STUDIES IN LOGIC, GRAMMAR AND RHETORIC 23 (36) 2011

Paweł Łoziński
Warsaw University of Technology

AN ALGORITHM FOR INCREMENTAL ARGUMENTATION ANALYSIS IN CARNEADES

Abstract: *Carneades* is an interactive application for argument construction, evaluation and visualization, integrating an knowledge-based inference engine and an argument mapping tool. Given the argument sources and the goal of argumentation process, *Carneades* conducts a resource-limited search for arguments for and against the given goal. The result of the search is an argumentation graph which can then be visualized and analized by the application user, e.g. an expert in the legal domain. This article presents a different, incremental approach to the exploration of argumentation space with a search algorithm using heuristics and search constraints for choosing the exploration paths. The article describes a motivation for such an approach, construction and implementation of the algorithm together with its comparison to argument construction in *Carneades*.

Keywords: argumentation theory, logic programming, reasoning, Carneades

Introduction

Carneades is an interactive application for argument construction, evaluation and visualization, integrating an knowledge-based inference engine and an argument mapping tool. As described in [3], *Carneades* is designed to handle multiple sources of arguments including ontologies, rules, cases and testimonial evidence.

Given the argument sources and the goal of the argumentation process, *Carneades* conducts a resource-limited search for arguments for and against the given goal. The result of the search is an argumentation graph which can then be visualized and analized by the application user, e.g. an expert in the legal domain. All the statements in the graph are labeled to indicate whether the statement and it's compilment are acceptable. This method gives the user a full (to the extent of resources avaliable during the search) picture of the subject of discourse, which is both an advantage and a potential source of problems. The advantage is that the user can see and analize the whole picture. However, as it can be seen from argumentation example presented in [4], the graph of arguments that is presented to the

ISBN 978–83–7431–305–6 ISSN 0860-150X

user can be very large, even for a relatively small argumentation case. This makes it difficult for the user to grasp and draw conclusions from. In the software licensing example presented in [4], the argumentation graph for goal *Exists copyright license that may be used by carneades engine* exceeds 100 nodes. Also, the time needed to construct the graph using e.g. arguments generated from ontology, which involves quering the OWL reasoner, can be substantial.

This article presents a different approach to the exploration of argumentation space with a search algorithm using heuristics and search constraints for choosing the exploration paths. The algorithm aims to find a minimal argumentation graph that allows for determining the acceptability of the given goal and to find it in minimal number of steps. In this approach, the analysis of an argumentation case takes an incremental form. After viewing one portion of information avaliable in the argumentation graph, the user may want to query the system with a different goal, using the obtained knowledge.

The rest of the paper is organized as follows. Section 1 briefly describes *Carneades* as defined in [6]. Section two introduces a reformulation of *Carneades* in terms of inference rules. It is important to note that *Carneades* has a notion of arguments *from* rules (e.g. legal rules) as opposed to e.g. arguments from ontologies. In this paper the term rule is used in a different meaning, as an inference pattern and in this sense an argument may be viewed as a rule. Section three presents a general approach to constructing an argumentation space search algorithm. In section four the RPA* search algorithm is presented. In section five the implementation of the algorithm is described. Section six contains a brief comparison of the argumentation graph generation using latest implementation of *Carneades* and $RPA*$. The paper concludes with a brief summary and description of future work on the subject.

1. *Carneades*

Definition 1 (Statements)

Let $(\mathcal{L}, =, \mathit{complement})$ be a structure where $\mathcal{L}$ denotes a set of declarative statements in some language, "=" is an equality relation modeled as a function of type $\mathcal{L} \times \mathcal{L} \rightarrow \mathit{boolean}$, and $\mathit{complement} : \mathcal{L} \rightarrow \mathcal{L}$ is a function mapping a statement to its logical complement. If s is a statement, the complement of s is denoted $\bar{s}$.

Definition 2 (Premises)

Let $\mathcal{P}_\mathcal{L}$ denote the set of premises. There are the following types of premises: (1) If $s \in \mathcal{L}$, then $\diamond s$, called *ordinary premise*, is a premise. (2) If $s \in \mathcal{L}$, then $\bullet s$, called *assumption*, is a premise. (3) If $s \in \mathcal{L}$, then $\circ s$, called *exception*, is a premise. (4) Nothing else is a premise.

Definition 3 (Arguments)

An *argument* is a tuple (c, d, P), where $c \in \mathcal{L}$, $d \in \{pro, con\}$ and $P \in 2^{\mathcal{P}_\mathcal{L}}$.

The key notion in *Carneades* framework is an argument graph, on the bases of which acceptability of statements can be determined.

