if either (a) $\mu(a) = in$; or (b) there is $b \in S$ with $\mu(b) = out$.

The label of an attack (S,a) can be directly inferred in case a is labeled *in*. When a is labeled *out* and (S,a) is the only attack on a or all other attacks attacking a are labelled *out* we can infer that (S,a) must be labelled *in*. Alternatively, one can consider the attacking set S: A label for the attack can be determined either if there is some argument $b \in S$ which is labelled *out* or in case every argument in S is labelled *in*.

Example 1. We consider the SETAF $F = (A,R)$ depicted in Figure 1 and a partial label μ where the attack $(\{a,d\},d)$ is initially labelled *out* since it is self-attacking and a is set to *in*. We can update argument d using the argument propagation rule 2.(b) and set $\mu'(d) = out$; moreover, we can label the attack $(\{c\},a)$ using the attack propagation rule 2.(a), that is, $\mu'((\{c\},a)) = out$ (cf. Figure 1b). ◊

Next we consider rules for arguments and attacks that have the same labels in all extensions and thus can be fixed in an initial phase. That is, instead of starting with an empty labelling, we compute the initial labelling $\mu_\emptyset$ by identifying arguments and attacks that can be labelled independently of their neighbors. Several initial rules can be defined as recent studies on collective attacks suggest (cf. [9]). In this work, we will consider the following initial rules to generate $\mu_\emptyset$:

- Each argument a with no incoming attacks must be labelled *in*.
- Each attack (S,a) with $a \in S$ must be labelled *out*.

Having set the initial labels, the propagation rules are applied as follows: The function update$(\mu,A,label)$ first updates $\mu(A)$ to *label*. Then if A is an argument we consider all incident attacks and test whether one could infer the labels of these attacks by one of the given propagation rules. Otherwise if A is an attack (S,a) we consider all arguments in $S \cup \{a\}$ and test whether one could infer the label of these attacks by one of the given propagation rules. For each argument or attack B where we have inferred new label $label'$ we apply the following: (a) If we have inferred different labels from different rules the labelling is inconsistent and update returns invalid. (b) If we have inferred a label that is incompatible with the current label in μ the labelling is inconsistent and update returns invalid. (c) If neither of the above applies we apply update$(\mu,B,label')$ and use the resulting labelling to proceed and returns invalid if update$(\mu,B,label')$ returns invalid.

Notice that the propagation rules ensure that once all arguments are labelled then either the last update computed is invalid or the labelling μ is a stable labelling. That is, first for each attack if all incident arguments are labeled then at least on of the rules applies and thus either we obtain that the labelling is invalid or that all attacks are labelled. Now for each argument if all the incident attacks are labeled then at least one of the rules applies and thus either we obtain an invalid labelling or that all arguments are labelled.

6. A Labelling-based Algorithm for Complete Semantics

In this section we present an algorithm for computing the complete labellings of a SETAF. The algorithm follows the same line as our algorithm for stable semantics, with some notable differences (see Algorithm 3). First, we have that complete labellings make use of all three labels $in, out, undec$, and in our algorithm we sometimes learn that we can exclude one of the labels. In order to encode this information in our partial labelling we allow two additional labels, *notin* indicating that the argument or attack must be labelled either *out* or *undec*, and *notout* indicating that the argument or attack must be labelled either *in* or *undec*. [2] We now have that certain labels are compatible, e.g., if an argument is labelled *notin* and we learn that it must be labelled *out* we can simply label it *out*, and others that are not, e.g., if an argument is labelled *undec* and we learn that it must be labelled *in* then the labelling cannot be extended to a complete labelling.

