Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

ABALearn: An automated system for learning logic-based argumentation

Author:
Cristina-Gabriela Tirsi

Supervisors:
Prof. Francesca Toni
Maurizio Proietti (external)

Second Marker:
Prof. Alessandra Russo

June 28, 2023

Abstract

We introduce ABALearn, an automated algorithm that learns ABA frameworks from training data consisting of positive and negative examples, and a given background knowledge. ABALearn's ability to generate comprehensible rules for decision-making promotes transparency and interpretability, addressing the challenges associated with the black-box nature of traditional machine learning models. This implementation is based on the strategy proposed in a previous work. The resulting ABA frameworks can be mapped onto logic programs with negation as failure. The main advantage of this algorithm is that it requires minimal information about the learning problem and it is also capable of learning circular debates. Our results show that this approach is competitive with state-of-the-art alternatives, demonstrating its potential to be used in real-world applications. Overall, this work contributes to the development of automated learning techniques for argumentation frameworks in the context of Explainable AI (XAI) and provides insights into how such learners can be applied to make predictions.

Acknowledgements

I would like to express my gratitude for my supervisors Prof. Francesca Toni and Maurizio Proietti for all the guidance and help they have offered me throughout this project. I would have not been able to bring this exciting piece of work anywhere close to its current state without their constant support.

I am also grateful for all the academics I have had the honor to learn from throughout my degree and have provided me with invaluable knowledge and skills, which are a big part of the reason I could even navigate through working on this project.

Lastly, I want to thank my family for the unconditional trust, love and support and I want to thank my friends for making this degree enjoyable and for pushing me to always do better.

Contents

Chapter 1

Introduction

1.1 Motivation

Within the field of artificial intelligence (AI), *computational argumentation* is a very powerful tool as it can be used to improve the performance and explainability of AI systems [1]. In general, argumentation is valuable through its interdisciplinary applications: legal and practical reasoning, game-theory, decision-theory, philosophy, etc.. It uses formal logic to manipulate and reason about arguments, which provides a clear and comprehensible explanation for the reasoning behind a conclusion [2]. When it comes to AI driven by data, standard machine learning (ML) systems easily become very expensive, they require huge amounts of data and it is difficult to inject human knowledge in them. On the other hand, logic-based learning works with less data, human knowledge can be easily injected, and, most importantly, it is explainable [3].

Assumption-Based Argumentation (ABA) is a popular structured argumentation used for knowledge representation and reasoning [4]. In ABA frameworks, the focus is on the assumptions underlying arguments built from rules, rather than the arguments themselves, and it uses contraries of assumptions to determine attacks between arguments. The acceptability of an argument is determined by the acceptability of its assumptions. It can capture a form of abductive logic programs, given that it accepts assumptions through negation as failure and, as highlighted in [5], there is a correlation between abduction and negation as failure (by using negations as hypotheses and then applying abduction, its semantics characterises the stable model semantics).
In this sense, *ABA learning* [6] (a novel methodology to learn ABA frameworks) has been proposed as a solution to support logic-based learning through the means of computational argumentation.

The main idea of ABA learning is that it uses a given background knowledge (made up of rules, assumptions and their contraries), a set of positive examples and a set of negative examples to extend the knowledge [6]. This is achieved through manipulating the background knowledge by altering the existing rules or introducing new ones using *transformation rules* which dictate how to perform these changes given that some preconditions are satisfied. More specifically, [6] proposes five such transformation rules (rote learning, equality removal, folding, assumption introduction and subsumption) along with a strategy that repeatedly applies these transformation rules, until the objectives of ABA learning are achieved.

ABA learning differs from related logic-based methods through its approach to *exceptions*. The FOLD algorithm, as proposed in [7] (as well as FOLD-R++ and FOLD-RM [8]), learns rules for exceptions (enabled by switching the labelling of the exceptional examples from positive to negative, and vice-versa). This ultimately results in stratified logic programs that allow the use of negation-as-failure.

Similarly, another proposed method is the one in [9], which also uses the interpolation of the positive and negative information. Additionally, this approach makes use of recursion and is interested in learning the negative concept symmetrically to the positive one. To illustrate, a rule like ‘if it is June and we are not in Australia, then it is summer‘ is transformed into ‘if it is June, then it is summer‘ and ‘if we are in Australia, then it is not summer‘. This transformation is motivated by viewing ‘we are not in Australia‘ as ‘unless we are in Australia‘.
ABA learning, on the other hand, makes use of assumptions and handles exceptions by creating conflicts that stop the default argument from applying in exceptional cases. This approach allows for learning ABA frameworks that can be mapped to *non-stratified logic programs*. This tactic closely mimics the human-like way of reasoning, where you can argue about a statement being true as long as there is no reason to believe the assumptions it is based on might be wrong. For example, when we say ‘if the restaurant is open, then we can have dinner there‘, this is a reasonable statement to make and is generally valid. However, if it is proven that the restaurant is fully booked, the deduction is not valid anymore, as it was based on the assumption that we could get a table at the restaurant (even though we did not explicitly mention it). This way of debating is generally more efficient, as it only requires reasoning about a minimal number of sentences.

1.2 Objectives

We are interested in exploring ABA learning in various ways.

The main objective is to investigate if there is a way to automate an ABA learning process. This represents the main focus of this project, as accomplishing to propose such an automated methodology enables us to further expand on its potential.

The second objective would be concerned with proving the correctness of such an automation. As with any other logic-based learning system, we are essentially modelling a human-like process of argumentation, which is, of course, not trivial. Thus, it is important that we justify why our algorithm does, in fact, manage to satisfy its objectives.

Lastly, we are interested in assessing how the implementation of this algorithm compares with other existing learners. In this sense, we are interested in looking at performance in terms of execution time and scalability, in measuring how accurate the returned solutions are, in discussing the cognitive complexity of each and to explore their range of applicability in terms of different types of learning tasks they can solve.

1.3 Contributions

To accomplish our main objective, we will derive a deterministic algorithm starting from the strategy proposed in [6], along with its implementation. This will also represent the biggest challenge of this project, as the initial strategy includes a number of steps that suggest multiple possibilities of proceeding, with no means to drive the decision based on the current state of the learning process.

Secondly, we want to provide an informal proof that allows us to safely conclude that this algorithm does achieve its goals. This informal proof will be presented as a discussion of the algorithm's correctness.

Finally, we will present an evaluation of this system against FOLD-RM and ILASP (two other related learners). We will look at user requirements from both a technical and a cognitive perspective, we will measure and compare execution times for inputs of various sizes in order to understand how each system scales, we will measure the accuracy of solutions for multiple inputs and, lastly we are interested in experimenting with learning tasks that capture different special cases.

1.4 Report structure

In Chapter 2, we briefly outline some ethical considerations that should be taken with regards to this work.

In Chapter 3, we will introduce all the theoretical aspects related to ABA frameworks that are required for this project, we will present the correspondence between ABA frameworks and logic programs, we will explain what ABA learning represents, and we will explain the previously proposed learning strategy, and, lastly, we will briefly outline how other related learners operate.

In Chapter 4, we present in detail the ABALearn system which is an implementation of a deterministic version of the proposed strategy, and we discuss challenges we have encountered in its development.

In Chapter 5, we conduct a discussion about the correctness of the algorithm. This is intended to informally prove that all of the algorithm's objectives are satisfied.

In Chapter 6, we analyse how ABALearn performs in comparison with two other learners. This includes empirical evaluation through assessing different metrics, as well as observational remarks.

In Chapter 7, we propose some possible directions in which the work on this project can be continued, addressing both limitations and low-priority improvements.

Chapter 2

Ethical discussion

The project is meant to explore and exemplify the power of computational argumentation. Under reasonable assumptions, it should not give rise to any ethical issues. However, it does fall under the field of Artificial Intelligence (AI). That being said, as with most AI systems, misuse can occur.

The product of this project is not meant to be used in scenarios of high importance (e.g. should not be used to decide the innocence of a person in a juridical setting, should not be used to diagnose patients, etc.). Its explainable nature should, however, partially mitigate a good amount of potential misuse, given that the arguments it generates are based on a human-readable set of rules, which means the user can use their own reasoning to decide if the output is safe to be trusted.

Given that our approach relies on labelled data (i.e. the training data is provided as positive and negative examples) to acquire knowledge, this gives rise to a number of concerns. There could be issues regarding their provenance, as it is essential to ensure reliability and trustworthiness of the data sources, avoiding bias in the learning process which can impact the outcome. Additionally, permission to use the data must be obtained from the relevant stakeholders, while complying with legal and ethical requirements. Another standard issue related to data is that of privacy. In cases where the training data contains sensitive or personal information, we must adhere to data protection regulations, employing anonymisation techniques. Thus, it is fundamental that all of these considerations are addressed in order to promote responsible use of data, minimising potential harms.

That being said, it is crucial to maintain a balance between the product's autonomy and human agency, ensuring that the human decision making is enhanced, rather than replaced. This requires human oversight in order to intervene, challenge, and contribute to the debates facilitated by the learner.

Chapter 3

Background

We will present in this chapter the background material used throughout the report.

3.1 Assumption-based argumentation frameworks

The core of this entire project is represented by assumption-based argumentation (ABA) frameworks and how deductions are carried out within them. For a good comprehension of this report, it is essential that a few theoretical concepts are well understood. In this section we aim to formally define those concepts.

According to [10, 6, 4], an ABA framework is a tuple $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ where

- $\langle \mathcal{L}, \mathcal{R} \rangle$ is a *deductive system*, with $\mathcal{L}$ being the **language** and $\mathcal{R}$ a set of (inference) **rules**;
- $\mathcal{A} \subseteq \mathcal{L}$ is a (*non-empty*) set of **assumptions**;
- $\overline{}$ is a *total mapping* from $\mathcal{A}$ into $\mathcal{L}$, where $\overline{a}$ is the **contrary** of a, for $a \in \mathcal{A}$.

Rules in $\mathcal{R}$ can have various representation, but for this project we will represent them as $\rho: \ h \leftarrow b_1, ..., b_n \ (n \geq 0, h \in \mathcal{L}$ and $b_i \in \mathcal{L}$, for $1 \leq i \leq n)$, where h is the head, $b_0, ..., b_n$ is the body and ρ is the identifier of the rule. If $n = 0$, then it is said that h is a *fact*.

The elements of $\mathcal{L}$ can be any sentences, but in this paper we will only focus on *ground atoms*. For sake of simplicity, however, we will use *schemata* for rules, contraries and assumptions. We will also assume $\mathcal{L}$ is *finite*.

Example 1 (ABA Framework)**.** Consider the following components:
$\mathcal{L} = \{p(X), q(X), r(X), s(X), t(X), v(X) \mid X \in \{a, b\}\}$
$\mathcal{R} = \{\rho_1 : p(X) \leftarrow s(X),$
$\rho_2 : q(X) \leftarrow r(X),$
$\rho_3 : q(X) \leftarrow t(X),$
$\rho_4 : t(b) \leftarrow s(b), v(b),$
$\rho_5 : t(a) \leftarrow,$
$\rho_6 : v(b) \leftarrow$
$\}$
$\mathcal{A} = \{r(X), s(X)\}$
$\overline{r(X)} = p(X) \quad \overline{s(X)} = q(X)$
Then, $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ is an ABA framework.

Note. As mentioned above, we are using schemata, therefore, for example, $r(X)$ is actually shorthand for $r(a)$ and $r(b)$.

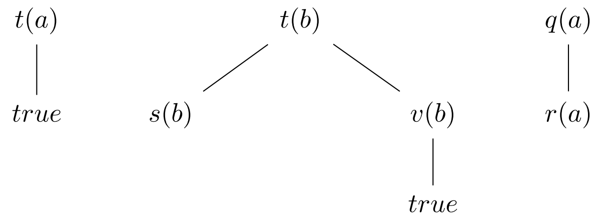


Figure 3.1: Arguments $\{\} \vdash_{\{\rho_5\}} t(a)$, $\{s(b)\} \vdash_{\{\rho_4, \rho_6\}} t(b)$, and, respectively, $\{r(a)\} \vdash_{\{\rho_2\}} q(a)$ represented as trees.

To illustrate examples for the upcoming definitions, we will consider the ABA framework defined in Example 1.

As in any type of argumentation, we are interested in how we can reason about sentences and how to decide when a claim is valid or not. For this purpose, we need to define what an argument represents.

Definition 1 (Argument)**.** An **argument** for (the claim) $\sigma \in \mathcal{L}$ supported by $A \subseteq \mathcal{A}$ and $R \subseteq \mathcal{R}$ (denoted $A \vdash_R \sigma$) is a finite tree with nodes labelled by sentences in $\mathcal{L}$ or by *true*, the root labelled by σ, leaves either *true* or assumptions in A, non-leaves σ' with, as children, the elements of the body of some rule in R with head σ', and A is the set of all assumptions labelling the leaves [11].

Example 2. Figure 3.1 shows trees illustrating some of the arguments in the ABA framework defined in Example 1.

Naturally, based on this definition, for any $\alpha \in \mathcal{A}$, the argument $\{\alpha\} \vdash_{\{\}} \alpha$ is always valid. However, we would be interested in keeping the acceptance of assumptions independent, and avoid cases where assuming a sentence forces us to accept another assumption (e.g. we do not want to deal with arguments of the form $\{\alpha\} \vdash_R \beta$ where $\alpha, \beta \in \mathcal{A}$). This preference allows us to have ABA frameworks that can be mapped onto logic programs (as we will see in Section 3.2). Formally, this restriction refers to *flat* ABA frameworks, defined as follows:

Definition 2 (Closed sets of assumptions)**.** A set of assumptions $A \subseteq \mathcal{A}$ is **closed** iff
$A = \{\alpha \in \mathcal{A} \mid \exists A' \vdash_R \alpha, A' \subseteq A, R \subseteq \mathcal{R}\}$ [4].

Definition 3 (Flat ABA framework)**.** An ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ is **flat** iff for every $A \subseteq \mathcal{A}$, A is *closed* [12].

Trivially, if we do not allow any assumption to be a conclusion (i.e. head of a rule), we are guaranteed to have a flat ABA framework. [13]

Note. The ABA framework defined in Example 1 is a flat ABA framework.

For the rest of the report, we will only consider flat ABA frameworks.
For convenience, we will also define the set *Args*.

Definition 4. Given a flat ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$, *Args* is the set of all arguments:
For all $\sigma \in \mathcal{L}, A \subseteq \mathcal{A}$ *and* $R \subseteq \mathcal{R}$, if $A \vdash_R \sigma$, then $A \vdash_R \sigma \in Args$

Example 3. Given the ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ defined above, the set of all arguments is:

$$Args = \{\{s(X)\} \vdash_{\{\rho_1\}} p(X), \quad \{r(X)\} \vdash_{\{\rho_2\}} q(X), \quad \{s(b)\} \vdash_{\{\rho_4,\rho_6\}} t(b),$$
$$\{\} \vdash_{\{\rho_5\}} t(a), \quad \{\} \vdash_{\{\rho_5,\rho_3\}} q(a), \quad \{s(b)\} \vdash_{\{\rho_6,\rho_4,\rho_3\}} q(b),$$
$$\{\} \vdash_{\{\rho_6\}} v(b), \quad \{r(X)\} \vdash_{\{\}} r(X), \quad \{s(X)\} \vdash_{\{\}} s(X)\}$$

In ABA, we are mostly concerned with *undercutting attacks* (that is, arguments that attack another argument and stop it from applying by disagreeing on the supporting assumptions). Attacks by rebuttal can also be represented in ABA, where arguments disagree on their claim (as shown in [14]), but undercutting attacks are stronger, so the preference is for the latter. Thus, for the rest of this report when the term *'attack'* is used, it refers to undercutting attacks, which are defined as follows:

Definition 5 (Attacks)**.** Argument $A_1 \vdash_{R_1} \sigma_1$ **attacks** argument $A_2 \vdash_{R_2} \sigma_2$ iff $\overline{a} = \sigma_1$ for some $a \in A_2$. [10]

As the set of rules required to build an argument is not relevant in deciding whether the argument attacks another or not, for sake of simplicity, for the rest of the report we will exclude R from the representation of the argument (i.e. $A \vdash_R \sigma$ becomes $A \vdash \sigma$).

Example 4. The attacks between the arguments defined above are:
$\{s(X)\} \vdash p(X)$ attacks $\{r(X)\} \vdash q(X)$, because $\overline{r(X)} = p(X)$
$\{s(X)\} \vdash p(X)$ attacks $\{r(X)\} \vdash r(X)$, because $\overline{r(X)} = p(X)$
$\{r(X)\} \vdash q(X)$ attacks $\{s(X)\} \vdash p(X)$, because $\overline{s(X)} = q(X)$
$\{r(b)\} \vdash q(b)$ attacks $\{s(b)\} \vdash t(b)$, because $\overline{s(b)} = q(b)$
$\{r(b)\} \vdash q(b)$ attacks $\{s(b)\} \vdash q(b)$, because $\overline{s(b)} = q(b)$
$\{r(X)\} \vdash q(X)$ attacks $\{s(X)\} \vdash s(X)$, because $\overline{s(X)} = q(X)$
$\{\} \vdash q(a)$ attacks $\{s(a)\} \vdash p(a)$, because $\overline{s(a)} = q(a)$
$\{\} \vdash q(a)$ attacks $\{s(a)\} \vdash s(a)$, because $\overline{s(a)} = q(a)$
$\{s(b)\} \vdash q(b)$ attacks $\{s(b)\} \vdash p(b)$, because $\overline{s(b)} = q(b)$
$\{s(b)\} \vdash q(b)$ attacks $\{s(b)\} \vdash t(b)$, because $\overline{s(b)} = q(b)$
$\{s(b)\} \vdash q(b)$ attacks $\{s(b)\} \vdash q(b)$, because $\overline{s(b)} = q(b)$
$\{s(b)\} \vdash q(b)$ attacks $\{s(b)\} \vdash s(b)$, because $\overline{s(b)} = q(b)$

Similarly to human reasoning, we can debate about facts based on assumptions, and that is a perfectly valid way to argue about them. But if we are proven our assumptions were wrong, the arguments fail. For example, say someone is accused of not following the law. According to a very well known legal principle, it is assumed that the person is innocent, unless proven guilty. This gives rise to two scenarios: one where they have been proven guilty so they deserve punishment, and one where nothing has been proven so they are free to go. These scenarios have a formal correspondence in ABA, and they are defined as follows:

Definition 6 (Stable Extension)**.** Given a flat ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$, let $Att = \{(\alpha, \beta) \in Args \times Args \mid \alpha \textit{ attacks } \beta\}$, for '$Args$' and 'attacks' as defined in Definition 4 and Definition 5, respectively. Then, $S \subseteq Args$ is a **stable extension** iff
(i) $\nexists \alpha, \beta \in S$ such that $(\alpha, \beta) \in Att$ (i.e. S is conflict-free)
(ii) $\forall \beta \in Args \setminus S, \exists \alpha \in S$ *such that* $(\alpha, \beta) \in Att$ (i.e. S "attacks" all arguments it does not contain) [12, 6].

