An algorithm for generating arguments in classical predicate logic

Vasiliki Efstathiou, Anthony Hunter

Department of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
{v.efstathiou, a.hunter}@cs.ucl.ac.uk

Abstract. There are a number of frameworks for modelling argumentation in logic. They incorporate a formal representation of individual arguments and techniques for comparing conflicting arguments. A common assumption for logic-based argumentation is that an argument is a pair $\langle \Phi, \alpha \rangle$ where Φ is a minimal subset of the knowledgebase such that Φ is consistent and Φ entails the claim α. Different logics provide different definitions for consistency and entailment and hence give us different options for argumentation. An appealing option is classical first-order logic which can express much more complex knowledge than possible with defeasible or classical propositional logics. However the computational viability of using classical first-order logic is an issue. Here we address this issue by using the notion of a connection graph and resolution with unification. We provide a theoretical framework and algorithm for this, together with some theoretical results.

1 Introduction

Argumentation is a vital aspect of intelligent behaviour by humans used to deal with conflicting information. There are a number of proposals for logic-based formalisations of argumentation (for reviews see [6, 14, 5]). These proposals allow for the representation of arguments and for counterargument relationships between arguments. In a number of key examples of argumentation systems, an argument is a pair where the first item in the pair is a minimal consistent set of formulae that proves the second item which is a formula (see for example [2, 9, 3, 1, 10, 4, 7, 13]). Algorithms have been developed for finding arguments from a knowledgebase using defeasible logic. However, there is a lack of viable algorithms for finding arguments for first-order classical logic.

We propose an approach to this problem by extending an existing proposal for propositional logic [8] based on the connection graph proof procedure [11, 12]. This extension is based on the idea of resolution with unification [15]. We use a connection graph structure where nodes are clauses from the knowledgebase and arcs link contradictory literals and then apply heuristic search strategies to follow the arcs of the graph and create partial instances of the visited clauses based on the unification of atoms that appear at either end of an arc. The

aim of the search is to retrieve an unsatisfiable grounded version of a subset of the knowledgebase and use the refutation completeness of the resolution rule and Herbrand's theorem to obtain a proof for the claim. The minimality and consistency of this proof is achieved according to some restrictions applied during the search.

2 Argumentation for a language of quantified clauses

In this section we review an existing proposal for argumentation based on classical logic [3] and in particular an extension of this [4] dealing with first-order logic. For a first-order language $\mathcal{F}$, the set of formulae that can be formed is given by the usual inductive definitions of classical logic.

In this paper we use a restricted function-free first-order language of quantified clauses $\mathcal{F}$ consisting of n-ary predicates ($n \geq 1$) where we allow both existential and universal quantifiers and we consider arguments whose claims consist of one disjunct (i.e. unit clauses). This language is composed of the set of n-ary ($n \geq 1$) predicates $\mathcal{P}$, a set of constant symbols $\mathcal{C}$, a set of variables $\mathcal{V}$, the quantifiers $\forall$ and $\exists$, the connectives $\neg$ and $\vee$ and the bracket symbols ( ). The clauses of $\mathcal{F}$ are in prenex normal form, consisting of a quantification string followed by a disjunction of literals. Literals are trivially defined as positive or negative atoms where an atom is an n-ary predicate. The quantification part consists of a sequence of quantified variables that appear as parameters of the predicates of the clause. These need not follow some ordering, that is any type of quantifier (existential or universal) can preceed any type of quantifier. Deduction in classical logic is denoted by the symbol $\vdash$.

Example 1. If $\{a, b, c, d, e\} \subset \mathcal{C}$ and $\{x, y, z, w\} \subset \mathcal{V}$, then each of the elements of Φ is a clause in $\mathcal{F}$ where $\Phi = \{\forall x \exists z(P(x) \vee \neg Q(z, a)), \exists x \exists z(P(x) \vee \neg Q(z, a), \forall w \exists x \exists z(P(x) \vee \neg Q(z, a) \vee P(b, w, x, z)), \forall w(\neg Q(w, b, a)), \neg Q(e, b, a) \vee R(d), \neg P(a, d)\}$. In addition $\forall w(\neg Q(w, b, a))$ is a unit clause, $\neg Q(e, b, a) \vee R(d)$ is a ground clause and $\neg P(a, d)$ is a ground unit clause.

