Parameterized Splitting: A Simple Modification-Based Approach*

Ringo Baumann[1], Gerhard Brewka[1], Wolfgang Dvořák[2], and Stefan Woltran[2]

[1] Leipzig University, Informatics Institute, Postfach 100920, 04009 Leipzig, Germany, {baumann,brewka}@informatik.uni-leipzig.de

[2] Vienna University of Technology, Institute of Information Systems, Favoritenstraße 9–11, A-1040 Vienna, Austria, {dvorak,woltran}@dbai.tuwien.ac.at

Abstract. In an important and much cited paper Vladimir Lifschitz and Hudson Turner have shown how, under certain conditions, logic programs under answer set semantics can be split into two disjoint parts, a "bottom" part and a "top" part. The bottom part can be evaluated independently of the top part. Results of the evaluation, i.e., answer sets of the bottom part, are then used to simplify the top part. To obtain answer sets of the original program one simply has to combine an answer set of the simplified top part with the answer set which was used to simplify this part. Similar splitting results were later proven for other nonmonotonic formalisms and also Dung style argumentation frameworks.

In this paper we show how the conditions under which splitting is possible can be relaxed. The main idea is to modify also the bottom part before the evaluation takes place. Additional atoms are used to encode conditions on answer sets of the top part that need to be fulfilled. This way we can split in cases where proper splitting is not possible. We demonstrate this idea for argumentation frameworks and logic programs.

1 Introduction

In an important and much cited paper Vladimir Lifschitz and Hudson Turner have shown how, under certain conditions, logic programs under answer set semantics can be split into two disjoint parts, a "bottom" part and a "top" part. The bottom part can be evaluated independently of the top part. Results of the evaluation, i.e., answer sets of the bottom part, are then used to simplify the top part. To obtain answer sets of the original program one simply has to combine an answer set of the simplified top part with the answer set which was used to simplify this part.

Splitting is a fundamental principle and has been investigated for several other non-monotonic formalisms, including answer-set programming [13, 7] default logic [15] and most recently argumentation [3, 4, 12]. It has important implications, both from the theoretical and from the practical point of view. On the the theoretical side, splitting allows for simplification of proofs showing properties of a particular formalism. On the practical side, splitting concepts are useful for solving since the possibility to compute

* This work has been funded by Vienna Science and Technology Fund (WWTF) through project ICT08-028.

solutions of parts of a program, and then to combine these solutions in a simple way, has obvious computational advantages.

In this paper we present a simple generalization of the "classical" splitting results for argumentation and logic programming. Generalizations for the latter have been investigated in depth in [11]. In fact, Janhunen et al. describe an entire rather impressive module theory for logic programs based on modules consisting of rules together with input, output and hidden variables. To a certain extent some of our results can be viewed as being "implicit" already in that paper. Nevertheless, we believe that the constructions we describe here complement the abstract theory in a useful manner. In particular, we present a generalized approach to splitting which is based on simple modifications of the components that are obtained through splitting. The general idea is to add certain constructs (new atoms in case of logic programs, new arguments in case of argumentation frameworks) representing meta-information to the first component. Solutions computed for the augmented first component thus will contain meta-information. This information is used, together with the standard propagation of results known from classical splitting, to modify the second component. The aim is to guarantee that solutions of the second part match the outcome for the first component. The main advantage of this approach is practical: standard solvers can directly be used for the relevant computations.

The outline of the paper is as follows. In Sect. 2 we provide the necessary background on classical splitting for logic programs and argumentation frameworks. Sect. 3 introduces quasi-splittings for argumentation frameworks and shows how stable extensions of AFs can be computed based on them. Sect. 4 defines quasi-splittings for normal logic programs and treats the computation of answer sets in a similar spirit. Finally, in Sect. 5 we propose a general theory of splitting abstracting away from concrete formalisms and take a closer look at computational issues concerned with quasi-splitting. We conclude the paper with a brief summary and an outlook on future work.

2 Background

In this section we give the necessary background on argumentation frameworks and logic programs.

Argumentation Frameworks. Abstract argumentation frameworks (AFs) have been introduced by Dung [6] and are widely used in formal argumentation. They treat arguments as atomic entities, abstracting away from their structure and content, and focus entirely on conflict resolution. Conflicts among arguments are represented via an attack relation. Different semantics have been defined for AFs. Each of them singles out sets of arguments which represent reasonable positions based on varying principles. We start with some notation.

Definition 1. *An* argumentation framework (AF) *is a pair $F = (A, R)$ where A is a finite set of arguments and $R \subseteq A \times A$ is the attack relation. For a given AF $F = (A, R)$ we use $A(F)$ to denote the set A of its arguments and $R(F)$ to denote its attack relation R. We sometimes write $(S, b) \in R$ in case there exists an $a \in S$, such that $(a, b) \in R$; likewise we use $(a, S) \in R$ and $(S, S') \in R$.*

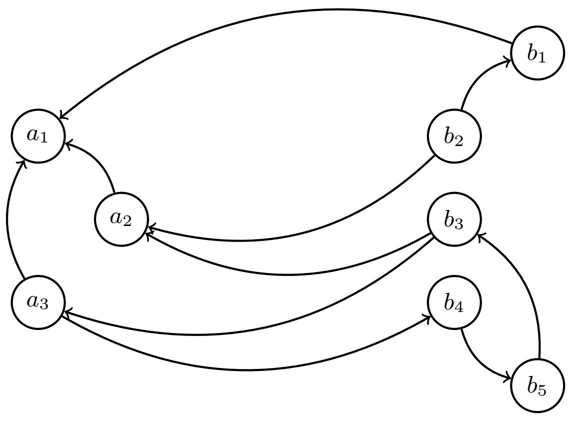

Fig. 1. An argumentation framework which is our running example.