Example 5. For easier visualisation, we can build a directed graph of attacks that occur in the given ABA framework, where $arg_1 \rightarrow arg_2$ means that arg_1 attacks arg_2, as in Figure 3.2.

The given ABA framework only accepts the following stable extension:
$$S = \{\{\} \vdash q(a), \quad \{r(a)\} \vdash q(a), \quad \{r(a)\} \vdash r(a), \quad \{r(b)\} \vdash r(b),$$
$$\{r(b)\} \vdash q(b), \quad \{\} \vdash v(b), \quad \{\} \vdash t(a)\}$$

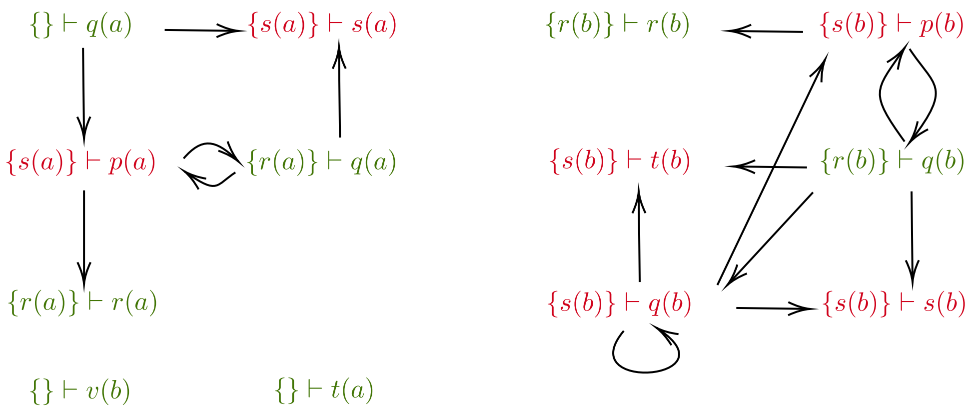


Figure 3.2: Attacks represented as a directed graph.
Arguments coloured green are part of the stable extension.
Arguments coloured red are *not* part of the stable extension.

However, as in any debate, when an argument that was made based on an assumption is proven wrong through an argument that suggests that the premise was false, in turn we might be able to back up our initial argument by showing that it was wrongly invalidated. This kind of defence is formally defined below.

Definition 7 (Defence)**.** A set $A \subseteq Args$ **defends** an argument $b \in Args$ iff $\forall c \in Args$, if c attacks b, then $\exists a \in A$ such that a attacks c. [10]

Example 6. These are the defences that occur in the given ABA framework:

1. No argument attacks $\{\} \vdash t(a)$.
   Therefore, $\{\}$ defends $\{\} \vdash t(a)$.
2. No argument attacks $\{\} \vdash q(a)$.
   Therefore, $\{\}$ defends $\{\} \vdash q(a)$.
3. No argument attacks $\{\} \vdash v(b)$.
   Therefore, $\{\}$ defends $\{\} \vdash v(b)$.
4. $\{r(a)\} \vdash q(a)$ is attacked by $\{s(a)\} \vdash p(a)$.
   $\{s(a)\} \vdash p(a)$ is attacked by $\{\} \vdash q(a)$.
   Therefore, $\{\{\} \vdash q(a)\}$ defends $\{r(a)\} \vdash q(a)$.
5. $\{r(a)\} \vdash q(a)$ is attacked by $\{s(a)\} \vdash p(a)$.
   $\{s(a)\} \vdash p(a)$ is attacked by $\{r(a)\} \vdash q(a)$.
   Therefore, $\{\{r(a)\} \vdash q(a)\}$ defends $\{r(a)\} \vdash q(a)$.
6. $\{r(a)\} \vdash r(a)$ is attacked by $\{s(a)\} \vdash p(a)$.
   $\{s(a)\} \vdash p(a)$ is attacked by $\{\} \vdash q(a)$.
   Therefore, $\{\{\} \vdash q(a)\}$ defends $\{r(a)\} \vdash r(a)$.
7. $\{r(a)\} \vdash r(a)$ is attacked by $\{s(a)\} \vdash p(a)$.
   $\{s(a)\} \vdash p(a)$ is attacked by $\{r(a)\} \vdash q(a)$.
   Therefore, $\{\{r(a)\} \vdash q(a)\}$ defends $\{r(a)\} \vdash r(a)$.
8. $\{r(b)\} \vdash r(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
   $\{s(b)\} \vdash p(b)$ is attacked by $\{s(b)\} \vdash q(b)$.
   Therefore, $\{\{s(b)\} \vdash q(b)\}$ defends $\{r(b)\} \vdash r(b)$.
9. $\{r(b)\} \vdash r(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
   $\{s(b)\} \vdash p(b)$ is attacked by $\{r(b)\} \vdash q(b)$.
   Therefore, $\{\{r(b)\} \vdash q(b)\}$ defends $\{r(b)\} \vdash r(b)$.

10. $\{s(b)\} \vdash p(b)$ is attacked by $\{r(b)\} \vdash q(b)$ and by $\{s(b)\} \vdash q(b)$.
$\{r(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
$\{s(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash q(b)$.
Therefore, $\{\{s(b)\} \vdash p(b), \quad \{s(b)\} \vdash q(b)\}$ defends $\{s(b)\} \vdash p(b)$.

11. $\{s(b)\} \vdash p(b)$ is attacked by $\{r(b)\} \vdash q(b)$ and by $\{s(b)\} \vdash q(b)$.
$\{r(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
$\{s(b)\} \vdash q(b)$ is attacked by $\{r(b)\} \vdash q(b)$.
Therefore, $\{\{s(b)\} \vdash p(b), \quad \{r(b)\} \vdash q(b)\}$ defends $\{s(b)\} \vdash p(b)$.

12. $\{r(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
$\{s(b)\} \vdash p(b)$ is attacked by $\{s(b)\} \vdash q(b)$.
Therefore, $\{\{s(b)\} \vdash q(b)\}$ defends $\{r(b)\} \vdash q(b)$.

13. $\{r(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
$\{s(b)\} \vdash p(b)$ is attacked by $\{r(b)\} \vdash q(b)$.
Therefore, $\{\{r(b)\} \vdash q(b)\}$ defends $\{r(b)\} \vdash q(b)$.

14. $\{s(b)\} \vdash t(b)$ is attacked by $\{r(b)\} \vdash q(b)$ an by $\{s(b)\} \vdash q(b)$.
$\{r(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
$\{s(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash q(b)$.
Therefore, $\{\{s(b)\} \vdash p(b), \quad \{s(b)\} \vdash q(b)\}$ defends $\{s(b)\} \vdash t(b)$.

15. $\{s(b)\} \vdash t(b)$ is attacked by $\{r(b)\} \vdash q(b)$ an by $\{s(b)\} \vdash q(b)$.
$\{r(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
$\{s(b)\} \vdash q(b)$ is attacked by $\{r(b)\} \vdash q(b)$.
Therefore, $\{\{s(b)\} \vdash p(b), \quad \{r(b)\} \vdash q(b)\}$ defends $\{s(b)\} \vdash t(b)$.

16. $\{s(b)\} \vdash q(b)$ is attacked by $\{r(b)\} \vdash q(b)$ an by $\{s(b)\} \vdash q(b)$.
$\{r(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
$\{s(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash q(b)$.
Therefore, $\{\{s(b)\} \vdash p(b), \quad \{s(b)\} \vdash q(b)\}$ defends $\{s(b)\} \vdash q(b)$.

17. $\{s(b)\} \vdash q(b)$ is attacked by $\{r(b)\} \vdash q(b)$ an by $\{s(b)\} \vdash q(b)$.
$\{r(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
$\{s(b)\} \vdash q(b)$ is attacked by $\{r(b)\} \vdash q(b)$.
Therefore, $\{\{s(b)\} \vdash p(b), \quad \{r(b)\} \vdash q(b)\}$ defends $\{s(b)\} \vdash q(b)$.

18. $\{s(b)\} \vdash s(b)$ is attacked by $\{r(b)\} \vdash q(b)$ an by $\{s(b)\} \vdash q(b)$.
$\{r(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
$\{s(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash q(b)$.
Therefore, $\{\{s(b)\} \vdash p(b), \quad \{s(b)\} \vdash q(b)\}$ defends $\{s(b)\} \vdash s(b)$.

19. $\{s(b)\} \vdash s(b)$ is attacked by $\{r(b)\} \vdash q(b)$ an by $\{s(b)\} \vdash q(b)$.
$\{r(b)\} \vdash q(b)$ is attacked by $\{s(b)\} \vdash p(b)$.
$\{s(b)\} \vdash q(b)$ is attacked by $\{r(b)\} \vdash q(b)$.
Therefore, $\{\{s(b)\} \vdash p(b), \quad \{r(b)\} \vdash q(b)\}$ defends $\{s(b)\} \vdash s(b)$.

Furthermore, with the same set of arguments we might be able to defend multiple arguments. Thus, we define:

Definition 8 (Defence set)**.** Given a set $A \subseteq Args$, the set of all the arguments it defends is denoted $Def(A)$ and is defined as follows:
$Def(A) = \{a \in Args \mid A \ defends \ a\}$ [12].

To continue our previous example, say the person was accused of not paying for parking, and it has been shown that they did indeed not pay, which makes them guilty. However, the person shows they have a permit which grants them free parking. In this case, as this is a factual argument (i.e. not based on any assumption), we now have a single valid scenario, and that is that the person has this permit and is innocent.

This type of scenario corresponds to the *grounded extension* of an ABA framework. Based on Definition 8, we notice that Def is monotonic, which, according to the Kleene fixed-point theorem, guarantees that Def admits a *unique* least fixed point, which is equal to $\bigcup_{i=0}^{\omega}\{Def^i(\emptyset)\}$, for some ordinal number ω [12]. It has been shown by Dung in [15] that this least fixed point corresponds to the grounded extension. Thus, we define the grounded extension as follows:

Definition 9 (Grounded Extension)**.** The **grounded extension** of an ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ is the *least fixed point of* Def.

Example 7. We start with $\omega = 0$:
$\bigcup_{i=0}^{0}\{Def^i(\emptyset)\} = \emptyset$

$\omega = 1$:
$\bigcup_{i=0}^{1}\{Def^i(\emptyset)\} = \emptyset \cup Def(\{\}) = \{\{\} \vdash t(a), \quad \{\} \vdash q(a), \quad \{\} \vdash v(b)\}$

$\omega = 2$ *(using items 4 and 6 from Example 6)*:
$\bigcup_{i=0}^{2}\{Def^i(\emptyset)\} = \bigcup_{i=0}^{1}\{Def^i(\emptyset)\} \cup Def(Def(\{\})) =$
$\{\{\} \vdash t(a), \quad \{\} \vdash q(a), \quad \{\} \vdash v(b)\} \cup \{\{r(a)\} \vdash q(a), \quad \{r(a)\} \vdash r(a)\}.$

$\omega = 3$:
$\bigcup_{i=0}^{3}\{Def^i(\emptyset)\} = \bigcup_{i=0}^{2}\{Def^i(\emptyset)\} \cup Def(Def(Def(\{\}))) = \bigcup_{i=0}^{2}\{Def^i(\emptyset)\} \cup \emptyset,$

i.e. $Def(Def(\{\}))$ does not defend anything that $Def(\{\})$ does not already defend.

Therefore, the grounded extension is:
$G = \bigcup_{i=0}^{2}\{Def^i(\emptyset)\} =$
$\{\{\} \vdash t(a), \quad \{\} \vdash q(a), \quad \{\} \vdash v(b), \quad \{r(a)\} \vdash q(a), \quad \{r(a)\} \vdash r(a)\}.$

Referring back to Figure 3.2, the grounded extension consists of the green arguments in the left half of the figure. Every argument in the right half of the figure is considered undecided, as all of them are attacked.

Theorem 1. For any flat ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$, there is a unique grounded extension and 0, 1 or multiple stable extensions [12].

Theorem 2. Any grounded extension accepted by an ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ is a subset of every stable extension accepted by $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ [4].

Theorem 3. If an ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ has a non-empty grounded extension, then it has at least one stable extension [4].

These extensions allow us to decide whether a sentence is accepted or not within their corresponding semantics.

Definition 10 (Coverage under grounded semantics)**.** An ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ is said to **cover** a sentence $\sigma \in \mathcal{L}$ *under the grounded semantics* (denoted as $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle \models_G \sigma$) if σ is the claim of an argument $\alpha \in G$, where G is the grounded extension of $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$.

Example 8. Given G as defined in Example 7, $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ covers all the given facts in $\mathcal{R}$ (i.e. $t(a)$, $q(a)$, $v(b)$), as well as $r(a)$.

Definition 11 (Coverage under stable semantics)**.** In this case we have two types of coverage:

- **Sceptical (cautious) semantics:** An ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ is said to *sceptically (cautiously)* **cover** a sentence $\sigma \in \mathcal{L}$ under the stable semantics if, *for every* stable extension $\mathcal{S}$ of $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$, σ is the claim of an argument $\alpha \in \mathcal{S}$.

- **Credulous (brave) semantics:** An ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ is said to *credulously (bravely)* **cover** a sentence $\sigma \in \mathcal{L}$ under the stable semantics if, there is a stable extension $\mathcal{S}$ of $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$, such that σ is the claim of an argument $\alpha \in \mathcal{S}$.

Example 9. In our case, as we have a unique stable extension (see Example 5), the covered sentences under sceptical semantics coincide with the ones covered under credulous semantics and they are: $q(a), q(b), r(a), r(b), t(a)$ and $v(b)$.

Definition 12 (Not covered sentences)**.** If a sentence is not the claim of any argument accepted under the chosen semantics, then we say the sentence is **not covered**.

3.2 Correspondences between assumption-based argumentation frameworks and logic programs

Given an ABA framework, if:

- it is *flat*;
- $\mathcal{L}$ is a set of *atoms*;
- $\overline{}$ is a *one-to-one* mapping (i.e. there are no two distinct assumptions that are mapped to the same contrary);
- none of the contraries are assumptions (i.e. $\forall \alpha \in \mathcal{A}, \overline{\alpha} \notin \mathcal{A}$);

then there is a *one-to-one* mapping between *ABA frameworks* and *logic programs.*

To transform an ABA framework to a logic program:
For all contraries $\overline{\alpha(X)} = p(X)$, replace $\alpha(X)$ with *not p(X)*, in the set of rules $\mathcal{R}$ and remove it from the language $\mathcal{L}$ to obtain the Herbrand Base.
To transform a logic program to an ABA framework:
For every negated atom in the logic program *not p(X)*, replace it with a new atom $\alpha(X)$, add $\alpha(X)$ to the Herbrand Base to obtain the language $\mathcal{L}$, add it to the set of assumptions $\mathcal{A}$ and add the contrary mapping $\overline{\alpha(X)} = p(X)$.

Example 10. Consider the following ABA framework:
$\mathcal{L} = \{p(X), q(X), r(X), \alpha(X), \beta(X), \gamma(X) | X \in \{a, b, c\}\}$
$\mathcal{R} = \{p(X) \leftarrow \beta(X) \qquad q(X) \leftarrow \alpha(X), \gamma(X) \qquad r(X) \leftarrow \beta(X)\}$
$\mathcal{A} = \{\alpha(X), \beta(X), \gamma(X)\}$
$\overline{\alpha(X)} = p(X) \qquad \overline{\beta(X)} = q(X) \qquad \overline{\gamma(X)} = r(X)$
This corresponds to the following logic program:
$p(X) \leftarrow$ *not q(X),*
$q(X) \leftarrow$ *not p(X), not r(X),*
$r(X) \leftarrow$ *not q(X)*
with the following Herbrand Base:
$HB = \{p(X), q(X), r(X) | X \in \{a, b, c\}\}$

There is also a correspondence between the grounded extension in an ABA framework and the well-founded model of the corresponding logic program.
Let P be a logic program .Then, for a given ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ the well-founded model $WFM(P)$ is a triple (T, F, U) where T is the set of sentences covered by $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$, F is the set of sentences whose contraries are covered by $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$, and U is the set of undecided sentences (i.e. sentences that are neither true nor false).
More formally:
Let G be the grounded extension of $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$.
$WFM(P) = (T, F, U)$
$HB = \mathcal{L} \setminus \mathcal{A}$

$T = \{s(X) \in HB | \exists A \subseteq \mathcal{A} \text{ such that } A \vdash \underline{s(X)} \in G\}$
$F = \{s(X) \in HB | \exists \alpha(X) \in \mathcal{L} \text{ such that } \overline{\alpha(X)} = s(X) \text{ and } \{\alpha(X)\} \vdash \alpha(X) \in G\}$
$U = HB \setminus (T \cup F)$

3.3 Assumption-based argumentation learning

In this section, we introduce the meaning of an ABA problem, by defining its components. Furthermore, we present the mechanism of ABA learning using transformation rules, which allow the manipulation of ABA framework in a manner that facilitates achieving the objectives of a learning task, and we defines those objectives. Finally, we discuss a proposed strategy that aims to solve such ABA problems.

ABA frameworks are representations of knowledge, which also provide means to reason with them [13]. We are interested in finding a way to learn how to conduct debates about concepts that we had no prior knowledge about. We call this task an ABA learning problem, which is defined below.