Given a set Δ of first-order clauses, we can define an argument as follows.

Definition 1. *An* **argument** *is a pair $\langle \Phi, \psi \rangle$ such that (1) $\Phi \subseteq \Delta$, (2) $\Phi \vdash \psi$, (3) $\Phi \nvdash \bot$ and (4) there is no $\Phi' \subset \Phi$ such that $\Phi' \vdash \psi$.*

Example 2. Let $\Delta = \{\forall x(\neg P(x) \vee Q(x)), P(a), \forall x \forall y(P(x, y) \vee \neg P(x)), R(a, b), \exists x(R(x, b)), \exists x(\neg S(x, b))\}$. Some arguments are:

$\langle \{\forall x(\neg P(x) \vee Q(x)), P(a)\}, Q(a) \rangle$ $\quad$ $\langle \{R(a, b)\}, \exists x(R(x, b)) \rangle$

$\langle \{\forall x \forall y(P(x, y) \vee \neg P(x)), P(a)\}, \forall y(P(a, y)) \rangle$ $\quad$ $\langle \{P(a)\}, \exists y(P(y)) \rangle$

3 Relations on clauses

In this section we define some relations on $\mathcal{F}$ that we use throughout this paper. We use the terms 'term', 'variable' and 'constant' in the usual way. We define

functions $\mathsf{Variables}(X)$ and $\mathsf{Constants}(X)$ to return the set of all the variables and constants respectively that appear in a literal or a clause or a set X.

Definition 2. *For a language $\mathcal{L}$, with variables $\mathcal{V}$ and constant symbols $\mathcal{C}$, the set of bindings $\mathcal{B}$ is $\{x/t \mid x \in \mathcal{V}$ and $t \in \mathcal{V} \cup \mathcal{C}\}$.*

Definition 3. *For a clause ϕ and a set of bindings $B \subseteq \mathcal{B}$, $\mathsf{Assign}(\phi, B)$ returns clause ϕ with the values of B assigned to the terms of ϕ. So, for each x/t, if x is a variable in ϕ, then x is replaced by t and the quantifier of x is removed.*

Example 3. Let $\phi = \exists x \forall y \exists z \forall w(P(c,x) \vee Q(x,y,z) \vee R(y,c,w))$. Some assignments for ϕ with the corresponding values assigned to ϕ are:

$B_1 = \{x/a, z/b\}$, $\mathsf{Assign}(\phi, B_1) = \forall y \forall w(P(c,a) \vee Q(a,y,b) \vee R(y,c,w))$
$B_2 = \{x/a, y/b, z/b\}$, $\mathsf{Assign}(\phi, B_2) = \forall w(P(c,a) \vee Q(a,b,b) \vee R(b,c,w))$
$B_3 = \{x/a, z/b, w/b\}$, $\mathsf{Assign}(\phi, B_3) = \forall y(P(c,a) \vee Q(a,y,b) \vee R(y,c,b))$
$B_4 = \{x/a, y/a, z/b, w/b\}$, $\mathsf{Assign}(\phi, B_4) = P(c,a) \vee Q(a,a,b) \vee R(a,c,b)$
$B_5 = \{w/z\}$, $\mathsf{Assign}(\phi, B_5) = \exists x \forall y \exists z(P(c,x) \vee Q(x,y,z) \vee R(y,c,z))$

Function $\mathsf{Assign}(\phi, B)$ gives a specific instance of ϕ, indicated by the bindings in B. We define next the function that returns all the possible instances for a clause ϕ and the function that returns all the possible instances for all the elements of a set of clauses Ψ.

Definition 4. *For a clause ϕ, $\mathsf{Assignments}(\phi)$ returns the set of all the possible instances of ϕ: $\mathsf{Assignments}(\phi) = \{\mathsf{Assign}(\phi, B_i) \mid B_i \in \wp(\mathcal{B})\}$. For a set of clauses Ψ, $\mathsf{SetAssignments}(\Psi) = \bigcup_{\phi \in \Psi} \{\mathsf{Assignments}(\phi)\}$.*

We use the assignment functions to create partial instances of the clauses from the knowledgebase during the search for arguments. As no restrictions apply to the order of the quantifiers in the quantification of a clause from $\mathcal{F}$, the order of interchanging universal and existential quantifiers in a clause ϕ is taken into account when a partial instance of ϕ is created. For this, we define function $\mathsf{Prohibited}(\phi)$ to return the sets of bindings that are not allowed for ϕ.

