An argumentation-based approach to generate domain-specific explanations

Nadin Kökciyan[1], Simon Parsons[2], Isabel Sassoon[3], Elizabeth Sklar[2], and Sanjay Modgil[4]

[1] University of Edinburgh, UK nadin.kokciyan@ed.ac.uk
[2] University of Lincoln, UK {sparsons,esklar}@lincoln.ac.uk
[3] Brunel University, London, UK isabel.sassoon@brunel.ac.uk
[4] King's College London, UK sanjay.modgil@kcl.ac.uk

Abstract. In argumentation theory, argument schemes are constructs to generalise common patterns of reasoning; whereas critical questions (CQs) capture the reasons why argument schemes might not generate arguments. Argument schemes together with CQs are widely used to instantiate arguments; however when it comes to making decisions, much less attention has been paid to the attacks among arguments. This paper provides a high-level description of the key elements necessary for the formalisation of argumentation frameworks such as argument schemes and CQs. Attack schemes are then introduced to represent attacks among arguments, which enable the definition of domain-specific attacks. One algorithm is articulated to operationalise the use of schemes to generate an argumentation framework, and another algorithm to support decision making by generating domain-specific explanations. Such algorithms can then be used by agents to make recommendations and to provide explanations for humans. The applicability of this approach is demonstrated within the context of a medical case study.

Keywords: Computational argumentation · Explainability · Human-agent systems.

1 Introduction

In recent years, artificial intelligence (AI) has made an increasing impact on decisions taken in day to day life. Many AI systems involve black-box models that make decisions on behalf of humans often without providing any explanations. This is problematic since AI does not always make fair or correct decisions [14]. There is an increasing focus on developing techniques to help humans to make better decisions while being assisted by AI models [20]. In situations where there are multiple recommendations and the decision as to what action to take depends on a human, then being able to reason with the justifications for the recommendations becomes crucial.

Computational argumentation [26], a well-founded logic methodology with roots in philosophy, has been applied in AI and multi-agent systems as a (structured) technique for reasoning in which conclusions are drawn from evidence

that supports the conclusions. Users find examining the arguments behind a recommendation to be helpful [28]. This makes a strong case for basing decision-support systems around argumentation to assist humans in making informed decisions. The fact that the General Data Protection Regulation [6] requires transparency for any automated decisions, further strengthens this case.

In existing argumentation-based approaches, an agent constructs an argumentation framework based on the information in its knowledge base; and computes a set of acceptable arguments. Most of the times, an acceptable argument does not provide any additional information such as why it was deemed to be acceptable. Furthermore, there is little information about the defeated arguments [7,32]. To provide such information requires a representation for attacks among arguments. Such a representation can give a better understanding of why certain decisions are made by agents.

On the other hand, it is common to carry out knowledge acquisition using argument schemes (AS) and critical questions (CQs) [1,29,17]. Argument schemes are a means to compactly represent all the arguments that may be generated in different situations; whereas CQs are a way of capturing all the reasons why argument schemes might not generate arguments, either as pre-conditions to the construction of arguments, or as a way of formulating counter-arguments. Despite the popularity of the argument schemes and critical questions approach, there is no consensus on a formal representation of these elements, nor on an approach to construct an argumentation framework (AF), in the sense of [4], and there is no clear method to use these elements to create explanations, what we term "explainability by design".

In this paper, we make the following contributions: (i) we propose a formal representation of arguments through their respective argument schemes and critical questions; (ii) we introduce the notion of *attack schemes* to account for the conflicts between arguments in a given domain; (iii) we propose one algorithm to construct an argumentation framework for decision support; and another algorithm to provide explanations for acceptable arguments and attacks by the use of explanation templates. Such algorithms can help agents to reason about (possibly conflicting) information, make a decision and explain this to humans. The rest of the paper is as follows. Section 2 discusses related work. In Section 3, we introduce a high-level description of AFs that support explainability; and we propose algorithms to construct AFs and explanations automatically. Sections 4 and 5 introduce a medical scenario from the hypertension domain to show the applicability of our approach. Section 6 concludes and details future directions.

2 Related work

Argumentation has been applied to many domains including medicine [8,10,17], multi-agent systems [1,28,18] and legal reasoning [24,11]. We now focus on the application of argumentation to support decision making.

Argument schemes and their associated critical questions are often modelled as defeasible inference rules. Prakken *et al.* model legal cases as argument

schemes together with their associated undercutting attacks within the ASPIC+ framework [24]. Similar to us, they model CQs as argument schemes that can be challenged by other CQs. Atkinson *et al.* propose an argumentation-based approach to reason with defeasible arguments [2]. Unlike us, the above authors do not provide a formal representation for the CQs, but they use CQs to manually construct arguments in natural language.

Various argumentation-based approaches focus on the medical domain and determining treatment options. Tolchinsky *et al.* propose an agent-based architecture, Carrel+, to help transplant physicians in deliberating over organ viability [29]. Gordon and Walton generalise Dung's abstract argumentation frameworks [4] such that the arguments have weights to help one to choose among alternative options [12]. Similar to us, they provide a formal model of a structured argument. However, in addition, we model different types of attacks through attack schemes and use explanation templates to explain the decisions that agents make in a specified domain. Glasspool *et al.* [9] construct and evaluate pro and con arguments on different treatment options independently. ArguEIRA [13] is an ASPIC based clinical decision support system aimed at detecting a patient's anomalous reaction to a medication.

Some work combines argumentation with explanations. Kakas, Moraitis and Spanoudakis propose an approach to take scenario-based preferences in a tabular format, and to translate these preferences into a *Gorgias* argumentation theory and code automatically [15]. Rago, Cocarascu and Toni propose an application of Tripolar argumentation frameworks to support argumentation based explanations in recommender systems [25]. The method relies on visual representations of the argumentation frameworks, which includes supporting and attacking arguments, and allows users to tweak the recommendations. Unlike our work, neither of these approaches provides a natural language explanation. In addition, we believe that whilst arguments can encapsulate premises and claims, even in the case where these are instantiated through the use of argument schemes, simply putting forward extensions or collections of arguments falls short of constituting an explanation [20]. We believe that our approach of using explanation templates to translate the contents of an argument and attack into a structured form of natural language is a promising step towards creating good explanations [27].