Note. Similarly to [6], we will assume all rules in $\mathcal{R}$ are normalised. Therefore they will have the following form:

$\rho: \ H \leftarrow Eqs, B.$

where $Eqs = eq_1, eq2, ..., eq_m$ $(m \geq 0)$, $B = b_1(X_1), ..., b_n(X_n)$ $(n \geq 0)$, $H = h(X)$ where h is a predicate of arity a, b_i (for $1 \leq i \leq n$) is a predicate of arity a_i, X and X_i (for $1 \leq i \leq n$) are tuples of a and, respectively, a_i variables, and eq_j (for $1 \leq j \leq m$) is an equality between variables in $X, X_1, ..., X_n$.

Note. For simplicity, we will drop the language component $\mathcal{L}$ when defining an ABA framework as it can be deduced from the sentences used in the rest of the components. Therefore, an ABA framework will be represented further as $\langle \mathcal{R}, \mathcal{A}, \overline{} \rangle$.

Definition 13. As proposed in [6], an ABA learning problem that aims to learn the concept $p(X)$ (where p is a predicate with arity $n \geq 0$ and $X = X_1, X_2, ..., X_n$) consists of:

- Two disjoint sets representing the **training data**: a (non-empty) set of *positive examples* $\mathcal{E}^+$ and a set of *negative examples* $\mathcal{E}^-$ for predicate p
- A *flat ABA framework* $\langle \mathcal{R}, \mathcal{A}, \overline{} \rangle$ representing the **background knowledge**

Note. As mentioned in Section 3.1, the set of assumptions in an ABA framework is non-empty. Thus, for the remainder of the report, if the set of assumptions is not explicitly stated, it is safe to assume that it is a set containing a single bogus assumption.

Example 11 (ABA learning problem)**.** The following is an example of an ABA learning problem, whose goal is to learn the concept of *robber(X)*:

$\mathcal{R} = \{$
$\rho_1 : seenAtBank(X) \leftarrow wasAtWork(X),$
$\rho_2 : wasAtWork(X) \leftarrow banker(X),$
$\rho_3 : banker(jane),$
$\rho_4 : banker(david),$
$\rho_5 : seenAtBank(ann),$
$\rho_6 : seenAtBank(taylor),$
$\rho_7 : wasAtWork(matt),$
$\}$
$\mathcal{E}^+ = \{robber(matt), \quad robber(ann), \quad robber(taylor)\}$
$\mathcal{E}^- = \{robber(jane), \quad robber(david)\}$

Note. All the examples in this section will be based on the ABA learning problem defined in Example 11.

The goal of ABA learning is, starting from the given problem, to construct an ABA framework $\langle \mathcal{R}', \mathcal{A}', \overline{}' \rangle$ (where $\mathcal{R} \subseteq \mathcal{R}', \mathcal{A} \subseteq \mathcal{A}'$ and, for all $\alpha \in \mathcal{A}, \overline{\alpha} = \overline{\alpha}'$) that is able to argue about the concept it was trying to learn. More formally, as defined in [6], the resulting framework should satisfy the properties of *existence, completeness* and *consistency*, defined as follows:

Definition 14 (Existence)**.** An ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ satisfies **existence**, if it admits at least one extension under the chosen ABA semantics.

Definition 15 (Completeness)**.** An ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ satisfies **completeness**, if it covers all the positive examples $e \in \mathcal{E}^+$ under the chosen ABA semantics.

Definition 16 (Consistency)**.** An ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$ satisfies **consistency**, if it covers none of the negative examples $e \in \mathcal{E}^-$ under the chosen ABA semantics.

The method proposed in [6] for achieving this ABA framework is using five transformation rules as follows:

- `Rote Learning:` creates an argument based on a ground atom (in our case, it usually chooses one of the positive examples). To achieve that, for the atom $p(a)$, it creates $\mathcal{R}' = \mathcal{R} \cup \{\rho :\ p(X) \leftarrow X = a\}$

  Example 12. Applying this rule to each of the given positive examples, we obtain:
  $\mathcal{R}_1 = \mathcal{R} \cup \{\rho_8 :\ robber(X) \leftarrow X = matt, \quad \rho_9 :\ robber(X) \leftarrow X = ann,$
  $\rho_{10} :\ robber(X) \leftarrow X = taylor\}.$

- `Equality Removal:` generalises a rule that is constrained by equalities. To do that, it removes one of the equalities from the body of the rule. More concretely, it replaces a rule $\rho_1 :\ h(X) \leftarrow eq_1, Eqs, B$ with $\rho_2 :\ h(X) \leftarrow Eqs, B$, thus $\mathcal{R}' = \{\mathcal{R} \setminus \{\rho_1\}\} \cup \{\rho_2\}$

  Example 13. We can apply this rule to ρ_8, in which case it becomes:
  $\rho_8' :\ robber(X) \leftarrow.$

- `Folding:` generalises a rule whose body contains an atom which, using $\mathcal{R}$, can deduct some additional conclusions. For this, assume $\mathcal{R}$ contains
  $\rho_1 :\ H_1 \leftarrow Eqs_1, B1, B2$ and $\rho_2 :\ H_2 \leftarrow Eqs_1, Eqs_2, B_1$
  Then, ρ_1 gets replaced by
  $\rho_3 :\ H_1 \leftarrow Eqs_2, H_2, B_2$
  (i.e. $\mathcal{R}' = \{\mathcal{R} \setminus \{\rho_1\}\} \cup \{\rho_3\}$).

  Example 14. Using ρ_5, ρ_9 becomes $\rho_{11} :\ robber(X) \leftarrow seenAtBank(X)$.
  So we have $\mathcal{R}_2 = \mathcal{R} \cup \{\rho_8, \rho_{10}, \rho_{11}\}$.

- `Subsumption:` allows the removal of a rule that is subsumed by other rules in the background knowledge. To illustrate, assume $\mathcal{R}$ contains rule ρ_1 which, w.l.o.g., has the atom $p(X)$ as the head. Let C_1 be the set of claims of all the arguments that have ρ_1 as the top rule (so C_1 will contain ground instances of $p(X)$), then let C_2 be the set of *ground* instances of $p(X)$ for which we can build an argument such that ρ_1 is not part of the supporting rules. If $C_1 \subseteq C_2$, we can say that ρ_1 is subsumed by other rules in $\mathcal{R}$ and can be deleted from $\mathcal{R}$, obtaining $\mathcal{R}' = \mathcal{R} \setminus \{\rho_1\}$. For an example of how this rule is applied, see Step 4 of Iteration 1 in Example 16.

  Example 15. Rule $\rho_{10} :\ robber(X) \leftarrow X = taylor$ is subsumed by other rules.

Notice the argument $\{\} \vdash_{\{\rho_{10}\}} robber(taylor)$ is the only argument that has ρ_{10} as the top rule. For the same claim $robber(taylor)$, we can also build the argument $\{\} \vdash_{\{\rho_{11},\rho_6\}} robber(taylor)$, which is not supported by the rule ρ_{10}. Therefore rule ρ_{10} is subsumed by other rules.

Thus, we can remove ρ_{10}, so $\mathcal{R}_3 = \mathcal{R} \cup \{\rho_8, \rho_{11}\}$.

- `Assumption Introduction:` aids in avoiding the coverage of negative examples by introducing a (possibly new) assumption in the body of a rule. It works by replacing a rule $\rho_1: \ H \leftarrow Eqs, B$ in $\mathcal{R}$ by $\rho_2: \ H \leftarrow Eqs, B, \alpha(X)$ where $X = X_1, X_2, ..., X_n$ with $X_i \in vars(H) \cup vars(B)$ $(0 \leq i \leq n)$ and $\overline{\alpha(X)}' = c\_\alpha(X)$. Thus, $\mathcal{R}' = \{\mathcal{R} \setminus \{\rho_1\}\} \cup \{\rho_2\}$, $\mathcal{A}' = \mathcal{A} \cup \{\alpha(X)\}$, $\overline{a}' = \overline{a}$ for all $a \in \mathcal{A}$ (i.e. the rest of the contraries remain the same).

As outlined, `Equality Removal`, `Folding` and `Subsumption` all generalise rules, which should enable us to cover the positive examples, while `Rote Learning` and `Assumption Introduction` should learn the exceptions by constraining the generality created by the former set of transformation rules.
In this sense, [6] proposes a strategy for applying these rules that should ensure that the goal of ABA learning is achieved and the resulting framework satisfies the properties of existence, completeness and consistency. The template in Figure 3.3 is adapted from [6] and describes how to apply the transformation rules in a favorable manner.

Input: Background knowledge: ABA framework $\langle \mathcal{R}, \mathcal{A}, \overline{} \rangle$
Training data: $\mathcal{E}^+$ and $\mathcal{E}^-$, with $\mathcal{E}^+ \cap \mathcal{E}^- = \emptyset$

Strategy:

Repeat until $\langle \mathcal{R}, \mathcal{A}, \overline{} \rangle$ is both complete and consistent:

Step 1 *[Select target for current iteration]* Choose a predicate p such that $\exists c.\ p(c) \in E^+$;

Step 2 *[Generate rules for p]* For each example $p(c) \in \mathcal{E}^+$ apply `Rote Learning`;

Step 3 *[Generalise]* For each rule ρ in $\mathcal{R}$, perform one of the following:

- Find rule ρ' that subsumes ρ and apply `Subsumption`;
- Repeatedly apply `Folding` and then `Equality Removal` until all constants are removed from ρ;

Step 4 *[Learn exceptions]* Repeat until for all $p(d) \in \mathcal{E}^-$, $\langle \mathcal{R}, \mathcal{A}, \overline{} \rangle \not\models p(d)$:

1. Select $p(d) \in \mathcal{E}^-$;
2. Select from $\mathcal{R}$ a rule $\rho: \ p(X) \leftarrow Eqs, B$. (w.l.o.g. $X \subseteq vars(B)$) such that we can construct an argument for $p(d)$ with ρ as top rule;
3. Construct a set $A = \{a_1(Y_1), ..., a_k(Y_k) \subseteq B$ that can generate two ground instances A^+ and A^- $(A^+ \cap A^- = \emptyset)$ such that:
   (a) For every example $p(e) \in \mathcal{E}^+$, we can build an argument for $p(e)$ with ρ as top rule and the ground atoms in A^+ are children of $p(e)$;
   (b) For every example $p(e) \in \mathcal{E}^-$, we can build an argument for $p(e)$ with ρ as top rule and the ground atoms in A^- are children of $p(e)$;
4. Apply `Assumption Introduction` by adding the (new) assumption $\alpha(Y_1, ..., Y_k)$ with contrary $c\_\alpha(Y_1, ..., Y_k)$ to the body of the rule ρ.
5. Add $c\_\alpha(consts(A^+))$ to $\mathcal{E}^-$. Add $c\_\alpha(consts(A^-))$ to $\mathcal{E}^+$.

Step 5 *[Remove examples]* Remove all examples for predicate p from training data.

Figure 3.3: ABA Learning Strategy Template

Example 16. Following the proposed strategy, here is a fully worked example using the problem defined in Example 11 and using the grounded semantics:

<u>`Iteration 1`</u>

$\langle \mathcal{R}', \mathcal{A}', \overline{}' \rangle = \langle \mathcal{R}, \mathcal{A}, \overline{} \rangle$
$\mathcal{E}^+ = \{robber(matt), \quad robber(ann), \quad robber(taylor)\}$
$\mathcal{E}^- = \{robber(jane), \quad robber(david)\}$

Step 1 *[Select target for current iteration]* Choose predicate *robber*
Step 2 *[Generate rules for robber]* As performed in Example 12, we obtain $\mathcal{R}_1 = \mathcal{R} \cup \{\rho_8 : \ robber(X) \leftarrow X = matt, \quad \rho_9 : \ robber(X) \leftarrow X = ann, \quad \rho_{10} : \ robber(X) \leftarrow X = taylor\}$.
Step 3 *[Generalise]* After performing `Folding` and `Subsumption` as we previously did in Examples 14 and 15, we obtain $\mathcal{R}_3 = \mathcal{R} \cup \{\rho_8, \rho_{11}\}$. We also apply `Folding` to ρ_7 and ρ_8, and we obtain $\mathcal{R}_4 = \mathcal{R} \cup \{\rho_{11}, \rho_{12} : \ robber(X) \leftarrow wasAtWork(X)\}$
Step 4 *[Learn exceptions]*
Iteration A
1. Select $robber(jane) \in \mathcal{E}^-$;
2. Since $\emptyset \vdash_{\{\rho_1,\rho_2,\rho_3,\rho_{11}\}} robber(jane)$ with ρ_{11} as top rule, we select $\rho_{11} : \ robber(X) \leftarrow seenAtBank(X)$
3. We construct $A = seenAtBank(Y)$ with $A^+ \in \{\{seenAtBank(matt)\}, \{seenAtBank(ann)\}, \{seenAtBank(taylor)\}\}$ and $A^- \in \{\{seenAtBank(jane)\}, \{seenAtBank(david)\}\}$
4. By performing `Assumption Introduction`, replace rule ρ_{11} with
$\rho_{13} : \ robber(X) \leftarrow seenAtBank(X), \alpha 1(X)$
where $\alpha 1(X)$ is the new assumption with contrary $c\_\alpha 1(X)$. The new mapping $\overline{\alpha 1(X)} = c\_\alpha 1(X)$ gets added to $\overline{}$.
5. We add $c\_\alpha 1(matt), c\_\alpha 1(ann), c\_\alpha 1(taylor)$ to $\mathcal{E}^-$ and $c\_\alpha 1(jane), c\_\alpha 1(david)$ to $\mathcal{E}^+$

Both negative examples are still covered.

Iteration B
1. Select $robber(jane) \in \mathcal{E}^-$;
2. Since $\emptyset \vdash_{\{\rho_2,\rho_3,\rho_{12}\}} robber(jane)$ with ρ_{12} as top rule, we select $\rho_{12} : \ robber(X) \leftarrow wasAtWork(X)$
3. We construct $A = wasAtWork(Y)$ with $A^+ \in \{\{wasAtWork(matt)\}\}$ and $A^- \in \{\{wasAtWork(jane)\}, \{wasAtWork(david)\}\}$
4. By performing `Assumption Introduction`, replace rule ρ_{12} with
$\rho_{14} : \ robber(X) \leftarrow wasAtWork(X), \alpha 2(X)$
where $\alpha 2(X)$ is the new assumption with contrary $c\_\alpha 2(X)$. The new mapping $\overline{\alpha 2(X)} = c\_\alpha 2(X)$ gets added to $\overline{}$.
5. We add $c\_\alpha 2(matt)$ to $\mathcal{E}^-$ and $c\_\alpha 2(jane), c\_\alpha 2(david)$ to $\mathcal{E}^+$

None of the negative examples are now covered.

Step 5 *[Remove examples]* Remove all examples for predicate *robber* from training data.

<u>`Iteration 2`</u>

$\langle \mathcal{R}', \mathcal{A}', \overline{}' \rangle = \langle \mathcal{R} \cup \{\rho_{13}, \rho_{14}\}, \mathcal{A} \cup \{\alpha 1(X), \alpha 2(X)\}, \overline{} \cup \{\overline{\alpha 1(X)} = c\_\alpha 1(X), \quad \overline{\alpha 2(X)} = c\_\alpha 2(X)\} \rangle$
$\mathcal{E}^+ = \{c\_\alpha 1(jane), \quad c\_\alpha 1(david), \quad c\_\alpha 2(jane), \quad c\_\alpha 2(david)\}$

$\mathcal{E}^- = \{c\_\alpha1(matt), \quad c\_\alpha1(ann), \quad c\_\alpha1(taylor), \quad c\_\alpha2(matt)\}$

Step 1 *[Select target for current iteration]* Choose predicate $c\_\alpha1$

Step 2 *[Generate rules for $c\_\alpha1$]* As performed in Example 12, we obtain $\mathcal{R}_5 = \mathcal{R}' \cup \{\rho_{15}: \ c\_\alpha1(X) \leftarrow X = jane, \quad \rho_{16}: \ c\_\alpha1(X) \leftarrow X = david\}$.

Step 3 *[Generalise]* After performing `Folding` and `Subsumption`, we obtain $\mathcal{R}_6 = \mathcal{R}' \cup \{\rho_{17}: \ c\_\alpha1(X) \leftarrow banker(X)\}$

Step 4 *[Learn exceptions]*

None of the negative examples are covered, so there is no need to perform this step.

Step 5 *[Remove examples]* Remove all examples for predicate $c\_\alpha1$ from training data.

<u>`Iteration 3`</u>

$\langle \mathcal{R}', \mathcal{A}', \overline{}' \rangle = \langle \mathcal{R}_6, \mathcal{A}', \overline{}' \rangle$
$\mathcal{E}^+ = \{c\_\alpha2(jane), \quad c\_\alpha2(david)\}$
$\mathcal{E}^- = \{c\_\alpha2(matt)\}$

Step 1 *[Select target for current iteration]* Choose predicate $c\_\alpha2$

Step 2 *[Generate rules for $c\_\alpha2$]* As performed in Example 12, we obtain $\mathcal{R}_7 = \mathcal{R}' \cup \{\rho_{18}: \ c\_\alpha2(X) \leftarrow X = jane, \quad \rho_{19}: \ c\_\alpha2(X) \leftarrow X = david\}$.

Step 3 *[Generalise]* After performing `Folding` and `Subsumption`, we obtain $\mathcal{R}_8 = \mathcal{R}' \cup \{\rho_{20}: \ c\_\alpha2(X) \leftarrow banker(X)\}$

Step 4 *[Learn exceptions]*

None of the negative examples are covered, so there is no need to perform this step.

Step 5 *[Remove examples]* Remove all examples for predicate $c\_\alpha2$ from training data.

We now have a complete and consistent ABA framework:
$\langle \mathcal{R}', \mathcal{A}', \overline{}' \rangle = \langle \mathcal{R}_8, \mathcal{A}', \overline{}' \rangle$

3.4 Other learners

In this section we present two other related logic-based learning systems, which will be later used for evaluation purposes.