Definition 5. *Let ϕ be a clause. Then, $\mathsf{Prohibited}(\phi) \subseteq \wp(\mathcal{B})$ returns the set of sets of bindings such that for each $B \in \mathsf{Prohibited}(\phi)$ there is at least one $y_i/t_i \in B$ such that y_i is a universally quantified variable which is in the scope of an existentially quantified variable x_i for which either $x_i = t_i$ or $x_i/t_i \in B$.*

Example 4. For the sets of bindings of example 3, $B_3, B_4, B_5 \in \mathsf{Prohibited}(\phi)$ and $B_1, B_2 \notin \mathsf{Prohibited}(\phi)$.

We now define a function that gives a partial instance of a clause ϕ where each of the existentially quantified variables is replaced by a distinct arbitrary constant from $\mathcal{C} \setminus \mathsf{Constants}(\phi)$. This is a form of Skolemization.

Definition 6. *For a clause* ϕ, $\mathsf{ExistentialGrounding}(\phi, B) = \mathsf{Assign}(\phi, B)$ *where* $B \in \wp(\mathcal{B})$ *is such that:* (1) $x_i/t_i \in B$ *iff* $x_i \in \mathsf{Variables}(\phi)$ *and* x_i *is existentially quantified* (2) $t_i \in \mathcal{C} \setminus \mathsf{Constants}(\phi)$ *and* (3) *for all* $x_j/t_j \in B$, *if* $x_j \neq x_i$ *then* $t_j \neq t_i$. *If* $\phi' = \mathsf{ExistentialGrounding}(\phi, B)$ *for some* $\phi \in \mathcal{F}$ *and* $B \in \wp(\mathcal{B})$, *we say that* ϕ' *is an* **existential instance** *of* ϕ.

Example 5. For $\phi = \exists x \forall y \exists z \forall w (P(c, x) \vee Q(x, y, z) \vee R(y, c, w))$, $\mathsf{Constants}(\phi) = \{c\}$ and so each of the elements of the set of constants $I = \{a, b\} \subset \mathcal{C} \setminus \mathsf{Constants}(\phi)$ can be used for the substitution of each of the existentially bound variables x, z. For $B = \{x/a, z/b\}$, $\mathsf{ExistentialGrounding}(\phi, B) = \forall y \forall w (P(c, a) \vee Q(a, y, b) \vee R(y, c, w))$.

Definition 7. *For a clause* $\phi = \mathsf{Q}_1 x_1, \ldots, \mathsf{Q}_m x_m (p_1 \vee \ldots \vee p_k)$, $\mathsf{Disjuncts}(\phi)$ *returns the set of disjuncts of* ϕ. $\mathsf{Disjuncts}(\phi) = \{p_1 \vee \ldots \vee p_k\}$. *For each* $p_i \in \mathsf{Disjuncts}(\phi)$, $\mathsf{Unit}(\phi, p_i)$ *returns the unit clause that consists of* p_i *as its unique disjunct and the part of the quantification* $\mathsf{Q}_1 x_1, \ldots, \mathsf{Q}_m x_m$ *of* ϕ *that involves the variables that occur in* p_i *as its quantification:* $\mathsf{Unit}(\phi, p_i) = \mathsf{Q}_j x_j \ldots \mathsf{Q}_l x_l (p_i)$ *where* $\{\mathsf{Q}_j x_j, \ldots, \mathsf{Q}_l x_l\} \subseteq \{\mathsf{Q}_1 x_1, \ldots, \mathsf{Q}_m x_m\}$ *and* $\{x_j, \ldots, x_l\} = \mathsf{Variables}(p_i)$.

