A Unifying Framework for Learning Argumenation Semantics
Supplementary Material

Appendix A

Proofs of Equivalence of Learned Semantics Encodings to the ASPARTIX encodings

In this appendix we provide the programs learned for the stable, the admissible and the complete semantics of AAF by ILASP and prove their equivalence to the manually engineered ASPARTIX encodings.

A.1 Learning of the Stable Semantics

We construct a LAS task with context-dependent examples, describing labellings of the arguments in the framework coming from the stable semantics (*stable.las* file). We then run the command:

```
ILASP --version=4 --learn-heuristics stable.las
```

The hypothesis that ILASP learns is:

```
out(X):-defeated(X).
in(X):-arg(X), not out(X).
```

and so the learned encoding of the stable semantics adding the background knowledge B_{AAF} is

```
out(X):-defeated(X).
in(X):-arg(X), not out(X).
defeated(X):-att(Y,X), in(Y).
not_defended(X):-att(Y,X), not defeated(Y).
```

We call this program P_{stb} and the ASPARTIX program that represents the stable semantics S_{stb}. The ASPARTIX encoding is the following:

```
in(X):-not out(X), arg(X).
out(X):-not in(X), arg(X).
defeated(X):-att(Y,X), in(Y).
not_defended(X):-att(Y,X), not defeated(Y).
```

```
:- in(X), in(Y), att(X,Y).
:- out(X), not defeated(X).
```

Evidently, the encoding, learned using ILASP, is more compact that the encoding from the ASPARTIX framework. We proceed with proving that, although it is more compact, this encoding has the same meaning as the ASPARTIX encoding. The following theorem shows that the two programs lead to the same answer sets.

Theorem A.1.1. *Given the encoding of an AAF as a set of* ***arg/att*** *facts F, A is an answer set of $S_{stb} \cup F$, if and only if A is an answer set of $P_{stb} \cup F$.*

Proof. We prove each direction of the double implication separately. First, assume that A is an answer set of $S_{stb} \cup F$. We now show that A is also an answer set of the program $P_{stb} \cup F$. For simplicity, as there are no function symbols, we can assume that $ground(P_{stb})$ returns all the ground instances of P_{stb} rather than the relevant ones produced by Clingo - irrelevant instances have no effect, so this is will not change the result. The reduct $(ground(P_{stb} \cup F))^A$ must contain:

```
(1) F.
(2) out(a):-defeated(a)., for each constant a.
(3) in(a):-arg(a).,for each constant a such
    that out(a)∉ A.
(4) defeated(a):-att(b,a), in(b)., for each pair
    of constants a and b.
(5) not_defended(a):-att(b,a)., for each pair of
    constants a and b, such that defeated(b)∉ A.
```

Since A is an answer set of $S_{stb} \cup F$, from the first two rules in S_{stb} we can conclude that for each constant a, $out(a) \in A$ iff $in(a) \notin A$ and $in(a) \in A$ iff $out(a) \notin A$. Hence (3) can be written as:

```
(3) in(a):-arg(a)., for each constant such that in(a)∈ A.
```

And as for all constants a, $arg(a) \in A$, since the only constants in the program are arguments, this can be further simplified to:

```
(3) in(a)., for each constant a such that in(a)∈ A.
```

Hence, all the ground instances of the predicate **in** in the reduct are the same as the ones in A.

Since the body conditions are proved to be the same in the definition of **defeated** for both S_{stb} and P_{stb}, the ground instances of rule (4) can be simplified to

```
(4) defeated(a)., for each constant a such that
    defeated(a)∈ A.
```

Similarly, the body conditions in the definition of **not_defended** are proved to be the same for both S_{stb} and P_{stb}, which means that rule (5) can be written as

```
(5) not_defended(a)., for each constant a such that
    not_defeanded(a)∈ A.
```

To be able now to simplify rule (2), we prove the following Lemma:

Lemma A.1.1. *Given that A is an answer set of $S_{stb} \cup F$ and a is an argument constant, $out(a) \in A$ iff $defeated(a) \in A$.*

Proof of Lemma. We first show that if $out(a) \in A$ then $defeated(a) \in A$. It is clear from the last constraint in S_{stb}, that if $out(a)$ is in A, $defeated(a)$ is in A.