There has been similar work done in this area that has led to the development of learners. Most relevant to this project are ILASP and FOLD-RM.

3.4.1 ILASP

ILASP is an automated system that, given a learning task, produces an *ASP program* employing ILP (inductive logic programming) techniques to perform the learning process [16]. This learner is relevant for our context because flat ABA frameworks can be mapped onto logic programs and the stable semantics for ABA corresponds to the answer sets for logic programs (see Section 3.2). Another similarity between the two is that they are both able to learn *non-stratified logic programs* with *negation as failure* [17]. However, while ABA learning can be considered to learn by abduction, ILASP uses mostly *inductive learning* (it does however use abduction in its searches).

In the case of ILASP the learning task has three components:

1. **Background knowledge:** This component plays the same role as it does in the ABA learning problem, but in this case it consists of an answer set programming

(ASP) program;

2. **Examples:** This corresponds to the training data in the ABA learning problem, and it describes a set of semantic properties that the learned ASP program should satisfy;

3. **Mode bias:** This is used to express the ASP programs that can be learned.

Similarly to other ILP systems, ILASP follows the principle of *Occam's Razor* [16] (i.e. the preferred solution is the simplest one), therefore it aims to find not only a program to satisfy the requirements, but an *optimal* one. The optimality of a solution is measured by the number of literals it uses, aiming for as few as possible.

One key feature of this learner is its approach to noise. As outlined in [16], it associates a penalty to the examples that are not covered by the learnt program. The sum of all these penalties represents the cost of the program. This cost along with the number of literals represents the final measure of optimality that is used when deciding on the learnt program the system will return.

From a theoretical perspective, ILASP has a similar goal to ABA learning. As defined in [17], an hypothesis H is an inductive solution of the given problem iff it satisfies the following:

- It follows the given *mode bias* (similar to the *existence* objective of ABA learning);
- There is an answer set of the learnt program that *covers all positive examples* (similar to the *completeness* objective of ABA learning);
- There is an answer set of the learnt program that *does not cover any of the negative examples* (similar to the *consistency* objective of ABA learning).

ILASP is a *conflict-driven* ILP system [16]. By "conflict" is meant a reason why the current learnt program is not optimal. In an *iterative* manner, this conflict generates a set of constraints, then the system learns a new program that satisfies these constraints, and if a conflict can still be found it repeats the process. In a similar manner, ABA learning repeats the process of generalising and learning exceptions until it satisfies all its objectives (or it concludes it is unable to learn more).

Thus, given the similarities between the two, ILASP makes a good candidate to evaluate ABA learning against.

3.4.2 FOLD-RM

FOLD-RM is an automated ILP algorithm for multi-category classification tasks, that takes in tabular data, and generates a set of ASP rules [8]. Its main advantage is that it is highly efficient and scalable.

In the case of FOLD-RM the learning task, similarly to our system, has two components:

- **Background knowledge:** It comes in the form of an extended logic program using *negation-as-failure* which is restricted to being *stratified*;
- **Examples:** These are two disjoint sets of ground target predicates representing the positive and negative examples.

The input, which comes in the form of tabular data, gets converted by the system into the background knowledge. While it does not require any encoding of the input, it does require the user to set up the model of the dataset by specifying the list of attributes (i.e. the column names) as well as which features are numerical.

Before starting the learning process, FOLD-RM splits the input data in two disjoint sets of training and test data with a ratio of 0.8. This split represents a source of randomness as different subsets of the input tend to generate different outputs, which, tested against the test data, achieve different levels of accuracy.

At each iteration FOLD-RM picks as the target literal the category with most examples among the current training set, as it is considered to have higher potential in generating rules that will get the algorithm closer to the goal faster. In the learning process, FOLD-RM tries to learn default rules that include as many positive examples as possible (that get rules out once implied). Then, FOLD-RM introduces 'abnormal predicates' in order to learn exceptions to default rules as a way to mitigate the impact noise may have over the result. The `Assumption Introduction` rule used by our system has a similar effect.

In the representation of the rules, for the atoms generated by the input data, they all use as predicate the corresponding attribute from the list provided in the model setup and they each have two arguments: the first is a reference to the data record itself, while the second represents the feature value of the record. The only other atoms that occur are the ones that have been introduced as abnormal predicates and those have a single argument, representing the reference to the data record.

The final output is a textually ordered answer set program which encourages lazy-evaluation, which also contributes to its efficiency.

Chapter 4

The ABALearn System Implementation

In this chapter, we present the implementation of an algorithm derived from the strategy in Figure 3.3.

To explore the potential of ABA learning, we introduce ABALearn, an automated logic-based learning system. Its algorithm is based on the proposed strategy in Figure 3.3, by adjusting it such that termination is enforced and there are no arbitrary choices at any point in the learning process. The source code of the system is available on GitHub at this link[1].

4.1 Design

In this section, we briefly outline the high level technical aspects of the system.

The implementation of the strategy uses both Python and Prolog [2]. The Python component is the main engine that sets up the learning problem, makes the decisions on which step should be performed next, what would be the optimal transformations and, ultimately, decides when the learning process is finished and outputs the learned ABA framework. When there is a transformation that needs to be performed, Python makes external calls to Prolog to perform it. Thus, the logistics are implemented in Python, but it is Prolog that handles the actual manipulation of the ABA framework.

4.2 Input

This section describes the accepted input formats for ABALearn.

The input is the ABA learning problem which can be given either as a Prolog file or as a CSV file.

4.2.1 Prolog files

ABALearn takes in a Prolog file containing the ABA learning problem written as follows:

1. **Background knowledge:** The given set of rules is written as individual statements of the form `my_rule(RuleId,RuleHead,RuleBody).`.

[1] https://github.com/CristinaGTW/ABALearn
[2] More specifically, we have used Python 3.10.7, along with SWI-Prolog 7.6.4.

For example the rule *r1: bird*$(X) \leftarrow$ *penguin*(X) would be written as `my_rule(r1,bird(X),[penguin(X)])..`

2. **Positive examples:** The set of positive examples are written as individual statements of the form `pos(PosExId, Example)..`
For example the positive example *flies(a)* would be written as `pos(p1,flies(a))..`

3. **Negative examples:** The set of negative examples are written as individual statements of the form `neg(NegExId, Example)..`
For example the negative example *flies(c)* would be written as `neg(n1,flies(c))..`

Therefore, an example of a file that can be taken as input by the program is the one in Figure 4.1.

```
% Rules:
my_rule(r1,bird(X),[penguin(X)]).
my_rule(r2,penguin(X),[superpenguin(X)]).
my_rule(r3,bird(X),[X=a]).
my_rule(r4,bird(X),[X=b]).
my_rule(r5,penguin(X),[X=c]).
my_rule(r6,penguin(X),[X=d]).
my_rule(r7,superpenguin(X),[X=e]).
my_rule(r8,superpenguin(X),[X=f]).

% Positive examples:
pos(p1, flies(a)).
pos(p2, flies(b)).
pos(p3, flies(e)).
pos(p4, flies(f)).

% Negative examples:
neg(n1, flies(c)).
neg(n2, flies(d)).
```

Figure 4.1: Example of input: `flies_example.pl`

4.2.2 Tabular data

Alternatively, ABALearn also supports taking *tabular data* as input under the form of CSV files, as shown in Figure 4.2 . In this case, it will need to parse the data first, converting it into an ABA framework represented in Prolog, in the same manner we have presented for the Prolog file input format. To do this, the background knowledge and the training data are generated from data-points.
To illustrate, consider the input in 4.2 which represents the classical weather-play prediction task (also shown in Figure 4.3 for easier visualisation). The system will first start constructing the background knowledge using all columns except the first and last one (first column is used solely to identify the scenario, while the last column is used to construct the training data), as follows:

- If a column only contains positive or negative values (e.g. 'yes'/'no' or 'true'/'false'), the data-points in that column that have *positive* values get converted into rules using the name of the column as the predicate of the head, and the body consisting of a single equality, binding the variable in the head to the scenario. For example, consider the 'windy' column: that will generate the rules $windy(X) \leftarrow X = 2$, $windy(X) \leftarrow X = 6$, $windy(X) \leftarrow X = 7$, $windy(X) \leftarrow X = 11$, $windy(X) \leftarrow X = 12$ and $windy(X) \leftarrow X = 14$. .

- If a column contains discrete values, the data-points in that column get converted into rules using the name of the column along with their value as the predicate of the head, and the body consisting of a single equality, binding the variable in the head to the scenario. For example, consider the 'outlook' column: *some* of the rules it will generate are $outlook\_sunny(X) \leftarrow X = 1$, $outlook\_overcast(X) \leftarrow X = 3$, and $outlook\_rainy(X) \leftarrow X = 4$.

Naturally, using this approach, the generated ABA framework amounts to facts only (i.e. rules with an empty body) when we use tabular data as input.

Once the background knowledge is constructed, the system constructs the training data using the last column (which should always only contain positive or negative values), by adding for each positive value a positive examples using the name of the column as the predicate of the atom and the scenario values to ground it, and, similarly, adding a negative example for each negative example. Thus, considering the last column 'play', part of the training data that gets generated in this case is: $play(3)$, $play(10)$ and $play(13)$ as positive examples and $play(1)$, $play(6)$ and $play(14)$ as negative examples.

```
scenario, outlook, temp, humidity, windy, play
1, sunny, hot, high, false, no
2, sunny, hot, high, true, no
3, overcast, hot, high, false, yes
4, rainy, mild, high, false, yes
5, rainy, cool, normal, false, yes
6, rainy, cool, normal, true, no
7, overcast, cool, normal, true, yes
8, sunny, mild, high, false, no
9, sunny, cool, normal, false, yes
10, rainy, mild, normal, false, yes
11, sunny, mild, normal, true, yes
12, overcast, mild, high, true, yes
13, overcast, hot, normal, false, yes
14, rainy, mild, high, true, no
```

Figure 4.2: Example of input: `weather_play.csv`

4.3 Algorithm

This section explains in detail the mechanism of the learning process, addressing termination conditions and steps taken towards achieving the objective of ABA learning.

Note. For this section, when we talk about coverage (and implicitly about completeness and consistency), we imply that they are meant under the grounded semantics.

4.3.1 Termination condition

The termination condition follows the one indicated in Figure 3.3. The system keeps repeating the iteration consisting of five steps until the resulting ABA framework

scenario	outlook	temp	humidity	windy	play
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

Figure 4.3: `weather_play.csv` viewed as table

is complete and consistent. It is worth mentioning that the completeness and consistency checks are carried out against the *initial* sets of examples. Therefore, the learning process is considered finished when all the initial positive examples are covered and all the initial negative examples are not covered.
However, for some learning tasks, completeness and consistency might never be achieved, due to the nature of the tasks. Such cases include noisy tasks, or tasks that have circular debates, where there is no way to construct an ABA framework that satisfies both properties, given that we are aiming for general rules (i.e. no equalities in the bodies). In these circumstances, the learning process may not terminate (caused by infinite looping). To mitigate such scenarios, there are additional checks that are performed at the beginning of each iteration (as explained in 4.3.2).

4.3.2 Training data maintenance and exceptional termination

Before carrying out any ABA framework manipulation, at the beginning of each iteration there are some additional steps that we must perform.

As indicated by the strategy in Figure 3.3, the general idea is that for each iteration, there is a target predicate p, and in that iteration changes are made to the ABA framework to provide new ways of reasoning about p. In order to accomplish this, the algorithm makes use of positive and negative examples about p. At the end of each iteration, the training data using the target predicate p gets removed.

As explained in 4.3.3, it might be the case that at some point in the learning process we will require to target a predicate that has already been targeted. If that predicate is the goal predicate, our only source of training data for it is the initial set of positive and negative examples. Hence, it is crucial that we do not lose the information, while also keeping it minimal. To achieve this, we must perform the following:

1. For each initial positive example that is not currently covered, we verify if it is part of the current training data. If it is not, we reintroduce it.

2. For each initial negative example that is currently covered, we verify if it is part of the current training data. If it is not, we reintroduce it.

Additionally, we have identified two situations which indicate that the condition presented in 4.3.1 will not ensure termination. Thus, at this stage we must also verify if the algorithm should trigger an exceptional termination. The two situations are:

1. The algorithm has just completed re-targeting each possible predicate, and at no point throughout that run has the ABA framework been complete.
2. The target in the previous iteration was the goal predicate and, after performing the above mentioned reintroduction of examples, we have actually reestablished the exact same training data used in the previous iteration.

In case 1, it is safe to assume that there will be no change in that respect if we keep iterating. For case 2, we can only end up in a state like that if the set of not covered initial positive examples and the set of covered initial negative examples remained unchanged after the iteration that targeted the goal predicate. What this suggests is that the algorithm did not manage to find any new ways to reason about it. Regardless, both cases indicate that the system will not be able to make any more progress towards achieving the goal, thus it stops the learning process here.

4.3.3 Step 1: Target Selection

Each iteration requires a target predicate to learn. The program selects it by looking at the positive examples in the current training data and selecting the predicate for which it has the most examples as it should provide more opportunity for learning. The program also keeps track of all the predicates that have been targeted previously and avoids choosing them as long as it can (i.e. as long as there are positive examples for predicates that haven't been targeted yet). If it cannot find an unused target, it will go through the list of already used ones in order.

4.3.4 Step 2: Generating rules

This step is carried out in accordance with the indication in Figure 3.3: the program applies `Rote Learning` for each positive example of the target predicate.

4.3.5 Step 3: Generalising

The approach for generalising the rules has been slightly altered from the one proposed in Figure 3.3. Instead of looking at each rule, one by one, and deciding how to generalise it, the program looks at the set of rules as a whole and decides how that can be generalised. It starts by finding rules that can be optimally folded (as explained in 4.3.5), then removing equalities from the resulting rules, and, finally, checking if there are any rules subsumed by other rules which can be removed.

Applying `Folding` and `Equality Removal`

For this step, the program goes through every rule in the background knowledge and if its head uses the target predicate, it finds all other rules it can be folded with and that wouldn't generate a semantic loop (we explain this term below, through an example).

To check that it is not likely to generate a semantic loop, if the target predicate is the contrary of an assumption α, it must not be the case that the predicate in the head of the rule ρ we want to fold with is in the body of the rule α has been introduced in.

To illustrate, consider we performed `Assumption Introduction` on the rule ρ_1: $flies(A) \leftarrow bird(A)$ and obtained ρ_2: $flies(A) \leftarrow bird(A), \alpha_1(A)$ with $c-\alpha_1(A)$ the contrary of $\alpha_1(A)$. This leads to adding the set $\{c-\alpha_1(c), c-\alpha_1(d)\}$ as positive examples and the set $\{c-\alpha_1(a), c-\alpha_1(b), c-\alpha_1(e), c-\alpha_1(f)\}$ as negative examples. Then, in a later iteration, we have $c-\alpha_1$ as the target predicate. We find that the rule ρ_3*:* $c - \alpha_1(A) \leftarrow penguin(A), \alpha_2(A)$ can be folded with the rule ρ_4*:* $bird(A) \leftarrow penguin(A)$. However, we will not allow this fold as the predicate *bird* in the head of ρ_4 is part of the body of ρ_2 where α_1 (whose contrary is the target predicate) has been introduced. If we were to allow this rule, we would get ρ_5*:* $c{-}\alpha_1(A) \leftarrow bird(A), \alpha_2(A)$ that would introduce the set $\{c - \alpha_2(a), c - \alpha_2(b), c - \alpha_2(e), c - \alpha_2(f)\}$ as positive examples and the set $\{c - \alpha_2(c), c - \alpha_2(d)\}$ as negative examples, which are the interpolation of the sets of examples introduced after the first assumption introduction. This eventually leads to an infinite loop of trying to learn exceptions to exceptions. This sort of loop is what we mean by 'semantic loop'.

Then, for each potential fold, the program does the following, in order to ultimately only perform the folds that result in the ABA framework that is closest to the objectives of the algorithm (i.e. that cover the most positive examples and the least negative examples):

1. **Minimising the number of variables:** The program wants to minimise the number of variables introduced. To achieve this, it will initially only consider a fold if it does not introduce any new variables. If such a fold is found, in this iteration it will only consider folds that also do not introduce any new variables. Otherwise, it will look for folds that would only introduce one new variable. This process continues, incrementing the number of allowed new variables by `1`, until the program finds folds that satisfy this condition.
   For example, if we have the rules *r1:* $path(A, B) \leftarrow A = 2, B = 3$,
   r2: $arc(A, B) \leftarrow A = 2, B = 5$ and *r3:* $arc(A, B) \leftarrow A = 2, B = 3$, it does notice that $r1$ can be folded with $r2$ and it would give
   r4: $path(A, B) \leftarrow arc(A, C), B = 5, C = 3$, but as this would introduce an additional variable C, it keeps searching. It then finds that $r1$ can be folded with $r3$ instead, and that would generate *r5:* $path(A, B) \leftarrow arc(A, B)$, which does not introduce any new variables. This fold is taken into consideration. Additionally, in this iteration, the program will only look at folds that introduce no additional variables.

2. **Double folding:** Say the program has found that rule ρ_1 can be folded with ρ_2 and obtains the rule ρ_3, which satisfies the condition in step 1, by only introducing n new variables. It will now check if rule ρ_3 can be folded with any other rule in the background knowledge, without introducing more than n new variables. Let ρ_4 be the resulting rule of such a double fold.
   For example, assume at some point in the learning process it has the rules
   r6: $path(A, B) \leftarrow A = 1, B = 7$, *r7:* $arc(A, B) \leftarrow A = 1, B = 3$ and
   r8: $arc(A, B) \leftarrow A = 3, B = 7$. It then finds the fold between $r6$ and $r7$ that obtains *r9:* $path(A, B) \leftarrow arc(A, C), C = 3, B = 7$. It then checks and finds that it can also fold this rule $r9$ with $r8$, and obtains
   r10: $path(A, B) \leftarrow arc(A, C), arc(C, B)$. Thus, it takes this fold into consideration as well.

3. **Removing equalities:** The program now performs `Equality Removal` on the new rule ρ_4 (or ρ_3 if it did not manage to perform a double fold), until it removes all the equalities from its body.

