Argumentation with Abduction

Neophytos Demetriou
Dept. of Computer Science
University of Cyprus
Nicosia CY-1678, Cyprus
nkd@cs.ucy.ac.cy
Tel.: (+357) 22892673

Antonis Kakas
Dept. of Computer Science
University of Cyprus
Nicosia CY-1678, Cyprus
antonis@cs.ucy.ac.cy
Tel.: (+357) 22892726

Abstract

This paper presents a general approach to combining argumentation and abduction where the different uses of argumentation for preference reasoning and abduction for reasoning under incomplete information are synthesized together in an enhancing way. This integrated approach of argumentation and abduction can form the basis for encoding adaptable preference policies in the face of incomplete information from dynamic and evolving environments. The paper shows how this integration can be achieved within a suitable logic based framework and studies a simple computational model to capture the combined form of reasoning.

1 Introduction

Argumentation has recently been shown to be a useful framework for formalizing non-monotonic reasoning and other forms of reasoning (see e.g. [BDKT97, Bre01, Dun95, KMD94, PSJ98, PS96]). In its abstract form, argumentation can be seen as a form of preference reasoning under some given relative strength of the arguments.

In this paper we study how we can develop a framework of argumentation that allows the strength of the arguments to be non static depending on factors that can change and that themselves can form part of the argumentative reasoning. Argumentation reasoning can then capture adaptable preference policies from dynamic and evolving environments.

In such dynamic environments it is often possible to have incomplete information. This can prevent us from constructing fully supported arguments as information that an argument needs maybe missing. In order to address this problem we study how abduction can be integrated with argumentation. This is done by extending the sets of arguments to include assumptions on abducible predicates as additional arguments and assigning them an appropriate strength with respect to the other arguments in the theory. This integration then allows us to develop a uniform computational model for argumentative reasoning and abduction.

2 Semantics

2.1 Argumentation frameworks

In general, an argumentation framework is a pair of a set of arguments and a binary attacking relation between conflicting arguments. Here, an argument is defined as a set of sentences whose role is solely determined by its relations to other arguments.

Definition 1 (Abstract Argumentation Framework) *An argumentation framework is a pair $(\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is a theory in some background logic, and $\mathcal{A}$ is a binary attacking relation on $2^{\mathcal{T}}$, i.e. $\mathcal{A} \subseteq 2^{\mathcal{T}} \times 2^{\mathcal{T}}$.*

We require that no set of sentences attacks the empty set and that the attacking relation is monotonic and compact [KT99]. In the sequel, we define the basic notion of an admissible subset of a theory within the abstract argumentation framework. A subset Δ of $\mathcal{T}$ is closed if it contains no rule whose conditions are not derived in Δ.

Definition 2 (Admissibility) *Let $(\mathcal{T}, \mathcal{A})$ be an argumentation framework. A closed subset Δ of $\mathcal{T}$ is admissible iff for all sets of sentences A, if A attacks Δ, then Δ attacks A and Δ is not self-attacking, i.e. it is consistent.*

Intuitively, a set is admissible if it defends itself against each attack. A theory may admit several admissible sets which may be incompatible with each other. Any proof theory for the admissibility extension semantics would therefore need to allow a non-deterministic choice among such sets.

2.2 The attacking relation

An attacking relation is realized via a notion of conflict and a notion of strength of an argument. Conflicting arguments, i.e. arguments that conclude a predicate with different negation status, may coexist in a knowledge base and thus a concrete scheme of the abstract attacking relation needs to employ an irreflexive strength (or qualification) relation that specify the scope of the conflict and the strength of the arguments under the given logical framework.

