On the Equivalence between Assumption-Based Argumentation and Logic Programming (Extended Abstract)*

Martin Caminada1, Claudia Schulz2

1 School of Computer Science and Informatics, Cardiff University, UK
2 Ubiquitous Knowledge Processing Lab (UKP-TUDA), Department of Computer Science, Technische Universität Darmstadt, Germany
CaminadaM@cardiff.ac.uk, schulz@ukp.informatik.tu-darmstadt.de

Abstract
In this work, we explain how Assumption-Based Argumentation (ABA) is subsumed by Logic Programming (LP). The translation from ABA to LP (with a few restrictions on the ABA framework) results in a normal logic program whose semantics coincide with the semantics of the underlying ABA framework. Although the precise technicalities are beyond the current extended abstract (these can be found in the associated full paper) we provide a number of examples to illustrate the general idea.

1 Introduction
Assumption-Based Argumentation (ABA) [Bondarenko et al., 1997; Dung et al., 2009; Toni, 2014] has become one of the leading approaches for formal argumentation. It provides methods for the construction of arguments from given inference rules and defeasible information, called assumptions, as well as for the identification of attacks between assumptions based on the notions of arguments and contraries of assumptions. The semantics of an ABA framework can be given in terms of assumption labellings [Schulz and Toni, 2014; 2017], which assign to each assumption a label IN, OUT or UNDEC.

ABA has a well-studied relationship to abstract argumentation (AA) [Dung, 1995], where arguments and attacks between them are given rather than constructed from knowledge and semantics can be defined in terms of sets of argument labellings [Caminada and Gabbay, 2009], in that both flat ABA is an instance of AA [Dung et al., 2007; Toni, 2014; Caminada et al., 2015] and AA is an instance of flat ABA [Toni, 2012] under many well-studied semantics.

In addition to its relationship with AA, it has been shown that ABA is powerful enough to capture various non-monotonic reasoning formalisms such as default logic, circumscriptive, autoepistemic logic, and – most importantly for this paper – logic programming (LP) [Bondarenko et al., 1997; Toni, 2007; 2008; Schulz and Toni, 2015]. More precisely, the aforementioned formalisms can be translated into ABA frameworks in such a way that the semantics of the resulting ABA framework and of the respective formalism correspond.

In our work, we investigate the opposite direction. That is, we define a translation from a (flat) ABA framework to an associated logic program and show that LP is powerful enough to capture the commonly used semantics of (flat) ABA. We here give the main idea of this work, using a number of running examples for illustration, and refer to the full paper [Caminada and Schulz, 2017] for details.

2 Assumption-Based Argumentation
An ABA framework \( \mathcal{F} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \gamma) \) [Toni, 2014; Bondarenko et al., 1997] consists of a set of inference rules \( \mathcal{R} \), based on a formal language \( \mathcal{L} \). An important part of this formal language is the set of assumptions \( \mathcal{A} \subseteq \mathcal{L} \). An assumption represents a defeasible piece of information that is assumed to be true by default, unless contrary information is available. Thus, each assumption is associated (through the function \( \gamma \)) with an element of the formal language \( \mathcal{L} \) that is called its contrary.

As an example, consider the ABA framework \( \mathcal{F}_{ex} \) with the set of assumptions \( \mathcal{A} = \{ \beta, \gamma, \delta, e, \varphi \} \), inference rules

\[
\begin{align*}
& a \leftarrow \quad b \leftarrow \gamma \quad c \leftarrow \beta \\
& d \leftarrow \gamma, \varphi \quad e \leftarrow a, \delta \quad f \leftarrow a, e
\end{align*}
\]

and contraries

\[
\begin{align*}
\overline{\beta} &= b \\
\overline{\gamma} &= e \\
\overline{\delta} &= d \\
\overline{e} &= f
\end{align*}
\]