4. **Fold statistics:** Say at the beginning of *Step 3* we had the ABA framework $\langle \mathcal{R}, \mathcal{A}, \overline{} \rangle$ and we are currently looking at the fold between rule ρ_1 and rule ρ_2 that resulted in rule ρ_3. We call the resulting rule after removing all equalities from rule ρ_3's body ρ'. Then we are interested in how many positive examples and how many negative examples $\langle (\mathcal{R} \setminus \{\rho_1\}) \cup \{\rho'\}, \mathcal{A}, \overline{} \rangle$ (further referred to as *the associated ABA framework* of the respective fold) covers. If this framework has achieved both the highest number of covered positive examples and the lowest number of covered negative examples so far, it will record those numbers as `max_pos` and `min_neg`. Additionally, for the rest of the search it will

only consider folds whose associated ABA framework cover at least `max_pos` positive examples and at most `min_neg` negative examples (naturally, `max_pos` and `min_neg` get their values updated accordingly).

Note. The number `min_neg` is strictly positive and while the program will take into considerations folds whose associated ABA framework cover `0` negative examples, `min_neg` only gets updated with strictly positive values. This way the program always allows for some generalisation to happen at this step.

At this point, the program is able to decide which folds it should perform. It goes through the list of folds it has taken into consideration and, for a fold that folds rule ρ_1 with rule ρ_2, it only performs it if it satisfies all of the following:

- The associated ABA framework covers max_pos positive examples and `0` or `min_neg` negative examples;
- Rule ρ_1 is not marked as `folded`;
- Either ρ_2 is not marked as `folded` or this fold is the first in a *double fold* (in which case it performs the whole double fold).

After the fold is performed, both ρ_1 and ρ_2 get marked as folded.

Once it finishes performing all the folds that satisfy these conditions, this part of *Step 3* is completed.

Applying `Subsumption`

At this step, the program goes through every rule ρ in the current background knowledge and checks which set of ground instances of the head of rule ρ are currently covered - call this set IN_{ρ}^{incl}. Then, it checks which set of ground instances of the head of rule ρ would be covered if rule ρ is not part of the background knowledge - call this set IN_{ρ}^{excl}. If IN_{ρ}^{incl} is a subset of IN_{ρ}^{excl}, it concludes that rule ρ is subsumed by other rules and it removes it from the background knowledge.

For example, say the program is looking at the rule ρ: $flies(X) \leftarrow superpenguin(X)$. It finds that the set of ground instances of $flies(X)$ that are currently covered is $IN_{\rho}^{incl} = \{flies(a), flies(b), flies(c), flies(d), flies(e), flies(f)\}$. Then it finds that, if rule ρ would get removed, the set of ground instances of $flies(X)$ that would be covered is $IN_{\rho}^{excl} = \{flies(a), flies(b), flies(c), flies(d), flies(e), flies(f)\}$. Since $IN_{\rho}^{incl} = IN_{\rho}^{excl}$, the program concludes that the rule ρ is subsumed by some other rules, so it removes it from the background knowledge.

4.3.6 Step 4: Learning exceptions

The execution of this step depends on the current state of the ABA framework, which can fall into one of two cases.

The first case is that the current ABA framework is consistent (i.e. does not cover any initial negative examples). If that is true, no further action is required for this step and we can move on to Step 5.

The second case is that the current ABA framework covers at least one of the initial negative examples. In this case, the program finds the top rules of all the arguments that have as a claim a negative example of the target predicate. Then, for each of these rules it does the following:

1. It constructs the two sets of constants $consts(A^{+})$ and $consts(A^{-})$, as described in Figure 3.3, by taking into consideration the variables of every atoms in the body of the current rule;

2. Performs `Assumption Introduction` on the current rule by introducing a new assumption α_0 with contrary c_α_0;

3. It checks if there is any previously introduced assumption α_1 with contrary c_α_1 for which the same sets $consts(A^+)$ and $consts(A^-)$ have been used, in which case it considers α_0 equivalent to α_1, and proceeds to replace α_0 with α_1;

4. If no equivalent assumption has been found, it adds examples for the contrary c_α_0 to the training data using the sets $consts(A^+)$ and $consts(A^-)$.

To illustrate why the addition of the check in item 3 was needed, we can refer to the worked example of a version of the Nixon-diamond example in [6] where `Assumption Introduction` is performed on the rule $abnormal\_republican(X) \leftarrow quacker(X)$. It introduces the assumption $normal\_quacker(X)$, which has been introduced at an earlier stage. In this scenario, the program would initially introduce a new assumption α in the body of the rule, obtaining $abnormal\_republican(X) \leftarrow quacker(X), \alpha(X)$. I would then notice that the sets $consts(A^+)$ and $consts(A^-)$ coincide with the ones that had been used when the assumption $normal\_quacker(X)$ has been introduced. Thus, it will replace α with $normal\_quacker$, and ultimately obtain the desired ABA framework.

4.3.7 Step 5: Removing examples

This step executes the same as in the proposed strategy: the program goes through the training data and removes all the examples for the target predicate of the current iteration.

4.3.8 Simplifying the ABA framework

After the system has managed to obtain an ABA framework that satisfies the objectives (existence, completeness, consistency), it will try to simplify the resulting framework. To do this it attempts removing redundant assumptions and performing `Folding` again, as described in the next two subsections.

Removing redundant assumptions

There are instances when the program will have achieved the objectives of ABA learning before *Step 4* (the one that introduces assumptions), but since the check only takes place before starting a new iteration, it will still perform *Step 4*. The program will terminate after this iteration. The ABA framework will have in the background knowledge a rule ρ containing a new assumption $\alpha(X)$ (with contrary $c\_\alpha(X)$) in the body, but no rules having $c\_\alpha(X)$ in the head. In this case, there can not be any arguments for claims with the predicate $c\_\alpha(X)$. In turn, this means any ground instance of $\alpha(X)$ can be assumed when building an argument with no chance of the argument being attacked (unless there is another assumption $\beta(X)$ supporting the argument, and there is an argument that has its contrary as the claim, but this is still independent of $\alpha(X)$). This behaviour makes the assumption $\alpha(X)$ redundant (i.e. has no effect over what is covered by the ABA framework), which means we can safely remove it, resulting in a simpler ABA framework (by simpler, it is meant that the ABA framework uses a more compact language).

In this sense, if the program finds one of these redundant assumptions, it will remove it from the rule it was introduced in, as well as from the set of assumptions and it will remove its mapping from the set of contraries.

Additional `Folding`

The program attempts to further generalise the rules that the learning process introduced by checking if there are any folds it can perform between them that would not introduce any additional variables. As there are no equalities in the bodies of these rules, performing such folds will not change what the ABA framework covers.

To illustrate, consider an ABA Learning task that resulted in introducing the rules ρ_1: $path(X,Y) \leftarrow arc(X,Y)$ and ρ_2: $path(X,Y) \leftarrow arc(X,Z), arc(Z,Y)$. The program will fold ρ_2 with ρ_1 in this case, obtaining ρ_3: $path(X,Y) \leftarrow path(X,Z), arc(Z,Y)$. Then, it will also fold this new rule ρ_3 with ρ_1, obtaining ultimately ρ_4: $path(X,Y) \leftarrow path(X,Z), path(Z,Y)$.

4.3.9 Enumeration

If the learning process was stopped before achieving completeness (triggered by the checks explained in 4.3.2), it will perform `Rote Learning` for the initial positive examples that remained uncovered. This guarantees completeness through enumeration.

4.3.10 Force-attack

If the learning process was stopped before achieving consistency (triggered by the checks explained in 4.3.2), we still want to ensure none of the negative examples are covered by forcing attacks on their arguments. This will partially sacrifice the generality of the learned rules, but will ensure consistency. To do this, we find the initial negative examples that are still covered along with the top rules of their arguments. It must be the case that these examples have previously been identified in some iteration targeting the goal predicate in *Step 4* (see 4.3.6). This means that the current top rules corresponding to their arguments contain assumptions in their bodies, but we have not managed to learn general rules for their contraries in order to attack those arguments that support the negative examples. Thus, we can force the attacks by rote learning rules to create arguments for the desired contraries.

For example, if we have a rule $flies(X) \leftarrow bird(X), alpha1(X)$, and one of the initial negative examples is $flies(c)$, but we have no accepted argument for $c_alpha1(c)$ (where c_alpha1 is $alpha1$'s contrary), we can force an attack by learning the rule $c_alpha1(X) \leftarrow X = c$.

4.4 Output

This section briefly presents what is the final output of the system.

Once it finished the learning process and performed the final optimisations, the program displays the learned rules along with some metrics and creates a Prolog file with the current ABA framework with similar format to the input file. For example, for the input in Figure 4.1, it would display in the terminal the output in Figure 4.4 and it would create a file like the one in Figure 4.5.

4.5 Coverage checker versions

This section discusses one of the main challenges encountered in the implementation of ABALearn and the various alternative approaches we have tried attempting to overcome it.

The focus in the implementation has been on the correctness of the algorithm: ensuring that the accepted arguments are constructed correctly in order to check what

```
-- Learned rules -- flies(A)<-bird(A),alpha1(A)
c_alpha1(A)<-penguin(A),alpha2(A)
c_alpha2(A)<-superpenguin(A)

-- Metrics --
Accuracy:  1.0
Precision:  1.0
Recall:  1.0

-- Learning time:  0.18405 seconds --
```

Figure 4.4: Example of terminal output for input `flies_example.pl`

```
% Background Knowledge
my_rule(r1,bird(A),[penguin(A)]).
my_rule(r2,penguin(A),[superpenguin(A)]).
my_rule(r3,bird(A),[A=a]).
my_rule(r4,bird(A),[A=b]).
my_rule(r5,penguin(A),[A=c]).
my_rule(r6,penguin(A),[A=d]).
my_rule(r7,superpenguin(A),[A=e]).
my_rule(r8,superpenguin(A),[A=f]).
my_rule(r_13,flies(A),[bird(A),alpha1(A)]).
my_rule(r_20,c_alpha1(A),[penguin(A),alpha2(A)]).
my_rule(r_23,c_alpha2(A),[superpenguin(A)]).

% Assumptions
my_asm(alpha1(A)).
my_asm(alpha2(A)).

% Contraries
contrary(alpha1(A),c_alpha1(A).
contrary(alpha2(A),c_alpha2(A)).
```

Figure 4.5: Example of generated file: `flies_example_solution.pl`

atoms are covered, performing the transformation rules in line with their definitions and ensuring that the final ABA framework satisfies all the objectives. However, we also aimed for a system that is still efficient and solves the learning task as fast as possible considering the complexity of the algorithm. This has represented a major challenge.

The first time we tried to run the system on a learning task with more than 50 rules in the background knowledge and more than 10 initial positive examples, it became obvious that the biggest bottleneck is checking the coverage of atoms. In this section we present some of the different approaches we have tried.

4.5.1 Coverage handled by Prolog (Version 1)

In this version, in order to check if an atom is covered by the current ABA framework, the system would make external calls to a Prolog script that handled this check. Prolog would try to find arguments that had the atom in question as the claim and then verify if it was part of the grounded extension.
Python uses the library `pyswip` to make these calls. The library keeps a Prolog instance running which would act as the source of truth for the current state of the ABA framework, as it records the current rules, assumptions, contrary mappings and training data. As previously outlined in Section 4.1, it is Prolog who handles the manipulation of the ABA framework by performing the transformation rules. Thus, in order for Python to access the current ABA framework, the system queried Prolog after every single transformation rule to retrieve every component of the ABA framework.
In Step 3, in order to decide which fold to perform, it had to be able to compute how many positive and negative examples it would introduce. When doing that, we are actually performing every single possible fold, then checking what examples are currently covered, then restoring the ABA framework to the state it was prior to the fold. This requires a huge amount of computationally-heavy calls to Prolog which tend to be very slow, partially because they are external and there is a lot of data flow and wiring that is happening, and partially because the search space is very big. In summary, this version turned out to be extremely slow, with no potential to scale.

4.5.2 Coverage handled by ASPforABA

We have also tried using ASPforABA [18] to handle the coverage checking. This system takes as input a file containing the problem specification (i.e. specifying that it is an ABA input file format and the number of atoms used in the ABA framework) along with every rule, assumption and contrary mapping in the ABA framework, where atoms are indexed by numbers.

Example 17 (Example from the specification of ICCMA 2023 [19]). The ABA framework with rules $(p \leftarrow q, a)$, $(q \leftarrow)$, $(r \leftarrow b, c)$, assumptions a, b, c, and contraries $\overline{a} = r$, $\overline{b} = s$, $\overline{c} = t$ is specified as follows, assuming the atom-indexing $a = 1, b = 2, c = 3, p = 4, q = 5, r = 6, s = 7, t = 8$.

```
p aba 8
a 1
a 2
a 3
c 1 6
c 2 7
c 3 8
r 4 5 1
r 5
r 6 2 3
```

We are interested in checking if an atom is sceptically accepted, therefore we run the system using the DS-σ subtrack for which we need to provide the `-a <query>` argument where `<query>` is equal to the number indexing the atom we want to check. The output of the system is `YES` if the atom is covered, and `NO` if it is not.

While ASPforABA itself is fast, the bottleneck in this case has come from creating the input for this solver. Given the required format, we had to maintain a mapping between every single existing atom and a number. What made this preprocessing truly expensive was the fact that, with this sort of representation of the ABA framework, we could not take advantage of the schemata representation of the rules. Instead, we had to get rid of any equalities from the body of a rule by using them to instantiate the atoms in the rule, and to further instantiate any remaining variable with every possible value. This significantly increased the complexity of this bit of the system. Thus, this approach has turned out to be even slower than the previous one, increasing the learning time to over 1 second even for the smallest inputs we were testing on.

4.5.3 Coverage handled by ASPARTIX

Another system that we have tried to pass the responsibility of coverage checking to was ASPARTIX [20].
This one, again, required some preprocessing in order to create the input for it. ASPARTIX requires to be provided with a file listing the arguments and attacks between arguments that take place in the ABA framework.

Example 18 (Example from the official documentation [21])**.** Consider an ABA framework with $Args = \{a, b, c\}$ where a attacks b, b attacks a and c attacks b. Then, the input file for ASPSARTIX should contain:

```
arg(a).
arg(b).
arg(c).
att(a,b).
att(b,a).
att(c,b).
```

Given the input, ASPARTIX then, if ran using Clingo along with the `ground.dl` encoding, outputs the corresponding grounded extension. Thus, in order to check if an atom is covered, we would check if the output from ASPARTIX contains any arguments who claims that atom.
This approach also turned out slower than the first version, as in order to provide it with the arguments and attacks, we would have to go through the same heuristics as the first version, given that we would identify the arguments and attacks using Prolog. This along with the file creation did not bring any improvement in our system's performance.

4.5.4 Coverage handled by Prolog (Version 2)

After concluding that any other system will most likely increase the execution time regardless of how fast it is because of the overhead of integrating it, we went back to the first approach and looked at how to reduce the number of external calls.
First optimisation was to store the current accepted arguments. This way, in order to check if an atom is covered we would not need to make external calls to Prolog anymore, but rather check by looking at the arguments. This improved the performance by a satisfying amount.
The other optimisation was to give Python the responsibility of updating the state of the ABA framework that Python uses, rather than constantly making external calls to Prolog after every transformation to retrieve the whole ABA framework. So, for example, in the first version we would retrieve the whole ABA framework after performing `Assumption Introduction`, now we just replace the old rule with the new one and add the assumption to the list of assumptions in the already stored ABA framework. This has also reduced the execution time noticeably.
However, while the improvement was significant, the performance was still not satisfying enough. The bottleneck in this approach is maintaining the set of accepted arguments. The set of accepted arguments has to be updated after each change of the background knowledge (which, as outlined previously, happens very frequently in Step 3 where we test out every single possible fold), which is done by making external calls to Prolog to identify the arguments.

4.5.5 Coverage handled by Python

Finally, we decided to try and implement the entire coverage checking in Python to fully avoid any external calls to Prolog for this purpose. While it was expected that removing the dependencies on Prolog will improve performance, the drawback of this approach is that we do not have the in-built variable unification that Prolog makes use of, making it difficult to develop a robust and reliable implementation.
To illustrate the improvement in execution time, we have run both the second approach of using Prolog to the one using Python on different sized subsets of the *acute* UCI dataset. The results can be observed in Figure 4.6.
We will further expand on the weaknesses of this implementation in Chapter 7, but based on empirical evaluation, this approach is the ideal one and has turned out to be significantly faster.

Note. This is the implementation version we will later use when evaluating our system.

Input Data Shape		**Execution time (seconds)**	
Rows	**Columns**	**Prolog (v2)**	**Python**
10	6	0.46	0.24
12	6	2.64	0.7
13	6	2.92	0.85
15	6	1.85	0.55
17	6	3.49	0.7
20	6	12.61	1.77
24	6	20.56	2.25
30	6	38.48	3.4
40	6	303.02	9.57
60	6	4256.23	33.24
120	6	>5000	326.35

Figure 4.6: Execution time comparison between checking coverage using Prolog (as explained in 4.5.4) and using Python (as explained in 4.5.5) when run on different sized subset of the *acute* UCI dataset.

4.6 Explainability

In this section we offer some insights about the explainability of ABALearn.

A main advantage that is specific to logic-based learners is that the learning output is interpretable. While ABALearn also showcases this characteristic, the learning output may include uninterpretable identifiers for assumptions and their contraries (e.g. `alpha1(A)` and `c_alpha1(A)` in Figure 4.4). This may potentially hinder its explainability.

In order to address this issue, we would need to choose more suggestive names for the predicates generated by the system. As this could only be achieved by a semantical prediction on what an exceptional scenario might represent, we have taken into consideration employing the abilities of large language models [3].

To further explore this idea, we conducted a simple experiment using OpenAI's ChatGPT [22]. This is a transcript of the chat where we attempted to rename the predicates in the solution for the `flies_3` task (see Appendix A):

[3] A large language model (LLM) consists of an artificial neural network, trained on large quantities of text using self-supervised or semi-supervised learning.