Algorithm 3 Complete-labellings(F)

Require: SETAF $F = (A,R)$, $\mu_\emptyset$ initial/empty partial labelling
Ensure: *Labs* corresponds to the set of complete labellings
1: $Labs \leftarrow \emptyset$
2: $PartialLabs \leftarrow \{\mu_\emptyset\}$
3: **while** $PartialLabs \neq \emptyset$ **do**
4: $\quad \mu \leftarrow PartialLabs.pull()$
5: $\quad$ **while** μ has has arguments that are unlabelled or labelled *notout* **do**
6: $\quad\quad a \leftarrow$ pick one of these arguments
7: $\quad\quad \mu', valid \leftarrow \text{update}(\mu, a, notin)$
8: $\quad\quad$ **if** $valid$ **then** $PartialLabs.push(\mu')$
9: $\quad\quad \mu, valid \leftarrow \text{update}(\mu, a, in)$
10: $\quad\quad$ **if** not $valid$ **then** GOTO Line 3
11: $\quad$ **end while**
12: $\quad Labs \leftarrow Labs \cup \{\mu\}$
13: **end while**
14: **return** Labs

Let us now consider the propagation rules that we can obtain for complete semantics. As for stable semantics initially we can label each unattacked argument with *in* and each attack (S,a) with $a \in S$ with *notin* and propagate these labels.

Now we consider propagation rules given a partial labelling μ. From the definition of complete labellings we can extract propagation rules for arguments and attacks to obtain new labels when just considering their incident attacks. Rules to infer *in*- and

[2] Notice that one could also introduce a label *notundec*, but such a label was of no additional value for our propagation rules.

out-labels generalize the rules for stable labellings by incorporating the additional labels *undec*, *notin*, *notout*; rules for *undec*-labels can be extracted from combining the argument constraints (respectively attack constraints) for complete labellings as discussed after Definition 7; rules for the labellings *notin* (*notout*) capture scenarios where either *out* or *undec* (respectively *in* or *undec*) can be learned, that is, they relax and combine the aforementioned *out* (respectively *in*) and *undec* labelling rules.

We will first consider propagation rules for an argument $a \in A$.

1. $a \in A$ must be labelled *in* if either (a) for all attacks $(S,a) \in R$, we have $\mu((S,a)) = out$; or (b) there is $(S,b) \in R$, $a \in S$, with $\mu((S,b)) = in$.
2. $a \in A$ must be labelled *out* if either (a) there is $(S,a) \in R$ with $\mu((S,a)) = in$; or (b) there is $(S,b) \in R$, $a \in S$, with $\mu((S,a)) = out$ and for all $c \in S \setminus \{a\}$, $\mu(c) \in \{in, undec, notout\}$.
3. $a \in A$ must be labelled *undec* if either (a) for all $(S,a) \in R$, $\mu((S,a)) \in \{out, undec, notin\}$ and there is $(S,a) \in R$ with $\mu((S,a)) = undec$; or (b) there is $(S,b) \in R$, $a \in S$, with $\mu((S,b)) = undec$ and for all $c \in S \setminus \{a\}$, $\mu(c) = in$.
4. $a \in A$ must be labelled *notin* if either (a) there is $(S,a) \in R$ with $\mu((S,a)) \in \{undec, notout\}$; or (b) there is $(S,b) \in R$, $a \in S$, with $\mu((S,b)) = notin$ and for all $c \in S \setminus \{a\}$, $\mu(c) = in$.
5. $a \in A$ must be labelled *notout* if either (a) for all $(S,a) \in R$, we have $\mu((S,a)) \in \{out, notin\}$; or (b) there is $(S,b) \in R$, $a \in S$, with $\mu((S,b)) \in \{undec, notout\}$.

Similar to the rules for stable labellings, one can infer the status of an argument a by either considering attacks on a or by considering attacks (S,b) where $a \in S$. We will briefly discuss rules which consider incoming attacks. Let us consider an argument a; in case there is some attack (S,a) which is labelled *in* or if all attacks (S,a) are labelled *out*, the propagation rules behave like those for stable labellings, that is, a is labelled *out* in the former case (cf. rule 2.(a)) and a is labelled *in* in the second case (cf. rule 1.(a)). In case we have that each attack (S,a) on a cannot be labelled *in* in the final label, i.e., each attack is either labelled *out*, *undec* or *notin*, then one can exclude the label *out*, that is, a must be labelled *notout* to avoid an invalid labelling. In case we furthermore know that there is some attack which is labelled *undec*, then we can conclude that a must be labelled *undec* (cf. rule 3.(a)). The former rule for *notout*-labellings is captured by rule 5.(a) where the last observation about *undec*-labellings is taken into account. Finally, a can be labelled *notin* if there is some incoming attack which is labelled *out* or *undec* in the final label (cf. rule 4.(a)).