Definition 4 (Argument graphs)

An *argument-graph* is a labeled, finite, directed, acyclic, bipartite graph, consisting of *argument* nodes and *statement* nodes. The edges link the argument nodes to the statements in the premises and conclusion of each argument. At most one statement node is allowed for each statement s and its complement, $\bar{s}$.

A fragment of an argument graph with one argument node and four statement nodes is shown in 1. Due to the limitation present in this definition, for the sake of graph representation, the notion of *negative premise* is introduced. Premise of every type can be linked to an argument as a negated premise, which is denoted: $\diamond\bar{s}$, $\bullet\bar{s}$, $\circ\bar{s}$.

Definition 5 (Argument context)

Let $\mathcal{C}$, the *argument context*, be a tuple $(status, ps, >)$, where *status* is a function of type $\mathcal{L} \rightarrow \{stated, questioned, accepted, rejected\}$, *ps* is a function of type $\mathcal{L} \rightarrow \mathcal{PS}$ and "$>$" is a strict partial ordering on arguments. $\mathcal{PS}$ is the set of proof standards. For every statement s and its complement $\bar{s}$, the proof standard assigned to $\bar{s}$ is the complement of the proof standard assigned to s and

- if $status(s) = stated$ then $status(\bar{s}) = stated$,
- if $status(s) = questioned$ then $status(\bar{s}) = questioned$,
- if $status(s) = accepted$ then $status(\bar{s}) = rejected$, and
- if $status(s) = rejected$ then $status(\bar{s}) = accepted$.

Statement is acceptable in the given argument graph if its proof standard is satisfied in this graph. In [6] three proof standards are defined (as stated in the article, this list is neither complete nor mandatory, a CAF-based system can have other proof standards defined).

SE **(Scintilla of Evidence)** A statement meets this standard iff it is supported by at least one defensible pro argument.

BA **(Best Argument)** A statement meets this standard iff it is supported by some defensible pro argument with priority over all defensible con arguments.

DV **(Dialectical Validity)** A statement meets this standard iff it is supported by at least one defensible pro argument and none of its con arguments are defensible.

The *complement* of a proof standard σ, denoted $\bar{\sigma}$, is a standard which results from switching the roles of pro and con arguments in the definition of σ.

An argument is *defensible* if all of its premises *hold*. Holding of a premise depends firstly on its type and status of its statement: premise $\diamond s$ holds if $status(s) = accepted$ and doesn't hold if $status(s) = rejected$; $\bullet s$ holds if $status(s) \in \{accepted, stated\}$ and doesn't hold if $status(s) = rejected$; $\circ s$ holds if $status(s) = rejected$ and doesn't hold if $status(s) = accepted$. Secondly, in remaining cases, premise $\diamond s$ or $\bullet s$ holds if statement s is acceptable, premise $\circ s$, holds if statement s is not acceptable.

Rule-based version of *Carneades*

The idea underlying rule-based version of *Carneades* is quite intuitive, i.e. an argument shown in figure 1 can be replaced with an inference rule that uses statements p, q and r to conclude s. Due to space limitations, we present in this section only a shortened definition, based much on intuition and analogy to the original *Carneades* model. The notion of statement's *status* in CAF can be translated to logical value of a statement, and the notion of *acceptability*, similarly as in other approaches to argument-based logic programming, replaces the notion of truthfulness of a statement. More formally speaking:

Definition 6 (RCAF)

Logic *RCAF* (rule version of CAF) is and ordered triple $(\mathcal{L}, \mathcal{S}, \mathcal{IM})$ where $\mathcal{L}$ is a simplified version of FOL language, $\mathcal{S}$ is semantics and $\mathcal{IM}$ is an inference mechanism.

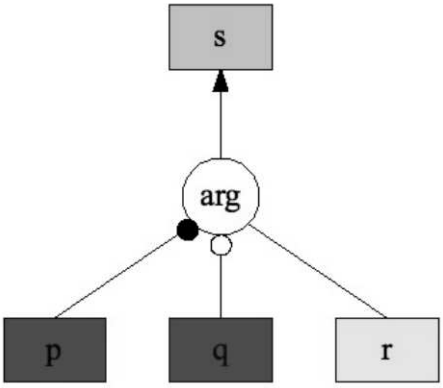

Figure 1. An argument in *Carneades* and a corresponding rule

Definition 7 (Language)

Language $\mathcal{L}$ is a traditional FOL language limited to constants, variables, predicates, negation and existential quantifier, e.g. $\neg\exists_x Likes(John, x)$.