Based on an ABA framework, one can start to construct arguments. These are basically derivations that use the rules of the ABA framework together with the set of assumptions to infer a conclusion. An argument is denoted \( \text{Asm} \vdash x \), where \( \text{Asm} \) is the set of all assumptions used in the derivation and \( x \) is the conclusion, i.e. the head of the sentence obtained in the last derivation step. For instance, in the above example \( \mathcal{F}_{ex} \), one can construct the following arguments: \( \emptyset \vdash a \) (applying rule \( a \leftarrow \)), \( \{ \gamma \} \vdash b \) (applying rule \( b \leftarrow \gamma \)), \( \{ \beta \} \vdash e \) (applying rule \( c \leftarrow \beta \)), \( \{ \gamma, \varphi \} \vdash d \) (applying rule \( d \leftarrow \gamma, \varphi \)), \( \{ \delta \} \vdash e \) (applying rules \( a \leftarrow e \) and \( e \leftarrow a, \delta \)) and \( \{ e \} \vdash f \) (applying

---

*This paper is an extended abstract of an article in the Journal of Artificial Intelligence Research [Caminada and Schulz, 2017].
rules \(a \leftarrow f \cdot a, e\). Apart from the arguments that are constructed using the inference rules, there are also “trivial” arguments that consist of a single assumption. In the example above, these are \(\{\beta\} \vdash \beta, \{\gamma\} \vdash \gamma, \{\delta\} \vdash \delta, \{\epsilon\} \vdash \epsilon\) and \(\{\varphi\} \vdash \varphi\).

The semantics of an ABA framework can be provided using assumption labellings [Schulz and Toni, 2014; 2017]. In essence, the idea is to label each assumption in \(A\) with IN, OUT or UNDEC. An assumption labelling is called a complete assumption labelling iff for each assumption \(x \in A\) it holds that:

- if \(x\) is labelled IN then each ABA argument for conclusion \(x\) has at least one assumption that is labelled OUT
- if \(x\) is labelled OUT then there exists an ABA argument for conclusion \(x\) that has all its assumptions labelled IN
- if \(x\) is labelled UNDEC then there exists an ABA argument for conclusion \(x\) without any assumption labelled OUT and there does not exist an ABA argument for conclusion \(x\) that has all its assumptions labelled IN

Given a (complete) assumption labelling \(Lab\), we write \(IN(\text{Lab})\) for the set of assumptions labelled \(\text{IN}\), \(OUT(\text{Lab})\) for the set of assertions labelled \(\text{OUT}\), and \(UNDEC(\text{Lab})\) for the set of assumptions labelled \(\text{UNDEC}\). We will sometimes write a (complete) assumption labelling \(Lab\) as a triplet \((\text{IN}(\text{Lab}), \text{OUT}(\text{Lab}), \text{UNDEC}(\text{Lab}))\).

In the above example of ABA framework \(F_{ex}\), there are three complete assumption labellings: \(Lab_1 = \{(\gamma), \{\beta\}, \{\delta, \epsilon, \varphi\}\}\), \(Lab_2 = \{(\beta, \delta, \varphi), \{\gamma, \epsilon\}, \emptyset\}\) and \(Lab_3 = (\emptyset, \emptyset, \{\beta, \gamma, \delta, \epsilon, \varphi\})\). A complete assumption labelling \(Lab\) of an ABA framework \(F\) is called:

- a preferred assumption labelling if \(IN(\text{Lab})\) is maximal (w.r.t. \(\subseteq\)) among all complete assumption labellings of \(F\)
- a grounded assumption labelling if \(IN(\text{Lab})\) is minimal (w.r.t. \(\subseteq\)) among all complete assumption labellings of \(F\)
- a semi-stable assumption labelling if \(UNDEC(\text{Lab})\) is minimal (w.r.t. \(\subseteq\)) among all complete assumption labellings of \(F\)
- a stable assumption labelling if \(UNDEC(\text{Lab}) = \emptyset\)
- an ideal assumption labelling if \(IN(\text{Lab})\) is maximal (w.r.t. \(\subseteq\)) among all complete assumption labellings of \(F\) satisfying that for each preferred assumption labelling \(Lab_{\text{pref}}\) of \(F\), \(IN(\text{Lab}) \subseteq IN(\text{Lab}_{\text{pref}})\)

In the example of \(F_{ex}\), \(Lab_1\) is a preferred assumption labelling, \(Lab_2\) is a preferred, semi-stable and stable assumption labelling, and \(Lab_3\) is a grounded and ideal assumption labelling.