For the other direction, let's assume $defeated(a) \in A$. The first constraint in S_{stb} is equivalent to ": $-in(Y), defeated(Y)$.", where the definition of the **defeated** predicate is used. From this constraint and the fact that $defeated(a) \in A$, it follows that $in(a)$ is not in A. Hence, by the second rule in S_{stb}, $out(a) \in A$. ■

Rule (2) can then be written as

```
(2) out(a)., for each constant a such that defeated(a)∈ A.
```

and using Lemma 1.1.1 this is transformed to

```
(2) out(a)., for each constant a such that out(a)∈ A.
```

The reduct $(ground(P_{stb} \cup F))^A$ becomes

```
(1) F.
(2) out(a)., for each constant a such that out(a)∈ A.
(3) in(a)., for each constant a such that in(a)∈ A.
(4) defeated(a)., for each constant a such that
    defeated(a)∈ A.
(5) not_defended(a)., for each constant a such that
    not_defended(a)∈ A.
```

Therefore, all ground instances of the predicates **arg, att, in, out, defeated, not_defended** are the same in the reduct and in A. Hence, the minimal model of the reduct must be equal to A, meaning that A is an answer set of $P_{stb} \cup F$.

It remains to show the other direction of the double implication, which is that if A is an answer set of $P_{stb} \cup F$ it must also be an answer set of $S_{stb} \cup F$. We begin by assuming that A is an answer set of $P_{stb} \cup F$. The reduct of $ground(S_{stb} \cup F)^A$ contains the following:

```
(1) F.
(2) in(a):-arg(a).,for each constant a such
    that out(a)∉ A.
(3) out(a):-arg(a).,for each constant a such
    that in(a)∉ A.
(4) defeated(a):-in(b), att(b,a)., for each pair
    of constants a and b.
(5) not_defended(a):-att(b,a)., for each pair of
    constants a and b, such that defeated(b)∉ A.
(6) ⊥:-in(a), in(b), att(a,b)., for each pair
    of constants a and b.
(7) ⊥:-out(a), not defeated(a)., for each constant a.
```

As each constant a is guaranteed to have a fact $arg(a)$ in F, this can be simplified to:

```
(1) F.
(2) in(a)., for each a such that out(a)∉ A.
(3) out(a)., for each a such that in(a)∉ A.
(4) defeated(a):-in(b), att(b,a)., for each pair
    of constants a and b.
(5) not_defended(a):-att(b,a)., for each pair of
    constants a and b, such that defeated(b)∉ A.
(6) ⊥:-in(a), in(b), att(a,b)., for each pair
    of constants a and b.
(7) ⊥:-out(a), not defeated(a)., for each constant a.
```

We first show that the constraints in (6) and (7) are respected by A. Clearly, (7) cannot produce $\bot$ using the ground instances of A, as $out(a)$ is defined to only be true if $defeated(a)$ is true, according to the first rule in P_{stb}.

Rule (6) is equivalent to:

```
(6) ⊥:-in(b), defeated(b).
```

using the definition of the **defeated** predicate. So for this rule to produce $\bot$, there must be a constant b such that $in(b) \in A$ and $defeated(b) \in A$. But if $in(b) \in (A)$, then by the second rule in P_{stb}, $out(b)$ must not be in A. Then by the first rule in P_{stb}, it follows that $defeated(b)$ must also not be in A. Hence, the constraint in (6) cannot be violated.

Next, from the second rule in P_{stb}, it follows that if $out(a) \notin A$, then $in(a) \in A$. Thus, rule (2) becomes

```
(2) in(a)., for each constant a such that in(a)∈ A.
```

For the transformation of rule (3) we prove the Lemma below.

Lemma A.1.2. *Given that A is an answer set of $P_{stb} \cup F$ and a is an argument constant, $in(a) \notin A$ iff $out(a) \in A$.*

Proof of Lemma. We prove each direction of the implication separately. First, assume that $in(a) \notin A$. Then by the second rule of P_{stb}, it follows that $out(a) \in A$.