Definition 8 (Semantics)

Semantics $\mathcal{S}$ is a standard semantics for $\mathcal{L}$, with difference in the set of logical values, which has four elements: $\{accepted, rejected, stated, questioned\}$. The values of propositions and their negations correspond to statuses of statements and their complements in CAF.

Definition 9 (Inference mechanism)

Inference mechanism $\mathcal{IM}$ consists of three elements: (i) Function $ps : \mathcal{L} \rightarrow \mathcal{PS}$, which assigns every proposition in $\mathcal{L}$ its proof standard. (ii) Set $\mathcal{R}$ of defined inference rules. (iii) Relation "$>$" $\subseteq \mathcal{R} \times \mathcal{R}$ of strict partial ordering among rules of inference.

Definition 10 (Inference rule)

Inference rule $\mathfrak{r} \in \mathcal{R}$ is a structure of two possible types $p_1, \ldots, p_n \xrightarrow{pro} c$ or $p_1, \ldots, p_n \xrightarrow{con} c$, where $p_1, \ldots, p_n$ are premises and $c \in \mathcal{L}$ is the conclusion. Rules of the first type are called *pro* rules, those of second type are called *con* rules. We say that rule $\mathfrak{r}$ *supports* conclusion c (regardless of the type of the rule). Type of rule $\mathfrak{r}$ is denoted with $T(\mathfrak{r})$, "$\rightarrow$" denotes a rule of either type.

Definition of rule's premises is analogous to def. 2, the proposition of the premise is sometimes refered to as the premise itself, type of the premise p

is denoted $T(p)$. Defensibility of rules and acceptability of propositions is analogous to defensibility of arguments and acceptability of statements in CAF. if its proof stanard is satisfied by ground rules supporting it.

Definition 11 (Satisfaction of proof standard)

We say that the proof standard of ground proposition s is *satisfied* if the set of ground rules supporting s satisfies the condition given in the proof standard's definition (e.g. for DV: there exists a *pro* rule that supports s and there is no *con* rule that supports s).

The condition for satisfaction of complement proof standard is generated from the proof standard's definition be switching types of rules and adding negation to the checked proposition.

Example 1

Let proposition $s =$"Cat is black" and $ps(s) = SE$. The proposition s is acceptable iff $SE(s) = true$, that is, iff there exists at least one *pro* rule supporting s. The proposition $\neg s =$"Cat is not black" is acceptable iff $\overline{SE}(\neg s) = true$, that is, iff there exists at least one *con* rule supporting $\neg\neg s$, that is, one *con* rule supporting s.

3. Logic programming in RCAF

Proving a ground proposition s in RCAF involves finding such substitution of variables in rules, that the set of all rules in $\mathcal{R}$ supporting s satisfies $ps(s)$. The task seems difficult, because it involves searching for several proofs[1] of s that together satisfy proof standard of s. The problem repeats recursively for premises of rules used to prove s.

The solution proposed in this paper is based on the idea presented e.g. in [1], that inference in logic can be modeled as a search problem, which in turn can be solved with one of many "off-shelf" search algorithms. This approach allows a clear formulation of the problem of inference in RCAF and makes relevant all the knowledge gathered in the well researched domain of search in AI.

3.1. Inference as search

The formulation of the inference problem in terms of search is based on the idea, that the set of proofs of a given proposition can be regarded as a

[1] In the classical sense of the word.

search space that needs to be explored in order to find a complete, sound proof of the proposition. For a general, formal definition of this search task the reader is referred to [1].

3.2 Formulation of the search problem in RCAF

Before moving forward, some details should be established, that — although not related to the particular idea of RCAF logic — are necessary to fully define the search task: (a) *Interpretation of variables.* Variables present in premises of proofs are interpreted as being tied with the existential quantifier (e.g. query $King(x)$ is interpreted as $\exists_x King(x)$ and read "Is there a King?"). (b) *Default values.* Even if all propositions in KB have the status and proof standard assigned, those assignments are not established for intermediate products of reasoning. This problem is here solved simply by the introduction of default values: status *stated* and proof standard DV. (c) *Priority of assumptions and exceptions.* Assumptions and exceptions are treated with low priority, that is, as long as the reasoning (e.g. about premise $\bullet R(A)$) can continue, this information is not used to determine holding of the premise.

3.2.1. Search space

Let RCAF $= (\mathcal{L}, \mathcal{S}, \mathcal{IM})$, KB a knowledge base defined in that logic and $q \in \mathcal{L}$ be a ground proposition that is a query asked to the knowledge base with a given proof standard. The query is given value *questioned* and interpreted as an ordinary premise $\diamond q$. Holding of this premise is equivalent to proving q.