### 3 From ABA to LP

Given an ABA framework \(F\), it is possible to define an associated (normal) logic program \(P_F\) by replacing each assumption \(x\) in a rule by its negated (using negation-as-failure) contrary, i.e. by \(\neg x\). As an example, when doing this for the ABA framework \(F_{ex}\), the result is the following logic program \(P_{F_{ex}}\):

\[
\begin{align*}
& a \leftarrow \\
& b \leftarrow \neg c \\
& c \leftarrow \neg b \\
& d \leftarrow \neg c, \neg f \\
& e \leftarrow a, \neg d \\
& f \leftarrow a, \neg e
\end{align*}
\]

Given a logic program \(P\), different semantics can be defined. For current purposes, we apply the concept of a 3-valued interpretation \((T, F)\) where \(T\) and \(F\) are subsets of the atoms occurring in the logic program, with \(T \cap F = \emptyset\). Given a logic program \(P\) and a 3-valued interpretation \((T, F)\), one can define the reduced logic program \(P(T, F)\) by replacing each occurrence of \(\neg x\) by TRUE if \(x \in F\), by FALSE if \(x \in T\) and by UNDEFINED otherwise. The resulting logic program \(P(T, F)\) does not contain any negation-as-failure, so it has a unique minimal 3-valued model [Przymusinski, 1990]. If this unique minimal 3-valued model is equal to \((T, F)\), then \((T, F)\) is said to be a 3-valued stable model of \(P\) [Przymusinski, 1990].

As an example, the logic program \(P_{F_{ex}}\) has three 3-valued stable models: \(Mod_1 = \{\{a, b\}, \{c\}\}\), \(Mod_2 = \{\{a, c, e\}, \{b, f, d\}\}\) and \(Mod_3 = \{\{a\}, \emptyset\}\). To verify that for instance \(Mod_1\) is a 3-valued stable model of \(P_{F_{ex}}\), we observe that \(P_{F_{ex}}^{\{\{a, b\}, \{c\}\}}\) is as follows.

\[
\begin{align*}
& a \leftarrow \\
& b \leftarrow \text{TRUE} \\
& c \leftarrow \text{FALSE} \\
& d \leftarrow \text{TRUE}, \text{UNDEFINED} \\
& e \leftarrow a, \text{UNDEFINED} \\
& f \leftarrow a, \text{UNDEFINED}
\end{align*}
\]

This logic program has the unique minimal 3-valued model \((\{a, b\}, \{c\})\) which is equal to \(Mod_1\), hence \(Mod_1\) is a 3-valued stable model of \(P_{F_{ex}}\). In a similar way, it can be verified that also \(Mod_2\) and \(Mod_3\) are 3-valued stable models of \(P_{F_{ex}}\).

A 3-valued stable model \(Mod = (T, F)\) of logic program \(P\) is called:

- a regular model if \(T\) is maximal (w.r.t. \(\subseteq\)) among all 3-valued stable models of \(P\)
- a well-founded model if \(T\) is minimal (w.r.t. \(\subseteq\)) among all 3-valued stable models of \(P\)
- an L-stable model if \(T \cup F\) is maximal (w.r.t. \(\subseteq\)) among all 3-valued stable models of \(P\)
- a (2-valued) stable model if \(T \cup F\) consists of all the atoms in \(P\)
- an ideal model if \(T\) is maximal (w.r.t. \(\subseteq\)) among all 3-valued stable models of \(P\) satisfying that for each regular model \((T_{reg}, F_{reg})\) of \(P\), \(T \subseteq T_{reg}\)

In the example of \(P_{F_{ex}}\), \(Mod_1\) is a regular model, \(Mod_2\) is a regular, L-stable and (2-valued) stable model, and \(Mod_3\) is a well-founded and ideal model.