In [21], Modgil and Bench-Capon explore the idea of attacking the attacks in AFs at the meta-level. Here, we focus on reasoning in the object-level. In both approaches, we can include arguments that represent human preferences and values to attack the attacks. We plan on exploring a comparison of different approaches in future work.

3 A formal model to represent argumentation frameworks

This section contains the main contribution of this paper, which is a high-level formal representation of an argumentation framework. This representation can: (i) be implemented in various ways (e.g. logic-based systems, in any programming language), (ii) enable the sharing of domain-specific knowledge across domains,

(iii) add explainability by design, where explanation templates are part of the model to generate domain-specific explanations. We then articulate algorithms so that agents can construct argumentation frameworks and generate explanations.

3.1 Formal model

We capture the semantics of an argument scheme in Definition 1. The premises and the conclusion are sentences, which can be represented in a logical language $\mathcal{L}$. Each of these sentences includes variables, which are then instantiated with elements of this language. We give an example of an argument scheme from the medical domain, Argument Scheme for a Proposed Treatment (ASPT) [17], in Table 1. ASPT represents an argument in support of each possible treatment `TR` within the current treatment step `S`, given the patient's treatment goal `G` to be realised. In the remaining of the paper, we will use the auxiliary function $\mathsf{Var}(AS)$ to refer to the set of variables used in AS.

Definition 1 (Argument Scheme). *$AS = \langle P, c, V\rangle$ denotes an argument scheme, where P is a set of premises, c is the conclusion, and $P \cup \{c\} \subseteq \mathcal{L}$. V is the set of variables used in the argument scheme.*

Table 1: Argument Scheme for a Proposed Treatment (ASPT) [17]

ASPT=$\langle\{p_1, p_2, p_3\}, c, \{G, TR, S\}\rangle$
p_1*:* Bringing about `G` is the goal.
p_2*:* Treatment `TR` promotes the goal `G`.
p_3*:* Treatment `TR` is indicated at step `S`.
c*:* Treatment `TR` should be offered.

An argument scheme can be associated with a set of critical questions (CQs) which provide reasons why that argument scheme might not generate arguments. We define CQs to themselves be argument schemes as suggested in [24,29]. We do not consider CQs to be part of a given argument scheme since this allows a CQ to be used by multiple argument schemes. We capture this structure in Definition 2.

Definition 2 (ASCQ). *$ASCQ : AS \rightarrow 2^{AS}$, is a function mapping an argument scheme to a set of argument schemes that represent the CQs of the original argument scheme.*

A knowledge base (KB) is the information store of an agent that includes premises, rules and the relationships between schemes, as captured in Definition 3. R is the set of rules written using the elements of the logical language. Rules and premises can be *strict*, in which case they admit no exceptions (we call strict premises "axioms"). For example, factual information about a patient may be considered to be axioms. Rules and premises can also be *defeasible*, in which

case they allow exceptions. Thus defeasible rules and facts can be falsified based on evidence. For example, argument schemes can be represented as defeasible rules, as we do in this paper, so that they represent tentative inferences that can be overturned. KB also has information about CQ relations among argument schemes as described via the $ASCQ$ function.

Definition 3 (Knowledge Base). *$KB = \langle P, R, ASCQ \rangle$ denotes a knowledge base; where P is the set of premises (e.g. facts), R is the set of rules and $ASCQ$ is the function as described in Definition 2.*

Arguments are constructed by instantiating each argument scheme according to ground terms that exist in the KB (i.e. terms that do not contain any free variables) (Definition 4). All the variables in AS are replaced with the ground terms to construct an argument (Definition 5). The notation $[X]$ will be used to denote the name of the scheme X (e.g. type of an argument).

Definition 4 (Argument Scheme Instantiation). *$AS_i = \langle AS, G, KB \rangle$ denotes an instantiation of the AS with $G \subseteq \mathcal{L}$ in the knowledge base KB, $AS\{v_i \mapsto g_i\}$ for all $i = 1, .., k$ where k is the size of $\mathsf{Var}(AS)$, v_i is the ith element in $\mathsf{Var}(AS)$ and g_i is the ith element in G. $\mathsf{Prem}(AS_i)$ returns the set of instantiated premises $AS.P$; $\mathsf{Conc}(AS_i)$ returns the instantiated conclusion $AS.c$; $\mathsf{Gr}(AS_i)$ returns the set of pairs (v_i, g_i); and $\mathsf{Gr}(AS_i)(v_i)$ returns g_i.*

Definition 5 (Argument). *$[AS]arg_i = \langle \mathsf{Prem}(AS_i), \mathsf{Conc}(AS_i) \rangle$ is an argument, which is derived from the argument scheme instantiation AS_i.*

Attacks among arguments are critical components of argumentation. In early works, attacks were defined syntactically, with an attack being between a formula and its negation. The well-known argumentation system ASPIC+ generalises this idea with the notion of a $\mathsf{contrary}$ function, which defines the set of formulae that conflict with a formula [22]. There are three forms of attacks among arguments: (i) a fallible premise of an argument can be attacked (*undermining*), (ii) the conclusion of a defeasible rule can be attacked (*rebuttal*), and (iii) a defeasible rule can be attacked as a whole (*undercutting*), for example denying its applicability in a particular setting.

While modelling well-known attacks is important, we introduce attack schemes (Definition 6) to: (i) provide flexibility to capture all the ways in which an attack may arise between two arguments, and (ii) explain the existence of attacks among arguments. This new representation allows for specific, domain dependent, forms of attack, such as drug contra-indications or the guideline conflicts of [31], as well as domain-independent attacks, such as undercuts.

Definition 6 (Attack Scheme). *$ATS = \langle \{p_1, p_2\} \cup P, c, V \rangle$ denotes an attack scheme with $P \cup \{c\} \subseteq \mathcal{L}$; where p_1 is an argument of type $\mathtt{X}$, p_2 is an argument of type $\mathtt{Y}$, P is a set of premises, c is the conclusion of the form 'p_1 attacks p_2' and $V = \mathsf{Var}(X) \cup \mathsf{Var}(Y)$. $\mathtt{X}$ and $\mathtt{Y}$ can be same type.*

Table 2: The attack schemes T_{cq} and ALT

(a) An undercutting attack

T_{cq}=$\langle\{p_1, p_2, p_3\}, c, \mathsf{Var}(X) \cup \mathsf{Var}(Y)\ \rangle$
p_1: An argument of type `X`.
p_2: An argument of type `Y`.
p_3: `X` challenges `Y` (i.e. `X` $\in ASCQ($`Y`$)$).
c: p_1 attacks p_2.