Conversely, if $out(a) \in A$, by constraint (7), which we showed is respected by A, $defeated(a) \in A$. We mentioned that constraint (6), which is also respected by A, is equivalent to "$\bot : -in(b), defeated(b)$.", using the definition of the defeated predicate. Considering this constraint and the fact that $defeated(a) \in A$, we can conclude that $in(a) \notin A$. ■

Using Lemma 1.1.2, rule (3) can be written as

```
(3) out(a)., for each constant a such that out(a)∈ A.
```

We proved that the body conditions in the definition of $defeated(a)$ are the same for both S_{stb} and P_{stb}, so the ground instances of rule (4) are simplified to

```
(4) defeated(a)., for each constant a such that
    defeated(a)∈ A.
```

Reasoning in the same way for the definition of **not_defended**, we transform rule (5) to

```
(5) not_defended(a)., for each constant a such that
    not_defended(a)∈ A.
```

We have successfully transformed the reduct $(ground(S_{stb} \cup F))^A$ to

```
(1) F.
(2) in(a)., for each constant a such that in(a)∈ A.
(3) out(a)., for each constant a such that out(a)∈ A.
(4) defeated(a)., for each constant a such that
    defeated(a)∈ A.
(5) not_defended(a)., for each constant a such that
    not_defended(a)∈ A.
(6) ⊥:-in(a), in(b), att(a,b)., for each pair
    of constants a and b.
(7) ⊥:-out(a), not defeated(a)., for each constant a.
```

Thus, we can conclude that all ground instances of the predicates **arg, att, in, out, defeated, not_defended** are the same in the reduct and in A. Moreover, the two rules in (6) and (7) do not produce $\bot$ ($\bot$ cannot be a part of any model), which means that the constraints are respected by A, so the minimal model of the reduct must be equal to A and then A is an answer set of $S_{stb} \cup F$.

We have now shown that A is an answer set of $S_{stb} \cup F$ iff A is an answer set of $P_{stb} \cup F$. □

A.2 Learning of the Admissible Semantics

Similarly to the stable semantics, ILASP learns a more compact program to represent the admissible semantics. Together with B_{AAF}, the learned program is the following:

```
out(X):-defeated(X).
out(X):-arg(X), not in(X).
in(X):-arg(X), not out(X), not not_defended(X).
defeated(X):-in(Y), att(Y,X).
not_defended(X):-att(Y,X), not defeated(Y).
```

We call this program P_{adm} and the program

```
in(X):-not out(X), arg(X).
out(X):-not in(X), arg(X).
defeated(X):-in(Y), att(Y,X).
not_defended(X):-att(Y,X), not defeated(Y).
```

```
:- in(X), in(Y), att(X,Y).
:- in(X), not_defended(X).
```

which is the ASPARTIX encoding for admissible semantics S_{adm}.

We prove the theorem given below, which shows that the two programs give us the same answer sets.

Theorem A.2.1. *Given the encoding of an AAF as a set of* ***arg/att*** *facts F, A is an answer set of $S_{adm} \cup F$, if and only if A is an answer set of $P_{adm} \cup F$.*

Proof. We first prove the implication from left to right. For this we assume that A is an answer set of $S_{adm} \cup F$. We now show that A is also an answer set of $P_{adm} \cup F$. Again for simplicity, as there are no function symbols, we can assume that $ground(P_{adm})$ returns all the ground instances of P_{adm} rather than the relevant ones produced by Clingo. The reduct $(ground(P_{adm} \cup F))^A$ must contain:

```
(1) F.
(2) out(a):-defeated(a)., for each constant a
(3) out(a):-arg(a)., for each constant a
    such that in(a)∉ A.
(4) in(a):-arg(a), for each constant a such
    that out(a)∉ A and not_defended(a)∉ A.
(5) defeated(a):-att(b,a), in(b)., for each pair
    of constants a and b.
(6) not_defended(a):-att(b,a)., for each pair of
    constants a and b, such that defeated(b)∉ A.
```

We see that (2) can be rewritten as:

```
(2) out(a)., for each constant a such that defeated(a)∈ A.
```

Since A is an answer set of $S_{adm} \cup F$, from the first two rules in S_{adm} it can be concluded that for each constant a, $out(a) \in A$ iff $in(a) \notin A$ and $in(a) \in A$ iff $out(a) \notin A$. This helps us to rewrite rule (3) as:

```
(3) out(a)., for each constant a such that out(a)∈ A.
```

We can combine (2) and (3) to get

```
(2-3) out(a)., for each constant a such
      that out(a)∈ A or defeated(a)∈ A.
```

To simplify this rule further, we will make use of a Lemma.

Lemma A.2.1. *Given that A is an answer set of $S_{adm} \cup F$ and a is an argument constant, $out(a) \in A$ iff $out(a) \in A$ or $defeated(a) \in A$.*

Proof of Lemma. The proof from left to right is trivial - if we have $out(a) \in A$, then $out(a) \in A$ or $defeated(a) \in A$.