One of our main findings is that it is possible convert an assumption labelling \(Lab\) of an ABA framework \(F\) to a 3-valued stable model \(Mod\) of the associated logic program \(P_F\), and vice versa.
To convert an assumption labelling to a 3-valued interpretation, we start by “inverting” the labelling. That is, we construct an interpretation \((T', F')\) where \(T'\) contains the contraries of the assumptions that are \(\text{OUT}\), whereas \(F'\) contains the contraries of the assumptions that are \(\text{IN}\) (with the additional condition that each of these contraries actually occurs in \(P_F\)). However, since we started with assumptions, this will only yield the status of atoms which are contraries of assumptions. In order to obtain the status of all atoms in the logic program (including those that are not the contrary of any assumption in the ABA framework) we perform a simple trick: apply the reduct. That is, we define \(\text{Lab2Mod}(\text{Lab})\) (the function that converts a labelling \(\text{Lab}\) of \(F\) into a 3-valued stable model \(\text{Mod}\) of \(P_F\)) as the unique minimal model of \(P_F^{(T',F')}\).

As an example, consider again \(F_{ex}\), which (as we have seen before) has the complete assumption labellings:

\[
\begin{align*}
\text{Lab}_1 &= \{\{\gamma\}, \{\beta\}, \{\delta, \epsilon, \varphi\}\}, \\
\text{Lab}_2 &= \{\{\beta, \delta, \varphi\}, \{\gamma, \epsilon\}, \emptyset\}, \\
\text{Lab}_3 &= \{\emptyset, \emptyset, \{\beta, \gamma, \delta, \epsilon, \varphi\}\}
\end{align*}
\]

It holds that

- \(\text{Lab2Mod}(\text{Lab}_1) = \{(a, b), \{c\}\} = \text{Mod}_1\) (with \((T', F') = (\{b\}, \{c\})\)),

- \(\text{Lab2Mod}(\text{Lab}_2) = \{(a, c, e), \{b, f, d\}\} = \text{Mod}_2\) (with \((T', F') = (\{c, e\}, \{b, f, d\})\)), and

- \(\text{Lab2Mod}(\text{Lab}_3) = \{(a), \emptyset\} = \text{Mod}_3\) (with \((T', F') = (\emptyset, \emptyset)\)).

In this example, the application of the reduct and minimal model thereof adds atom \(a\) as a true atom to each interpretation \((T', F')\).

It is also possible to go the other way around. That is, given a 3-valued stable model \(\text{Mod}\) of \(P_F\), one can define the associated complete assumption labelling \(\text{Lab}\) of \(F\), using a function called \(\text{Mod2Lab}\). The idea is that:

- the assumptions whose contrary is in \(F\) are labelled \(\text{IN}\)
- the assumptions whose contrary is in \(T\) are labelled \(\text{OUT}\)
- the assumptions whose contrary does occur in \(P_F\) but neither in \(T\) or \(F\) are labelled \(\text{UNDEC}\)
- the assumptions whose contrary does not even occur in \(P_F\) are labelled \(\text{IN}\) as well

As an example, for \(F\) and \(P_F\) it can be observed that

\[\text{Mod2Lab}(\text{Mod}_1) = \text{Lab}_1, \text{Mod2Lab}(\text{Mod}_2) = \text{Lab}_2\]

and

\[\text{Mod2Lab}(\text{Mod}_3) = (\text{Lab}_3)\]

We have proven that when the domain of \(\text{Lab2Mod}\) consists of the complete assumption labellings of \(F\), and the domain of \(\text{Mod2Lab}\) consists of the 3-valued stable models of \(P_F\), \(\text{Lab2Mod}\) and \(\text{Mod2Lab}\) are bijective functions that are each other’s inverse. Moreover,