One simple example of a qualification relation for sets of sentences is "prefer monotonic rules". Intuitively, "prefer monotonic rules" sets the truth value of a predicate to the one suggested by a monotonic rule rather than a default rule. All semantics we consider can be formulated in terms of the abstract attacking relation defined as follows:

Definition 3 (Abstract Attacking Relation) *Given $\phi, \psi \subseteq \mathcal{T}$ and a strength relation $\mathcal{Q} \subseteq 2^{\mathcal{T}} \times 2^{\mathcal{T}}$, ϕ attacks ψ with respect to $\mathcal{Q}$, denoted by $\phi \xrightarrow{\mathcal{Q}} \psi$, iff $conflict(\phi, \psi)$ and $(\phi, \psi) \in \mathcal{Q}$, denoted by $\phi \succeq_{\mathcal{Q}} \psi$ or $\psi \preceq_{\mathcal{Q}} \phi$.*

Furthermore, we can consider the problem of reducing the strength relation in terms of a given priority relation $<$ on the sentences in a theory where $r < r'$ means that r has lower priority than r'. The role of the priority relation is to encode locally the relative strength of rules in the theory. As we will see in the next section, the priority relation can be reasoned about, just like any other predicate, and thus it can be classified as either static or dynamic and first- or higher-order.

2.3 Logic programming without negation as failure

In this section we study a concrete scheme of the abstract argumentation framework, namely the framework of logic programming without negation as failure (LPwNF) introduced in [KMD94, DK95], which uses explicit negation but does not contain negation as failure (NAF) in its object level language. We will extend this framework in two ways: (a) to generalize the attacking relation to be dynamic, and (b) we will integrate abduction. The background logic and a strength relation for LPwNF are formalized as follows:

Definition 4 (Background Logic) *Formulae in the background logic $\mathcal{L}$ of the LPwNF framework are defined as labelled rules of the form $label : l \leftarrow l_1, ..., l_n$ where $l, l_1, ..., l_n$ are positive or explicit negative literals and $label$ is a functional term.*

Definition 5 (Strength Relation via Priorities) *Let $\phi, \psi \subseteq \mathcal{T}$ be two conflicting arguments. Then, $\phi \preceq_{DYN} \psi$ iff $(\exists r \in \phi, r' \in \psi : \phi \vdash r' < r) \Rightarrow (\exists r \in \phi, r' \in \psi : \psi \vdash r < r')$.*

2.3.1 Example: Static priorities

The canonical Tweety problem – infering that Tweety can fly from the facts that Tweety is a bird and that birds typically fly; and retracting that conclusion upon discovering that Tweety is a penguin – is formulated as follows:

$$\begin{aligned} f_1 &: bird(tweety). \\ f_2 &: penguin(tweety). \\ r_1(X) &: fly(X) \leftarrow bird(X). \\ r_2(X) &: \neg fly(X) \leftarrow penguin(X). \\ r_3(X) &: r_1(X) < r_2(X). \end{aligned}$$

The last rule in the description above states that the rule $r_2(X)$ is stronger than $r_1(X)$, i.e. X cannot fly if it is both a bird and a penguin. Such priorities are said to be static since they are not subject to any preconditions. It is easy to verify that $\{f_2, r_2(tweety), r_3(tweety)\}$ is an admissible set since its only conflicting argument $\{f_1, r_1(tweety)\}$ does not qualify as an attack under $\preceq_{DYN}$. Note, however, that $\{f_1, r_1(tweety)\}$ attacks $\{f_2, r_2(tweety)\}$ but when we include $r_3(tweety)$ in the latter it does not.

2.3.2 Example: Dynamic priorities

Now let us consider a simple inheritance hierarchy of the form depicted in figure 1. An inheritance hierarchy consists of an acyclic graph representing the proper subclass relation between classes and a collection of properties of objects from these subclasses. The hierarchy from figure 1 consists of domain dependent axioms:

$$\begin{aligned} d_1(X) &: has(X, p) \leftarrow in(X, b). \\ d_2(X) &: \neg has(X, p) \leftarrow in(X, c). \\ d_3(X) &: d_1(X) < d_2(X) \leftarrow in(X, c). \end{aligned}$$