(b) Attack between ASPT Arguments

ALT=$\langle\{p_1$-$p_5\}, c, \{A.TR, B.TR, S\}\rangle$
p_1: `A` is an argument of $ASPT$
p_2: `B` is an argument of $ASPT$
p_3: `A.TR` is offered at step S.
p_4: `B.TR` is offered at step S.
p_5: `A.TR` is an alternative to `B.TR`.
c: `A` attacks `B`.

Table 2 shows two different attack types. In Table 2a, we provide the attack scheme T_{cq} to represent an undercutting attack between two arguments, where the argument scheme X is a critical question of Y. Note that each argument scheme can be represented as a defeasible rule in the knowledge base. Therefore, challenging one argument through a critical question would mean an undercutting attack. Table 2b gives an example of a domain-specific attack scheme, where an attack exists between two arguments because two treatments promoting the same goal are alternatives to each other. Definition 7 captures the idea of an attack, which is constructed when an attack scheme is initialised.

Definition 7 (Attack). $[ATS]att_i = \langle \mathsf{Prem}(ATS_i), \mathsf{Conc}(ATS_i)\rangle$ *is an attack, which is derived from the attack scheme instantiation* ATS_i.

We make use of Dung's abstract argumentation framework [4], captured in Definition 8, to evaluate the arguments and attacks generated by the schemes in a KB. In a Dung AF, the idea is represent arguments as nodes, and attacks among them with arrows in a directed graph; it is abstract in the sense that the internal structure of arguments and attacks is not defined.

Definition 8 (Dung Argumentation Framework, Dung AF). *A Dung AF is a tuple* $\langle \mathcal{A}', \mathcal{R}'\rangle$*, where* $\mathcal{A}'$ *is the set of arguments and* $\mathcal{R}' \subseteq \mathcal{A}' \times \mathcal{A}'$ *is a relation such that for arguments* a *and* b, $(a, b) \in \mathcal{R}'$ *iff* $\{a, b\} \subseteq \mathcal{A}'$ *and* a *attacks* b.

Having defined the notions of argument and attack in a structured way previously, we can map these concepts into a Dung AF (Definition 9). The aim of this translation will be to compute acceptable arguments in the structured AF that we construct. Note that $\mathsf{Prem}(x)[i]$ returns the ith premise of x, where x is a scheme instantiation.

Definition 9 (Argumentation Framework, AF). *An argumentation framework is a tuple* $\langle \mathcal{A}, \mathcal{R}\rangle$*, where* $\mathcal{A}$ *and* $\mathcal{R}$ *are, respectively, the set of arguments (Definition 5) and the set of attacks (Definition 7). The mapping to a Dung AF* $\langle \mathcal{A}', \mathcal{R}'\rangle$ *is as follows:* $\mathcal{A}' = \mathcal{A}$*;* $\mathcal{R}' = \{(\mathsf{Prem}(r)[0], \mathsf{Prem}(r)[1]) \mid r \in \mathcal{R}\}$.

Given a Dung AF, it is typical to evaluate it by computing the *acceptable* arguments according to the chosen Dung semantics [3,4]. For example we might

use the *grounded* or the *preferred* semantics. The grounded semantics is sceptical in the sense that one can only accept arguments that cannot be rejected for any reason; whereas one can accept mutually exclusive alternative arguments (each set represented in different *extensions*) while using preferred semantics. Under the chosen semantics, the acceptable arguments are the ones that can be considered to hold for that AF. In this paper, we introduce the idea of an *acceptable attack* in Definition 10.

Definition 10 (Acceptable attack). *An attack is acceptable, if* $\forall r \in \mathcal{R}$, $\mathsf{Prem}(r)[0]$ *is an acceptable argument in Dung AF,* $\mathcal{R}$ *being the set of attacks.*

We distinguish such attacks because we believe that they are key to understanding, and *explaining* why a particular set of arguments is acceptable. More than one extension may hold when evaluating an AF under the chosen semantics such as the preferred semantics. Therefore, each extension will consist of a pair of acceptable arguments and attacks. Definition 11 captures this.

Definition 11 (Acceptability). $ACC = \langle AF, \mathcal{S} \rangle$ *denotes the set of* $(A_{arg}, A_{att})_i$ *where:* $\mathcal{S}$ *is the chosen semantics to evaluate* AF, $(A_{arg}, A_{att})_i$ *is the pair of acceptable arguments and attacks in the ith extension of* AF.

Now that we have the sets of acceptable arguments and attacks, we can use argument and attack schemes to give rationales behind the existence of arguments and attacks within the argumentation framework. The basic idea is to map the acceptable arguments and attacks into the *explanation templates* that we introduced in [27]. Definition 12 captures the idea of an explanation template for an argument scheme. An explanation template for the ASPT scheme can be described as: $e_1 = \langle ASPT,$ *"Treatment* $\{TR\}$ *should be considered at step* $\{S\}$ *as it promotes the goal of* $\{G\}$*."*$\rangle$. The variables are shown in curly brackets in textual representation, and the template includes all the variables (TR, S, G) that exist in ASPT scheme. The explanation definitions below are similar for attacks, where AS and $[AS]arg_i$ are replaced by ATS and $[ATS]att_i$ respectively.

Definition 12 (Explanation template). *An explanation template is a tuple* $E{=}\langle AS, t \rangle$*, where* AS *is an argument scheme, and* t *is a text in natural language that can include variables* V *such that* $V{\subseteq}\mathsf{Var}(AS)$.

We build explanations from explanation templates by instantiating them with acceptable arguments (and attacks). Each variable in the explanation text ($E.t$) is replaced by a ground term found in the argument (attack) scheme instantiation, giving us Definition 13. An explanation for an ASPT argument can be represented as: $\langle e_1, \langle \{goal(rbp), promotes(d, rbp), indicatedAt(d, s1)\}, offer(d) \rangle \rangle$. In this case, $e_1.t$ will become *"Treatment d should be considered at step 1 as it promotes the goal of reducing blood pressure."*

Definition 13 (Explanation). *An explanation is a tuple* $\langle E, [AS]arg_i \rangle$*, where* E *is an explanation template of the argument scheme* AS, $[AS]arg_i$ *is an acceptable argument (Definition 11); and for each variable* $v \in E.t$, $E.t\{v \mapsto \mathsf{Gr}(AS_i)(v)\}$.