1. when \(\text{Lab}\) is a preferred assumption labelling of \(F\), \(\text{Lab2Mod}(\text{Lab})\) is a regular model of \(P_F\), and when \(\text{Mod}\) is a regular model of \(P_F\), \(\text{Mod2Lab}(\text{Mod})\) is a preferred assumption labelling of \(F\);

2. when \(\text{Lab}\) is a stable assumption labelling of \(F\), \(\text{Lab2Mod}(\text{Lab})\) is a (2-valued) stable model of \(P_F\), and when \(\text{Mod}\) is a (2-valued) stable model of \(P_F\), \(\text{Mod2Lab}(\text{Mod})\) is a stable assumption labelling of \(F\);

<table>
<thead>
<tr>
<th>assumption labelling of (F)</th>
<th>logic programming model of (P_F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete</td>
<td>3-valued stable</td>
</tr>
<tr>
<td>preferred</td>
<td>regular</td>
</tr>
<tr>
<td>stable</td>
<td>(2-valued) stable</td>
</tr>
<tr>
<td>grounded</td>
<td>well-founded</td>
</tr>
<tr>
<td>ideal</td>
<td>ideal</td>
</tr>
</tbody>
</table>

Table 1: Semantic equivalences when translating ABA to LP

3. when \(\text{Lab}\) is a grounded assumption labelling of \(F\), \(\text{Lab2Mod}(\text{Lab})\) is a well-founded model of \(P_F\), and when \(\text{Mod}\) is a well-founded model of \(P_F\), \(\text{Mod2Lab}(\text{Mod})\) is a grounded assumption labelling of \(F\);

4. when \(\text{Lab}\) is an ideal assumption labelling of \(F\), \(\text{Lab2Mod}(\text{Lab})\) is an ideal model of \(P_F\), and when \(\text{Mod}\) is an ideal model of \(P_F\), \(\text{Mod2Lab}(\text{Mod})\) is an ideal assumption labelling of \(F\).

Overall, the results regarding formal correspondences between assumption labellings and logic programming models are summed up in Table 1.

4 Limitations

For our translation from ABA to LP to work, there need to be a few restrictions on the ABA framework. The first restriction is that the ABA framework should be flat [Toni, 2013]. This means that no assumption appears in the head of an ABA rule. This is important, for if the head of an ABA rule was an assumption, the resulting LP rule would have negation-as-failure in its head, so the result would not be a normal logic program.

The second restriction is that each assumption should have just a single contrary, rather than a set of contraries. This restriction is shared by most of the ABA literature (like [Bondeirenko et al., 1997; Dung et al., 2009; 2007; Toni, 2014; Schulz and Toni, 2014]), although some works on ABA (like [Gaertner and Toni, 2007; 2008; Fan and Toni, 2014; 2015]) allow an assumption to have more than one contrary. Our current translation from ABA to LP would not be well-defined in the latter case.

The third restriction is that the contrary of an assumption should not be an assumption itself. Although this restriction is not required for the well-definedness of the resulting logic program, examples exist where violating this restriction results in breaking the link between the assumption labellings of the ABA framework and the logic programming models of the associated logic program.

For the second and third restriction, workarounds exist. For instance, it is possible to convert an ABA framework where assumptions have multiple contraries to an equivalent ABA framework\(^1\) where each assumption has a single contrary [Gaertner and Toni, 2008]. Furthermore, it is possible to

\(^1\)With an equivalent ABA framework we mean an ABA framework that has the same complete (resp. preferred, semi-stable, stable, grounded and ideal) assumption labellings.
convert an ABA framework where the contrary of an assumption may be an assumption to an equivalent ABA framework where this is not the case. Details of the respective translations can be found in the full paper. For an ABA framework that does not comply with the second or third restriction, the workaround idea is to first translate it to an equivalent ABA framework that does comply with the restriction, and then translate this to the associated logic program. The logic programming models (2-valued) stable, well-founded and ideal semantics, respectively) of this logic program will coincide with the assumption labellings of the original ABA framework as previously described.