Given $F = (A, R)$ and a set $S \subseteq A$, we write $F|_S$ to denote the AF $(S, R \cap (S \times S))$ induced by S. Furthermore, $S_R^{\oplus} = \{b \mid (S, b) \in R\}$ and $S_R^{\ominus} = \{a \mid (a, S) \in R\}$; as well $S_R^{+} = S \cup S_R^{\oplus}$ and $S_R^{-} = S \cup S_R^{\ominus}$.

Several semantics for argumentation frameworks have been studied in the literature, see e.g. [1, 2]; we focus here on the stable semantics as introduced already in [6], not the least due to its close connection to the ASP semantics. Intuitively, a set S of arguments is a stable extension if no argument in S attacks another argument in S, and moreover all arguments not in S are attacked by an element of S.

Definition 2. *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is* conflict-free (in F)*, denoted as $S \in cf(F)$, iff there are no $a, b \in S$, such that $(a, b) \in R$. Moreover, S is called a stable extension of F iff $S \in cf(F)$ and $S_R^{+} = A$. The set of all stable extensions of F is given by $stb(F)$.*

Example 1. Figure 1 shows an example AF with stable extensions $\{a_3, b_2, b_5\}$ and $\{a_1, b_2, b_3, b_4\}$.

Logic Programs. We restrict the discussion in this paper to normal logic programs. Such programs are sets of rules of the form

$$a \leftarrow b_1, \ldots, b_m, \text{ not } c_1, \ldots, \text{ not } c_n \qquad (1)$$

where a and all b_i's and c_j's are atoms. Intuitively, the rule is a justification to "establish" or "derive" that a (the so called *head*) is true, if all default literals to the right of $\leftarrow$ (the so called *body*) are true in the following sense: a non-negated atom b_i is true if it has a derivation, a negated one, not c_j, is true if c_j does not have one. Variables can appear in rules, however they are just convenient abbreviations for the collection of their ground instantiations.

The semantics of (ground) normal logic programs [8, 9] is defined in terms of answer sets, also called stable models for this class of programs. Programs without negation in the bodies have a unique answer set, namely the smallest set of atoms closed under the rules. Equivalently, this set can be characterized as the least model of the rules, reading $\leftarrow$ as classical implication and the comma in the body as conjunction.

For programs with negation in the body, one needs to guess a candidate set of atoms S and then verifies the choice. This is achieved by evaluating negation with respect to S and checking whether the "reduced" negation-free program corresponding to this evaluation has S as answer set. If this is the case, it is guaranteed that all applicable rules were applied, and that each atom in S has a valid derivation based on appropriate rules. Here is the formal definition:

Definition 3. *Let P be a normal logic program, S a set of atoms. The Gelfond/Lifschitz-reduct (GL-reduct) of P and S is the negation-free program P^S obtained from P by*

1. *deleting all rules $r \in P$ with not c_j in the body for some $c_j \in S$,*
2. *deleting all negated atoms from the remaining rules.*

S is an answer set of P iff S is the answer set of P^S. We denote the collection of answer sets of a program P by $AS(P)$.

For convenience we will use rules without head (constraints) of the form

$$\leftarrow b_1, \ldots, b_m, \text{ not } c_1, \ldots, \text{ not } c_n \tag{2}$$

as abbreviation for

$$f \leftarrow \text{ not } f, b_1, \ldots, b_m, \text{ not } c_1, \ldots, \text{ not } c_n \tag{3}$$

where f is an atom not appearing anywhere else in the program. Adding rule 2 to a program has the effect of eliminating those answer sets of the original program which contain all of the b_is and none of the c_js.

For other types of programs and an introduction to answer set programming, a problem solving paradigm based on the notion of answer sets, the reader is referred to [5].

3 Quasi-splittings for argumentation frameworks

In this section we develop our approach for argumentation frameworks under stable semantics. We start with the definition of quasi-splittings.

Definition 4. *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is a* quasi-splitting *of F. Moreover, let $\bar{S} = A \setminus S$, $R^S_{\rightarrow} = R \cap (S \times \bar{S})$ and $R^S_{\leftarrow} = R \cap (\bar{S} \times S)$. Then, S is called*

- k-splitting *of F, if $|R^S_{\leftarrow}| = k$;*
- *(proper)* splitting *of F, if $|R^S_{\leftarrow}| = 0$.*

Note that above definition also allows for trivial splittings, i.e. $S = \emptyset$ or $S = A$. In what follows, we often tacitly assume quasi-splittings to be non-trivial.

A quasi-splitting S of F induces two sub-frameworks of F, namely $F_1^S = F|_S$ and $F_2^S = F|_{\bar{S}}$, together with the sets of links $R_{\rightarrow}^S$ and $R_{\leftarrow}^S$ connecting the sub-frameworks in the two possible directions.[3] All these components are implicitly defined by S. For this reason specifying this set is sufficient.

Our goal is to use - under the assumption that k is reasonably small - a k-splitting to potentially reduce the effort needed to compute the stable extensions of F. The case $k = 0$, that is proper splitting, was considered by Baumann [3][4].

The basic idea is as follows. We first find a stable extension of F_1^S. However, we have to take into account that one of the elements of S may be attacked by an argument of F_2^S. For this reason we must provide, for each argument a in F_1^S attacked by an argument in F_2^S, the possibility *to assume it is attacked*. To this end we add for a a new argument $att(a)$ such that a and $att(a)$ attack each other. Now we may choose to include the new argument in an extension which corresponds to the assumption that a is attacked. Later, when extensions of F_2^S are computed, we need to check whether the assumptions we made actually are satisfied. Only if they are, we can safely combine the extensions of the sub-frameworks we have found.