Definition 12 (Proof)

Proof of q is a directed graph $P = (V^P, E^P)$, where V^P is a set of vertices represented by ground premises. If edge (p_1, p_2) exists in E^P, then there exists $\mathfrak{r} \in KB$, which has one premise equal to p_1 and conclusion equal to the proposition of p_2 (after appropriate variable substitution).

Subgoal of proof P is a premise $q' \in V^P$ such, that $deg_{in}(q') = 0$ and the proof standard of q' needs to be checked. Proof is called *complete* iff it has no subgoals, otherwise it is *incomplete*. The only vertex of P that has no outgoing edges is q ($deg_{out}(q) = 0$). Number of subgoals in P is denoted with $\delta(P)$.

Definition 13 (Proof space)

Proof space $\mathcal{P}(q)$ is a set of proofs of q. We say that proofs P_1 and P_2 are *neighbouring* in $\mathcal{P}(q)$ iff P_2 is constructed from P_1 with one *inference*

step made using subgoal $q_1 \in V_1^P$. Let $V_1^P = V \cup \{q_i\}_{i=1}^m$, $E_1^P = E \cup E'$, where $E' = \{(q_i, p_i)\}_{i=1}^m$, $\{q_i\}_{i=1}^m$ are subgoals of P_1 and $\{p_i\}_{i=1}^m$ are some premises of P_1 that are incident with subgoals. Sets V, E, E' may equal $\emptyset$. Two types of inference steps are distinguished:

Through proposition: Matching q_1 with a proposition $s \in KB$. In this case we switch q_1 with a new premise $q_1' = T(q_1)s$: $V_2^P = V \cup \{q_i'\}_{i=1}^m$, $E_2^P = E \cup \{(q_i', p_i)\}_{i=1}^m$.

Through rule: Matching q_1 with head of rule $\mathfrak{r} \in KB$, where $\mathfrak{r} = r_1, \ldots, r_n \rightarrow c$. In this case we switch q_1 with a new premise $q_1' = T(q_1)c'$ and add rule body: $V_2^P = V \cup \{q_i'\}_{i=1}^m \cup \{r_i'\}_{i=1}^n$, $E_2^P = E \cup \{(q_i', p_i)\}_{i=1}^m \cup \{(r_i', q_1')\}_{i=1}^n$.

Propositions $c', r_1', \ldots, r_n', q_2', \ldots, q_m'$ denote accordingly: elements of rule $\mathfrak{r}$ and subgoals of P_1 with applied substitution of variables. In case of the inference step through rule $\mathfrak{r}$ edges from the set $\{(r_i', q_1')\}_{i=1}^n$ are labeled with $\mathfrak{r}$. Proof P_1 is called a *predecessor* of P_2, which in turn is called a *successor* of P_1.

Example 2

Let $q = P(x)$ and the knowledge base is

$$KB = \{P(A) : \mathit{accepted}, R(B) : \mathit{accepted}\} \cup \{[\diamond Q(y,z), \bullet R(z)] \xrightarrow{pro} P(z)\}$$

The incomplete proof $P_0 = (\{\diamond P(x)\}, \emptyset)$ has two neighbouring proofs (successors of P_0), P_1 and P_2, which in turn has successor P_3:

$$P_1 = (\{\diamond P(A)\}, \emptyset),$$
$$P_2 = (\{\diamond P(x), \bullet R(x), \diamond Q(y,x)\}, \{(\diamond Q(y,x), \diamond P(x)), (\bullet R(x), \diamond P(x))\}),$$
$$P_3 = (\{\diamond P(x), \bullet R(B), \diamond Q(y,B)\}, \{(\diamond Q(y,B), \diamond P(x)), (\bullet R(B), \diamond P(x))\}).$$

P_1 is a complete proof, P_2 has two subgoals: $Q(y,x)$ i $R(x)$, P_3 has one subgoal: $Q(y,B)$ (which is read: $\exists_y Q(y,B)$). The proof space can be interpreted as an (undirected in this case) graph:

$$\mathcal{P}(q) = (\{P_0, P_1, P_2, P_3\}, \{\{P_0, P_1\}, \{P_0, P_2\}, \{P_2, P_3\}\}).$$

The default starting point for search in the proof space $\mathcal{P}(q)$ equals $P_0 = (\{\diamond q\}, \emptyset)$. As shown in the example 2, proof space $\mathcal{P}(q)$ can be represented as a graph, whose vertices are proofs and edges represent the neighbouring relation. The edges can be given a direction, e.g. form the predecessor to the successor.