Q: Consider the following logic program:
flies(X) ← bird(X), not c_alpha1(X)
c_alpha1(X) ← feathered(X), not c_alpha2(X)
c_alpha2(X) ← bird(X), not c_alpha3(X)
c_alpha3(X) ← brokenwings(X)
Can you come up with more suggestive names for the predicates c_alpha1, c_alpha2 and c_alpha3 such that the rules are more intuitive for a human reading them?

A: Certainly! Here are some alternative names for the predicates c_alpha1, c_alpha2, and c_alpha3 that might make the rules more intuitive:

flies(X) ← bird(X), not incapable(X)
incapable(X) ← feathered(X), not can_fly_well(X)
can_fly_well(X) ← bird(X), not injured(X)
injured(X) ← brokenwings(X)

With these alternative names, the rules become more human-readable and easier to understand in terms of bird behavior and wing conditions.

Note. We provided ChatGPT with the equivalent logic program (see 3.2), given that it deals better with that sort of representation, as it does not need to handle the concept of contraries.

This outcome goes to show that there is potential in making use of readily available LLM to mitigate the potential obstacles in logic-based learners' explainability.

Chapter 5

Objective satisfaction

This chapter represents an informal proof that the algorithm does achieve the objectives of existence, completeness and consistency, in accordance to Definitions 14, 15 and 16.

In order to provide a justification in this sense, we must first clarify the chosen semantics. The algorithm is operating by default under the *grounded semantics*. However, as we have mentioned in 4.3.1, for some learning tasks, completeness and consistency under the grounded semantics can never be achieved. For those cases, termination happens under exceptional conditions (as presented in 4.3.2) and the algorithm automatically switches to achieving its objectives under *credulous semantics*. Thus, we will prove the satisfaction of the objectives accordingly.

Each property, will be proved assuming termination of the algorithm. Then, in Section 5.4 we provide a proof that the algorithm does, indeed, terminate.
We will split each proof into two cases:

- **Default case**: Assumes termination under the condition in 4.3.1 and proves the property under the *grounded semantics*;
- **Exceptional case**: Assumes exceptional termination (see 4.3.2) and proves the property under the *credulous semantics*.

5.1 Existence

5.1.1 Default case

From Theorem 1, it follows that every flat ABA framework has a unique grounded extension, therefore the existence objective is trivially achieved for grounded semantics. Additionally, the termination of the learning process is assumed to had been triggered by the termination condition in 4.3.1. Thus, we know that completeness has been achieved so, as the initial set of positive examples is non-empty, we can also guarantee that the grounded extension is non-empty (see Definition 10 and Definition 15).

5.1.2 Exceptional case

If the exceptional termination is triggered, we must be in one of the following scenarios:

1. **Completeness under the grounded semantics is unachievable:** In this case, enumeration will be performed (see 4.3.9). This ensures that every initial positive example $p(a)$ that is not covered under the grounded semantics becomes

covered by introducing the rule $p(X) \leftarrow X = a$. Naturally, the argument for this example will be supported by an empty set of assumptions. As a consequence, the argument cannot be attacked. So, by Definition 6, it must be the case that the argument is part of a stable extension admitted by the ABA framework (a stable extension attacks all the arguments it does not contain). Thus, the ABA framework does admit at least one stable extension.

2. **Consistency under the grounded semantics is unachievable:** If, at the time of the exceptional termination, completeness under the grounded semantics has also not been achieved yet, enumeration will be performed. Then, according to the same justification as above, the final ABA framework admits at least one stable extension. Otherwise, completeness under the grounded semantics has been achieved. If that is the case, as pointed out in 5.1.1, the ABA framework admits a non-empty grounded extension. Then, according to Theorem 3 and Theorem 2, it also has at least one stable extension.

Thus, we have exhaustively proven that it satisfies existence under credulous semantics.

5.2 Completeness

5.2.1 Default case

The termination condition in the default case is that the ABA framework is both complete and consistent under the grounded semantics. Then, the ABA framework is indeed complete at the end of the algorithm, as required.

5.2.2 Exceptional case

In this case, again, there are two situations:

1. **Completeness under the grounded semantics is unachievable:** Let P_{cov} be the set of initial positive examples that are covered under the grounded semantics at the time of termination and $P_{rest} = \mathcal{E}_0^+ \setminus P_{cov}$ where $\mathcal{E}_0^+$ is the initial set of positive examples. In this case, enumeration will be performed (see 4.3.9). This ensures that every initial positive example $p(a) \in P_{rest}$ becomes covered by introducing the rule $p(X) \leftarrow X = a$. Naturally, the argument for this example will be supported by an empty set of assumptions. As a consequence, the argument cannot be attacked. So, by Definition 6, it must be the case that the argument is part of a stable extension admitted by the ABA framework (a stable extension attacks all the arguments it does not contain). Additionally, by Definition 10 and Theorem 2, we know that, for every $p(a) \in P_{cov}$, every stable extension of the ABA framework contains an argument that has $p(a)$ as the claim. Thus, by Definition 11 and because $P_{cov} \cup P_{rest} = \mathcal{E}_0^+$, completeness under credulous semantics is achieved.

2. **Consistency under the grounded semantics is unachievable:** If, at the time of the exceptional termination, completeness under the grounded semantics has also not been achieved yet, enumeration will be performed. Then, according to the same justification as above, the final ABA framework satisfies completeness under credulous semantics. Otherwise, completeness under the grounded semantics has been achieved. If that is the case, as pointed out in 5.2.1, the ABA framework satisfies completeness under the grounded semantics. Then, according to Theorem 2 and Definition 11, it also satisfies completeness under credulous semantics.

Thus, we have exhaustively proven that it satisfies completeness under credulous semantics.

5.3 Consistency

5.3.1 Default case

The termination condition in the default case is that the ABA framework is both complete and consistent under the grounded semantics. Then, the ABA framework is indeed consistent at the end of the algorithm, as required.

5.3.2 Exceptional case

We are looking again at the two cases:

1. **Completeness under the grounded semantics is unachievable:** If consistency under the grounded semantics has been achieved at the time of termination, then we know by Definitions 9, 12 and 16 that there are no arguments for any initial negative example in the grounded extension G of the ABA framework. Otherwise, consistency under the grounded semantics has not been achieved, but, following the same justification as below, we conclude consistency under the credulous semantics is still satisfied. Regardless of the satisfaction of consistency under the grounded semantics, the algorithm will further perform enumeration (see 4.3.9) at this stage. Let $Args_{enum}$ be the set of new arguments that become accepted as a consequence of introducing these new rules. As mentioned in 5.2.2, these arguments will be part of all stable extensions as they are not attacked. Additionally, according to Theorem 2, G will also be a subset of each stable extension. Then, by Definition 6, each stable extension can be written as $S = G \cup Args_{enum} \cup Args_{rest}$, where each $Args_{rest}$ are arguments that ensure all the arguments that are not in S are attacked. However, by Definition 5, arguments can only be attacked by arguments whose claims are contraries. Since the goal predicate is not part of the contrary mapping of the ABA framework, we know that there are no arguments in $Args_{rest}$ whose claim uses the goal predicate. Thus, there is no argument in S whose claim is an initial negative example, which proves consistency.

2. **Consistency under the grounded semantics is unachievable:** If consistency under grounded semantics has not been achieved throughout the iteration, it means that it could not be achieved via general rules. However, if this is the case, we will force attacks performing the step explained in 4.3.10. This ensures that the arguments that cover initial negative examples are attacked (and cannot be defended since the arguments that attack them are not supported by any assumptions). Moreover, since the arguments that attack them cannot be attacked, they must be part of every stable extension. If that is the case, then no stable extension will contain arguments for any negative examples, given that it has to be conflict-free. This proves the property of consistency.

Thus, we have exhaustively proven that it achieves consistency under credulous semantics.

5.4 Termination

As each discussion so far has been based on the assumption of termination, we must prove that our algorithm does in fact terminate in order for the justifications to be valid.

As explained in Section 4.3, there are two situations that may cause termination: one presented in 4.3.1 and the other presented in 4.3.2. We are interested in proving that at least one of them will eventually get triggered.

In order to be able to prove termination, we must first prove an additional characteristic of the algorithm:
As explained in *Step 1* (see 4.3.3), once the algorithm can no longer find new predicates to target, it will go through the list of already used ones. We want to prove that it will eventually not be able to target new predicates, and that the list of already used ones is finite.
The initial background knowledge uses a finite language. Therefore, in order for the algorithm to always be able to find new predicates to target, it would have to keep introducing new assumptions in *Step 4* (see 4.3.6). However, as mentioned in 4.3.6, if the new assumption it is attempting to introduce uses the same sets $consts(A^+)$ and $consts(A^-)$ as a previously introduced assumption, the new one is no longer introduced. Given that these sets are constructed from distinct permutations of constants (which also come from a finite set), the possible values of $consts(A^+)$ and $consts(A^-)$ are finite.
Thus, the algorithm can only introduce a finite number of assumptions.
This implies that the ABA framework will always have a finite language, meaning that the algorithm will indeed have to eventually target already used predicates, which also represent a finite choice.

Without loss of generality, consider we are running a learning task whose goal predicate is p.
Assume the algorithm does not terminate. Then, it must never trigger any of the termination conditions. Then, it must satisfy all of the following propositions:

1. At the end of an iteration, the ABA framework is never both complete and consistent under the grounded semantics.

2. The algorithm cannot target each possible predicate and never satisfy completeness at the beginning of any iteration.

3. Let IN_{goal}^{start} be the set of all ground instances of the goal predicate that are covered at the beginning of an iteration and IN_{goal}^{end} be the set of all ground instances of the goal predicate that are covered at the end of an iteration. For every iteration that targets the goal predicate, $IN_{goal}^{start} \neq IN_{goal}^{end}$.

It is enough for one of these propositions to not hold in order for the assumption that the algorithm does not terminate to be proven false.

According to the second proposition, and because the list of predicates that can be targeted is finite (as proven above), we will eventually reach an iteration I where the ABA framework is complete at its beginning and the target predicate has been targeted before. Then, $IN_{goal}^{start} = \mathcal{E}_0^+$, where $\mathcal{E}_0^+$ is the initial set of positive examples. We now have two cases:

Case 1: The target predicate of I is p
In this case, since $\mathcal{E}_0^+ \subseteq IN_{goal}^{start}$, performing the training data maintenance (see 4.3.2) will not reintroduce any positive examples. Thus, there is no `Rote Learning` to be performed. This means that, equally, there is no generalisation that can be done.
Given that the ABA framework is complete, according to the first proposition, it must be the case that it is not consistent. Thus, the algorithm is able to find initial negative examples that are still covered. This means `Assumption Introduction` will be performed.
If the introduced assumption is new, we notice that $IN_{goal}^{start} = IN_{goal}^{end}$, as we are yet to learn rules for its contraries in order to attack the arguments supported by ground instances of the new assumption. More specifically, we can still cover the

same negative examples using the same arguments, except replacing the top rule with the new one and supporting them with the new assumption. Consequently, the third proposition is now invalidated, triggering termination. Thus, the assumption that the algorithm does not terminate is false.

If the introduced assumption has been used already and we have already learned the rules to reason about its contrary, then, at the end of the iteration, the arguments for negative examples are now attacked. Thus, we achieve consistency under the grounded semantics. This means that the ABA framework is both complete and consistent, which invalidates the first proposition. This triggers termination, therefore the assumption that the algorithm does not terminate is false.

If, however, we have not already learned the rules to reason about its contrary, using the same justification as for introducing a new assumption, $IN_{goal}^{start} = IN_{goal}^{end}$. Thus, at the end of the iteration, the third proposition gets invalidated. This triggers termination, therefore the assumption that the algorithm does not terminate is false.

Case 2: The target predicate of I is r ($r \neq p$)

All potential target predicates that are not p are contraries of assumptions. Thus, given that we are now in the stage when the algorithm 're-targets' predicates, the training data will not contain any positive examples of r (as they have been removed at the end of the first iteration that targeted r). Therefore, similarly as in *Case 1*, the algorithm will not perform `Rote Learning`, nor any generalisation.

If no negative examples of r are covered, then it does not perform `Assumption Introduction` either. Thus, the set of accepted arguments remains unchanged. Therefore, at the end of the iteration $IN_{goal}^{start} = IN_{goal}^{end}$ hold, invalidating the third proposition and triggering termination. So, the assumption that the algorithm does not terminate is false.

If there any negative examples of r that are currently covered, the algorithm will perform `Assumption Introduction`. In this case, following the same justification as in *Case 1*, this will either cause $IN_{goal}^{start} = IN_{goal}^{end}$ to hold (which invalidates the third proposition), or will cause consistency to hold (which invalidates the first proposition). Regardless, termination gets triggered. Thus, the assumption that the algorithm does not terminate is false.

Thus, we have proved that the algorithm does indeed terminate every time.

5.5 Summary

We have proven that our algorithm always eventually terminates (see Section 4.3.1). Then, according to Sections 5.2 and 5.3 (which assumed termination), the algorithm always returns an ABA framework that satisfies both completeness and consistency. We have also proven that the property of existence is satisfied (see Section 5.1).

Thus, we have proven that the algorithm satisfies all of its objectives.

Chapter 6

Evaluation

In order to evaluate our system we will compare it with ILASP and FOLD-RM (the learners presented in Section 3.4) by looking at four different aspects:

1. **Ease of use:** We will look at both technical prerequisites, as well as at the cognitive complexity required in order to run each system.

2. **Response time and scalability:** We will compare the systems based on execution times on inputs of various sizes.

3. **Reliability:** We will compare the solutions in terms of accuracy and will also observe how stable each system in terms of replicability.

4. **Range of applicability:** We will look at how each systems deals with noise, circular debates, and other types of potential obstacles.

6.1 Ease of use

6.1.1 ABALearn

In terms of prerequisites, ABALearn is developed in Python3.10 and only requires `pyswip` (which requires SWI-Prolog 7.2.x or higher).
As presented in Section 4.1, ABALearn supports two types of inputs: either a Prolog file defining the components of the learning task, or tabular data (a CSV file) from which the system will define a learning task.

6.1.2 ILASP

ILASP is available to download from GitHub and has no prerequisites.
In terms of the required input, it takes a `.las` file containing the definitions of each component of the learning task.

6.1.3 FOLD-RM

FOLD-RM is developed with Python3 and can either be installed as only the function library via `pip` or the entire repository can be cloned from GitHub. The only prerequisite is `numpy` in this case.
FOLD-RM does not really have any way to take an input without preparing it by adding code to the source code. It does have many UCI datasets prepared which can be run directly. However, if the user wants to give it its own input, it has to be a CSV file containing tabular data and in order to prepare the dataset they have to add code that defines the attributes, identifies the numerical values, sets up the `Classifier` and loads the data.

```
#pos({flies(a),flies(b),flies(e),flies(f)},
     {flies(c),flies(d)}).

bird(X) :- penguin(X).
penguin(X) :- superpenguin(X).
bird(a).
bird(b).
penguin(c).
penguid(d).
superpenguin(e).
superpenguin(f).

#modeh(flies(var(bird))).
#modeb(bird(var(bird))).
#modeb(penguin(var(bird))).
#modeb(superpenguin(var(bird))).

#constant(bird,a).
#constant(bird,b).
#constant(bird,c).
#constant(bird,d).
#constant(bird,e).
#constant(bird,f).
```

Figure 6.1: Equivalent ILASP input to `flies_example.pl`

```
id,bird,penguin,superpenguin,flies
a,yes,no,no,yes
b,yes,no,no,yes
c,yes,yes,no,no
d,yes,yes,no,no
e,yes,yes,yes,yes
f,yes,yes,yes,yes
```

Figure 6.2: Equivalent FOLD-RM input to `flies_example.pl`

6.1.4 Comparison

As far as prerequisites are concerned, ILASP is undoubtedly at advantage, since it requires none.

Looking at the input required by each system, there are more significant differences.
ABALearn only requires essential information about a learning task (background knowledge and training data), and is even able to define the learning problem on its own from tabular data.
ILASP requires similar information, but in addition it also needs to be provided with the mode bias, which we would argue represents a higher amount of cognitive demand from the user.
FOLD-RM has the highest cognitive complexity, as the user has to modify the source code for any input it does not have already prepared.

For example, consider the example input file in Figure 4.1. The input for an equivalent learning task for the other two systems would be the ones in Figure 6.1 and Figure 6.2.

Given these arguments, we would argue ABALearn is the easiest to set up and run.

6.2 Response time and scalability

We will look at the execution time[1] of all three systems for both small learning tasks with a more complex background knowledge from a structural point of view, and larger learning tasks with the background knowledge consisting of just facts.

6.2.1 Small learning tasks

For this category, we have run all three systems on the same set of simple inductive tasks (see Appendix A for details on the learning task specifications as well as the heuristics of converting them in equivalent inputs for each system). The execution times are shown in Figure 6.3.

Input	ABALearn	ILASP	FOLD-RM
flies_1	0.1	0.11	0.0003
flies_2	0.21	0.32	0.0007
flies_3	0.27	0.31	0.0004
path	0.1	0.51	N/A
robber	0.07	0.1	0.0002
free	0.15	0.3	N/A

Figure 6.3: Execution times on different inputs

We notice that at this scale ABALearn is slightly faster than ILASP in all tasks. FOLD-RM outperforms both my a high margin.

6.2.2 Learning tasks from non-noisy UCI datasets

For this part of the evaluation we have picked a couple of UCI datasets (which are perfectly labeled) and pruned them in order to be able to create inputs for all three systems as fairly as possible. Thus, we have only kept columns containing discrete values and have picked different sized subsets of the datasets. The results of this experiment can be seen in Figures 6.4 and 6.5. See Appendix B for details on these learning tasks.