A given argument scheme might have different explanations in different contexts [19,32]. For example, patients and healthcare professionals may see different explanations for the same set of acceptable arguments and attacks concerning a medical decision. For now, however, we assume that each scheme is associated with a single explanation template, leaving the question of handling context-specific explanations for future work.

3.2 Mapping ASPIC+ theory into our formal model

In this section, we show how an existing argumentation theory, the well-known ASPIC+, can be mapped into our formal model. We do this to demonstrate the expressibility of our approach. Our formal model includes explainability features by design, which cannot be represented in existing approaches directly. Therefore, we only define mappings for arguments and attacks.

Proposition 1 *An ASPIC+ argument can be represented as an argument constructed according to Definition 5.*

Proof Sketch. Assume that we have a defeasible rule r, where the conjunction of predicates implies the conclusion ($p_1, ..., p_n \Rightarrow c$) and $r \in \mathcal{R}_d$, where $\mathcal{R}_d$ is the set of defeasible rules in an ASPIC+ argumentation theory. In this theory, the knowledge base K includes all the predicates p_i, where i=1, .., n. $Prem$, $Conc$, Sub, $DefRules$, and $TopRule$ are functions defined in the theory; where $Prem$ is the set of premises and $Conc$ is the conclusion of the argument, Sub returns all sub-arguments, $DefRules$ returns all the defeasible rules and $TopRule$ returns the last rule to construct the argument. Assume that A is an argument on the basis of this theory such that $Sub(A) = \{A\}$, $DefRules(A) = r$, $TopRule(A) = r$. Hence, the corresponding ASPIC+ argument is '$A : p_1, ..., p_n \Rightarrow c$'.

With our formal model, we can represent r as the argument scheme $as = \langle\{p_1, ..., p_n\}, c, \{\}\rangle$, the scheme instantiation as $as_1 = \langle as, \{\}, KB\rangle$; where the knowledge base KB includes all the predicates p_i and the argument scheme as. The mapping from an ASPIC+ argument to our formal model is then straightforward: $Prem(A) = \mathsf{Prem}(as_1)$, $Conc(A) = \mathsf{Conc}(as_1)$. The ASPIC+ argument A can then be represented as $\langle\{p_1, ..., p_n\}, c\rangle$.

Proposition 2 *ASPIC+ attack types (undermining, rebuttal and undercutting) can be represented as attack schemes according to Definition 6.*

Proof Sketch. Assume that A and B are two ASPIC+ arguments. A premise in argument A can be a contrary of the conclusion of an argument B (rebuttal), or a premise in argument A can be a contrary of a premise in argument B (undermining). All these attack types are represented by the use of the $\mathsf{contrary}$ function in ASPIC+. Proposition 1 ensures that ASPIC+ arguments can be represented as arguments in our formal model. Table 2a shows an undercutting attack; whereas other attack types can be represented through the use of additional predicates in attack schemes as well. However, domain-specific attacks, such as the one in Table 2b, can only be represented with our formal model.

Algorithm 1 EvalAf $(\mathcal{X}, \mathcal{S})$

Input: $\mathcal{X}$, the set of schemes of interest
Input: $\mathcal{S}$, chosen semantics
Output: ACC, the sets of acceptable arguments and attacks
Require: KB, the knowledge base

1: $\mathcal{A} \leftarrow \{\}$, $\mathcal{R} \leftarrow \{\}$
2: $I \leftarrow \mathsf{instantiateSchemes}(\mathcal{X}, KB)$
3: **for all** i in I **do**
4: $\quad x \leftarrow \mathsf{arg}(i.sname, \mathsf{Prem}(i), \mathsf{Conc}(i))$
5: $\quad \mathcal{A} \leftarrow$ ExtendArg$(x, \mathcal{A})$
6: $K \leftarrow \mathsf{instantiateAttSchemes}(\mathcal{A}, KB)$
7: **for all** k in K **do**
8: $\quad at \leftarrow \mathsf{att}(k.sname, \mathsf{Prem}(k), \mathsf{Conc}(k))$
9: $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{at\}$
10: $AF \leftarrow \mathsf{computeAF}(\mathcal{A}, \mathcal{R})$
11: $ACC \leftarrow \mathsf{getAccepted}(AF, \mathcal{S})$
12: **return** ACC
13: **function** ExtendArg$(arg, \mathcal{A})$
14: $\quad Q \leftarrow \mathsf{getCQs}(arg.sname)$
15: $\quad$ **for all** q in Q **do**
16: $\quad\quad J \leftarrow \mathsf{instantiateScheme}(q.name, KB)$
17: $\quad\quad$ **for all** j in J **do**
18: $\quad\quad\quad a \leftarrow \mathsf{arg}(j.sname, \mathsf{Prem}(j), \mathsf{Conc}(j))$
19: $\quad\quad\quad \mathcal{A} \leftarrow \mathcal{A} \cup \{a\}$
20: $\quad\quad\quad \mathcal{A} \leftarrow$ ExtendArg$(a, \mathcal{A})$
21: $\quad$ **return** $\mathcal{A}$

3.3 The EvalAF algorithm

Having introduced our representation, we propose the EvalAf algorithm. An agent can employ this algorithm to: (i) generate an argumentation framework from a knowledge base, and (ii) compute extensions under a chosen semantics. EvalAf thus provides an operational semantics for our system of schemes and critical questions, showing how they translate into arguments and attacks that conform to the proposed formal model.