Example 6. Let $\phi = \forall x \forall y \exists z (P(x) \vee Q(a) \vee \neg R(x, y, z, b) \vee S(a, b, c))$ and let $p = P(x), q = Q(a), r = \neg R(x, y, z, b)$ and $s = S(a, b, c)$. Then, $\mathsf{Disjuncts}(\phi) = \{p, q, r, s\}$ and

$$\mathsf{Unit}(\phi, p) = \forall x (P(x)) \qquad \mathsf{Unit}(\phi, r) = \forall x \forall y \exists z (\neg R(x, y, z, b))$$
$$\mathsf{Unit}(\phi, q) = Q(a) \qquad \mathsf{Unit}(\phi, s) = S(a, b, c)$$

Definition 8. *For a clause* ϕ, $\mathsf{Units}(\phi) = \{\mathsf{Unit}(\phi, p_i) \mid p_i \in \mathsf{Disjuncts}(\phi)\}$.

Example 7. Continuing example 6, for $\phi = \forall x \forall y \exists z (P(x) \vee Q(a) \vee \neg R(x, y, z, b) \vee S(a, b, c))$, $\mathsf{Units}(\phi) = \{\forall x (P(x)), Q(a), \forall x \forall y \exists z (\neg R(x, y, z, b)), S(a, b, c)\}$

We now define some binary relations that express contradiction between clauses. For this we define contradiction between unit clauses ϕ and ψ as follows: ϕ and ψ **contradict** each other iff $\phi \vdash \neg\psi$. Then we say that ψ is a complement of ϕ and we write $\phi = \overline{\psi}$. Using the contradiction relation between the units of a pair of clauses, we define the following relations of attack.

Definition 9. *Let* ϕ, ψ *be clauses. Then,* $\mathsf{Preattacks}(\phi, \psi) = \{a_i \in \mathsf{Units}(\phi) \mid \exists a_j \in \mathsf{Units}(\psi)$ *s.t.* $a_i = \overline{a}_j\}$.

Example 8. According to definition 9, the following relations hold.
8.1) $\mathsf{Preattacks}(\forall x(\neg N(x) \vee R(x)), N(a) \vee \neg R(b)) = \{\forall x(\neg N(x)), \forall x(R(x))\}$
8.2) $\mathsf{Preattacks}(\forall x(\neg N(x) \vee R(x)), N(a) \vee \neg R(a)) = \{\forall x(\neg N(x)), \forall x(R(x))\}$
8.3) $\mathsf{Preattacks}(P(a) \vee \neg Q(b), \neg P(a) \vee Q(b)) = \{P(a), \neg Q(b)\}$
8.4) $\mathsf{Preattacks}(\forall x(P(x) \vee \neg Q(a, x)), \exists x(\neg P(a) \vee Q(x, b))) = \{\forall x(P(x))\}$
8.5) $\mathsf{Preattacks}(\exists x(\neg P(a) \vee Q(x, b)), \forall x(P(x) \vee \neg Q(a, x)) = \{\neg P(a)\}$

We now define a special case of the preattacks relation which we use to define arcs for trees in the next section.

Definition 10. *For clauses ϕ, ψ, if* $|\mathsf{Preattacks}(\phi, \psi)| = 1 = |\mathsf{Preattacks}(\psi, \phi)|$ *then* $\mathsf{Attacks}(\phi, \psi) = \alpha$*, where* $\alpha \in \mathsf{Preattacks}(\phi, \psi)$*, otherwise* $\mathsf{Attacks}(\phi, \psi) = \mathsf{Attacks}(\psi, \phi) = null$*.*

Example 9. For examples 8.1, 8.2 and 8.3, $\mathsf{Attacks}(\phi, \psi) = null$. For examples 8.4-8.5, $\mathsf{Attacks}(\phi, \psi) = \mathsf{Preattacks}(\phi, \psi)$.

Although the Attacks relation might be *null* for a pair of clauses ϕ, ψ, it can sometimes hold for instances of ϕ and ψ.

Example 10. In example 8.1, let $\phi = \forall x(\neg N(x) \vee R(x))$ and $\psi = N(a) \vee \neg R(b)$. Then $|\mathsf{Preattacks}(\phi, \psi)| > 1$ and so, $\mathsf{Attacks}(\phi, \psi) = null$. There are instances ϕ' of ϕ though for which $\mathsf{Attacks}(\phi', \psi) \neq null$. Let $B_1 = \{x/a\}$, and $B_2 = \{x/b\}$. Then for $\phi_1 = \mathsf{Assign}(\phi, B_1) = \neg N(a) \vee R(a)$ and $\phi_2 = \mathsf{Assign}(\phi, B_2) = \neg N(b) \vee R(b)$, $\mathsf{Attacks}(\phi_1, \psi) = \neg N(a)$ and $\mathsf{Attacks}(\phi_2, \psi) = R(b)$. For all other instances ϕ' of ϕ $\mathsf{Attacks}(\phi', \phi) = null$.