Next we consider an attack $(S,a) \in R$, and obtain the following propagation rules.

1. $(S,a) \in R$ must be labelled *in* if either (a) $\mu(a) = out$ and for all $(S',a) \in R$, $S' \neq S$, we have $\mu((S',a)) \in \{out, undec, notin\}$; or (b) for all $b \in S$, we have $\mu(b) = in$.
2. $(S,a) \in R$ must be labelled *out* if either (a) $\mu(a) = in$; or (b) there is $b \in S$ with $\mu(b) = out$.
3. $(S,a) \in R$ must be labelled *undec* if either (a) $\mu(a) = undec$ and for all $(S',a) \in R$, $S' \neq S$, we have $\mu((S',a)) = out$; or (b) for all $c \in S$, we have $\mu(c) \in \{in, undec, notout\}$ and there is $c \in S$ with $\mu(c) = undec$.
4. $(S,a) \in R$ must be labelled *notin* if either (a) $\mu(a) \in \{undec, notout\}$; or (b) there is $b \in S$ with $\mu(b) \in \{undec, notin\}$.
5. $(S,a) \in R$ must be labelled *notout* if either (a) $\mu(a) = notin$ and for all $(S',a) \in R$, $S' \neq S$, we have $\mu((S',a)) = out$; or (b) for all $b \in S$, $\mu(b) \in \{in, notout\}$.

While for certain rules it suffice to consider either the label of argument a or the labels od the set S to determine new labels, in general we have that attack rules for complete labellings require additional information about further attacks (S',a) to infer the label of (S,a) via the target a (cf. rules 1.(a), 3.(a), 5.(a)). As an example, assume that a is labelled *out*. Recall that for stable labellings, an attack (S,a) must be labeled *in* in case a is labelled *out*. The situation gets more complex for complete labellings since they allow for *undec*-labellings: Here, the label of the attack (S,a) also depends on other attacks (S',a), $S' \neq S$, i.e., only if every (S',a) is not labelled *in* in the final labelling (that is, each (S',a) is labelled *out*, *undec* or *notin* in the partial labelling) we can conclude that (S,a) must be labelled *in* (cf. rule 1.(a)).

The function update$(\mu,A,label)$ works as for stable semantics but applies the above propagation rules. Moreover, when inferring several labellings it only yields invalid if they are not compatible and otherwise returns the most specific label. That is, the *in* label is only compatible with *notout* (combining them results *in*); the *out* label is only compatible with *notin* (combining them results *out*); the *undec* label is compatible with both *notin* and *notout* (combining them results *undec* in both cases); the *notin* label is compatible with *undec*, *out* and *notout* (combining *notout* and *notin* results *undec*); and the *notout* label is compatible with *in*, *out* and *notin*. Again the propagation rules ensure that once all arguments are labelled also all attacks are labelled and either the last update computes invalid or the resulting labelling μ is a complete labelling.

Finally, notice that the discussed propagation rules can be applied to all semantics that propose subsets of the complete extensions and Algorithm 3 can extended to compute preferred or semi-stable extensions in the same way as for Dung AFs.

7. Conclusions

In this work we introduced argument-attack labellings of SETAFs and show their potential for computational purposes by provided a linear time algorithm for grounded semantics, and a systematic analysis of labelling propagation rules for stable and complete-based semantics. As for Dung AFs our algorithms for stable and complete semantics branch only on the arguments and not on attacks and thus the worst case running time is exponentially in the number of arguments but only polynomially depends on the number and size of the attacks. That is, we can bound the running time by $O(2^{|A|} \cdot poly(\|F\|))$ where the actual polynomial will depend on the choice and implementation of the propagation rules. Notice, that these exponential-time algorithms reflect the facts that (a) a SETAF might have exponentially many stable, complete respectively, extensions and (b) there are reasoning problems for stable and complete semantics which are NP or coNP-complete [8].

We identify several directions for future research: (a) tailored data-structures for labelling propagation for stable and complete labellings, like the counters for grounded and the *mout* [16] label for AFs; (b) extending the approach to other semantics based on conflict-free labellings; (c) once tailored data-structures have been settled, implementations and an empirical comparison with existing systems would be of interest; and (d) in the light of the previous point it is crucial to establish standardized data formats and benchmarks sets for SETAFs.