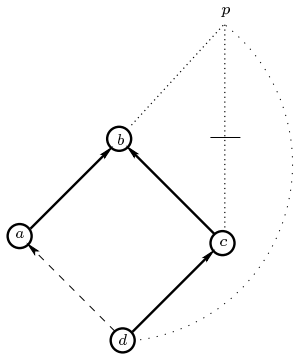

Figure 1: A simple inheritance hierarchy. Solid lines denote a proper subclass relation between classes. Dotted lines denote properties of objects while the dashed arrow denotes a possible addition of a as a superclass of d.

where $has(X, P)$ stands for "element X has property P", $in(X, C)$ stands for "X is an instance of class C", $subclass(A, B)$ stands for "class A is a subclass of class B" and the domain independent axioms:

$$\begin{aligned} r_1(C_0, C_2) &: subclass(C_0, C_2) \leftarrow subclass(C_0, C_1), subclass(C_1, C_2). \\ r_2(X, C_1) &: in(X, C_1) \leftarrow subclass(C_0, C_1), in(X, C_0). \\ r_3(X, C) &: \neg in(X, C). \\ r_4(A, B) &: \neg subclass(A, B). \end{aligned}$$

The first two rules represent general properties of $subclass$ and in. The last two rules express the closed world assumptions for simple hierarchies. In contrast to the previous section the priority rule $d_3(X)$ has a non-empty body which means that the priority is dynamic, i.e. it's truth value changes with respect to the given object. To complete, our formalization, we need the following facts:

$$\begin{aligned} f_1 &: subclass(a, b). & f_4 &: in(x_1, b). \\ f_2 &: subclass(c, b). & f_5 &: in(x_2, b). \\ f_3 &: subclass(d, c). & f_6 &: in(x_3, c). \end{aligned}$$

2.3.3 Example: Higher-order priorities

Consider the program in the previous section extended by $f_7 : subclass(d, a)$ to represent the fact that class d is also a subclass of a and $d_4(X) : d_2(X) < d_1(X) \leftarrow in(X, a)$ to express that instances of d have the property p. Now, both $d_1(X) < d_2(X)$ and $d_2(X) < d_1(X)$ hold for x_3 and thus both $has(x_3, p)$ and $\neg has(x_3, p)$ are consequences of the program. One way to resolve this conflict is to rewrite the rules $d_3(X)$ and $d_4(X)$ as follows:

$$\begin{aligned} d'_3(X) &: d_1(X) < d_2(X) \leftarrow in(X, c), \neg in(X, a). \\ d'_4(X) &: d_2(X) < d_1(X) \leftarrow in(X, a), \neg in(X, c). \end{aligned}$$

However, these rewritten rules do not represent the specification of the given domain in a natural way since such a practice would lead into a combinatorial explosion on the number of literals required in the body of the priority rules and thus degrading the high-intentionality of the language.

In contrast our framework, can accept rules stating priorities over priorities, i.e. higher-order priorities. For example, the following rule $d_5(X) : d_3(X) < d_4(X) \leftarrow in(X, a)$ states that an instance of d has the property p beside the fact that d is also a subclass of c.

2.4 Abduction

In several cases the admissibility of an argument depends on whether we have or not some background information about the specific case in which we are reasoning. However, this information maybe just unknown (or incomplete) and thus we need to find assumptions related to the unknown information under which we can build an admissible argument. Furthermore, this type of information may itself be dynamic and change while the rest of the theory remains fixed. To address this problem we can use abductive reasoning directly at the primary level.

For abduction in logic programming, we separate a distinguished set of predicates in the language of the theory, called abducible predicates, that express the incomplete information of the given domain of discourse. Then, given a goal, abduction extends the theory with ground abducibles so that it can satisfy the goal.

To integrate abduction with argumentation, we can consider the abducibles as a special type of predicates with the following strength relation on abductive arguments:

Definition 6 (Strength Relation via Assumptions) *Let α be an abducible predicate and $\phi, \psi \subseteq \mathcal{T}$. Then, $\phi \preceq_{ABD} \psi$ iff $\alpha \in \psi$ for some $\neg\alpha \in \phi$.*

Furthermore, we extend the framework to allow the split of the theory into a definitional part consisting of the logic program and an assertional part consisting of a set of integrity constraints on the abducible predicates which can be in the form of a rule with an abducible predicate in its head.