3.2.2. Search task

As it was mentioned at the beginning of section 3, the goal of search in the RCAF's proof space – unlike in the standard case – is not to find a single complete proof of proposition q, but rather to find a set of complete proofs $\mathcal{P}^t = \{P_1^t, \ldots, P_n^t\}$ that together satisfy proof standard $ps(q)$. Moreover, $\mathcal{P}^t$ should be maximal in the sense, that proof space cannot contain any other complete proof P^t such that $\mathcal{P}^t \cup \{P^t\}$ would no longer satisfy $ps(q)$.

Finding the set $\mathcal{P}^t$ involves repeated searching for its elements in $\mathcal{P}(q)$. The found proofs must not require contradictory substitutions, so after finding the complete proof P_1^t, the substitution θ^t of variables used in the proof must be stored. Next, the search should continue from the beginning with θ^t treated as a constraint: substitutions used in the explored proofs *must not* be contradictory with θ^t. In case of finding the next complete proof P_2^t, substitution θ_2^t should be composed[2] with the previous one: $\theta^t := \theta^t \cup \theta_2^t$, an so on with subsequent proofs.

The search task ends if one of two possible situations occurs:

1. the maximal set $\mathcal{P}^t$ satisfying $ps(q)$ was found (and so q is acceptable),
2. the set $\mathcal{P}^t$ is not a subset of $\mathcal{P}(q)$.

If the constructed set $\mathcal{P}^t$ cannot satisfy $ps(q)$ (e.g. just found proof P_n^t uses rule $A \xrightarrow{con} q$, when $ps(q) = DV$), then the following steps should be taken: (a) the constructed substitution θ^t should be stored in a different record: $\theta^f := \theta^t$; (b) search results should be cleared: $\mathcal{P}^t := \emptyset$ and $\theta^t := \emptyset$; (c) the search should continue from the beginning with θ^f treated as a constraint: substitutions used in the explored proofs *must* be contradictory with θ^f. If the situation will repeat, that is, subsequent "false paths" will be found, then every subsequent "forbidden substitution" should be stored: $\Theta^f = \{\theta^f, \theta_1^f, \ldots\}$. Substitution used in the explored proofs must be contradictory with each element of Θ^f.

Finally, if a given subgoal q' is already acceptable in $\mathcal{P}^t$, then it can be memorized (e.g. added to set Q) and omitted in further search. The Q set should be cleared everytime $\mathcal{P}^t$ is cleared.

To summarize, given this representation, the inference task in RCAF reduces to the task of repeated search with two types of constraints:

1. non-contradiction with hitherto found proofs,
2. contradiction with proofs leading to refutation of q.

[2] See [9, p. 288] for details on substitution composition.

4. The RPA* search algorithm

Space $\mathcal{P}(q)$ has a natural starting point for search in $P_0 = (\{\diamond q\}, \emptyset)$ and it contains some information that allows guessing in which direction the complete proof should be looked for (e.g. number of subgoals, proof standards of the subgoals, etc.). Therefore a natural candidate for a search algorithm is A^* (see e.g. [7]).

In its basic version, A^* chooses among possible points of exploration the point n, such that $f(n)$ is minimum, and expands it. For every successor n' of n the value $f(n')$ is calculated: $f(n') = g(n') + h(n')$. Value $g(n') = g(n) + c(n, n')$ is a cost of reaching point n' and the value $h(n')$ is an estimated cost of reaching the goal of the search from this point.

While reasoning in RCAF, the function f must carry out two tasks:

Task 1. Analogously to the classical case, it should estimate the cost of reaching a complete proof.

Task 2. The function should direct the search in such a way, that the algorithm could verify as quickly as possible if the constructed set $\mathcal{P}^t$ can satisfy $ps(q)$.

Accomplishing task 1 allows to find a complete proof in a single search fast, while accomplishing task 2 minimizes the total number of searches needed to construct $\mathcal{P}^t$. This task can be carried out by first checking those proofs, which can guarantee satisfaction or non-satisfaction of the proof standard.

As task 2 exceeds the classical application of A^* algorithm, function $f : \mathcal{P}(q) \to \Re$ will be responsible only for task 1. Let P be a proof, then:

$$f(P) = |V(P)| + \delta(P), \tag{1}$$

where $\delta(P)$ (the h function) is the number of subgoals in P. The heuristic assumption is made, that the estimated future cost is equal to the number of subgoals of the proof. So defined function h is not *admissible* (see [7, p. 77]) because, in the optimistic case, substitution in one inference step can prove even all subgoals of the proof being extended.

For addressing task 2, a different function, $p : \mathcal{P}(q) \to \Re$, is defined. It is used to prioritize points of space exploration. A^* algorithm is modified as to first choose points with the highest value of p, and among them, the one which has lowest value of f.