Acknowledgements. The authors want to thank the reviewers for their valuable feedback which helped to improve the presentation of the results.

This research has been funded by the Austrian Science Fund (FWF): P30168 and W1255-N23.

References

[1] Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors. *Handbook of Formal Argumentation*. 2018.

[2] Gerhard Brewka, Sylwia Polberg, and Stefan Woltran. Generalizations of Dung frameworks and their role in formal argumentation. *IEEE Intell. Syst.*, 29(1):30–38, 2014.

[3] Federico Cerutti, Sarah A. Gaggl, Matthias Thimm, and Johannes P. Wallner. Foundations of implementations for formal argumentation. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 15, pages 688–767. 2018.

[4] Federico Cerutti, Mauro Vallati, and Massimiliano Giacomin. Where are we now? state of the art and future trends of solvers for hard argumentation problems. In *Proc. COMMA*, volume 287 of *FAIA*, pages 207–218, 2016.

[5] Günther Charwat, Wolfgang Dvořák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation - A survey. *Artif. Intell.*, 220:28–63, 2015.

[6] Sylvie Doutre and Jérôme Mengin. Preferred extensions of argumentation frameworks: Query answering and computation. In *Proc. IJCAR*, volume 2083 of *LNCS*, pages 272–288, 2001.

[7] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.

[8] Wolfgang Dvořák, Alexander Greßler, and Stefan Woltran. Evaluating SETAFs via answer-set programming. In *Proc. SAFA*, volume 2171 of *CEUR Workshop Proceedings*, pages 10–21, 2018.

[9] Wolfgang Dvořák, Anna Rapberger, and Stefan Woltran. Strong equivalence for argumentation frameworks with collective attacks. In *Proc. KI*, volume 11793 of *LNCS*, pages 131–145, 2019.

[10] Giorgos Flouris and Antonis Bikakis. A comprehensive study of argumentation frameworks with sets of attacking arguments. *Int. J. Approx. Reason.*, 109, 03 2019.

[11] Sarah Alice Gaggl, Thomas Linsbichler, Marco Maratea, and Stefan Woltran. Design and results of the second international competition on computational models of argumentation. *Artif. Intell.*, 279, 2020.

[12] Sanjay Modgil and Martin Caminada. Proof theories and algorithms for abstract argumentation frameworks. In Iyad Rahwan and Guillermo Simari, editors, *Argumentation in Artificial Intelligence*, pages 105–132. 2009.

[13] Søren Holbech Nielsen and Simon Parsons. Computing preferred extensions for argumentation systems with sets of attacking arguments. In *Proc. COMMA*, volume 144 of *FAIA*, pages 97–108, 2006.

[14] Søren Holbech Nielsen and Simon Parsons. A generalization of Dung's abstract framework for argumentation: Arguing with sets of attacking arguments. In *Proc. ArgMAS*, volume 4766 of *LNCS*, pages 54–73, 2006.

[15] Samer Nofal, Katie Atkinson, and Paul E. Dunne. Algorithms for argumentation semantics: Labeling attacks as a generalization of labeling arguments. *J. Artif. Intell. Res.*, 49:635–668, 2014.

[16] Samer Nofal, Katie Atkinson, and Paul E. Dunne. Algorithms for decision problems in argument systems under preferred semantics. *Artif. Intell.*, 207:23–51, 2014.

[17] Samer Nofal, Katie Atkinson, and Paul E Dunne. Computing grounded extensions of abstract argumentation frameworks. *The Computer Journal*, 11 2019.

[18] Samer Nofal, Katie Atkinson, and Paul E. Dunne. On checking skeptical and ideal admissibility in abstract argumentation frameworks. *Inf. Process. Lett.*, 148:7–12, 2019.

[19] Matthias Thimm and Serena Villata. The first international competition on computational models of argumentation: Results and analysis. *Artif. Intell.*, 252:267–294, 2017.

[20] Bart Verheij. A labeling approach to the computation of credulous acceptance in argumentation. In *Proc. IJCAI*, pages 623–628, 2007.