The EvalAf algorithm requires two inputs: the set of schemes of interest ($\mathcal{X}$) to initialise arguments and a semantics ($\mathcal{S}$) to compute the extensions in the argumentation framework. In other words, $\mathcal{X}$ includes the scheme set to initialise the construction of an AF; therefore, only relevant arguments are constructed. The output of the algorithm is the sets of acceptable arguments and acceptable attacks. KB is the knowledge base that includes domain-specific content such as schemes, critical questions, facts and rules (Definition 3). The set of arguments ($\mathcal{A}$) and the set of attacks ($\mathcal{R}$) are initialised as empty sets (line 1). The function instantiateSchemes is used to instantiate the schemes in $\mathcal{X}$ (Definition 4) (line 2). Each instantiation is translated into an argument x (Definition 5), and the set of arguments is updated to include more arguments as a result of applying critical

questions (line 5). So far, all possible arguments are constructed regarding argument schemes; in line 6, the attack schemes are instantiated to generate attacks among arguments. For each of these instantiations, an attack relation is formed and added to the set of attacks (lines 8-9). The auxiliary function computeAF generates an argumentation framework by using the sets of arguments and attacks (Definition 9). At this point, the AF can be used to make a decision under the chosen semantics $\mathcal{S}$. getAccepted returns the sets of acceptable arguments and attacks (line 11) (Definition 11).

The function ExtendArg is described between the lines 13 and 21. The inputs are an argument arg, and the current set $\mathcal{A}$. Since each argument is constructed according to an argument scheme, the set of CQs are collected in order to challenge the current argument arg (Definition 2)(line 14). Each CQ is a scheme to be initialised (line 16), and an argument is generated accordingly (line 18). The set of arguments is updated (line 19). Each new CQ argument can be challenged by other CQs as well, hence ExtendArg is invoked recursively (line 20). Note that at this point the order in which we ask critical questions is not important, because KB does not change but the argumentation framework is updated by the construction of new arguments and attacks.

It is easy to show that the algorithm is sound in the sense that it only returns arguments that are acceptable:

Proposition 3 (Soundness) *Given a set of argument schemes and a semantics, the set of arguments returned by* EvalAf *are acceptable arguments.*

Proof Sketch. getAccepted ensures the mapping into a Dung AF (Definitions 8 and 9). Existing reasoning tools, such as *Aspartix* [5], can be used to compute acceptable arguments. Therefore, EvalAf returns acceptable arguments as well.

Note that EvalAf will always construct all the arguments and attacks given an initial set of argument schemes:

Proposition 4 (Completeness/AF) EvalAf *returns the complete AF when arguments are instantiated from an argument scheme in* $\mathcal{X}$.

Proof Sketch. Assume that $\mathcal{F}_1$ is the complete Dung AF that can be constructed given a specific $\mathcal{X}$ (Definition 8); i.e, $\mathcal{F}_1$ will include all the possible arguments and attacks. When an agent invokes EvalAf each argument scheme in $\mathcal{X}$ will be instantiated to construct an argument (Definition 5). ExtendArg is then invoked recursively to instantiate all the argument schemes and add the resulting arguments to the set of arguments. Hence, all possible arguments will be constructed. instantiateAttSchemes will initialise all possible attacks. computeAF will then construct an AF, which can then mapped into the Dung AF, $\mathcal{F}_2$ (Definition 9). Since there can only exist one Dung AF, $\mathcal{F}_1$ and $\mathcal{F}_2$ should be the same. Therefore, EvalAf constructs the complete AF given a specific $\mathcal{X}$.

Proposition 5 (Completeness/Acceptability) EvalAf *returns all the acceptable arguments that are instantiated from an argument scheme in* $\mathcal{X}$. EvalAf *returns all the acceptable attacks that are instantiated from acceptable arguments.*

Proof Sketch. EvalAf returns all acceptable arguments in a complete AF (follows from Propositions 3 and 4). Definition 10 ensures that there is an acceptable attack when an argument is acceptable. Hence, EvalAf returns all the acceptable arguments and attacks in an argumentation framework.

3.4 The ExpAF algorithm

The next important step is to map acceptable arguments and attacks into explanations in natural language. In this section, we propose ExpAf algorithm that conforms to Definitions 12 and 13. The algorithm requires two inputs the set of acceptable arguments ($\mathcal{A}'$) and the set of acceptable attacks ($\mathcal{R}'$). KB is the knowledge base that includes domain-specific information as before. EvalAf ensures that only relevant arguments and attacks will be constructed. In other words, the agent will not try to initialise all the schemes in its KB but it will start constructing the argumentation framework with the ones specified in $\mathcal{X}$. Hence, when an agent provides outputs of EvalAf algorithm as inputs to ExpAf algorithm, it will get explanations that are relevant to the problem instance.

In line 1, the sets of explanations for arguments and attacks are initialised as empty sets. *objects* keeps a set of all the inputs. For each object o, getSchemeName returns the scheme name (line 4), getExpTemplate returns the explanation template e (Definition 12) (line 5); and the explanation tuple exp (Definition 13) is generated by the generateExp (line 6). The explanation tuple is added to E_{arg} if o is an acceptable argument; otherwise, exp is added to E_{att}. The algorithm returns the explanation sets for each object (line 11).

Proposition 6 ExpAf *always returns explanations for acceptable arguments and attacks.*

Proof Sketch. The input $\mathcal{A}'$ includes acceptable arguments; for each argument, there will be an instantiated scheme. If there is an instantiated scheme, an explanation template will exist; and this template will have an instance initialised with ground terms, which will constitute an explanation (follows from Definitions 4, 5, 12 and 13). ExpAf conforms to these definitions; hence, it provides an explanation for any acceptable argument. Similar reasoning holds for attacks.

The output of ExpAf can then be used by tools to provide explanations for acceptable arguments and/or attacks.

4 Arguments and attacks

The formal model introduced above can be used in order to describe a particular domain. We represent the hypertension domain via first-order language predicates as in our previous work [17]. This language consists of predicates of different arities. Variables are denoted as capital letters, the predicates are written in italic text and the constants are in lower case. The knowledge base (KB) includes information such as the clinical guidelines, patient information, argument and attack schemes in terms of facts and rules.

