Artificial Intelligence 205 (2013) 39–70

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

# Approximating operators and semantics for abstract dialectical frameworks

Hannes Strass

*Computer Science Institute, Leipzig University, Augustusplatz 10, 04109 Leipzig, Germany*

ARTICLE INFO

*Article history:*
Received 9 October 2012
Received in revised form 10 September 2013
Accepted 17 September 2013
Available online 23 September 2013

*Keywords:*
Abstract dialectical frameworks
Abstract argumentation frameworks
Logic programming
Fixpoint semantics
Approximations
Nonmonotonic reasoning

ABSTRACT

We provide a systematic in-depth study of the semantics of abstract dialectical frameworks (ADFs), a recent generalisation of Dung's abstract argumentation frameworks. This is done by associating with an ADF its characteristic one-step consequence operator and defining various semantics for ADFs as different fixpoints of this operator. We first show that several existing semantical notions are faithfully captured by our definition, then proceed to define new ADF semantics and show that they are proper generalisations of existing argumentation semantics from the literature. Most remarkably, this operator-based approach allows us to compare ADFs to related nonmonotonic formalisms like Dung argumentation frameworks and propositional logic programs. We use polynomial, faithful and modular translations to relate the formalisms, and our results show that both abstract argumentation frameworks and abstract dialectical frameworks are at most as expressive as propositional normal logic programs.



## 1. Introduction

In recent years, abstract argumentation frameworks (AFs) [14] have become increasingly popular in the artificial intelligence community. An AF can be seen as a directed graph where the nodes are arguments whose internal structure is abstracted away, and where the edges encode a notion of attack between arguments. Part of the reason for the interest in AFs may be that in spite of their conceptual simplicity, there exist many different semantics with different properties in terms of characterisation, existence and uniqueness. Notwithstanding their success, their expressive capabilities are somewhat limited, as has been recognised many times in the literature: often it is inadequate to model argumentation scenarios having as only means of expression arguments *attacking* each other. There have been several proposals towards generalising AFs. To cite only a few examples, Prakken and Sartor [34] add *priorities* amongst arguments that are constructed from prioritised logic programming rules; Nielsen and Parsons [30] introduced attacks from *sets* of arguments; Cayrol and Lagasquie-Schiex [9] presented bipolar argumentation frameworks, in which arguments can also *support* each other; and Modgil [28] proposed attacks on *attacks* with the aim of reasoning about preferences on the object level.

As a general way to overcome the restrictions of Dung's AFs while staying on the abstract level, Brewka and Woltran [3] introduced abstract dialectical frameworks (ADFs). Just like AFs, these ADFs treat arguments (called *statements* there) as abstract, atomic entities whose contents are not further analysed. But instead of expressing for an argument only its attackers, ADFs associate with each statement an *acceptance condition* that determines the acceptance status of a statement given the acceptance status of its parent statements. These parents are the statements which have a say on whether the statement in question can or must (not) be accepted. In this way, AFs are recovered in the language of ADFs by specifying for each statement the acceptance condition "accept if and only if none of the attackers is accepted."

*E-mail address:* strass@informatik.uni-leipzig.de.

0004-3702/$ – see front matter 
http://dx.doi.org/10.1016/j.artint.2013.09.004

The abstract nature of Dung's AFs makes them well-suited as a target language for translations from more expressive formalisms. To be more precise, it is common to use expressive languages to model more concrete (argumentation) scenarios, and to provide these original expressive languages with semantics by translating them into Dung AFs [8,44,33,40]. However, Caminada and Amgoud [8] observed that it is not always immediately clear how such translations into AFs should be defined, even for a fairly simple source formalism. A major problem that they encountered were unintended conclusions that indirectly led to inconsistency. In the same paper, Caminada and Amgoud also proposed solutions to these problems, where during translation additional precautions have to be taken to avoid undesired anomalies. Let us explain in more detail what this means in general for abstractions among knowledge representation (KR) languages.

First of all, by an abstraction we mean a translation between languages that may disregard some information. Instantiating an abstract language is then the process of translating a more concrete, more expressive language into the abstract, less expressive language. This entails that there is no dichotomy "knowledge representation language vs. abstraction formalism" – any KR language abstracts to a greater or lesser extent, and can thus be used for abstraction purposes. Whether any specific language is to be used for direct, concrete representation or for abstraction of another language depends entirely on the application domain at hand.