When integrity constraints are introduced in the formalism, one must define how they constrain the abductive arguments. There is need to assign a separate role to the logic program and the integrity constraints so that they filter the models of the extended theory by requiring that these are true formulae in such models:

Definition 7 (Strength Relation via Integrity Constraints) *Let α be an abducible predicate and $\phi, \psi \subseteq \mathcal{T}$ such that $\alpha \notin \psi$. Then, $\phi \preceq_{IC} \psi$ if and only if $\psi \vdash \alpha$ for some $\neg\alpha \in \phi$.*

3 Computing Argumentation with Abduction

In this section we develop a proof theory for the abstract argumentation framework introduced in the previous section. A proof theory can be used to decide whether a given (variable-free) goal holds with respect to the given semantics. The different proof theories are based upon a common computational framework, parametric with respect to the attacking relation. Different instances of the attacking relation correspond to different semantics.

The proof theory then aims to detect whether a given goal admits solutions. In this section we will assume that an argument Δ_0 for $\mathcal{G}$ is given and we will concetrate on the questions "Is Δ_0 admissible?", "If not, can Δ_0 be made admissible?", or "Is there an admissible superset Δ of Δ_0?".

The proof theory is based upon the construction of admissible trees via derivation of partial trees. In the trees we will consider, nodes are sets of sentences with each node labelled as 'attack' or 'defence'. In the sequel, we will use the term node to refer both to a location in the tree and to the set of sentences at this location.

Let us now define the notion of derivation of partial trees formally. In this definition we assume a selection strategy identifying a node (in the current partial tree) to be handled next, and we mark nodes that should not be selected further. Moreover, we record culprits chosen in selected (and marked) attack nodes.

Definition 8 (Derivation) *A derivation for a set of sentences Δ_0 is a sequence of partial trees $\mathcal{T}_0, ..., \mathcal{T}_n$ such that $\mathcal{T}_0$ consists only of the (unmarked) root Δ_0 labelled as defence, and, given $\mathcal{T}_i (i \geq 0)$, if N is the selected (unmarked) node in $\mathcal{T}_i$ then $\mathcal{T}_{i+1}$ is obtained as follows:*

(α) if N is an attack node, choose a culprit $c \in closure(N)$ and a minimal argument D against c such that D attacks N with respect to some qualification relation. Then $\mathcal{T}_i + 1$ is $\mathcal{T}_i$ where N is marked, c is recorded as the culprit of N, and D is added as the (unmarked) defence node child of N.

(δ) if N is a defence node, $closure(N) \cap culprits(T_i) = \emptyset$, then $\mathcal{T}_{i+1}$ is $\mathcal{T}_i$ where N is marked, the root is extended by N, and if $A_1, ..., A_m (m \geq 0)$ are all minimal attacks against N then $A_1, ..., A_m$ are added as additional (unmarked) attack nodes children of the root.

Definition 9 *A* **successful derivation** *for a set of sentences Δ_0 is a derivation of trees $T_1, T_2, ..., T$ for Δ_0 such that all nodes in T are marked and all leaves in T are labelled as defence. If the root of T is Δ, then we say that the derivation computes Δ from Δ_0.*

Theorem 1 (Soundness) *Let Δ_0 be a closed set of sentences. If there exists a successful derivation computing Δ from Δ_0, then Δ is admissible and $\Delta_0 \subseteq \Delta$.*

Theorem 2 (Completeness) *Let Δ_0 be a closed set of sentences. If Δ is an admissible set of sentences such that $\Delta_0 \subseteq \Delta$, then there exists a successful derivation computing Δ' from Δ_0, such that $\Delta_0 \subseteq \Delta' \subseteq \Delta$ and Δ' is admissible.*

3.1 A proof procedure for LPwNF

The proof theory developed in the previous section can be specialized to give a proof procedure for LPwNF by incorportating a specific way of computing minimal attacks.