Algorithm 2 ExpAf $(\mathcal{A}', \mathcal{R}')$

Input: $\mathcal{A}'$, the set of acceptable arguments
Input: $\mathcal{R}'$, the set of acceptable attacks
Output: E_{arg}, E_{att}, sets of explanations for arguments and attacks
Require: KB, the knowledge base

$E_{arg} \leftarrow \{\}$, $E_{att} \leftarrow \{\}$
$objects \leftarrow \mathcal{A}' \cup \mathcal{R}'$
for all o in $objects$ **do**
    $sname \leftarrow o.\mathsf{getSchemeName}(KB)$
    $e \leftarrow \mathsf{getExpTemplate}(sname, KB)$
    $exp \leftarrow \mathsf{generateExp}(e, o)$
    if $o \in \mathcal{A}'$ **then** ▷ o is an acceptable argument
        $E_{arg} \leftarrow E_{arg} \cup exp$
    else ▷ o is an acceptable attack
        $E_{att} \leftarrow E_{att} \cup exp$
return E_{arg}, E_{att}

4.1 Guideline representation

In the domain and example that follow we refer to the NICE hypertension guidelines [23]. NICE[5] has a set of guidelines to help healthcare professionals in diagnosing and treating primary hypertension. The guideline includes treatment steps, such that a patient progresses to the next step and takes a new drug if their blood pressure control does not improve in the previous step. It provides guidance on which of the treatments or treatment combinations should be considered at each step. For example, c (Calcium-Channel Blocker) and d (Thiazide-like Diuretic) are two treatment options that may be offered if the patient facts indicate a goal of blood pressure reduction. A treatment that *promotes* a *goal*, can be offered or not offered (predicates are *offer* and *notoffer* respectively). Moreover, a treatment can be marked as *offered* at a specific time. A treatment can be *indicatedAt* a specific step according to guidelines. *greater* is used to define an ordering between different time points. A treatment that is previously prescribed *may_cause* a side effect.

4.2 Patient information

The choice of a treatment may depend on the *fact*s about a patient. In the hypertension domain; *age*, *ethnic_origin*, the current treatment *step* in the treatment process and an *observation* about the patient are important facts to consider before recommending a particular treatment. Observations include information such as if any side-effect has been reported or the desired goal (e.g. reduction in blood pressure) has been achieved. Such information can dynamically be added to the knowledge base. For example, in our previous work, we showed that the

[5] `https://www.nice.org.uk/`

knowledge base can be populated with patient facts collected via wellness sensors [16].

4.3 Argument schemes

We use the ASPT scheme in order to construct arguments in support of different treatment options (Table 1). There are different reasons precluding a treatment from being an option for a specific patient, so there are multiple critical questions associated with ASPT–we just show one in the following. SE scheme ascertains that no treatment will be offered if a side effect is observed (i.e. $ASCQ(\text{ASPT}) = \{\text{SE}\}$). SE is challenged by SEF scheme in situations where the treatment is effective so should not be excluded as an option despite the side effect (i.e. $ASCQ(\text{SE}) = \{\text{SEF}\}$). In Table 3, the first frame shows these schemes in a first-order language.

4.4 Attack schemes

In Section 3, we have discussed different types of attacks that could exist among arguments, and Table 2a gives an example of an undercutting attack between arguments. Now, we give an example of an attack scheme that is domain-specific and describes the rationale behind an attack in terms of domain-specific premises. Table 2b shows the attack scheme, ALT, belonging to the hypertension domain. The ALT scheme defines an attack between two ASPT arguments when two treatments promoting the same goal are offered at a specific step. ALT has a similar intuition as *Argument from Alternative* scheme proposed by Walton [30]. The instantiation of this attack scheme will result in attacks among alternative treatment arguments in the argumentation framework.

5 A stroke survivor: Baula

We will work through the case of Baula a 32-year-old person of African origin. Baula suffered a stroke and has hypertension. In order to prevent secondary stroke, Baula's blood pressure (BP) needs to be controlled. Baula has started using a new medication c to control blood pressure as suggested by a GP. During a follow up visit, Baula's BP is 130/90 (indicating the treatment is having the desired BP lowering effect) but there is a side effect (swollen ankles). In the light of this information, *what* are the treatment options to consider and *why*?

We now illustrate the use of EvalAf algorithm on the example. The two inputs provided to the algorithm are $ASPT$ and $preferred$, respectively. In our running example, the goal is set to reducing blood pressure (rbp) by default. The construction of the arguments will start by initialising $ASPT$ according to the information available in the knowledge base. The use of $preferred$ semantics will ensure that there can be multiple acceptable sets of arguments and attacks. The human user (e.g. Baula's GP) will make a final decision in the light of

Table 3: Arguments schemes and arguments used in the running example

Schemes	
ASPT	$\langle$ {*goal*(G), *promotes*(TR,G), *indicatedAt*(TR,S)}, *offer*(TR), {G,TR,S} $\rangle$
SE	$\langle$ {*greater*(T,T'), *offered*(TR,T'), *may_cause*(TR,S)}, *notoffer*(TR), {T,T',TR,S} $\rangle$
SEF	$\langle$ {*greater*(T,T'), *effective*(TR,T')}, *offer*(TR), {T,T',TR} $\rangle$
Arguments	
$[ASPT]arg_1$:	$\langle$ {*goal*(rbp), *promotes*(*c*,rbp), *indicatedAt*(*c*,s1)}, *offer*(*c*) $\rangle$
$[ASPT]arg_2$:	$\langle$ {*goal*(rbp), *promotes*(*d*,rbp), *indicatedAt*(*d*,s1)}, *offer*(*d*) $\rangle$
$[SE]arg_{1.1}$:	$\langle$ {*greater*(t_2,t_1), *offered*(*c*,t_1), *caused*(*c*,swollen-ankles,t_1)}, *notoffer*(*c*) $\rangle$
$[SEF]arg_{1.1.1}$:	$\langle$ {*greater*(t_2,t_1), *effective*(*c*,t_1)}, *offer*(*c*) $\rangle$

the suggested possible solutions. The bottom part of Table 3 includes all the arguments generated by the algorithm.

The arguments arg_1 and arg_2 are constructed as a result of the instantiation of ASPT. The CQs for each scheme are considered in the following steps. SE is relevant only to arg_1, and given the side effects there is an attack generated on arg_1 from $arg_{1.1}$. Even if Baula reports side effects, c is still effective in *rbp*. Therefore, $arg_{1.1.1}$ attacks $arg_{1.1}$ as well. Figure 1 depicts the resulting AF; where arguments are displayed as boxes, the solid arrows represent attacks instantiated by T_{cq}, and the dashed arrows show attacks instantiated by the ALT scheme. For simplicity, the attacks are annotated without scheme names. Each attack has a unique label att_i. att_4, which conforms to the attack scheme T_{cq}, is instantiated as: $\langle$ {$[SEF]arg_{1.1.1}$, $[SE]arg_{1.1}$, *challenges*(SEF, SE)}, *attacks*($arg_{1.1.1}$, arg_1) $\rangle$. att_1, which conforms to the attack scheme ALT, is instantiated as: $\langle$ {$[ASPT]arg_2$, $[ASPT]arg_1$, *alt*(c, d), *indicatedAt*(c, $s1$), *indicatedAt*(d, $s1$)}, *attacks*(arg_2, arg_1) $\rangle$. att_2 and att_4 are instantiated similarly.