Naturally, we are interested in those abstractions that preserve the meaning of translated language elements in some sense. As an example, consider the language $\{yes, no\}$. It is very simple and can abstract from any decision problem whatsoever. Furthermore it is trivial to devise an intuitively correct semantics for it. But to faithfully instantiate this language to a particular decision problem – say, the satisfiability problem of propositional logic –, the problem must be solved during translation, for otherwise the abstraction would not be meaningful at all. At the other end of the spectrum, for any language $\mathcal{L}$, an "abstraction" is provided by $\mathcal{L}$ itself. In contrast to the two-element target language $\{yes, no\}$, using $\mathcal{L}$ as target language makes it trivial to translate $\mathcal{L}$ into the abstraction, but the target language does in fact not abstract at all and devising a semantics for the abstraction is as hard as devising a semantics for the original language.

Thus abstraction proper should indeed disregard some information, but not too much of it. In the example above, the fact that the language $\{yes, no\}$ can abstract away from any decision problem is no argument for its usefulness as an abstraction formalism, since its expressive power is clearly too poor to model real problems (meaning problems that are syntactically different from their solutions). Consequently the expressiveness of a language is important when using it as a target language for abstraction. More specifically, a suitable target language for abstraction must be expressive enough to model important problem aspects, while being sufficiently abstract to ignore irrelevant details.

So to be able to use a formalism for abstraction, we obviously need a clear picture of its capabilities as a KR language, especially its expressive power in comparison to other languages, and about the properties of its semantics. It is the main objective of this paper to provide this information for abstract dialectical frameworks. For this purpose, we technically view ADFs as KR languages – but of course our work has ramifications for ADFs as abstraction formalisms. In the same way as there is no single intended semantics for argumentation frameworks, there is also no single perfect formalism for abstraction. But to be able to make an informed choice, it is of great importance to understand the inherent relationships between different available options. Our results will facilitate this choice and be an aid to anyone wishing to abstract from concrete argumentation languages; especially, our results will help them decide if they want to translate into AFs or into ADFs.

But why, after all, should there be a choice to be made between AFs and ADFs? Here, the additional expressiveness of ADFs in comparison to AFs comes into play. As we will see throughout the paper, the well-known distinction between supported and stable models from logic programming is present in ADFs but is missing in AFs. In a different disguise, this same distinction also materialises as Moore expansions vs. Reiter extensions in nonmonotonic logics [12]. To summarise it in a nutshell, there are basically two ways in which the major nonmonotonic KR formalisms deal with cyclic positive dependencies between pieces of knowledge. To explain what such cyclic support dependencies are and why they can be problematic, let us look at a study from the literature where researchers applied several logic-based knowledge representation techniques in a medium-sized practical application.

Nogueira et al. [32] describe a declarative rule-based system that controls some of the functions of a space shuttle. More specifically, the system operates the space shuttle's reaction control system, whose primary responsibility is to manoeuvre the shuttle through space. Part of the rule-based specification represents the plumbing system of this reaction control system. The plumbing system consists of a collection of tanks, jets and pipe junctions, which are connected through pipes. The flow of fluids through pipes is controlled by valves. The purpose of the plumbing system is to deliver fuel and oxidiser from tanks to the jets needed to perform a manoeuvre. The structure of the plumbing system is described by a directed graph whose nodes are tanks, jets and pipe junctions, and whose edges are labelled by valves. The description of the plumbing system should predict how the positions of valves affect the pressure of tanks, jets and junctions. For tanks themselves, the pressure resulting from pressurising certain (other) tanks is easy to specify. For all other nodes in the graph the definition is recursive: roughly, any non-tank node is pressurised by a tank if the node is connected by an open valve to a node which is pressurised by the tank. Nogueira et al. [32] explicitly recognise that modelling this is non-trivial because the connection graph of the plumbing system can contain cycles. That is, there may be nodes in the graph that are mutually connected to each other, and accurately modelling this is not straightforward:

**Example 1.1** *(Under pressure).* Consider the following easy setup where two nodes $n_1, n_2$ with associated tanks are connected to each other. The connection between a node $n_i$ and its tank is controlled by the valve $v_i$ in between.