Definition 5. *Let $F = (A, R)$ be an AF, S a* quasi-splitting *of F. A conditional extension of F_1^S is a stable extension of the modified AF $[F_1^S] = (A_S, R_S)$ where*

- $A_S = S \cup \{att(a) \mid a \in A^+_{R_{\leftarrow}^S}\}$, *and*
- $R_S = (R \cap (S \times S)) \cup \{(att(a), a), (a, att(a)) \mid a \in A^+_{R_{\leftarrow}^S}\}$.

In other words, $[F_1^S]$ is obtained from F_1^S by adding a copy $att(a)$ for each argument a attacked from F_2^S, and providing for each such a a mutual attack between a and $att(a)$. For a k-splitting we thus add k nodes and $2k$ links to F_1^S - a tolerable augmentation for reasonably small k. For a proper splitting S, note that $[F_1^S] = F_1^S$.

Example 2. Consider $S = \{a_1, a_2, a_3\}$, a splitting of our example AF F, and the resulting frameworks $[F_1^S]$ and F_2^S as depicted in Figure 2. We have $R_{\rightarrow}^S = \{(a_3, b_4)\}$ and $R_{\leftarrow}^S = \{(b_1, a_1), (b_2, a_2), (b_3, a_2), (b_3, a_3)\}$. The set of the grey highlighted arguments $E = \{att(a_1), att(a_2), a_3\}$ is a conditional extension of F_1^S, i.e. a stable extension of the modified framework $[F_1^S]$. Further stable extensions of $[F_1^S]$ are $E_1 = \{a_1, att(a_2), att(a_3)\}$, $E_2 = \{att(a_1), a_2, att(a_3)\}$, $E_3 = \{att(a_1), a_2, a_3\}$ and $E_4 = \{att(a_1), att(a_2), att(a_3)\}$.

Each conditional extension E of F_1^S may contain meta-information in the form of $att(x)$ elements. This information will later be disregarded, but is important to verify the assumptions extensions of F_2^S need to fulfill so that E can be augmented to an

[3] Unlike Lifschitz and Turner we enumerate the sub-frameworks rather than calling them bottom and top framework. For quasi-splittings there does not seem to be a clear sense in which one is below the other.

[4] Note that Baumann used a more involved definition of a splitting. The definition we use here is not only simpler but also closer to the one used by Lifschitz and Turner.

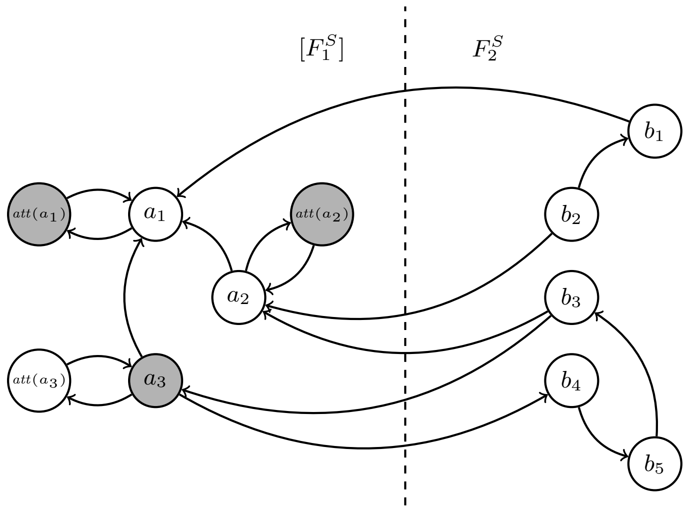

Fig. 2. A splitting of our example framework.

extension of the entire argumentation framework F. As in [3], E will be used to modify F_2^S accordingly, thus propagating effects of elements in E on F_2^S. In the generalized case we also need to take the meta-information in E into account to make sure the assumptions we made are valid. In particular,

- if $att(a)$ is in E yet a is not attacked by another element in $E \cap S$, then we know that a must be externally attacked from an element of F_2^S. In other words, we are only interested in extensions of F_2^S which contain at least one attacker of a.
- if b is in E yet it is attacked by some argument in F_2^S, then we are only interested in extensions of F_2^S not containing any of the attackers of b.

Before we turn these ideas into a definition we present a useful lemma which helps to understand our definition.

Lemma 1. *Let $F = (A, R)$ be an AF. Let further B and $C_1, \ldots, C_n$ be sets s.t. $B, C_1, \ldots, C_n \subseteq A$, and $D = \{d_1, \ldots, d_n\}$ s.t. $D \cap A = \emptyset$. The stable extensions of the AF*

$$F' = (A \cup D, R \cup \{(b, b) \mid b \in B \textit{ or } b \in D\} \cup \{(c, d_j) \mid c \in C_j, 1 \leq j \leq n\})$$

are exactly the stable extensions E of F containing no element of B and at least one element of every C_i, i.e. $C_i \cap E \neq \emptyset$ for every $i \in \{1, \ldots, n\}$.

Proof. Let $E \in stb(F)$, s.t. E does not contain an element of B (+) and $C_i \cap E \neq \emptyset$ for every $i \in \{1, \ldots, n\}$ (*). We observe $E \in cf(F')$ because of $E \subseteq A$ and (+), and furthermore, $E^+_{R(F')} = E^+_{R(F)} \cup D = A \cup D$ because of (*). Thus, $E \in stb(F')$.