Another type of restriction is regarding the semantics applied to the ABA framework. Although our translation works for complete, preferred, stable, grounded and ideal assumption labellings (which coincide with the 3-valued stable, regular, (2-valued) stable, well-founded and ideal models, respectively, of the associated logic program through the functions Lab2Mod and Mod2Lab) it is not the case that the semi-stable assumption labellings of an ABA framework coincide with the L-stable models of the associated logic program. However, it can be observed that semi-stable assumption labellings do coincide with L-stable models when the ABA framework is assumption-spanning, that is, when each non-assumption in the language is the contrary of an assumption.

5 From LP to ABA

Apart from the translation from ABA to LP, it is also possible to translate from LP to ABA, as investigated in previous work [Bondarenko et al., 1997; Schulz and Toni, 2015; 2016]. The idea is that, given a logic program, one can translate it to an ABA framework by replacing each occurrence of not \( x \) by an assumption not \( x \).

For instance, for the logic program \( P_{ex} \) mentioned above, the corresponding ABA framework is as follows,

\[
\begin{align*}
\text{a} & \leftarrow \text{not}_c \\
\text{b} & \leftarrow \text{not}_c \\
\text{c} & \leftarrow \text{not}_b \\
\text{d} & \leftarrow \text{not}_c, \text{not}_f \\
\text{e} & \leftarrow \text{a}, \text{not}_d \\
\text{f} & \leftarrow \text{a}, \text{not}_e
\end{align*}
\]

with the set of assumptions in the underlying language being \{not\_a, not\_b, not\_c, not\_d, not\_e, not\_f\}. Note that the assumption not\_a does not occur in any of the ABA rules.

We observe that when translating a logic program to an ABA framework and then back to a logic program, the final logic program is precisely the same as the one at the start. This makes it possible to reuse part of our existing technical results from the “ABA to LP” direction to apply to the “LP to ABA” direction (see the full paper for details). In particular, it turns out that the various types of models of a logic program \( P \) coincide (through the functions Mod2Lab and Lab2Mod) with the various types of assumption labellings of the associated ABA framework \( F_P \), thus generalising existing work [Bondarenko et al., 1997; Schulz and Toni, 2015; 2016]. An overview of the semantic correspondence is provided in Table 2.

As indicated in Table 2, for the “LP to ABA” direction, we do obtain equivalence between L-stable models and semi-stable assumption labellings (which is not the case for the “ABA to LP” direction, see Table 1). This is because translating a logic program results in an ABA framework that is assumption-spanning.

6 Discussion

In this work we re-examined the relationship between ABA and LP and found that the most frequently studied fragment of ABA, namely flat ABA frameworks, does not only subsume normal logic programming as previously shown [Bondarenko et al., 1997; Toni, 2007; 2008; Schulz and Toni, 2015], but is in fact also itself subsumed by normal logic programming through a straightforward translation. That is, flat ABA can be seen as LP with a slightly different syntax since the outcome that is yielded by a flat ABA framework under common ABA semantics (that is: complete, grounded, preferred, stable and ideal) is essentially the same as the outcome yielded by its (syntactically nearly identical) associated normal logic program under common LP semantics, and vice versa. The difference is that in ABA the outcome is defined in terms of assumptions (which correspond to negation-as-failure atoms in the associated logic program) whereas in LP the outcome is defined in terms of atoms in the logic program. However, since in LP the status of the atoms is determined solely by the status of the negation-as-failure atoms, flat ABA and normal LP are semantically equivalent. As a by-product of our work, we prove that the ideal semantics for ABA is not just inspired by, but coincides with, the ideal scenario semantics for LP [Alferes et al., 1993].

ABA methods have already been successfully applied to LP, e.g. for explaining [Schulz and Toni, 2016] as well as visualizing [Schulz, 2015] logic programs under certain semantics. Our results pave the way for the application of LP methods to (flat) ABA frameworks, since our translation prevents an exponential blow-up while preserving the semantics. Efficient computation methods for LP semantics [Janhunen et al., 2006; Gebser et al., 2011] could for instance be used to determine the semantics of ABA frameworks, which is a promising direction for future work.
References