Example 11. In example 8.2, let $\gamma = \forall x(\neg N(x) \vee R(x))$ and $\delta = N(a) \vee \neg R(a)$. Then, for all the instances γ' of γ, $|\mathsf{Preattacks}(\gamma', \delta)| \neq 1$ and so there is no instance γ' of γ for which $\mathsf{Attacks}(\gamma', \delta) \neq null$.

4 Assignment Trees

Using the attack relations defined in section 3, we define in this section the notion of an assignment tree which represents a tentative proof of an argument. The definition is designed for use with the algorithm we introduce in section 5 for searching for arguments.

Definition 11. *Let Δ be a clause knowledgebase and α be a unit clause and let $\Delta' = \Delta \cup \{\neg\alpha\}$. An* **assignment tree** *for Δ and α is tuple (N, A, e, f, g, h) where N is a set of nodes and A is a set of arcs such that (N, A) is a tree and e, f, g, h are functions such that: $e : N \mapsto \Delta'$, $f : N \mapsto \mathsf{SetAssignments}(\Delta')$, $g : N \mapsto \wp(\mathcal{B})$, $h : N \mapsto \mathsf{SetAssignments}(\Delta')$ and*

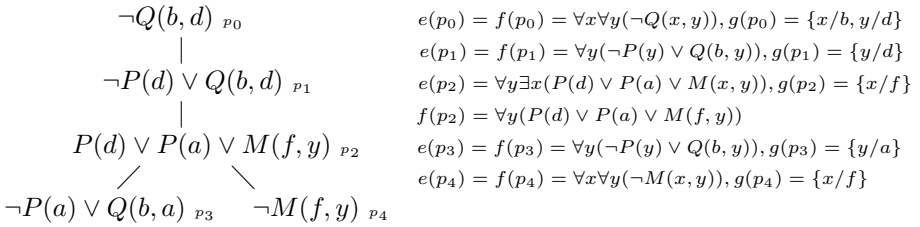
(1) *if p is the root of the tree, then $e(p) = \neg\alpha$*
(2) *f(p) is an existential instance of $e(p)$ s.t. $\mathsf{Constants}(f(p)) \subseteq \mathsf{Constants}(g(p))$*
(3) *for any nodes p, q in the same branch, if $e(p) = e(q)$ then $g(p) \neq g(q)$*
(4) *for all $p \in N, g(p) \cap \mathsf{Prohibited}(e(p)) = \emptyset$*
(5) *for all $p \in N, h(p) = \mathsf{Assign}(f(p), g(p))$*
(6) *for all $p, q \in N$, if p is the parent of q, then $\mathsf{Attacks}(h(q), h(p)) \neq null$*
(7) *for all $p, q \in N$, $(\mathsf{Constants}(f(p)) \setminus \mathsf{Constants}(e(p))) \bigcap \mathsf{Constants}(\Delta') = \emptyset$, & $(\mathsf{Constants}(f(p)) \setminus \mathsf{Constants}(e(p))) \bigcap (\mathsf{Constants}(f(q)) \setminus \mathsf{Constants}(e(q))) = \emptyset$*

Each of the functions e, f, g, h for a node p gives the state of the tentative proof for an argument for α. Function $e(p)$ identifies for p the clause ϕ from $\Delta \cup \{\neg\alpha\}$ and $f(p)$ is an existential instance of $e(p)$. $g(p)$ is a set of bindings that when assigned to $e(p)$ creates the instance $h(p)$ of $e(p)$. Hence, $g(p)$ contains

the set of bindings that create the existential instance $f(p)$ of $e(p)$ together with the bindings that unify atoms of contradictory literals connected with arcs on the tree as condition 6 indicates. Condition 7 ensures that the existential instances used in the proof are created by assigning to the existentially quantified variables of a clause $e(p)$ constants that do not appear anywhere else in $\Delta \cup \{\neg\alpha\}$ or the other instances of the clauses of the tentative proof. Finally, condition 3 ensures that an infinite sequence of identical nodes on a branch will be avoided.