Assume now $E \in stb(F')$. We observe that $E \cap B = \emptyset$ since conflict-freeness has to be fulfilled. Furthermore, $C_i \cap E \neq \emptyset$ for every $i \in \{1, \ldots, n\}$ has to hold because $E^+_{R(F')} = A \cup \{d_1, \ldots, d_n\}$ and only arguments in C_i attack the argument d_i

by construction. Obviously, $E \subseteq A$ since the elements of D are self-attacking. Furthermore, $E \in cf(F)$ because $R(F) \subseteq R(F')$. Consider now the attack-relation $R(F')$. We obtain $E^+_{R(F)} = E^+_{R(F')} \setminus D = (A \cup D) \setminus D = A$ which proves $E \in stb(F)$. $\square$

Based on the lemma we can now define the modification of F_2^S that is needed to compute those extensions which comply with a conditional extension E, capturing also the assumptions made in E. First, we can eliminate all arguments attacked by an element of E. This step corresponds to the usual propagation needed for proper splittings as well. In addition, we make sure that only those extensions of the resulting framework are generated which (1) contain an attacker for all externally attacked nodes of S, and (2) do not contain an attacker for any element in E. For this purpose the techniques of the lemma are applied.

Definition 6. *Let $F = (A, R)$ be an AF, S a* quasi-splitting *of F, and let E be a conditional extension of F_1^S. Furthermore, let*

$$EA(S, E) = \{a \in S \setminus E \mid a \notin (S \cap E)^{\oplus}_{R}\}$$

denote the set of arguments from F_1^S not contained in E because they are externally attacked. An (E, S)-match of F is a stable extension of the AF $[F_2^S]_E = (A', R')$ where

- *$A' = (\bar{S} \setminus E^+_{R^S_{\rightarrow}}) \cup \{in(a) \mid a \in EA(S, E)\}$, and*
- *$R' = (R \cap (A' \times A')) \cup \{(in(a), in(a)), (b, in(a)) \mid a \in EA(S, E), (b, a) \in R^S_{\leftarrow}\} \cup \{(c, c) \mid (c, E) \in R^S_{\leftarrow}\}$.*

In other words, we take the framework F_2^S and modify it w.r.t. to a given conditional extension E of F_1^S. To this end, we remove those arguments from F_2^S which are attacked by E via $R^S_{\rightarrow}$ but we make a copy of each argument a in F_1^S externally attacked by F_2^S via $R^S_{\leftarrow}$. These additional self-attacking arguments $in(a)$ are used to represent the forbidden situation where an externally attacked argument a actually remains unattacked. Finally, we exclude those arguments in F_2^S from potential extensions which attack an argument in E located in F_1^S; these are the self-loops (c, c) for arguments s with $(c, E) \in R^S_{\leftarrow}$. Again the size of the modification is small whenever k is small: we add at most k nodes and $2k$ links to F_2^S.

Example 3. We continue our running example with Figure 3. On the right-hand side we have the modification of F_2^S w.r.t. the conditional extension $E = \{att(a_1), att(a_2), a_3\}$, i.e. the AF $[F_2^S]_E$. Observe that $EA(S, E) = \{a_2\}$ because a_2 is not an element of E and furthermore, it is not attacked by an argument in $E \cap S = \{a_3\}$. Hence, we have to add a self-attacking node $in(a_2)$ to F_2^S which is attacked by the attackers of a_2, namely the arguments b_2 and b_3. The argument a_3 (which belongs to the extension E) is attacked by the argument b_3 and attacks the argument b_4. Hence, we have to add a self-loop for b_3 and further, we have to delete b_4 and its corresponding attacks. The set of the light-grey highlighted arguments $E' = \{b_2, b_5\}$ is an (E, S)-match of F, i.e. a stable extension of $[F_2^S]_E$; in fact, it is the only (E, S)-match of F. Recall that $(E \cap S) \cup E' = \{a_3, b_2, b_5\}$ is a stable extension of the initial AF F; we will below

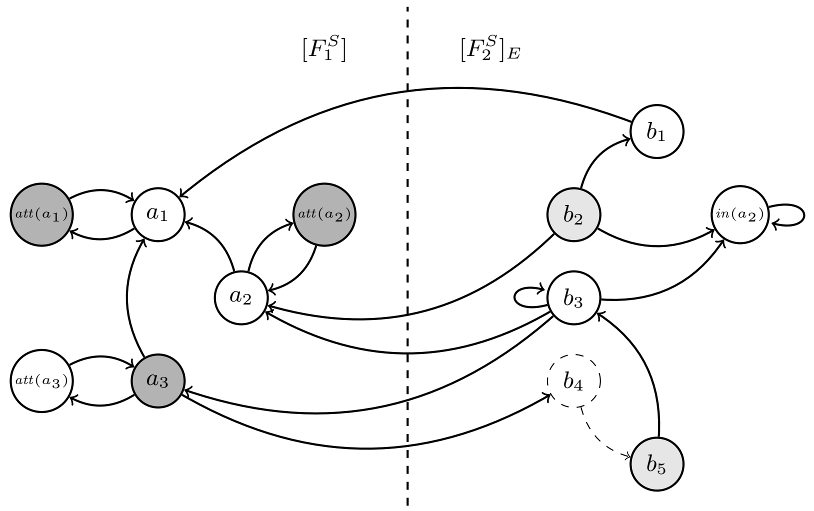

Fig. 3. Propagating conditional extensions to the second AF.

show this result in general. One can further check that $\{b_2, b_3, b_4\}$ is an (E_1, S)-match of F with E_1 as given in the previous example. On the other hand, for the remaining conditional extensions of F_1^S, no corresponding matches exist. Take, for instance, $E_2 = \{att(a_1), a_2, att(a_3)\}$; here we have to put self-loops for b_2 and b_3, but b_2 remains unattacked in $[F_2^S]_{E_2}$. Thus no stable extension for $[F_2^S]_{E_2}$ exists.