Definition 14 (PA^* algorithm)

Let N denote the set of points of search space exploration. PA^* (*prioritized* A^*) *algorithm* is a modification of A^* where the choice of the next

point of exploration is limited to set $\{n \in N : \forall_{m \in N} p(n) \geq p(m)\}$, where $p : \mathcal{P}(q) \rightarrow \Re$ is a function that assigns priorities to points of search space exploration.

To construct the p function it is necessary to look more closely at the differences between different successors of a given proof. These differences form a hierarchy:

1. successors can use different subgoals of their predecessors;
2. for the same subgoal q', successors can differ in the type of inference step (through proposition or through rule),
3. for the same type of inference step, successors can use different elements of knowledge base (different rules or different propositions).

Starting from the top level of this hierarchy, for realization of task 2 the difference in used subgoal is irrelevant, whereas the type of inference clearly is: it is generally faster to prove subgoal by matching it with a proposition in knowledge base. If so, the p function should prefer inference through proposition over inference through rule. On the bottom level: (a) among different propositions, the algorithm should prefer those which end proof of the subgoal (those which have value *acceptable* or *rejected*, for assumptions also *stated*); (b) among different rules four levels of priority can be distinguished:

- rules whose defensibility is a *sufficient condition* for not holding of premise q' (level 3);
- rules whose defensibility is a *sufficient condition* for holding of q' (level 2);
- rules whose defensibility is a *necessary condition* for holding or not holding of premise q' (level 1);
- other rules (level 0).

Rules that definitively prevent holding of the premise have the highest priority, because defensibility of such rule for any subgoal means automatic failure of the chosen exploration path. Depending only on the proof standard, the assignment of priorities to rules is as follows:

SE: *pro* rules: level 2, *con* rules: level 0;

BA: maximal w.r.t. "$>$" rules *con*: level 3, maximal w.r.t. "$>$" rules *pro*: level 1, other rules: level 0;

DV: *con* rules: level 3, *pro* rules: level 1.

For complementary proof standards the above mentioned types of rules should be switched. For $T(q') = \circ$, the above assignments of level 2 and 3 to proof standards should be switched, because an exception holds if its proof standard is *not* satisfied. Of course, it is possible to differentiate rules priorities with other criteria (e.g. number of premises), it is also possible to construct a more refined rule hierarchy for the BA proof standard; these can be regarded as optimization steps, that are not crucial for the concept of PA^* algorithm.

After defining levels of priority, it is possible to formally define the p function. First, $p(P_0) = 0$, where $P_0 = (\{\diamond q\}, \emptyset)$ is the starting point of search. Let P' be a proof, which was constructed from P (is its successor) by matching subgoal q_P with (i) proposition s, (ii) head of rule $\mathfrak{r}$. Function p is defined as follows.

$$p(P') = \begin{cases} p(P) + 4 & \text{in case (i)}, \\ p(P) + prir(q_P, \mathfrak{r}) & \text{in case (ii)}. \end{cases} \tag{2}$$

Help function $prir : \mathcal{L} \times \mathcal{R} \to \Re$ equals ($\mathcal{R}^{q_P}_{max}$ denotes the set of maximal (w.r.t ">") rules supporting q_P):

- If $ps(q_P) = SE$ then

$$prir(q_P, \mathfrak{r}) = \begin{cases} 3 & \text{if } T(q_P) = \circ \wedge T(\mathfrak{r}) = pro, \\ 2 & \text{if } T(q_P) \neq \circ \wedge T(\mathfrak{r}) = pro, \\ 0 & \text{if } T(\mathfrak{r}) = con. \end{cases}$$

- If $ps(q_P) = BA$ then

$$prir(q_P, \mathfrak{r}) = \begin{cases} 3 & \text{if } T(q_P) \neq \circ \wedge T(\mathfrak{r}) = con \wedge \mathfrak{r} \in \mathcal{R}^{q_P}_{max}, \\ 2 & \text{if } T(q_P) = \circ \wedge T(\mathfrak{r}) = con \wedge \mathfrak{r} \in \mathcal{R}^{q_P}_{max}, \\ 1 & \text{if } T(\mathfrak{r}) = pro \wedge \mathfrak{r} \in \mathcal{R}^{q_P}_{max}, \\ 0 & \text{otherwise.} \end{cases}$$

- If $ps(q_P) = DV$ then

$$prir(q_P, \mathfrak{r}) = \begin{cases} 3 & \text{if } T(q_P) \neq \circ \wedge T(\mathfrak{r}) = con, \\ 2 & \text{if } T(q_P) = \circ \wedge T(\mathfrak{r}) = con, \\ 1 & \text{if } T(\mathfrak{r}) = pro. \end{cases}$$