As the input size increases, it is interesting to observe how ILASP compares with ABALearn. We notice that for the *acute* dataset ILASP outperforms ABALearn in almost all subset sizes, with significant differences for the three largest subsets. However, for the *krkp* dataset, ABALearn outperforms ILASP up until a certain size. If we look at the structure of the two inputs we notice that the *acute* dataset only uses 5 predicates in the given background knowledge, while the *krkp* one uses 36. What seems to happen is that although ABALearn does not generally scale well as the size of the input increases (in terms of number of rules in the background knowledge and number of positive examples), it does scale with the size of the language in the given background knowledge better than ILASP.
It is important to note that the performance of ILASP is also highly impacted by the input, as there are multiple ways to specify a learning task and the user can control

[1] All learning processes on all systems have been run on the same machine with AMD Ryzen 7 3700U @ 2.3GHz and 16GB RAM.

Input Data Shape		ABALearn Input		Execution time (seconds)		
Rows	**Columns**	**BK**	**E^+**	**ABALearn**	**ILASP**	**FOLD-RM**
120	6	288	59	306.25	0.16	0.002
60	6	154	31	30.36	0.12	0.002
40	6	96	20	8.93	0.11	0.001
30	6	72	15	3.2	0.09	0.001
24	6	57	13	2.13	0.09	0.0007
20	6	57	11	1.65	0.09	0.001
17	6	38	7	0.65	0.09	0.0006
15	6	39	7	0.42	0.12	0.0008
13	6	33	9	0.68	0.08	0.0003
12	6	35	8	0.62	0.07	0.0002
10	6	24	5	0.19	0.08	0.0007

Figure 6.4: Execution times on different subsets of *acute* dataset
(Note: "Rows" and "Columns" specify the number of rows and columns in the input CSV file; "BK" and E^+ specify the number of facts in the background knowledge and the number of initial positive examples of the equivalent ABALearn learning task)

Input Data Shape		ABALearn Input		Execution time (seconds)		
Rows	**Columns**	**BK**	**E^+**	**ABALearn**	**ILASP**	**FOLD-RM**
63	37	682	33	333.24	25.84	0.008
31	37	328	16	40.91	17.81	0.006
15	37	155	7	4.45	11.91	0.005
14	37	141	7	4.9	15.65	0.003
12	37	135	6	4.4	9.78	0.002
10	37	100	5	2.3	6.55	0.003
9	37	92	4	1.99	9.89	0.001
7	37	73	3	0.96	5.94	0.001

Figure 6.5: Execution times on different subsets of *krkp* dataset
(Note: "Rows" and "Columns" specify the number of rows and columns in the input CSV file; "BK" and E^+ specify the number of facts in the background knowledge and the number of initial positive examples of the equivalent ABALearn learning task)

the size of the search space through choices in the mode declaration. To illustrate this we have also experimented with specifying the learning tasks using two different approaches: using the *autism* dataset, we have given it a learning task where the background knowledge was made up of facts (very similar to how we specified the learning task for ABALearn) and an equivalent learning task using the "context-dependent" representation, where each row from the CSV file is represented as a distinct example. The outcomes of this experiment can be observed in Figure 6.6 and we notice that the difference in performance is very significant, with one approach clearly outperforming ABALearn and, the other clearly outperformed by ABALearn.

FOLD-RM outperforms both by a high margin, as expected.

6.2.3 Comparison

At this aspect FOLD-RM has the best performance, this is very much expected, as it has been built with scalability as one of the main objectives. ILASP also scales far better than ABALearn, but we have also observed that for certain tasks types, up

Input Data Shape		ABALearn Input		Execution time (seconds)		
Rows	**Columns**	**BK**	**E^+**	**ABALearn**	**ILASP - A**	**ILASP - B**
100	17	717	27	577.89	>2000	2.47
70	17	510	19	137.47	>2000	3.00
54	17	394	15	155.68	>2000	2.68
35	17	234	7	8.23	551.57	1.98
23	17	176	7	5.8	203.49	1.89
17	17	114	4	2.13	197.92	1.85
14	17	113	5	1.49	274.44	2.36
11	17	83	3	1.11	116.73	1.31

Figure 6.6: Execution times on different subsets of *autism* dataset
(Note: "Rows" and "Columns" specify the number of rows and columns in the input CSV file; "BK" and E^+ specify the number of facts in the background knowledge and the number of initial positive examples of the equivalent ABALearn learning task; "ILASP-A" specifies the execution times with the factual representation and "ILASP-B" with the context-dependent representation; We have used 2000 seconds as a timeout)

until a certain input size, ABALearn is able to outperform ILASP (as it scales better with the language size), and that ILASP's scalability is highly dependent on how the task is specified.

6.3 Reliability

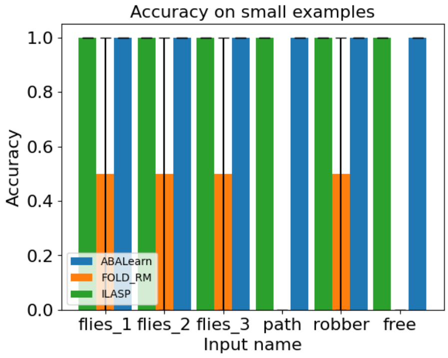
Note. By "accuracy" in this section we mean the result of computing $(TruePostives + TrueNegatives) \div$ *Number of examples*, where each term represents the following in each case respectively:

- **ABALearn and FOLD-RM:** $TruePositives$ is the number of initial positive examples covered by the solution; $TrueNegatives$ is the number of initial negative examples that are *not* covered by the solution; *Number of examples* is the total number of examples in the given training data;

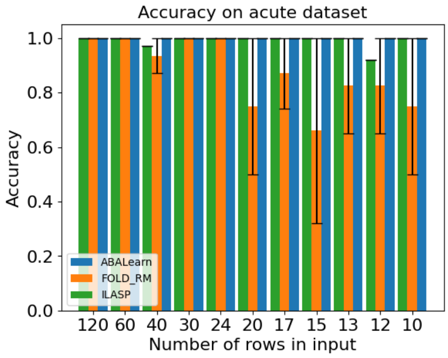
- **ILASP:** $TruePositives$ is the number of given positive examples that can be entailed using the background knowledge along with the generated hypothesis; $TrueNegatives$ is the number of given negative examples that cannot be entailed using the background knowledge along with the generated hypothesis; *Number of examples* is the total number of given examples.

At this section we are interested in observing what accuracies the solutions given by each system achieves. For this we have measured the accuracy of each solution for each system over 10 consecutive runs of the same input. The inputs used are the same as in the previous section (both the small examples and the UCI datasets). We can see the outcomes of these experiments in Figure 6.7 where we have plotted the mean accuracy along with its standard deviation.

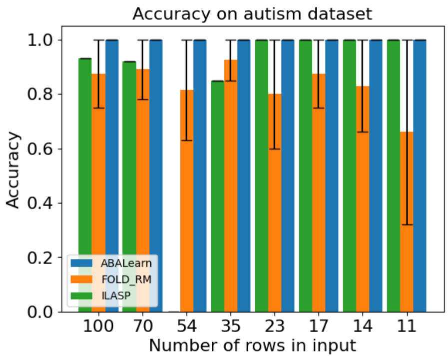
First thing that stands out is that both ABALearn and ILASP have 0 standard deviation across all runs, while FOLD-RM has very high standard deviations, with accuracies ranging from 0 to 1 for certain values. The reason why this happens for FOLD-RM is that its algorithm starts by randomly splitting the data into training and testing data, which although mitigates the issue of overfitting, it causes it to return different solutions for the same input.



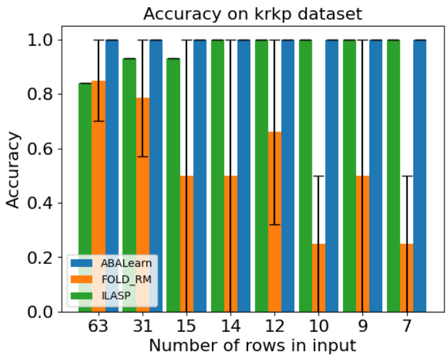
(a) Small examples



(b) The acute dataset



(c) The autism dataset



(d) The krkp dataset

Figure 6.7: Accuracy measurements

Another observation is that ABALearn managed to constantly achieve 100% accuracy across all runs. FOLD-RM has also managed to at least occasionally achieve 100% accuracy for every run, with the exception of two runs. ILASP has achieved very high accuracies, with 100% accuracy in 24 out of the entire 33 runs, and none below 85% (except for the one subset of the *autism* dataset that it found unsatisfiable - thus, missing in the corresponding plot). We do, however, acknowledge that in ILASP's case the lower accuracies are most likely caused by how the mode bias was declared, as the generated hypothesis space might have not allowed for any way to build a hypothesis that would give maximum accuracy.

These measurements go to show that both ABALearn and ILASP are more reliable than FOLD-RM, in the sense that they are deterministic, returning the same solution for the same input, and the solutions achieve very high accuracies (with ABALearn leading in this perspective with 100% accuracy in all runs), successfully achieving their objectives.

6.4 Range of applicability

6.4.1 Non-stratified, circular debates

One of the main advantages of ABALearn is that it manages to handle learning tasks that require to conduct circular debates. To illustrate consider the Nixon-diamond example [23] that gives rise to the following equivalent ABA learning problem:
$\mathcal{L} = \{quacker(X), republican(X) | X \in \{a, b\}\}$
$\mathcal{R} = \{r1 : quacker(a),$

$r2 : republican(a),$
$r3 : quacker(b),$
$r4 : republican(b)$
$\}$
$E^+ = \{pacifist(a)\}$
$E^- = \{pacifist(b)\}$
For this learning task, ABALearn comes up with the following ABA framework as the solution:
$\mathcal{L}' = \mathcal{L} \cup \{alpha1(X), alpha2(X), c\_alpha1(X), c\_alpha2(X) | X \in \{a, b\}\}$
$\mathcal{R}' = \mathcal{R} \cup \{r5 : pacfist(X) \leftarrow quacker(X), alpha1(X),$
$r6 : c\_alpha1(X) \leftarrow republican(X), alpha2(X),$
$r7 : c\_alpha2(X) \leftarrow pacifist(X),$
$r8 : c\_alpha2(X) \leftarrow X = b$
$\}$
$\mathcal{A} = \{alpha1(X), alpha2(X) | X \in \{a, b\}\}$
$\overline{alpha1(X)} = c\_alpha1(X) \qquad \overline{alpha2(X)} = c\_alpha2(X)$
One of the stable extensions admitted by this framework is:
$S = \{\{alpha1(a)\} \vdash pacifist(a),$
$\{alpha1(a)\} \vdash alpha1(a),$
$\{alpha1(a)\} \vdash c\_alpha2(a),$
$\{\} \vdash c\_alpha1(b),$
$\{alpha2(b)\} \vdash c\_alpha1(b),$
$\{alpha2(b)\} \vdash alpha2(b)$
$\}$
Thus, because *pacifist(a)* is the claim of the first argument in S and *pacifist(b)* is not the claim on any of the arguments in S, the goal of ABALearn is achieved under *credulous* reasoning as all the positive examples are covered, and all negative examples are not covered.

```
#pos({pacifist(a)},
     {pacifist(b)}).

quacker(a).
republican(a).
quacker(b).
republican(b).

#modeh(pacifist(var(t1))).
#modeh(c_alpha1(var(t1))).
#modeh(c_alpha2(var(t1))).
#modeb(pacifist(var(t1))).
#modeb(quacker(var(t1))).
#modeb(republican(var(t1))).
#modeb(c_alpha1(var(t1))).
#modeb(c_alpha2(var(t1))).

#constant(t1,a).
#constant(t1,b).
```

Figure 6.8: Equivalent ILASP input to the Nixon-diamond example

Initially we gave the equivalent learning task to ILASP using the same heuristics that we used for the small examples. With that input ILASP returned `UNSATISFIABLE`,

being unable to build a solution. However, after we change the mode declaration to match the solution we get from ABALearn[2] (see Figure 6.8), ILASP manages to successfully learn rules to reason about this circular debate. The solution it returns is:
`pacifist(V1) - quacker(V1); not c_alpha2(V1)`
`c_alpha2(V1) - republican(V1); not pacifist(V1)`
It is easy to see that this solution is semantically equivalent to the one returned by ABALearn.

For FOLD-RM, we gave it the equivalent input, using the same heuristics as in the small examples. The learned rule it returned was:
`pacifist(X,'no') :- quacker(X,'yes')` This solution does not achieve the goal as it cannot deduct `pacifist(a,'yes')` (equivalent to *pacifist(a)*).

6.4.2 Handling noise

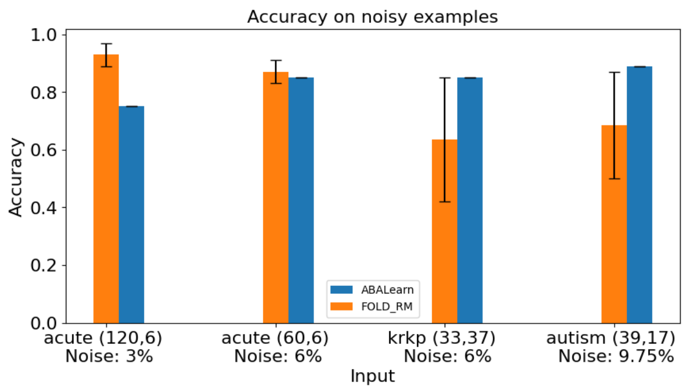
For this aspect, we took a few of the tabular data inputs we used in Sections 6.2 and 6.3 and introduced a few more duplicate rows with opposite labeling (e.g. if the input contains the row `1, yes, no, yes, yes, TRUE` we may introduce another row `2, yes, no, yes, yes, FALSE` which introduces noise given that identical properties result in different outcomes).
While we know ILASP is able to handle noise, using the same heuristics to specify the learning task will cause ILASP to always return `UNSATISFIABLE` when the data is not perfectly labelled. Thus, we will only compare ABALearn with FOLD-RM for this category.
To measure how well each system handles the noise, we looked again at the achieved accuracies. The results of the experiments can be seen in Figures 6.9.

Given that in *Step 3* of its algorithm ABALearn chooses to perform the folds that lead to covering as many positive examples and as few negative examples possible, and it also performs the Enumeration and Force-attack step (see 4.3.9 and 4.3.10), we expected it to perform reasonably well with noise. The results in the Figure 6.9 confirm this, and we see that it performs generally better than FOLD-RM in terms of accuracy achieved.

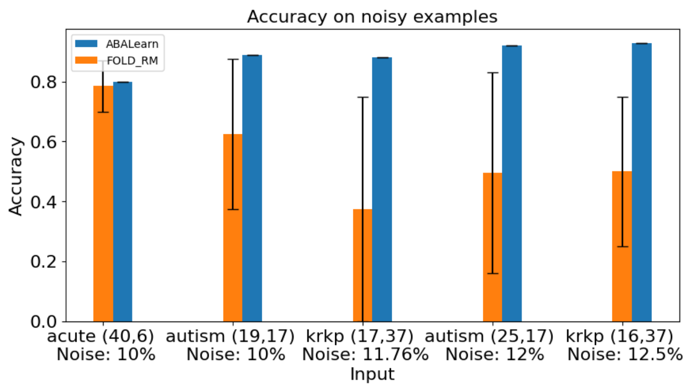
6.5 Summary

Having conducted the above evaluation, we can conclude that ABALearn has a good performance, managing to achieve high accuracies. Its ability to scale is by far its weakest point in the current state, but we are confident that can be improved.
In terms of how it compares to the other learners we have chosen (i.e. ILASP and FOLD-RM), ABALearn has the lowest cognitive complexity (as it requires the least amount of information in order to specify a learning task) and achieves generally the highest accuracy.

[2] We transformed the ABA framework in the equivalent logic program as explained in 3.2.



(a) Noise $< 10\%$



(b) Noise ≥ 10 %

Figure 6.9: Outcome of experiments with noisy inputs

Chapter 7

Future work

In this chapter, we propose potential directions in which this project can be extended, taking into consideration both limitations and improvements.

While we did manage to make significant progress in coming up with an implementation of the proposed learning strategy for ABA frameworks, there are still obvious limitations that require further study, as well as improvements that can be easily implemented, but have simply not been a priority for this project.

7.1 Limitations

7.1.1 Coverage checker

As we have previously pointed out, this component has been the biggest obstacle in our implementation. While we did conclude it should ideally be implemented in Python to improve the performance of the system, the current implementation is in a prototype state.
Purely based on empirical evaluation, if the language is made up of purely unary predicates, it behaves as expected. Additionally, for the `free` task which uses two unary predicates and one binary predicate it also behaves correctly and manages to succesfully complete the learning task. However, it does not manage to solve the `path` example which only uses two binary predicates. That is caused by some faulty variable unification that takes place at some point in the learning process. Therefore, currently, if the learning task is small in size or if the long execution time is not an issue, we would still recommend using the second version of the Prolog coverage checker.

7.1.2 Scalability

We have concluded from the evaluation that the biggest disadvantage of the current state of ABALearn is its ability to scale. The bottleneck is known, and it is the part in *Step 3* where we need to decide which rules to fold in that iteration. Currently the complexity of that algorithm segment alone is $\mathcal{O}(n^3)$ where n is the number of rules in the current background knowledge (which increases as we progress through the learning process). We are yet to come up with a more efficient way to perform this step. While we can't expect ABALearn to scale as well as FOLD-RM does, we do however believe it can scale similarly to ILASP.

7.2 Improvements

7.2.1 Packaging

The system is not yet packaged and in order to be used it would require the user to clone the entire GitHub repository, then compile and run it locally. Ideally, it should get packaged and published to `pip`.

7.2.2 Explainability

We have presented in Section 4.6 an experiment to improve the explainability of ABALearn's solutions. However, we have not yet integrated it into our system. Ideally, we would prefer to make use of OpenAI's GPT-4 API instead of using ChatGPT. The API is (at the time of writing this report) yet to be released, therefore further work in this direction is not possible at the current time. We do however believe it is a truly valuable improvement which explores the idea of enhancing XAI through traditional machine-learning models.

Chapter 8

Conclusion

Referring back to the objectives we set at the beginning of this work (as presented in the Introduction - see 1.2), our main target was to introduce a deterministic algorithm that is capable of learning ABA framework that achieve the properties of existence, completeness and consistency, given a learning task specified as an ABA learning problem (as defined in Definition 13). To accomplish that, we have proposed *ABALearn*, an automated system based on a previously proposed strategy.