Theorem 1. *Let $F = (A, R)$ be an AF and let S be a* quasi-splitting *of F.*

1. *If E is a conditional extension of F_1^S and E' an (E, S)-match of F, then $(E \cap S) \cup E'$ is a stable extension of F.*
2. *If H is an extension of F, then there is a set $X \subseteq \{att(a) \mid a \in A^+_{R^S_{\leftarrow}}\}$ such that $E = (H \cap S) \cup X$ is a conditional extension of F_1^S and $H \cap \bar{S}$ is an (E, S)-match of F.*

Proof. ad 1. First we will show that $(E \cap S) \cup E' \in cf(F)$. Given that E is a conditional extension of F_1^S we deduce $E \in cf([F_1^S])$. Consequently, $E \cap S \in cf([F_1^S])$ (less arguments) and thus, $E \cap S \in cf(F_1^S)$ (less attacks). Finally, $E \cap S \in cf(F)$ since $(F_1^S)|_S = F|_S$ holds. Let E' be an (E, S)-match of F, i.e. $E' \in stb([F_2^S]_E)$. According to Lemma 1, $E' \in stb(F')$ where $F' = (\bar{S} \setminus E^{\oplus}_{R^S_{\rightarrow}}, R(F) \setminus (E^{\oplus}_{R^S_{\rightarrow}}, E^{\oplus}_{R^S_{\rightarrow}}))$. Thus, $E' \in cf(F)$. Obviously, $(E \cap S, E') \notin R(F)$ since $E \cap S \subseteq S$ and $E' \subseteq \bar{S} \setminus E^{\oplus}_{R^S_{\rightarrow}}$. Assume now $(E', E \cap S) \in R(F)$. This means, there are arguments $e' \in E'$ and $e \in E \cap S$, s.t. $(e', e) \in R$. This contradicts the conflict-freeness of E' in $[F_2^S]_E$ because $R([F_2^S]_E)$ contains the set $\{(c, c) \mid (c, E) \in R^S_{\leftarrow}\}$. Thus, $(E', E \cap S) \notin R(F)$ and $(E \cap S) \cup E' \in cf(F)$ is shown.

We now show that $((E \cap S) \cup E')^+_{R(F)} = S \cup \bar{S} = A$. Let us consider an argument s, s.t. $s \notin ((E \cap S) \cup E')^+_{R(F)}$. Assume $s \in S$. Consequently, $s \in EA(S, E)$. Since

E' is an (E,S)-match of F, by Lemma 1, $E \in stb(F')$ such that $(E, in(s)) \in R(F')$. By definition of F' we have $R(F') \subseteq R(F)$ and thus $(E, in(s)) \in R(F)$ contradicting the assumption. Assume now $s \in \bar{S}$. Obviously, $s \in \bar{S} \setminus E^{\oplus}_{R^S_{\rightarrow}}$. Since E' is an (E,S)-match of F we deduce by Lemma 1 that E' is a stable extension of F' as defined above. Thus, $s \in (E')^{+}_{R(F)}$ contradicting the assumption. Altogether, we have shown that $(E \cap S) \cup E'$ is a stable extension of F.

ad 2. Let $(H \cap S)^{+}_{R(F)} \dot{\cup} B = S$. Since $H \in stb(F)$ is assumed it follows $B \subseteq (H \cap \bar{S})^{+}_{F}$. This means, $B \subseteq A^{+}_{R^S_{\leftarrow}}$. Consider now $X = \{att(b) \mid b \in B\}$. It can be easily seen that $E = (H \cap S) \cup X$ is a conditional extension of $[F^S_1]$. Note that $B = EA(S, E)$. Since $H \in stb(F)$ it follows $H \cap \bar{S} \in stb(F')$ where F' is as above. Furthermore, there is no argument $c \in H \cap \bar{S}$, s.t. $(c, d) \in R^S_{\leftarrow}$ with $d \in E$. Remember that $B = EA(S, E)$. Hence, for every $b \in B$, $(H \cap \bar{S}) \cap \{b\}^{\oplus}_{R^S_{\leftarrow}} \neq \emptyset$, since $H \in stb(F)$. Thus, Lemma 1 is applicable which implies that $H \cap \bar{S}$ is an (E,S)-match of F. $\square$

4 Quasi-splittings for logic programs

We restrict ourselves here to normal logic programs. A splitting S can then be defined as a set of atoms dividing a program P into two disjoint subprograms P^S_1 and P^S_2 such that no head of a rule in P^S_2 appears anywhere in P^S_1.

One way to visualize this is via the dependency graph of P. The nodes of the dependency graph are the atoms in P. In addition, there are two kinds of links, positive and negative ones: whenever b appears positively (negatively) in a rule with head c, then there is a positive (negative) link from b to c in the dependency graph. A (proper) splitting S now is a set of atoms such that the dependency graph has no links - positive or negative - to an atom in S from any atom outside S.[5]

Now let us consider the situation where a (small) number k of atoms in the heads of rules in P^S_2 appear negatively in bodies of P^S_1. This means that the dependency graph of the program has a small number of negative links *pointing in the wrong direction*. As we will see methods similar to those we used for argumentation can be applied here as well.