The proposed algorithm for reasoning in RCAF is RPA^*:

Definition 15 (RPA^* algorithm)

RPA^* (*repeated* PA^*) *algorithm* is a repeated execution of PA^* algorithm with function p given by equation 2 and function f given by equation 1. The set of legal points of space exploration is modified according to constraints given in section 3.2.2. Repeated execution of PA^* algorithm ends when one of the stop conditions given in section 3.2.2 is fulfilled. RPA^* concludes that the found $\mathcal{P}^t$ set is maximal if $\mathcal{P}(q)$ contains no points of search space exploration that are not contradictory with θ^t.

All elements of $\mathcal{P}^t$ are directed, acyclic argument graphs with root equal to q. A single argumentation graph, which should be the result of argumentation space search can be obtained be summing together elements of $\mathcal{P}^t$.

5. Implementation

The above algorithm was implemented in Java language, using JUNG framework (`http://jung.sourceforge.net`) for graph manipulation and visualization. It is avaliable for download at: `http://www.ii.pw.edu.pl/~plozinsk/materialy/rcaf-demo.jar`. The implementation uses a compression method for search space storage which is based on the observation that neighbouring proofs differ only with one inference step, so they can be stored in a differential manner. The program is equipped with a GUI which supports (a) editing the knowledge base (in a textual form); (b) loading text with the knowledge base to the reasoner; (c) asking queries with specified proof standard. A GUI shows visualization of the elements of $\mathcal{P}^t$ mapped into one directed graph. The colours of propositions in the graph follow the street lights metaphor: green represents status *accepted*, yellow is for *questioned*, red for *rejected* and additionally grey for *stated*. Green is also used to mark *pro* rules, while *con* rules are marked with read.

The format for textual editing of knowladge base is as follows. Predicates and constants must begin with uppercase, variables must begin with lowercase. Every proposition is entered in separate line and has two optional parameters: `s` used to assign the proposition's status and `p` to assign the proof standard. Default values are `s=STATED` and `p=DV`. Rules have optional labels that can be used in specifying the partial ordering, e.g. $r1 < r2$. Rules of type *pro* are denoted with "`->`", *con* with "`-<`". Assumptions are marked with "`+`", and exceptions with "`-`". Negation is marked with "`!`". An example knowledge base will be given in the next section.

6. Short comparison

The results of argumentation space search with *Carneades*'s abductive construction of arguments and $RPA*$ algorithm will be compared using rather abstract, but concise example. For the purpose of the comparison, the latest implementation of *Carneades* is used. The system is implemented in *Clojure* functional programming language, and avaliable for download form `http://carneades.berlios.de/`. Let us consider a knowledge base defined in *Carneades* as simply as possible:

```
(def abstract-rb
    (rulebase
      (rule r1
            (if (B ?x) (A ?x)))
      (rule r2
            (if (and (D ?x) (E ?x ?y)) (B ?x)))
      (rule r3
            (if (I ?x) (not (B ?x))))
      (rule r4
            (if (and (D ?x) (F ?x ?y)) (I ?x)))
      (rule r5
            (if (C ?x) (not (A ?x))))
      (rule r6
            (if (and (G ?x) (E ?x ?y)) (C ?x)))
      ))
```

Using this knowledge base for the only source of arguments, we ask if statement $A(X)$ is acceptable with proof standard DV. First, we accept the following facts: $D(X)$, $E(X,Y)$ and $F(X,Y)$. Then we ask the question. After the search for arguments we set desired proof standards DV for $A(X)$ and SE for $B(X)$ (this cannot be done in advance, as before the search those statements don't exist in any argumentation graph). Response of *Carneades*'s abductive construction of arguments, shown in figure 2, presents the full[3] information regarding this issue. The statement happens to be acceptable, because there is no defensible con argument and there is a pro argument from $B(X)$, which in turn is acceptable, because its proof standard (SE) is satisfied even in the presense of an argument con $B(X)$.

[3] Given avaliable search resources.