In all the examples that follow, assignment trees are represented by the value $h(p)$ for each node p. Hence, all the variables that appear in a tree representation are universally quantified and so universal quantifiers are omitted for simplicity.

Example 12. Let $\Delta = \{\forall y(\neg P(y) \vee Q(b,y)),\ \forall y \exists x(P(d) \vee P(a) \vee M(x,y)),\ R(c),$ $\forall x \forall y(\neg M(x,y)),\ \exists x \forall y(Q(x,y) \vee R(x,y)),\ Q(a,b) \vee \neg N(a,b),\ \forall x \forall y(L(x,y,a)),$ $\forall x(\neg R(x,x) \vee S(x,y)),\ \neg Q(a,b) \vee N(a,b),\ \forall x \forall y(\neg S(x,y)),\ \neg L(c,d,a),\ \forall x(P(x)),$ $\neg R(c,a)\}$. The following is an assignment tree for Δ and $\alpha = \exists x \exists y(Q(x,y))$

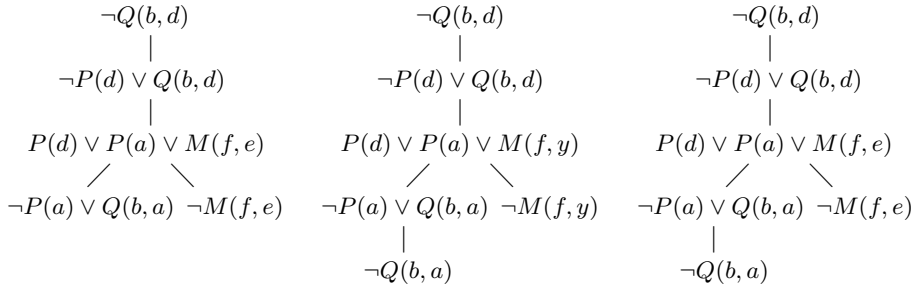


Definition 12. *A* **complete assignment tree** (N, A, e, f, g, h) *is an assignment tree such that for any* $x \in N$ *if* y *a child of* x *then there is a* $\overline{b}_i \in \mathsf{Units}(h(x))$ *such that* $\mathsf{Attacks}(h(y), h(x)) = b_i$ *and for each* $b_j \in \mathsf{Units}(h(y)) \setminus \{b_i\}$
(1) either there is exactly one child z *of* y *s.t.* $\mathsf{Attacks}(h(z), h(y)) = \overline{b}_j$
(2) or there is a node w *in the branch containing* y *s.t.* $b_j = \mathsf{Attacks}(h(y), h(w))$

Definition 13. *A* **grounded assignment tree** (N, A, e, f, g, h) *is an assignment tree such that for any* $x \in N$, $h(x)$ *is a ground clause.*

Example 13. The assignment tree of example 12 is neither complete nor grounded. It is not a complete assignment tree because for $Q(b,a) \in \mathsf{Units}(h(p_3))$ the conditions of definition 12 do not hold. Adding a node p_5 as a child of p_3 with $e(p_5) = f(p_5) = \forall x \forall y(\neg Q(x,y))$, $g(p_5) = \{x/b, y/a\}$ for which $h(p_5) = \neg Q(b,a)$ gives a complete assignment tree. It is not a grounded assignment tree because for nodes p_2 and p_4 $h(p_2) = P(d) \vee P(a) \vee M(f,y)$ and $h(p_4) = \neg M(f,y)$ are non-ground clauses. If we substitute the non-ground term y in $h(p_2)$ and $h(p_4)$ with the same arbitrary constant value ($e \in \mathcal{C}$ for instance), the resulting tree still satisfies the conditions for an assignment tree and it is also a grounded

assignment tree.



For a complete grounded assignment tree we have the following result on the entailment of a claim α for an argument.

Proposition 1. *If (N, A, e, f, g, h) is a complete grounded assignment tree for Δ and α, then $\{e(p) \mid p \in N\} \setminus \{\neg\alpha\} \vdash \alpha$.*

Example 14. For the complete and grounded assignment tree of example 13, $\{e(p) \mid p \in N\} \setminus \{\neg\alpha\} = \{\forall y(\neg P(y) \vee Q(b, y)), \forall y \exists x(P(d) \vee P(a) \vee M(x, y)), \forall x \forall y(\neg Q(x, y)), \forall x \forall y(\neg M(x, y))\} \vdash \exists x \exists y(Q(x, y))$.