In the following $head(r)$ denotes the head of rule r, $At(r)$ the atoms appearing in r and $pos(r)$ (respectively $neg(r)$) the positive (respectively negative) atoms in the body of r. We also write $head(P)$ for the set $\{head(r) \mid r \in P\}$ and $At(P)$ for $\{At(r) \mid r \in P\}$.

Definition 7. *Let P be a normal logic program. A set $S \subseteq At(P)$ is a* quasi-splitting *of P if, for each rule $r \in P$,*

- *$head(r) \in S$ implies $pos(r) \subseteq S$.*

Let $\bar{S} = At(P) \setminus S$ and $V_S = \{c \in \bar{S} \mid r \in P, head(r) \in S, c \in neg(r)\}$. S is called

[5] An argumentation framework F can be represented as a logic program P as follows: for each argument a with attackers $b_1, \ldots, b_n$, P contains the rule $a \leftarrow$ not $b_1, \ldots,$ not b_n. Now the graph of F corresponds exactly to the dependency graph of P with all links negative.

– k-splitting *of P, if* $|V_S| = k$;
– *(proper)* splitting *of P, if* $|V_S| = 0$.

Analogously to AFs where a splitting induces two disjoint sub-frameworks, here the set S induces two disjoint sub-programs, P_1^S having heads in S and P_2^S having heads in $\bar{S}$. Whenever $|V_S| \neq 0$, there are some heads in P_2^S which may appear in bodies of P_1^S, but only in negated form. This is not allowed in standard splittings which thus correspond to 0-splittings.

Note an important distinction between splittings for AFs and for logic programs: an arbitrary subset of arguments is an AF splitting, whereas a splitting S for a program P needs to fulfill the additional requirement that for each rule $r \in P$ with its head contained in S, also the entire positive body stems from S.

Example 4. Consider the following simple program

(1) $a \leftarrow$ not b
(2) $b \leftarrow$ not a
(3) $c \leftarrow a$

The program does not have a (nontrivial) classical splitting. However, it possesses the quasi-splitting $S = \{a, c\}$ (together with the complementary quasi-splitting $\bar{S} = \{b\}$). P_1^S consists of rules (1) and (3), P_2^S of rule (2). It is easily verified that $V_S = \{b\}$.

For the computation of answer sets we proceed in the same spirit as before by adding rules to P_1^S which allow answer sets to contain meta-information about the assumptions which need to hold for an answer set. We introduce atoms $ndr(b)$ to express the assumption that b will be underivable from P_2^S.

Definition 8. *Let P be a normal logic program, let S be a quasi-splitting of P and let P_1^S, respectively P_2^S, be the sets of rules in P with heads in S, respectively in $\bar{S}$. Moreover, let V be the set of atoms in $\bar{S}$ appearing negatively in P_1^S.*

$[P_1^S]$ is the program obtained from P_1^S by adding, for each $b \in V_S$, the following two rules:

$b \leftarrow not\ ndr(b)$
$ndr(b) \leftarrow not\ b$

E is called conditional answer set of P_1^S iff E is an extension of $[P_1^S]$.

Intuitively, $ndr(b)$ represents the assumption that b is not derivable from P_2^S (since $b \in V_S$ it cannot be derivable from P_1^S anyway). The additional rules allow us to assume b - a condition which we later need to verify. The construction is similar to the one we used for AFs where it was possible to assume that an argument is attacked.

Now, given an answer set E of $[P_1^S]$, we use E to modify P_2^S in the following way. We first use E to simplify P_2^S in exactly the same way this was done by Lifschitz and Turner: we replace atoms in S appearing positively (negatively) in rule bodies of P_2^S with $true$ whenever they are (are not) contained in E. Moreover, we delete each rule with a positive (negative) occurrence of an S-atom in the body which is not (which is) in E. We call the resulting program E-evaluation of P_2^S. Next we add adequate rules (constraints) which guarantee that answer sets generated by the modification of P_2^S match the conditions expressed in the meta-atoms of E. This is captured in the following definition:

Definition 9. *Let P, S, and P_2^S be as in Def. 8, and let E be a conditional answer set of P_1^S. Let $[P_2^S]_E$, the E-modification of P_2^S, be obtained from the E-evaluation of P_2^S by adding the following k rules*

$$\{\leftarrow \mathit{not}\ b \mid b \in E \cap V_S\} \cup \{\leftarrow b \mid ndr(b) \in E\}.$$

E' is called (E, S)-match iff E' is an answer set of $[P_2^S]_E$.

Now we can verify that answer sets of P can be obtained by computing an answer set E_1 of $[P_1^S]$ and an answer set E_2 of the E_1-modification of P_2^S (we just have to eliminate the meta-atoms in E_1).

Theorem 2. *Let P be a normal logic program and let S be a quasi-splitting of P.*

1. *If E is a conditional answer set of P_1^S and F an (E, S)-match, then $(E \cap S) \cup F$ is an answer set of P.*
2. *If H is an answer set of P, then there is a set $X \subseteq \{ndr(a) \mid a \in V_S\}$ such that $E = (H \cap S) \cup X$ is a conditional answer set of P_1^S and $H \cap \bar{S}$ is an (E, S)-match of P.*

Proof. Thanks to the richer syntax of logic programs compared to AFs, the proof of this result is conceptually simpler than the one for Theorem 1. Due to space restrictions, we give here only a sketch. The proof is done in two main steps.