For the other direction, we have to separately show that if $out(a) \in A$, then $out(a) \in A$ and that if $defeated(a) \in A$, then $out(a) \in A$. Clearly, if $out(a) \in A$, then $out(a) \in A$. Let's assume $defeated(a) \in A$. The first constraint in S_{adm} is equivalent

to ": $-in(Y), defeated(Y)$.", where the definition of the **defeated** predicate is used. Then having $defeated(a) \in A$ means that $in(a)$ is not in A and by the second rule in S_{adm}, $out(a) \in A$. Hence, $out(a) \in A$ iff $out(a) \in A$ or $defeated(a) \in A$. ■

Applying Lemma 1.1.3, the rule (2-3) can be rewritten as:

```
(2-3) out(a)., for each constant a such that out(a)∈ A.
```

We can also transform (4) to:

```
(4) in(a)., for each constant a such
    that in(a)∈ A and not_defended(a)∉ A.
```

We again proceed with proving a Lemma to help us transform this rule further.

Lemma A.2.2. *Given that A is an answer set of $S_{adm} \cup F$ and a is an argument constant, $in(a) \in A$ iff $in(a) \in A$ and $not\_defended(a) \notin A$.*

Proof of Lemma. This time the proof from right to left is trivial - if we have $in(a) \in A$ and $not\_defended(a) \notin A$, then $in(a) \in A$.

For the other direction, clearly, if $in(a) \in A$, then $in(a) \in A$, so we are left to show that if $in(a) \in A$, then we also have $not\_defended(a) \notin A$. For this we can use the last constraint in S_{adm} - if $in(a) \in A$, then $not\_defended(a) \notin A$. It follows that if $in(a) \in A$, then $in(a) \in A$ and $not\_defended(a) \notin A$. ■

Applying Lemma 1.1.4, rule (4) is equivalent to:

```
(4) in(a)., for each constant a such that in(a)∈ A.
```

As everything in the body has been proven to be the same in the definitions of **defeated** in both S_{adm} and P_{adm}, rule (5) can be simplified to:

```
(5) defeated(a)., for each constant a such that
    defeated(a)∈ A.
```

This also means that everything in the body of rule (6) is the same in both cases, so (6) becomes

```
(6) not_defended(a)., for each constant a
    such that not_defended(a)∈ A.
```

Like for the stable semantics, we have simplified each rule so that we can conclude that all ground instances of the predicates **arg, att, in, out, defeated, not_defended** are the same in the reduct and in A. Hence, the minimal model of the reduct must be equal to A, meaning that A is an answer set of $P_{adm} \cup F$.

Next, we need to prove the implication from right to left, so we have to show that if A is an answer set of $P_{adm} \cup F$, it must also be an answer set of $S_{adm} \cup F$. For the purpose of the proof, let's assume that A is an answer set of $P_{adm} \cup F$. The reduct of ground$((S_{adm} \cup F)^A$ contains the following:

```
(1) F.
(2) in(a)., for each constant a such that out(a)∉ A.
```

```
(3) out(a)., for each constant a such that in(a)∉ A.
(4) defeated(a):-in(b), att(b,a)., for each pair
    of constants a and b.
(5) not_defended(a):-att(b,a)., for each pair of
    constants a and b, such that defeated(b)∉ A.
(6) ⊥:-in(a), in(b), att(a,b)., for each
    pair of constants a and b.
(7) ⊥:-in(a), not_defended(a)., for each constant a.
```

where for simplifying (2) and (3) we used the fact that each constant a has $arg(a)$ in F.

We have to show that the constraints in (6) and (7) are respected by A, which means that they should not produce $\bot$ ($\bot$ cannot be a part of any model). For (6), first note that it is equivalent to:

```
(6) ⊥:-in(b), defeated(b).
```

where we make use of the definition of the **defeated** predicate. For this constraint to be violated, there must be a constant b such that $in(b) \in A$ and $defeated(b) \in A$. But if $defeated(b) \in A$, then by the first rule in P_{adm}, $out(b)$ must also be in A, meaning that $in(b)$ could not be in A, by the third rule in P_{adm}. Hence, the constraint cannot be violated.

Rule (7) cannot produce $\bot$ using the ground instances of A, as $in(X)$ is defined to only be true if $not\_defended(X)$ is false (by the third rule in P_{adm}).