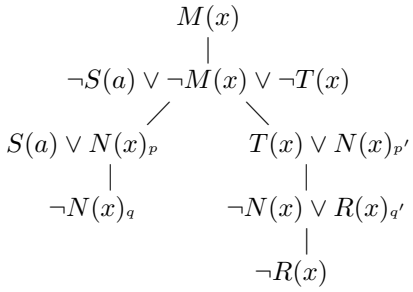
Although all the assignment trees in example 13 correspond to the same subset of clauses $e(p)$ from Δ, it is not always the case that a non-grounded or non-complete assignment tree is sufficient to indicate a proof for α.

The following definitions introduce additional constraints on the definition of a complete assignment tree for Δ and α that give properties related to the minimality and the consistency of the proof for α indicated by the set of nodes in the assignment tree.

Definition 14. (N, A, e, f, g, h) *is* **a minimal assignment tree** *for Δ and α if for any arcs (p, q), (p', q') such that* $\mathsf{Attacks}(h(q), h(p)) = \mathsf{Assign}(\beta, g(q))$ *for some $\beta \in \mathsf{Units}(e(q))$, and* $\mathsf{Attacks}(h(q'), h(p')) = \mathsf{Assign}(\beta', g(q'))$ *for some $\beta' \in \mathsf{Units}(e(q'))$, $\beta \vdash \beta'$ holds iff $e(q) = e(q')$.*

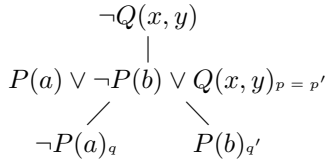
Example 15. The following (N, A, e, f, g, h) is a complete assignment tree for a knowledgebase Δ and $\alpha = \exists x(\neg M(x))$, with $\{e(p) \mid p \in N\} = \{\forall x(M(x)), \forall x(\neg S(a) \vee \neg M(x) \vee \neg T(x)), \forall x(S(a) \vee N(x)), \forall x(T(x) \vee N(x)), \forall x(\neg N(x)), \forall x(\neg N(x) \vee R(x)), \forall x(\neg R(x))\}$. (N, A, e, f, g, h) is not minimal because of $\beta = \forall x(\neg N(x)) \in \mathsf{Units}(e(q))$ and $\beta' = \forall x(\neg N(x)) \in \mathsf{Units}(e(q'))$. If a copy of the subtree rooted at p in (N, A, e, f, g, h) is substituted by the subtree rooted at p', a minimal assignment tree (N', A', e', f', g', h') with $\{e(p) \mid p \in N'\} = \{\forall x(M(x)), \forall x(\neg R(x)), \forall x(\neg S(a) \vee \neg M(x) \vee \neg T(x)), \forall x(S(a) \vee N(x)), \forall x(T(x) \vee N(x)), \forall x(\neg N(x) \vee R(x))\}$ is obtained. Similarly, if a copy of the subtree rooted at p' is substituted by the subtree rooted at p, another minimal assignment tree

$(N'', A'', e'', f'', g'', h'')$ is obtained, with $\{e(p) \mid p \in N''\} = \{\forall x(M(x)), \forall x(\neg S(a) \vee \neg M(x) \vee \neg T(x)), \forall x(S(a) \vee N(x)), \forall x(T(x) \vee N(x)), \forall x(\neg N(x))\}$.



Definition 15. *Let (N, A, e, f, g, h) be a minimal assignment tree for Δ and α. Then, (N, A, e, f, g, h) is* **a consistent assignment tree** *if for any arcs (p, q), (p', q') where $\mathsf{Attacks}(h(q), h(p)) = \mathsf{Assign}(\beta, g(q))$ for some $\beta \in \mathsf{Units}(e(q))$ and $\mathsf{Attacks}(h(q'), h(p')) = \mathsf{Assign}(\beta', g(q'))$ for some $\beta' \in \mathsf{Units}(e(q'))$, $\beta \vdash \overline{\beta}'$ holds iff $e(q) = e(p')$.*

Example 16. The following minimal assignment tree (N, A, e, f, g, h) with $\{e(p) \mid p \in N\} = \{\forall x \forall y(\neg Q(x, y)), \forall x \forall y(P(a) \vee \neg P(b) \vee Q(x, y)), \forall x(\neg P(x)), \forall x(P(x))\}$ is not consistent because for q, q', $\beta = \forall x(\neg P(x)) \in \mathsf{Units}(e(q))$, $\beta' = \forall x(P(x)) \in \mathsf{Units}(e(q'))$, $\beta \vdash \overline{\beta}'$ but $e(q) \neq e(p')$.