First, given a normal program P and S a quasi-splitting of P, define the program $P_S = [P_1^S] \cup \overline{P}_2^S \cup \{\leftarrow \text{ not } b', b; \leftarrow b', ndr(b) \mid b \in V_S\}$ where $\overline{P}_2^S$ results from P_2^S by replacing each atom $s \in \bar{S}$ with a fresh atom s'. One can show that P and P_S are equivalent in the following sense: (1) if E is an answer set of P then $X = (E \cap S) \cup \{s' \mid s \in E \cap \bar{S}\} \cup \{ndr(s) \mid s \in V_S \setminus E\}$ is an answer set of P_S; in particular, by the definition of X and $V_S \subseteq \bar{S}$, we have, for each $s \in V_S$, $s' \in X$ iff $s \in X$ iff $ndr(s) \notin X$; (2) if H is an answer set of P_S, then $(H \cap S) \cup \{s \mid s' \in H\}$ is an answer set of P.

Next, we observe that P_S has a proper split T with $T = S \cup \{ndr(s) \mid s \in V_S\}$. With this result at hand, it can be shown that after evaluating the bottom part of this split, i.e. $(P_S)_1^T = [P_1^S]$, the rules $\{\leftarrow \text{ not } b', b; \leftarrow b', ndr(b) \mid b \in V_S\}$ in P_S play exactly the role of the replacements defined in $[P_2^S]_E$. In other words, we have for each $E \in AS((P_S)_1^T) = AS([P_1^S])$, the E-evaluation of $(P_S)_2^T$ is equal to $([P_2^S]_E)'$, where $([P_2^S]_E)'$ denotes $[P_2^S]_E$ replacing all atoms s by s'. Together with the above relation between answer sets of P and P_S the assertion is now shown in a quite straightforward way. □

Example 5. We follow up on Example 4. To determine the conditional answer sets of P_1^S we use the program

$a \leftarrow$ not b
$c \leftarrow a$
$b \leftarrow$ not $ndr(b)$
$ndr(b) \leftarrow$ not b

This program has two answer sets, namely $E_1 = \{a, c, ndr(b)\}$ and $E_2 = \{b\}$.

The E_1-modification of P_2^S consists of the single rule $\leftarrow b$ and its single answer set $\emptyset$ is an (S, E_1)-match. We thus obtain the first answer set of P, namely $(E_1 \cap S) \cup \emptyset = \{a, c\}$.

The E_2-modification of P_2^S is the program:

b
$\leftarrow not\ b$

Its single answer set $\{b\}$ is an (S, E_2)-match. We thus obtain the second answer set of P, namely $(E_2 \cap S) \cup \{b\} = \{b\}$.

Of course, the example is meant to illustrate the basic ideas, not to show the potential computational benefits of our approach. However, if we find a k-splitting with small k for large programs with hundreds of rules such benefits may be tremendous.

As the thoughtful reader may have recognized our results are stated for normal logic programs while Lifschitz and Turner define splittings for the more general class of disjunctive programs. However, our results can be extended to disjunctive programs in a straightforward way.

5 Algorithms for quasi-splittings

For the computation of quasi-splittings it is most beneficial to apply some of the existing highly efficient graph algorithms. Since AFs actually are graphs, such algorithms can be directly applied here. For logic programs we have to make a slight detour via their dependency graphs. Intuitively, a dependency graph is a directed graph with two types of links, E_n and E_p, the negative respectively positive links. Recall that the dependency graph of a logic program P is given by

$$(At(P), \{(a, head(r)) \mid a \in neg(r), r \in P\}, \{(a, head(r)) \mid a \in pos(r), r \in P\}).$$

The concepts of quasi-splittings, k-splittings and proper splittings can be defined for dependency graphs in a straightforward way:

Definition 10. *Let $D = (V, E_n, E_p)$ be a dependency graph and $S \subseteq V$. Let $\bar{S} = V \setminus S$, $E = E_n \cup E_p$, $R_{\rightarrow}^S = E \cap (S \times \bar{S})$ and $R_{\leftarrow}^S = E \cap (\bar{S} \times S)$. S is called*

- quasi-splitting *of D, if $R_{\leftarrow}^S \cap E_p = \emptyset$*
- k-splitting *of D, if $|R_{\leftarrow}^S| = k$ and $R_{\leftarrow}^S \cap E_p = \emptyset$.*
- *(proper)* splitting *of D, if $|R_{\leftarrow}^S| = 0$.*

Now it is not difficult to see that each quasi-splitting S of the dependency graph of a program P corresponds to a quasi-splitting of P and vice versa.[6] Since all dependencies in AFs are negative, we can also identify an AF (A, R) with the dependency graph $(A, R, \emptyset)$.

We now address the problem of finding splittings of a given dependency graph. For proper splittings the distinction between negative and positive dependencies is irrelevant. Proper splittings are given by the strongly connected components (SCCs) of the graph $(V, E_n \cup E_p)$ (see [3, 4] for more details). It is well known that the SCCs of a graph can efficiently be computed using, for instance, the Tarjan-algorithm [14].

While proper splittings do not allow splitting within an SCC S, with a k-splitting we can split S as long as the edge-connectivity of S is $\leq k$. However we additionally have to respect the condition $R^S_{\leftarrow} \cap E_p = \emptyset$, i.e. that there are no positive dependencies from $\bar{S}$ to S.

For computing a k-splitting of a dependency graph with minimal k, we can apply existing polynomial-time algorithms for computing minimum cuts in directed graphs (see e.g. [10]). Let $G = (V, E)$ be a directed graph where each arc (i, j) has an associated weight u_{ij}, which is a nonnegative real number. A cut is a partition of the node set into two nonempty parts S and $V \setminus S$. The capacity of the cut $(S, V \setminus S)$ is $\sum_{i \in S, j \in V \setminus S} u_{ij}$, and we denote the capacity of the cut as $u(S, V \setminus S)$. The *minimum unrestricted cut problem* is to find a partition of V into two nonempty parts, S^* and $V \setminus S^*$, so as to minimize $u(S^*, V \setminus S^*)$.