Moreover, we have also said we would present a *justification* for the algorithm's ability of satisfying the three properties. In this sense, we have argued the satisfaction of each property through a discussion which is intended to represent an informal proof of this claim.

Lastly, we wanted to *evaluate* the implementation of said algorithm against existing learners that can solve similar equivalent tasks.

8.1 Contributions and challenges

8.1.1 ABALearn

In Chapter 4, we have introduced a novel system, called ABALearn, that represents the implementation of the algorithm we have proposed. The details of the algorithm are presented in Section 4.3, which explain the additions and changes we have brought to the initial proposed strategy in order to make it deterministic.

This amounts for the biggest part of the work and represents the main contribution, as, according to our research, there are no other existing algorithms that are able to achieve the goals of ABA learning given an ABA learning task.

Challenges

A challenge in this sense has been proposing a reliable termination condition which would ensure both that it does not terminate before the goal is achieved, and that the condition will eventually be met (i.e. it will never lead to non-termination). The solution to this is presented in Section 4.3.1, where we explain in detail what triggers the system to end the learning process. Additionally, we discuss in Section 5.4 the correctness of this condition.

Another challenge that required extensive research and multiple attempts to solve was the bottleneck generated by coverage checking. We have addressed this issue in

Section 4.5, and have eventually concluded that the optimal solution is to have an in-built coverage checker, which avoids any external calls.

8.1.2 Objective satisfaction

In Chapter 5, we have reasoned about the correctness of the algorithm in terms of its ability of learning an ABA framework that satisfies existence, completeness and consistency with respect to the grounded and stable semantics. To do this we have exhaustively analysed the states we might be in during the learning process and how each of them still guarantees reaching the goal.

8.1.3 Evaluation

In Chapter 6, we have conducted both observational and empirical evaluation in order to compare ABALearn to two other existing learners, namely ILASP and FOLD-RM. We have considered four different aspects to analyse for each system: ease of use (see Section 6.1), response time and scalability (see Section 6.2), reliability (see Section 6.3) and range of applicability (see Section 6.4).

In this regard, we have drawn satisfactory conclusions, which prove the potential of ABALearn, but also acknowledge its limitations.

We have seen that ABALearn requires minimal cognitive complexity in order to solve a learning task, and we have also looked at the types of learning tasks it is capable of solving, which has outlined its potential for being used in various contexts.

The evaluation on reliability showed that in terms of accuracy of the solutions it often outperforms the other learners, thus providing valid means for conducting debates, while also being stable, in the sense that the results are replicable, returning the same solution every time for the same input (which does not happen in the case of FOLD-RM).

In terms of scalability, we have concluded that it remains the biggest drawback of ABALearn, as it is not optimised in this direction. We have, however, been pleased to conclude it scales well with the language size. Nevertheless, we have identified the bottleneck and believe there is potential for improvement, as suggested in Subsection 7.1.2.

Challenges

Conducting this evaluation has required us to overcome a number of obstacles.
The one that was expected and unavoidable was that we had to first get familiar with the other two systems and understand how they operate, as well as deal with the technical aspects (i.e installing and running the systems).
Secondly, in order to fairly evaluate our system against them, we had to come up with a method to derive equivalent learning tasks for all three systems, as none of them accept the same input as the others. This has involved writing scripts to automate the conversion of tabular data to ABALearn inputs (which we have eventually integrated in the system in order to allow tabular data as an accepted input format) and to ILASP inputs, as well as mapping ABALearn inputs to ILASP and FOLD-RM inputs. The exact heuristics we have used in order to address these issues are presented in Appendices A and B.

8.2 Concluding remarks

Overall, this report has brought significant insights into the potential of ABA learning and, more generally, of XAI. The development of ABALearn represents more exciting proof that we can closely model human-like methods of logical reasoning, which goes to show that computational argumentation is truly powerful. While there are still undeniable limitations to our algorithm, we believe that there are good prospects that, with more research and development, it can have real-life applications under reasonable ethical considerations.

Appendix A

Small Learning Tasks

The small learning tasks used throughout this report are the following (represented as ABA learning tasks):

1. `flies_1`:
   $\mathcal{L} = \{bird(X), penguin(X), superpenguin(X) | X \in \{a, b, c, d, e, f\}\}$
   $\mathcal{R} = \{r1 : bird(X) \leftarrow penguin(X),$
   $r2 : penguin(X) \leftarrow superpenguin(X),$
   $r3 : bird(a),$
   $r4 : bird(b),$
   $r5 : penguin(c),$
   $r6 : penguin(d),$
   $r7 : superpenguin(e),$
   $r8 : superpenguin(f)$
   $\}$
   $E^+ = \{flies(a), flies(b), flies(e), flies(f)\}$
   $E^- = \{flies(c), flies(d)\}$

2. `flies_2`:
   $\mathcal{L} = \{bird(X), penguin(X), superpenguin(X), plane(X), damaged(X) | X \in \{a, b, c, d, e, f, g, h, k, m\}\}$
   $\mathcal{R} = \{r1 : bird(X) \leftarrow penguin(X),$
   $r2 : penguin(X) \leftarrow superpenguin(X),$
   $r3 : bird(a),$
   $r4 : bird(b),$
   $r5 : penguin(c),$
   $r6 : penguin(d),$
   $r7 : superpenguin(e),$
   $r8 : superpenguin(f),$
   $r9 : plane(g),$
   $r10 : plane(h),$
   $r11 : plane(k),$
   $r12 : plane(m),$
   $r13 : damaged(k),$
   $r14 : damaged(m)$
   $\}$
   $E^+ = \{flies(a), flies(b), flies(e), flies(f), flies(g), flies(h)\}$
   $E^- = \{flies(c), flies(d), flies(k), flies(m)\}$

3. `flies_3`:
   $\mathcal{L} = \{bird(X), feathered(X), light(X), brokenwings(X), big(X) | X \in \{a, a1, b, b1, c, c1, d, d1\}\}$
   $\mathcal{R} = \{r1 : bird(a),$

$r2 : feathered(a),$
$r3 : bird(a1),$
$r4 : feathered(a1),$
$r5 : bird(b),$
$r6 : light(b),$
$r7 : bird(b1),$
$r8 : light(b1),$
$r9 : bird(c),$
$r10 : feathered(c),$
$r11 : brokenwings(c),$
$r12 : bird(c1),$
$r13 : feathered(c1),$
$r14 : brokenwings(c1),$
$r15 : bird(d),$
$r16 : light(d),$
$r17 : big(d),$
$r18 : bird(c1),$
$r19 : light(c1),$
$r20 : big(c1)$
$\}$
$E^+ = \{flies(a), flies(b), flies(a1), flies(b1)\}$
$E^- = \{flies(c), flies(d), flies(c1), flies(d1)\}$

4. `path`:
$\mathcal{L} = \{arc(X, Y)|X, Y \in \{1, 2, 3, 7\}\}$
$\mathcal{R} = \{r1 : arc(1, 3),$
$r2 : arc(3, 7),$
$r3 : arc(2, 2)\}$
$E^+ = \{path(1, 1), path(1, 3), path(3, 7), path(1, 7)\}$
$E^- = \{path(2, 1), path(7, 1), path(7, 3)\}$

5. `robber`:
$\mathcal{R} = \{$
$r1 : seenAtBank(X) \leftarrow wasAtWork(X),$
$r2 : wasAtWork(X) \leftarrow banker(X),$
$r3 : banker(jane),$
$r4 : banker(david),$
$r5 : seenAtBank(ann),$
$r6 : seenAtBank(taylor),$
$r7 : wasAtWork(matt),$
$\}$
$\mathcal{E}^+ = \{robber(matt), \quad robber(ann), \quad robber(taylor)\}$
$\mathcal{E}^- = \{robber(jane), \quad robber(david)\}$

6. `free`:
$\mathcal{L} = \{step(X, Y), busy(X)|X, Y \in \{1, 2, 3, 4, 5, 6\}\}$
$\mathcal{R} = \{r1 : step(1, 2),$
$r2 : step(1, 3),$
$r3 : step(2, 4),$
$r4 : step(2, 5),$
$r5 : step(4, 6),$
$r6 : step(5, 2),$
$r7 : busy(3),$
$r8 : busy(6)$
$\}$

$E^+ = \{free(1), free(2), free(5)\}$
$E^- = \{free(3), free(4), free(6)\}$

A.1 ILASP equivalent learning tasks

For evaluation purposes, we had to create equivalent learning tasks to the ones mentioned above for ILASP. To do this, for each task we specified the background knowledge by adapting the rules in $\mathcal{R}$ to the required syntax, the examples are specified as `#pos(`E^+`,`E^-`).` and for the mode declaration we set the goal predicate as the only `modeh`, all the predicates in $p \in \mathcal{L}$ become `#modeb(p(var(t1))).` (with the appropriate arity) and all the constants $c \in \mathcal{L}$ become `#constant(t1,c).`.

Example 19. For background knowledge: $r1 : seenAtBank(X) \leftarrow wasAtWork(X)$ becomes `seenAtBank(X) - wasAtWork(X).`
For examples: the training data in `free` becomes
`#pos({free(1),free(2),free(5)},{free(3),free(4),free(6)}).`
For mode declaration: for the task `path` we get:

```
#modeh(path(var(t1),var(t1))).
#modeb(arc(var(t1),var(t1))).
#constant(t1,1).
#constant(t1,2).
#constant(t1,3).
#constant(t1,7).
```

A.2 FOLD-RM equivalent learning tasks

FOLD-RM is not designed to take logic programs as inputs. Instead, it only accepts tabular data. Therefore, while it does not necessarily capture the inductive nature of the background knowledge of a learning task, we made the translation by creating csv files where predicates in $\mathcal{L}$ along with the goal predicate become headers (and adding an `id` header as well) and the rows get filled out with `true` or `false` values if the atom built using the predicate in the respective column and the value in the `id` column to ground it is the claim of an accepted argument of the ABA framework we are attempting to translate. The last column (the one corresponding to the goal predicate) indicates whether the atom built using the goal predicate and the value in the `id` column represents a positive (`true`) or negative (`false`) example.

Example 20. The `robber` task becomes a CSV file with the following content:

```
id, seentAtBank, wasAtWork, banker, robber
matt, true, true, false, true
ann, true, false, false, true
taylor, true, false, false, true
jane, true, true, true, false
david, true, true, true, false
```

Appendix B

UCI Dataset-based Learning Tasks

For evaluation purposes we are also interested in experimenting with larger learning tasks. To do that we chose to convert a few UCI datasets (which are accepted as inputs as they are by FOLD-RM) into equivalent learning tasks for ILASP and ABALearn. The way we did this is by creating purely factual background knowledges. This does means that structurally we end up with very simple background knowledges which do not take advantage of the abilities of ILASP and ABALearn, however they should still be able to correctly handle them.
We have chosen UCI datasets that had mostly discrete values and have fully removed any columns that contained continuous (numerical) values, as neither ILASP nor ABALearn can capture those type of values.

B.1 ABALearn equivalent learning tasks

The translation is explained in Subsection 4.2.2.

B.2 ILASP equivalent learning tasks

After some investigation on the various ways we can specify learning tasks to ILASP, we have concluded that for this specific case the ideal representation is the context-dependent one. Thus, the task can be represented propositionally by representing each row of the CSV file as a distinct example.

Example 21. For a CSV file that contains the headers: `id,a1,a2,a3,a4,a5,label`, a row with the values `1,no,yes,no,yes,yes,yes` becomes:
`#pos({label},{},{a2. a4. a5.})` as it corresponds to a positive example.
Another row `2,yes,yes,no,no,no,no` in the same file would become:
`#pos({},{label},{a1. a2.})`.
Alternatively, if the row contains other discrete values (not positive, negative), the corresponding column will be encoded as a single predicate as follows:
Consider a file that contains the headers: `id, gender, ethnicity, label`, a row with the values `1,male,white,no` would get encoded as `#pos({},{label},{gender(male). ethnicity(white).})` and a row with the values `2,female, black, yes` would get encoded as `#pos({label},{},{gender(female). ethnicity(black).})`

Bibliography

[1] Atkinson K, Baroni P, Giacomin M, Hunter A, Prakken H, Reed C, et al. Towards Artificial Argumentation. AI Mag. 2017;38(3):25-36. Available from: https://doi.org/10.1609/aimag.v38i3.2704.

[2] Cyras K, Rago A, Albini E, Baroni P, Toni F. Argumentative XAI: A Survey. In: Zhou Z, editor. Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021. ijcai.org; 2021. p. 4392-9. Available from: https://doi.org/10.24963/ijcai.2021/600.

[3] Law M, Russo A, Broda K. Logic-Based Learning of Answer Set Programs. In: Krötzsch M, Stepanova D, editors. Reasoning Web. Explainable Artificial Intelligence - 15th International Summer School 2019, Bolzano, Italy, September 20-24, 2019, Tutorial Lectures. vol. 11810 of Lecture Notes in Computer Science. Springer; 2019. p. 196-231. Available from: https://doi.org/10.1007/978-3-030-31423-1_6.

[4] Cyras K, Fan X, Schulz C, Toni F. Assumption-based Argumentation: Disputes, Explanations, Preferences. FLAP. 2017;4(8). Available from: http://www.collegepublications.co.uk/downloads/ifcolog00017.pdf.

[5] Eshghi K, Kowalski RA. Abduction Compared with Negation by Failure. In: Levi G, Martelli M, editors. Logic Programming, Proceedings of the Sixth International Conference, Lisbon, Portugal, June 19-23, 1989. MIT Press; 1989. p. 234-54.

[6] Proietti M, Toni F. Learning Assumption-based Argumentation Frameworks. CoRR. 2023;abs/2305.15921. To appear in Proc. ILP '22, LNCS, Springer. Available from: https://doi.org/10.48550/arXiv.2305.15921.

[7] Shakerin F, Salazar E, Gupta G. A new algorithm to automate inductive learning of default theories. Theory Pract Log Program. 2017;17(5-6):1010-26. Available from: https://doi.org/10.1017/S1471068417000333.

[8] Wang H, Shakerin F, Gupta G. FOLD-RM: A Scalable, Efficient, and Explainable Inductive Learning Algorithm for Multi-Category Classification of Mixed Data. Theory Pract Log Program. 2022;22(5):658-77. Available from: https://doi.org/10.1017/S1471068422000205.

[9] Dimopoulos Y, Kakas AC. Learning Non-Monotonic Logic Programs: Learning Exceptions. In: Lavrac N, Wrobel S, editors. Machine Learning: ECML-95, 8th European Conference on Machine Learning, Heraclion, Crete, Greece, April 25-27, 1995, Proceedings. vol. 912 of Lecture Notes in Computer Science. Springer; 1995. p. 122-37. Available from: https://doi.org/10.1007/3-540-59286-5_53.

[10] Dung PM, Kowalski RA, Toni F. Assumption-Based Argumentation. In: Simari G, Rahwan I, editors. Argumentation in Artificial Intelligence. Boston, MA:

Springer US; 2009. p. 199-218. Available from: https://doi.org/10.1007/978-0-387-98197-0_10.

[11] Fan X, Toni F. A general framework for sound assumption-based argumentation dialogues. Artificial Intelligence. 2014;216:20-54. Available from: https://www.sciencedirect.com/science/article/pii/S0004370214000782.

[12] Bondarenko A, Dung PM, Kowalski RA, Toni F. An Abstract, Argumentation-Theoretic Approach to Default Reasoning. Artif Intell. 1997;93:63-101. Available from: https://doi.org/10.1016/S0004-3702(97)00015-5.

[13] Toni F. A tutorial on assumption-based argumentation. Argument Comput. 2014;5(1):89-117. Available from: https://doi.org/10.1080/19462166.2013.869878.

[14] Kowalski RA, Toni F. Abstract Argumentation. Artif Intell Law. 1996;4(3-4):275-96. Available from: https://doi.org/10.1007/BF00118494.

[15] Dung PM. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. Artif Intell. 1995;77(2):321-58. Available from: https://doi.org/10.1016/0004-3702(94)00041-X.

[16] Law M, Russo A, Broda K. The ILASP system for Inductive Learning of Answer Set Programs. CoRR. 2020;abs/2005.00904. Available from: https://arxiv.org/abs/2005.00904.

[17] Law M, Russo A, Broda K. Inductive Learning of Answer Set Programs. In: Fermé E, Leite J, editors. Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings. vol. 8761 of Lecture Notes in Computer Science. Springer; 2014. p. 311-25. Available from: https://doi.org/10.1007/978-3-319-11558-0_22.

[18] Lehtonen T, Wallner JP, Järvisalo M. Reasoning over Assumption-Based Argumentation Frameworks via Direct Answer Set Programming Encodings. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. AAAI Press; 2019. p. 2938-45. Available from: https://doi.org/10.1609/aaai.v33i01.33012938.

[19] ICCMA 2023: 5th International Competition on Computational Models of Argumentation;. Available from: https://iccma2023.github.io/rules.html [cited 05.06.2023].

[20] Dvorák W, Rapberger A, Wallner JP, Woltran S. ASPARTIX-V19 - An Answer-Set Programming Based System for Abstract Argumentation. In: Herzig A, Kontinen J, editors. Foundations of Information and Knowledge Systems - 11th International Symposium, FoIKS 2020, Dortmund, Germany, February 17-21, 2020, Proceedings. vol. 12012 of Lecture Notes in Computer Science. Springer; 2020. p. 79-89. Available from: https://doi.org/10.1007/978-3-030-39951-1_5.

[21] ASPARTIX for Dung AFs;. Available from: https://www.dbai.tuwien.ac.at/proj/argumentation/systempage/dung.html [cited 05.06.2023].

[22] OpenAI (2023). ChatGPT (May 24 Version) [Large Language Model] - personal communication;. Available from: https://chat.openai.com.

[23] Reiter R, Criscuolo G. On Interacting Defaults. In: Hayes PJ, editor. Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI '81, Vancouver, BC, Canada, August 24-28, 1981. William Kaufmann; 1981. p. 270-6. Available from: http://ijcai.org/Proceedings/81-1/Papers/054.pdf.