To simplify rule (2) we will prove the Lemma below.

Lemma A.2.3. *Given that A is an answer set of $P_{adm} \cup F$ and a is an argument constant, $out(a) \notin A$ iff $in(a) \in A$.*

Proof of Lemma. For the proof from left to right, notice that if $in(a) \notin A$ the second rule of P_{adm} will produce $out(a)$. This means that if $out(a) \notin A$, then $in(a) \in A$.

For the proof from right to left we use the third rule in P_{adm} - if we have $in(a) \in A$, then $out(a)$ should not be in A. ■

Using Lemma 1.1.5, rule (2) is transformed to

```
(2) in(a)., for each constant a such that in(a)∈ A.
```

Also, by the second rule in P_{adm}, if $in(X) \notin A$, then $out(X) \in A$, so rule (3) becomes:

```
(3) out(a)., for each constant a such that out(a)∈ A.
```

Every body condition in the definition of the **defeated** predicate is the same for both S_{adm} and P_{adm}, so (4) can be rewritten as:

```
(4) defeated(a)., for defeated(a)∈ A.
```

Now everything in the body of rule (5) (the definition of the **not_ defended** predicate) is the same in both cases so (5) can be simplified to:

```
(5) not_defended(a)., for not_defended(a)∈ A.
```

We have simplified each rule to be able conclude that all ground instances of the predicates **arg, att, in, out, defeated, not_defended** are the same in the reduct and in A. Moreover, the two rules in (6) and (7) never produce $\bot$, which means that the minimal model of the reduct must be equal to A, and so A is an answer set of $S_{adm} \cup F$.

Based on the above proofs of the two-direction implication, A is an answer set of $S_{adm} \cup F$ iff A is an answer set of $P_{adm} \cup F$. □

A.3 Learning of the Complete Semantics

For the complete semantics, ILASP learns the following program:

```
out(X):-not_defended(X).
in(X):-arg(X), not out(X), not defeated(X).
```

Combined with B_{AAF}, this gives us the program P_{com}, describing the complete semantics:

```
out(X):-not_defended(X).
in(X):-arg(X), not out(X), not defeated(X).
defeated(X):-in(Y), att(Y,X).
not_defended(X):-att(Y,X), not defeated(Y).
```

We call the program

```
in(X):-not out(X), arg(X).
out(X):-not in(X), arg(X).
defeated(X):-in(Y), att(Y,X).
not_defended(X):-att(Y,X), not defeated(Y).
:- in(X), in(Y), att(X,Y).
:- in(X), not_defended(X).
:- out(X), not not_defended(X).
```

that is the ASPARTIX encoding for the complete semantics S_{com}.

We want to show that the two programs, together with the background knowledge that comes from the argumentation framework - a set of **arg**/**att** facts, have the same answer sets. This is captured by the theorem below.

Theorem A.3.1. *Given the encoding of an AAF as a set of **arg/att** facts F, A is an answer set of $S_{com} \cup F$, if and only if A is an answer set of $P_{com} \cup F$.*

Proof. We prove each direction of the double implication separately. We start by assuming that A is an answer set of $S_{com} \cup F$. Now we show that A is also an answer set of $P_{com} \cup F$. There are no function symbols, so we can again assume that $ground(P_{com})$ returns all the ground instances of P_{com} rather than the relevant ones produced by Clingo. The reduct $(ground(P_{com} \cup F))^A$ must contain:

```
(1) F.
(2) out(a):-not_defended(a)., for each constant a.
(3) in(a)., for each constant a
```

```
    such that out(a)∉ A and defeated(a)∉ A.
(4) defeated(a):-att(b,a), in(b)., for each
    pair of constants a and b.
(5) not_defended(a):-att(b,a)., for each pair of
    constants a and b, such that defeated(b)∉ A.
```

where rule (3) was simplified using the fact that $arg(a)$ is in A for each constant a.

We first focus on transforming rule (2) and prove a helper Lemma.

Lemma A.3.1. *Given that A is the answer set of $S_{com} \cup F$ and a is an argument constant, $out(a) \in A$ iff $not\_defended(a) \in A$.*

Proof of Lemma. We show each direction of the double implication separately. From the second constraint in S_{com}, we know that if $out(a)$ is in A, then $not\_defended(a)$ is in A.