To ensure that minimal cuts do not contain forbidden edges we use the weight function $w : E \mapsto \{1, \infty\}$ such that $w(e) = \infty$ for $e \in E_p$ and $w(e) = 1$ for $e \in E_n \setminus E_p$. Now, if $(S^*, V \setminus S^*)$ is a solution to the problem, then $V \setminus S^*$ is a quasi-splitting of the graph with smallest possible k. More generally, each cut with finite capacity corresponds to a quasi-splitting, and if the minimal cut has weight ∞, then no quasi-splitting exists.

Recursive Splitting. We observe that quasi-splittings allow for a recursive procedure to compute the stable extensions of AFs, resp. the answer-sets of logic programs. This is in contrast to proper splittings, where a decomposition into SCCs is the "best" we can do. To exemplify this observation consider an AF with an even cycle

$$F = (\{a, b, c, d\}, \{(a, b), (b, c), (c, d), (d, a)\}).$$

Then we first can find a quasi split, e.g. $S = \{a, b\}$. If we now consider $[F^S_1]$, we observe that we now find even a proper splitting of this AF, namely $S' = \{a, att(a)\}$. The value of such a recursive approach is even more drastic if we consider a, b, c, d being huge DAGs which are only linked via single edges which however yield a cycle going through all arguments. This suggests the following simple recursive procedure to compute stable extensions of an AF.

[6] Note, however, that the program quasi-splitting corresponding to a k-splitting of the dependency graph may actually be a k'-splitting for some $k' \neq k$. This is due the different ways to count the size of a splitting. For splittings on the dependency graph we use the number of (negative) edges going form $\bar{S}$ to S while in the definition of program quasi-splittings we used the number of atoms in $\bar{S}$ such that there is an edge to S.

Function RS; input is an AF $F = (A, R)$; output is a set of extensions:

```
1. find non-trivial quasi-splitting $S$ of $F$ (s.t. $|R^S_{\leftarrow}|$ is minimal);
2. if $size([F^S_1]) \geq size(F)$ return[7] $stb(F)$ via some standard method;
3. otherwise, let $\mathcal{E} = \emptyset$ and do for each $E \in RS([F^S_1])$:
     $\mathcal{E} = \mathcal{E} \cup \{(E \cup E') \cap A \mid E' \in RS([F^S_2]_E)\}$.
4. return $\mathcal{E}$.
```

Note that the function terminates since the size of the involved AFs decreases in each recursion. The procedure is directly applicable to splitting of programs, by just replacing F by a program P and $stb(F)$ by $AS(P)$.

6 Discussion

In this paper, we proposed a generalization of the splitting concept introduced by Lifschitz and Turner which we showed to be applicable to two prominent nonmonotonic reasoning formalisms. Compared to other work in the area, the novel contributions of our work are twofold:

1. with the concept of quasi-splitting, we significantly extended the range of applicability of splitting without increasing the computational cost of finding a split;
2. the concept of a modification-based approach allows to evaluate the components of a split within the formalism under consideration.

We believe that the approach we have introduced here is applicable to further formalisms and semantics. In Section 5 we have already discussed some computation methods which might pave the way towards a uniform and general theory of splitting. Future work includes the refinement of our methods in the sense that splitting should, whenever possible, be performed in such a way that one of the resulting components is an easy-to-compute fragment.

Finally, we also want to provide empirical support for our claim that quasi-splittings have computational benefits. Note that quasi-splittings as well as proper splittings, on the one hand, divide the search space into smaller fractions in many cases, but on the other hand this might result in the computation of more models (i.e., stable extensions or answer sets) which turn out to be "useless" when propagated from the first to the second part of the split (i.e. they do not contribute to the models of the entire framework (or program). So from the theoretical side it is not clear how splitting effects the computation times. However for classical splitting we have empirical evidence [4] that splitting improves the average computation time. We thus plan to perform an empirical analysis of the effects of quasi-splitting on computation times in the spirit of [4].

Acknowledgments

We want to thank the reviewer of this paper for various useful remarks which helped to improve the presentation.

[7] $size(F)$ denotes the number of arguments in F.

References

1. Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowledge Eng. Review*, 26(4):365–410, 2011.
2. Pietro Baroni and Massimiliano Giacomin. Semantics of abstract argument systems. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 25–44. Springer, 2009.
3. Ringo Baumann. Splitting an argumentation framework. In James P. Delgrande and Wolfgang Faber, editors, *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011)*, volume 6645 of *LNCS*, pages 40–53. Springer, 2011.
4. Ringo Baumann, Gerhard Brewka, and Renata Wong. Splitting argumentation frameworks: An empirical evaluation. In *Proc. TAFA*, LNCS. Springer, 2012. to appear.
5. Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.
6. 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.
7. Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.
8. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.
9. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.
10. Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424–446, 1994.
11. Tomi Janhunen, Emilia Oikarinen, Hans Tompits, and Stefan Woltran. Modularity aspects of disjunctive stable models. *J. Artif. Intell. Res.*, 35:813–857, 2009.
12. Bei Shui Liao, Li Jin, and Robert C. Koons. Dynamics of argumentation systems: A division-based method. *Artif. Intell.*, 175(11):1790–1814, 2011.
13. Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proceedings of the Eleventh International Conference on Logic Programming (ICLP 1994)*, pages 23–38. MIT-Press, 1994.
14. Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
15. Hudson Turner. Splitting a default theory. In William J. Clancey and Daniel S. Weld, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI) Vol. 1*, pages 645–651. AAAI Press / The MIT Press, 1996.