The EvalAf algorithm returns the set of acceptable arguments and attacks (Definition 11). Under the preferred semantics, there are two extensions: ({arg_1, $arg_{1.1.1}$}, {att_2, att_4}) and ({arg_2, $arg_{1.1.1}$}, {att_1, att_4}).

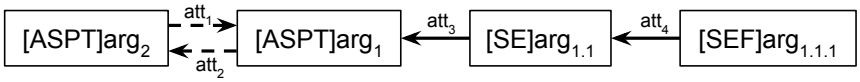

Fig. 1: Argumentation framework constructed by EvalAf algorithm.

5.1 Explanations

As argument and attack schemes are associated with explanation templates (Definition 12), agents can instantiate them with an algorithm like ExpAf to generate explanations in natural language (Definition 13). In Section 3, we introduced

e_1 as an explanation template for ASPT. In a similar way, we can describe an explanation template for the ALT scheme as: $\langle ALT$, *"Since* $\{A.TR\}$ *and* $\{B.TR\}$ *promote the same goal at step* $\{S\}$, $\{A.TR\}$ *is an alternative to* $\{B.TR\}$*; hence, they should not be offered together."*$\rangle$.

When we consider the following extension ($\{arg_2, arg_{1.1.1}\}$, $\{att_1, att_4\}$), ExpAf will generate an explanation for each acceptable argument and attack. For example, an explanation for arg_2 will be *"Treatment d should be considered at step 1 as it promotes the goal of reducing blood pressure."*; and an explanation for att_1 will be *"Since d and c promote the same goal at step 1, d is an alternative to c; hence, they should not be offered together."*, applying the explanation templates described above. The remaining explanations are generated in a similar way by the use of explanation templates. $arg_{1.1.1}$ can be explained as *"Treatment ccb can be considered as it was an effective treatment at time t1."*; and att_4 can be explained as *"The scheme sef is a critical question of the scheme se"*.

Given the suggested treatments and explanations, one option is to continue the current treatment c as it is effective, another option is to offer a new treatment such as d. At this point, the GP should also consider Baula's preferences when making a decision.

5.2 Demonstration of the proposed approach

We provide an example implementation of the proposed approach in our GitLab repository[6]. Baula's example is also provided to demonstrate the applicability of the proposed algorithms. By running the code, one can get the textual explanations for acceptable arguments and attacks as described in this paper. In our implementation, we make use of *Aspartix*, an answer set programming approach to find the justified arguments of an argumentation framework [5]. In a first-order language, we describe the knowledge base, the argument and attack schemes, and data about Baula in terms of facts and rules. More use cases can be described in a similar manner. We also make our Python-based implementation public; we share an implementation of the proposed algorithms that use explanation templates to generate textual explanations. Moreover, our implementation provides means to export the generated Dung AF as a graph, which is useful in providing a visual explanation of the constructed arguments and attacks.

Figure 2 depicts one extension as a Dung AF where the recommended action is offering a new treatment d (thiazide). Each box represents an argument constructed using argument schemes, each arrow represents an attack between arguments constructed using attack schemes. The acceptable arguments are highlighted with a green color. Note that since we are using preferred semantics in this example, there is also another extension (i.e. another graph) supporting the idea of using the current treatment c (ccb).

[6] `https://git.ecdf.ed.ac.uk/nkokciya/explainable-argumentation/`

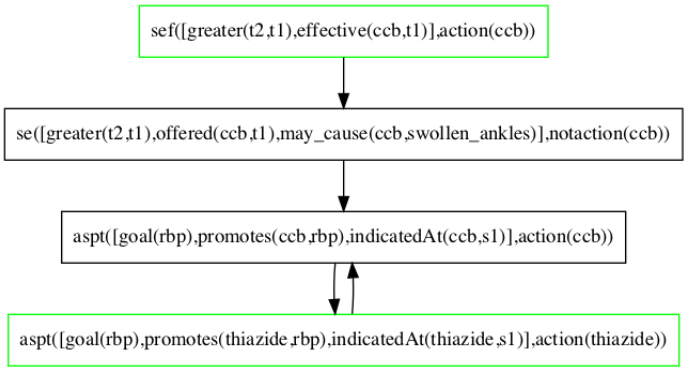

Fig. 2: One extension recommends offering a new treatment thiazide (d)

6 Discussion and conclusion

We proposed a formalism to describe arguments and attacks in a given domain through the use of schemes. We introduced the notion of attack schemes to capture domain-specific conflicts between two arguments. We articulated an algorithm that generates an AF from a set of schemes to establish the set of acceptable arguments and attacks and provided soundness and completeness proof sketches. We also introduced another algorithm for generating explanations, and illustrated our approach through an example. We showed that further explanations can be generated by extending the acceptable set of arguments with the acceptable attacks. Intuitively, this enables the explainability of both accepted and defeated arguments through the instantiation of argument schemes and attack schemes respectively. We shared a prototype implementation to demonstrate how our approach works in practice. In this work, the initialisation of arguments and attacks is performed according to the information available in the knowledge base. In this paper, the description of schemes or guideline rules is static; however, such information can be automatically learned from data.

There are two important steps to achieve explainability by design. First, we need a formal model that captures the essential components of a decision, and we propose the use of argument and attack schemes in this paper. Second, we need methods to deliver explanations for end-users. In this paper, we propose a simple algorithm to generate textual explanations based on explanation templates. However, machine learning techniques could be used to construct explanations in a dynamic way, something which is out of the scope of this paper. In future work, we will extend the explainability definitions to cover cases such as how an attack affects the status of an argument in an AF. We are planning to develop user interfaces where we will show graphs and explanations together, similar to [28]. Moreover, we will evaluate the quality of generated explanations by conducting user studies. Finally, we will extend our theoretical results to fully explain the translation of existing argumentation frameworks into our proposed approach.