To prove the implication in the other direction, let's assume that $not\_defended(a)$ is in A. Then by the second constraint in S_{com} (": $-in(X), not\_defended(X)$."), it follows that $in(a) \notin A$ and from the second rule in S_{com}, we can conclude that $out(a) \in A$. ■

Applying Lemma 1.1.6, rule (2) can be rewritten as

```
(2) out(a)., for each constant a such that out(a)∈ A.
```

Since A is an answer set of $S_{com} \cup F$, it is true that $in(a) \in A$ iff $out(a) \notin A$. Therefore, rule (3) becomes:

```
(3) in(a)., for each constant a such that in(a)∈ A
    and defeated(a)∉ A.
```

The following Lemma helps us simplify this rule further.

Lemma A.3.2. *Given that A is the answer set of $S_{com} \cup F$ and a is an argument constant, $in(a) \in A$ iff $in(a) \in A$ and $defeated(X) \notin A$.*

Proof of Lemma. It is trivial to show the implication from right to left. If $in(a) \in A$ and $defeated(X) \notin A$, then $in(a) \in A$.

The other direction requires the use of the first constraint in S_{com}. Using the definition of the predicate **defeated**, the first constraint in S_{com} is equivalent to ": $-in(Y), defeated(Y)$.", so if $in(a)$ is in A, then $defeated(a)$ is not in A. Clearly, if $in(a) \in A$, then $in(a) \in A$, so it follows that if $in(a) \in A$, we have $in(a) \in A$ and $defeated(a) \notin A$. ■

Applying Lemma 1.1.7, rule (3) can be transformed to:

```
(3) in(a)., for each constant a such that in(a)∈ A.
```

Everything in the body of the definition of **defeated** has been proven to be the same for both programs, so rule (4) is simplified to:

```
(4) defeated(a)., for each constant a such that
    defeated(X)∈ A.
```

Consequently, everything in the body the definition of **not_defended** is the same in both cases, so rule (5) becomes

```
(5) not_defended(a)., for each constant a such
    that not_defended(a)∈ A.
```

After we have performed the above steps, each rule is simplified so that it is clear that all ground instances of the predicates **arg, att, in, out, defeated, not_defended** are the same in the reduct and in A. Therefore, the minimal model of the reduct must be equal to A, meaning that A is an answer set of $P_{com} \cup F$.

Using similar simplifications, we will prove the other direction of the double implication - we will show that if A is an answer set of $P_{com} \cup F$ it must also be an answer set of $S_{com} \cup F$. To do that, we assume that A is an answer set of $P_{com} \cup F$. The reduct of $ground((S_{com} \cup F)^A$ contains the following:

```
(1) F.
(2) in(a)., for each constant a such that out(a)∉ A.
(3) out(a)., for each constant a such that in(a)∉ A.
(4) defeated(a):-in(b), att(b,a)., for each
    pair of constants a and b.
(5) not_defended(a):-att(b,a)., for each pair of
    constants a and b, such that defeated(b)∉ A.
(6) ⊥:-in(a), in(b), att(a,b)., for each
    pair of constants a and b.
(7) ⊥:-in(a), not_defended(a)., for each constant a.
(8) ⊥:-out(a), not not_defended(a)., for each constant a.
```

where rules (2) and (3) were simplified using the fact that for each constant a, $arg(a)$ $\in A$.

We proceed by first showing that the three rules (6), (7) and (8) do not produce $\bot$, as $\bot$ cannot be a part of any model.

For (6), first note that it is equivalent to:

```
(6) ⊥:-in(b), defeated(b).
```

But by the second rule in P_{com}, $in(b)$ is defined to only be true if $defeated(b)$ is not true. Hence, rule (6) cannot lead to $\bot$.

We can see that (7) could not produce $\bot$ using instances of the predicates in A, as if $not\_defended(a)$ is true, by the first rule in P_{com}, $out(a)$ is in A, which means that $in(a)$ is not in A (using the second rule in P_{com}).

Clearly, (8) also could not produce $\bot$ using instances of the predicates in A, as by the first rule in P_{com}, $out(a)$ is defined to be true only if $not\_defended(a)$ is true.

Rule (2) needs to be simplified and for that we will use a Lemma.

Lemma A.3.3. *Given that A is the answer set of $P_{com} \cup F$ and a is an argument constant, $out(a) \notin A$ iff $in(a) \in A$.*

Proof of Lemma. First, notice that the second rule in P_{com} is the only one that can produce $in(a)$. For the proof from right to left, from this rule it follows that if $in(a)$, then $out(a) \notin A$.