The construction of an admissible set Δ is done incrementally, starting from a given set of sentences Δ_0, by adding to Δ_0 suitable defences for it. The existence of several admissible sets reflects itself on the existence of several defences for

a given Δ_0, and imposes a non-deterministic choice among defences in the proof procedure. However, not every potential defence can be promoted to Δ_0 as shown in [KT99].

We will assume that, actual nodes result from reducing (by resolution) a goal $\mathcal{G}$ into a closed and minimal set that concludes $\mathcal{G}$. Moreover, during computation, we use the special predicate "not" to record the non-existence of a priority constraint between rules in an argument and its counterargument. This permits the monotonic growth of the admissible set Δ during the computation. We construct a counterargument for a given node, as formalized below:

Definition 10 *Given an argument N, a counterargument $N' \cup S' \cup N'' \cup S''$ of N is obtained as follows:*

Basis: Choose a literal $c \in closure(N)$ and construct a closed and minimal set N' such that $N' \cup S' \vdash c'$ where c' is in conflict with c and S' is a minimal set of assumptions.

Qualification: Choose any of the following such that $N'' \cup S''$ is a non-empty set:

- *If there exists an abducible $\alpha \in N$ then $S'' = \{\neg\alpha\}$.*
- *If there exists an abducible $\alpha \in N$ then N'' is a closed and minimal subset of $\mathcal{T}$ such that $N'' \cup S'' \vdash \neg\alpha$ where S'' is a minimal set of assumptions, $\neg\alpha \notin N''$, and $\neg\alpha \notin S''$.*
- *If there exist rules $r \in N$ and $r' \in N'$ such that $N \vdash r' < r$ then N'' is a closed and minimal subset of $\mathcal{T}$ such that $N'' \cup S'' \vdash \tau < \tau''$, where S'' is a minimal set of assumptions, $\tau \in N$, and $\tau'' \in N''$. Otherwise, $N'' = \{not(r' < r) \mid r \in N \text{ and } r' \in N'\}$.*
- *If there exists $not(L) \in N$ for some functional term L then N'' is a closed and minimal subset of $\mathcal{T}$ such that $N'' \cup S'' \vdash L$, where S'' is a minimal set of assumptions.*

Note that, the first three options in the qualification step correspond to the definitions of $\preceq_{ABD}$, $\preceq_{IC}$, and $\preceq_{DYN}$. The soundness and completeness of the proof procedure for the framework of LPwNF, follow from the proofs of Theorem 1 and Theorem 2, respectively.

4 Conclusions and Future Work

We have proposed a general approach to combine argumentation and abduction in a way that preserves the benefits of both of them. The proposed argumentation framework and its integration with abduction has been implemented in a general system for argumentative deliberation and is available at `http://www.cs.ucy.ac.cy/~nkd/gorgias/`. Currently, we are improving its interface and using this system to develop a platform for implementing autonomous agents.

5 Acknowledgments

This work is partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-32250 SOCS project.

References

[BDKT97] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract, argumentation-theoretic framework for default reasoning. *Artificial Intelligence*, 93(1-2):63–101, 1997.

[Bre01] G. Brewka. Dynamic argument systems: a formal model of argumentation process based on situation calculus. In *Journal of Logic and Computation, 11(2), pp. 257-282*, 2001.

[DK95] Y. Dimopoulos and Antonis Kakas. Logic programming without negation as failure. In John Lloyd, editor, *Proceedings of the 12th International Logic Programming Symposium*, pages 369–383. MIT Press, Portland, Oregon, USA, 1995.

[Dun95] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.

[KMD94] A.C. Kakas, P. Mancarella, and P.M. Dung. The acceptability semantics for logic programs. *In Proceedings of the Eleventh International Conference on Logic Programming (ICLP'94), MIT Press*, pages 504–519, 1994.

[KT99] Antonis C. Kakas and Francesca Toni. Computing argumentation in logic programming. *Journal of Logic and Computation*, 9(4):515–562, 1999.