Acknowledgements

This work was supported by the UK Engineering & Physical Sciences Research Council (EPSRC) under grant #EP/P010105/1.

References

1. Atkinson, K., Bench-Capon, T.: Practical reasoning as presumptive argumentation using action based alternating transition systems. Artificial Intelligence **171**(10-15), 855–874 (2007)
2. Atkinson, K., Bench-Capon, T., Modgil, S.: Argumentation for decision support. In: Database and Expert Systems Applications. pp. 822–831. Springer (2006)
3. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. The Knowledge Engineering Review **26**(4), 365–410 (2011)
4. Dung, P.M.: 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)
5. Egly, U., Gaggl, S., Woltran, S.: Aspartix: Implementing argumentation frameworks using answer-set programming. Logic Programming pp. 734–738 (2008)
6. European Parliament and Council of the European Union: General Data Protection Regulation (GDPR). `https://gdpr-info.eu` (2016), last accessed 12/02/2020
7. Fan, X., Toni, F.: On explanations for non-acceptable arguments. In: International Workshop on Theory and Applications of Formal Argumentation. pp. 112–127. Springer (2015)
8. Fox, J., Glasspool, D., Grecu, D., Modgil, S., South, M., Patkar, V.: Argumentation-based inference and decision making–A medical perspective. IEEE Intelligent Systems **22**(6), 34–41 (2007)
9. Glasspool, D., Fox, J., Oettinger, A., Smith-Spark, J.: Argumentation in decision support for medical care planning for patients and clinicians. In: AAAI Spring Symposium: Argumentation for Consumers of Healthcare. pp. 58–63 (2006)
10. Glasspool, D., Oettinger, A., Smith-Spark, J., Castillo, F., Monaghan, V., Fox, J., et al.: Supporting medical planning by mitigating cognitive load. Methods of Information in Medicine **46**(6), 636–640 (2007)
11. Gordon, T.F., Walton, D.: Legal reasoning with argumentation schemes. In: Proceedings of the 12th International Conference on Artificial Intelligence and Law. pp. 137–146. ACM (2009)
12. Gordon, T.F., Walton, D.: Formalizing balancing arguments. In: Proceedings of the Conference on Computational Models of Argument. pp. 327–338 (2016)
13. Grando, M.A., Moss, L., Sleeman, D., Kinsella, J.: Argumentation-logic for creating and explaining medical hypotheses. Artificial Intelligence in Medicine **58**(1), 1–13 (2013)
14. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Computing Surveys **51**(5), 1–42 (2019)
15. Kakas, A.C., Moraitis, P., Spanoudakis, N.I.: Gorgias: Applying argumentation. Argument & Computation **10**(1), 55–81 (2019)
16. Kökciyan, N., Chapman, M., Balatsoukas, P., Sassoon, I., Essers, K., Ashworth, M., Curcin, V., Modgil, S., Parsons, S., Sklar, E.: A collaborative decision support tool for managing chronic conditions. In: MEDINFO 2019: Health and Wellbeing e-Networks for All. vol. 264, pp. 644–648 (2019)

17. Kökciyan, N., Sassoon, I., Young, A., Chapman, M., Porat, T., Ashworth, M., Curcin, V., Modgil, S., Parsons, S., Sklar, E.: Towards an argumentation system for supporting patients in self-managing their chronic conditions. In: AAAI Joint Workshop on Health Intelligence (2018)
18. Kökciyan, N., Yaglikci, N., Yolum, P.: An argumentation approach for resolving privacy disputes in online social networks. ACM Transactions on Internet Technology **17**(3), 27:1–27:22 (2017)
19. Kökciyan, N., Yolum, P.: Context-based reasoning on privacy in internet of things. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. pp. 4738–4744 (2017)
20. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence **267**, 1–38 (2019)
21. Modgil, S., Bench-Capon, T.: Metalevel Argumentation. Journal of Logic and Computation **21**(6), 959–1003 (2011)
22. Modgil, S., Prakken, H.: A general account of argumentation with preferences. Artificial Intelligence **195**, 361–397 (2013)
23. NICE: Hypertension in adults: diagnosis and management. `https://www.nice.org.uk/guidance/cg127` (2016), last accessed 12/02/2020
24. Prakken, H., Wyner, A., Bench-Capon, T., Atkinson, K.: A formalization of argumentation schemes for legal case-based reasoning in ASPIC+. Journal of Logic and Computation **25**(5), 1141–1166 (2013)
25. Rago, A., Cocarascu, O., Toni, F.: Argumentation-based recommendations: fantastic explanations and how to find them. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. pp. 1949–1955. AAAI Press (2018)
26. Rahwan, I., Simari, G.R.: Argumentation in Artificial Intelligence. Springer (2009)
27. Sassoon, I., Kökciyan, N., Sklar, E., Parsons, S.: Explainable argumentation for wellness consultation. In: Calvaresi, D., Najjar, A., Schumacher, M., Främling, K. (eds.) Explainable, Transparent Autonomous Agents and Multi-Agent Systems. pp. 186–202. Springer International Publishing (2019)
28. Sklar, E., Parsons, S., Li, Z., Salvit, J., Wall, H., Mangels, J.: Evaluation of a trust-modulated argumentation-based interactive decision-making tool. Journal of Autonomous and Multi-Agent Systems **30**(1), 136–173 (2016)
29. Tolchinsky, P., Cortes, U., Modgil, S., Caballero, F., Lopez-Navidad, A.: Increasing human-organ transplant availability: Argumentation-based agent deliberation. IEEE Intelligent Systems **21**(6), 30–37 (2006)
30. Walton, D., Reed, C., Macagno, F.: Argumentation Schemes. Cambridge University Press (2008)
31. Zamborlini, V., da Silveira, M., Pruski, C., ten Teije, A., Geleijn, E., van der Leeden, M., Stuiver, M., van Harmelen, F.: Analyzing interactions on combining multiple clinical guidelines. Artificial Intelligence in Medicine **81**, 78–93 (2017)
32. Zeng, Z., Fan, X., Miao, C., Leung, C., Jih, C.J., Soon, O.Y.: Context-based and explainable decision making with argumentation. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. pp. 1114–1122 (2018)