For the proof from left to right, we will again use this second rule in P_{com}. Assume that $out(a) \notin A$ and assume for proof by contradiction that we also have $in(a) \notin A$. Then for the second rule of P_{com} to not produce $in(a)$, we need to have $defeated(a) \in A$. But then since only the third rule in P_{com} can produce $defeated(a)$, from this rule we must have that there exists a constant b, such that $in(b) \in A$ and $att(b, a) \in A$.

In addition, since $out(a) \notin A$, from the first rule in P_{com}, $not\_defended(a) \notin A$. Now if we look at the definition of **not_defended** in P_{com}, we know that $att(b, a) \in A$, so in order for $not\_defended(a)$ to not be in A, we must have $defeated(b) \in A$.

We showed that there exists a constant b such that $in(b) \in A$ and $defeated(b) \in A$. However, as we already mentioned, rule (6), which is respected by A can be transformed to "$\bot : -in(b), defeated(b)$.". This is a contradiction to $in(b) \in A$ and $defeated(b) \in A$ that we showed, so our assumption that $in(a) \notin A$ is not correct and it follows that $in(a) \in A$.

We can conclude that $out(a) \notin A$ iff $in(a) \in A$. ■

Applying Lemma 1.1.8, rule (2) becomes:

```
(2) in(a)., for each constant a such that in(a)∈ A.
```

To transform rule (3), we will use two Lemmas, so that we can split the proof of the important results in smaller parts.

Lemma A.3.4. *Given that A is the answer set of $P_{com} \cup F$ and a is an argument constant, if $defeated(a) \in A$, then $out(a) \in A$.*

Proof of Lemma. For the constant a the third rule in P_{com} will become "$defeated(a) : -in(Y), att(Y, a)$". Assume that $defeated(a) \in A$, so there exists a constant c, such that $in(c) \in A$ and $att(c, a) \in A$. Earlier we transformed rule (6) to "$\bot : -in(b), defeated(b)$.", using the definition of the **defeated** predicate and showed that it represents a constraint, respected by A. For the constant c, $in(c) \in A$, so by the transformed rule (6), $defeated(c) \notin A$. This makes $att(c, a)$ true and $defeated(c)$ not true with respect to A. Thus, by the fourth rule of P_{com}, $not\_defended(a)$. Finally, by the first rule in P_{com}, we conclude that $out(a) \in A$. ■

Lemma A.3.5. *Given that A is the answer set of $P_{com} \cup F$ and a is an argument constant, $out(a) \in A$ iff $in(a) \notin A$.*

Proof of Lemma. We separately prove each direction of the double implication. First, assume that $out(a) \in A$. Then by rule (8), which is in fact a constraint respected by A (we proved this above), it follows that $not\_defended(a) \in A$. Moreover, rule (7) is also a constraint respected by A, and because $not\_defended(a) \in A$, we have $in(a) \notin A$.

Conversely, assume that $in(a) \notin A$. Rules (6), (7) and (8) are constraints respected by A. Also, earlier we transformed rule (6) to "$\bot : -in(b), defeated(b)$.", using the definition of the **defeated** predicate. We assumed that $in(a) \notin A$ and using the second rule in P_{com} and that $arg(a) \in A$ for each constant a, it follows that $out(a) \in A$ or $defeated(a) \in A$. Applying Lemma 1.1.8, from $defeated(a) \in A$, $out(a) \in A$ follows. Therefore, both cases of the disjunction $out(a) \in A$ or $defeated(a) \in A$ lead to $out(a) \in A$, which means that if $in(a) \notin A$, then $out(a) \in A$. ■

With the help of Lemma 1.1.10, rule (3) is transformed to:

```
(3) out(a)., for each constant a such that out(a)∈ A.
```

And as the body predicates in the definition of the **defeated** are the same in both programs, rule (4) can be simplified to:

```
(4) defeated(a)., for each constant a such that
    defeated(a)∈ A.
```

Consequently, everything in the body of (5) (the definition of the **not_defended** predicate) is the same in both programs, so rule (5) becomes:

```
(5) not_defended(a)., for each constant such that
    not_defended(a)∈ A.
```

After simplifying each rule in the reduct and showing that the constraints are respected by A, we can conclude that the ground instances of the predicates **arg, att, in, out, defeated, not_defended** are the same in the reduct and in A. This means that the minimal model of the reduct must be equal to A, so A is an answer set of $S_{com} \cup F$.