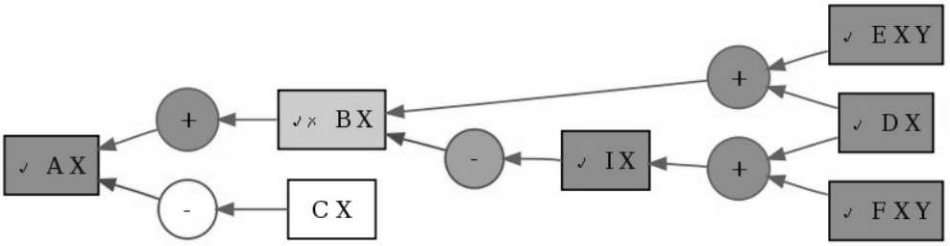

Figure 2. ***Carneades*'s Argumentation graph for $A(X)$ with proof standard DV**

Now, let us consider the same example using the implementation of $RPA*$. The above knowledge base, written down in the textual format described in section 5, looks as follows:

```
r1: B(x) -> A(x)
r2: D(x), E(x, y) -> B(x)
r3: I(x) -< B(x)
r4: D(x), F(x, y) -> I(x)
r5: C(x) -< A(x)
r6: G(x), E(x, y) -> C(x)
B(X) p=SE
D(X) s=ACCEPTED
E(X, Y) s=ACCEPTED
F(X, Y) s=ACCEPTED
```

The information about proof standards is given in advance. After asking the query $A(X)$ with proof standard DV the application returns result shown in figure 3, which is the minimal information required to determine acceptability of $A(X)$.

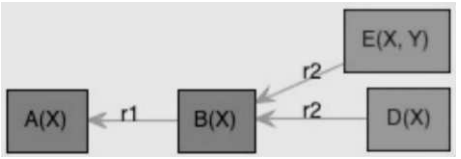

Figure 3. Result for query $A(X)$ with proof standard DV returned by $RPA*$

When we deal with relatively small cases, it is best for argumentation analysis to have as much information as possible avaliable at once. However, in argumentation cases more close to real-life situations, both the size of resulting graph and the time spent on it's computation become significant enough to consider incremental analysis of the case using partial information provided e.g. by the algorithm presented here.

Conclusions and future work

This article presents an algorithm for incremental argumentation analysis in *Carneades*. It is proposed, that such analysis can be conducted using a search algorithm for finding a minimal argumentation graph that allows for determining acceptability of the given goal. Because steps in the search space are considered potentially expensive (e.g. involving OWL reasoner), it is important to complete the search in the minimal number of steps.

The solution for incremental argumentation analysis consists of three steps: (a) proposing a defeasible logic (called RCAF) based on *Carneades*; (b) proposing a general approach for construction of RCAF reasoners by formulating the problem of inference in RCAF as a generic search task; (c) designing and implementing a sample algorithm for reasoning in RCAF, which is $RPA*$ It uses heuristics and search constraints for choosing the exploration paths of argumentation search space.

It is claimed that this algorithm can be helpfull in argumentation analysis in cases where full argumentation graphs are too large to grasp for the application user and in situations where it takes a long time to generate them.

As of the end of 2010, the implementation of the newest version of *Carneades* (as presented in [8] and [2]) is avaliable at the project's website. Future work will mainly focus on integrating the $RPA*$ algorithm with this implementation which will allow its more extended evaluation using resources avaliable in the main *Carneades* project.

References

[1] Jarosław Arabas and Paweł Cichosz, Search-Based View of Artificial Intelligence. In *Artificial Intelligence Studies*, volume 26, pages 13–29, Siedlce 2006.

[2] S. Ballnat and T. F. Gordon. Goal selection in argumentation processes – a formal model of abduction in argument evaluation structures. In *Computational Models of Argument – Proceedings of COMMA 2010*, pages 51–62, IOS Press, 2010.

[3] Thomas Gordon. Hybrid reasoning with argumentation schemes. In *8th Workshop on Computational Models of Natural Argument*, 2008.

[4] Thomas Gordon. Report on prototype decision support system for oss license compatibility issues. Technical report, Fraunhofer FOKUS, Berlin, 2010.

[5] Thomas F. Gordon and Douglas Walton. The Carneades argumentation framework – using presumptions and exceptions to model critical questions. In Paul E. Dunne and Trevor J. M. Bench-Capon, editors, *Computational Models of Argument – Proceedings of COMMA 2006*, volume 144, of *Frontiers in Artificial Intelligence and Applications*, pages 195–207, Amsterdam, 2006, COMMA, IOS Press.

[6] Thomas G. Gordon and Henry Prakken, and Douglas Walton. The Carneades model of argument and burden of proof. *Artificial Intelligence*, 171(10–15): 875–896, 2007.

[7] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[8] Iyad Rahwan and Guillermo R. Simari. *Argumentation in Artificial Intelligence*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[9] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

Paweł Łoziński
Institute of Computer Science,
Warsaw University of Technology,
ul. Nowowiejska 15/19, 00-665 Warsaw, Poland;
plozinsk@ii.pw.edu.pl