Thus, we have now shown that A is an answer set of $S_{com} \cup F$ iff A is an answer set of $P_{com} \cup F$. □

Appendix B

Proof of Simplification of the Background Knowledge of the Unified LAS Task Encoding

In this appendix we show that for each of AAF, BAF and VAF, the background knowledge of this unified LAS task encoding simplifies significantly (given the context of the examples in the task).

The unified LAS task encoding background knowledge B is the following:

```
(1) support(X,Z) :- support(X,Y), support(Y,Z).
(2) supported(X) :- support(Y,X), in(Y).
(3) valpref(X,Y) :- valpref(X,Z), valpref(Z,Y).
(4) pref(X,Y) :- valpref(U,V), val(X,U), val(Y,V).
(5) pref(X,Y) :- pref(X,Z), pref(Z,Y).
(6) defeat(X,Y) :- att(Z,Y), support(X,Z).
(7) defeat(X,Y) :- att(X,Z), support(Z,Y).
(8) defeat(X,Y) :- att(X,Y), not pref(Y,X).
(9) defeated(X) :- in(Y), defeat(Y,X).
(10) not_defended(X) :- defeat(Y,X), not defeated(Y).
```

B.1 Simplification for AAF

There will be no instances of the predicates **valpref** and **support** - they will not be present in the context of the examples E^+ and E^-. Therefore, rules (2), (4) and (5), will not produce any instances of the predicates **supported** and **pref**. Consequently, rules (6) and (7) won't produce any instances of the predicate **defeat**, leaving only one rule (8) that may produce instances of this predicate. As there are no instances of the predicate **pref**, rule (8) will be equivalent to

```
defeat(Y,X) :- att(Y,X).
```

From this simplified rule, if we have $att(Y, X)$, then we have $defeat(Y, X)$ and this is the only definition of the predicate **defeat** that can produce it. This means that if we substitute rule (9) with

```
defeated(X) :- in(Y), att(Y,X).
```

and rule (10) with

```
not_defended(X) :- att(Y,X), not defeated(Y).
```

there will be no new instances of the **defeated** and **not_defended** predicates produced. Now we can drop the definition of **defeat** (rule (8)) since it is not in the body of any rule. Thus, the only two rules we are left with, that will influence the learning are:

```
defeated(X) :- in(Y), att(Y,X).
not_defended(X) :- att(Y,X), not defeated(Y).
```

which is exactly the simplified background knowledge B_{AAF} discussed in the paper.

B.2 Simplification for BAF

There will be no instances of the predicate **valpref**, because it won't be present in the context of the examples E^+ and E^-. Therefore, rules (4) and (5) will not produce any instances of the predicate **pref**. This means that rule (8) to

```
defeat(Y,X) :- att(Y,X).
```

Thus, the rules that are left, that will impact the learning are:

```
support(X,Z) :- support(X,Y), support(Y,Z).
supported(X) :- support(Y,X), in(Y).
defeat(X,Y) :- att(Z,Y), support(X,Z).
defeat(X,Y) :- att(X,Z), support(Z,Y).
defeat(X,Y) :- att(X,Y).
defeated(X) :- in(Y), defeat(Y,X).
not_defended(X) :- defeat(Y,X), not defeated(Y).
```

which is the background knowledge B_{BAF} discussed in the paper.

B.3 Simplification for VAF

We notice that there will be no instances of the predicate **support**, as it won't be present in the context of the examples E^+ and E^-. Consequently, rule (2) won't produce any instances of the predicate **supported**. Also, rules (6) and (7) will not produce any instances of the **defeat** predicate, so we the only rule that is left and might produce instances of this predicate is (8). Therefore, the only rules that are left to influence the learning are:

```
valpref(X,Y) :- valpref(X,Z), valpref(Z,Y).
pref(X,Y) :- valpref(U,V), val(X,U), val(Y,V).
pref(X,Y) :- pref(X,Z), pref(Z,Y).
```

```
defeat(X,Y) :- att(X,Y), not pref(Y,X).
defeated(X) :- in(Y), defeat(Y,X).
not_defended(X) :- defeat(Y,X), not defeated(Y).
```

and this is the background knowledge B_{VAF} from the paper.