An assignment tree (N', A', e', f', g', h') with the same tree structure as above can be formed from the set of clauses $\{e(p) \mid p \in N'\} = \{\forall x \forall y(\neg Q(x, y)), \forall x \forall y(P(a) \vee \neg P(b) \vee Q(x, y)), \neg P(a), P(b)\}$. In this case, (N', A', e', f', g', h') satisfies the conditions of definition 15.

Using the definitions for minimality and consistency for an assignment tree we have the following result.

Proposition 2. *Let (N, A, e, f, g, h) be a complete, consistent grounded assignment tree. Then $\langle \Phi, \alpha \rangle$ with $\Phi = \{e(p) \mid p \in N\} \setminus \{\neg\alpha\}$ is an argument.*

5 Algorithms

In this section we present an algorithm to search for all the minimal and consistent complete assignment trees for a unit clause α from a given knowledgebase Δ. If a grounded version of a complete assignment trees exists, then according to proposition 2 this gives an argument for α.

Algorithm 1 builds a depth-first search tree T that represents the steps of the search for arguments for a claim α from a knowledgebase Δ. Every node in T is an assignment tree, every node is an extension of the assignment tree in its parent node. The leaf node of every complete accepted branch is a complete consistent assignment tree.

$\mathsf{Reject}(T)$, and $\mathsf{Accept}(T)$ are boolean functions which, given the current state T of the search tree, test whether the leaf node of the currently built branch can be expanded further. $\mathsf{Reject}(T)$ rejects the current branch of the search tree if the assignment tree in its leaf node does not satisfy the conditions for an assignment tree. $\mathsf{Accept}(T)$ checks whether a solution has been found. Hence, $\mathsf{Accept}(T)$ tests whether the assignment tree in the leaf node of the currently built branch is a complete assignment tree. When either of these functions returns true, the algorithm rejects or outputs the current branch accordingly and the algorithm backtracks and continues to the next node of tree T to be expanded. $\mathsf{NextChild}(T)$ adds to T one of the next possible nodes for its current leaf. A next possible node for the current branch can be any extension of the assignment tree contained in its current leaf which satisfies the conditions of definition 11.

Algorithm 1 $Build(T)$

if $\mathsf{Reject}(T)$ **then**
  return $T = null$
end if
if $\mathsf{Accept}(T)$ **then**
  return T
  $S = \mathsf{NextChild}(T)$
end if
while $S \neq null$ **do**
  $Build(S)$
  $S = \mathsf{NextChild}(T)$
end while

The search is based on a graph structure whose vertices are represented by clauses from $\Delta \cup \{\neg\alpha\}$ and arcs link clauses ϕ, ψ for which $\mathsf{Preattacks}(\phi, \psi) \neq \emptyset$. In fact, the algorithm works by visiting a subgraph of this graph which we call the **query graph** of α in Δ. The query graph is the component (N, A) of the graph where for each node $\phi \in N$: (1) ϕ is linked to $\neg\alpha$ through a path in A and (2) $\forall a_i \in \mathsf{Units}(\phi)$ there is a $\psi \in N$ with $a_i \in \mathsf{Preattacks}(\phi, \psi)$. Hence, each unit in each clause of the search space has a link in the query graph associated to it. Figure 1 illustrates the structure of the query graph of $\alpha = \exists x \exists y(Q(x, y))$ in Δ from example 12. The idea in building an assignment tree by using the structure of the query graph, is to start from the negation of the claim and walk over the graph by following the links and unifying the atoms of pairs of contradictory literals connected with arcs. Hence, the algorithm at the same time follows the arcs of the graph and also produces partial instances of the clauses it visits as the

unification of atoms indicates. The partial instances produced while walking over

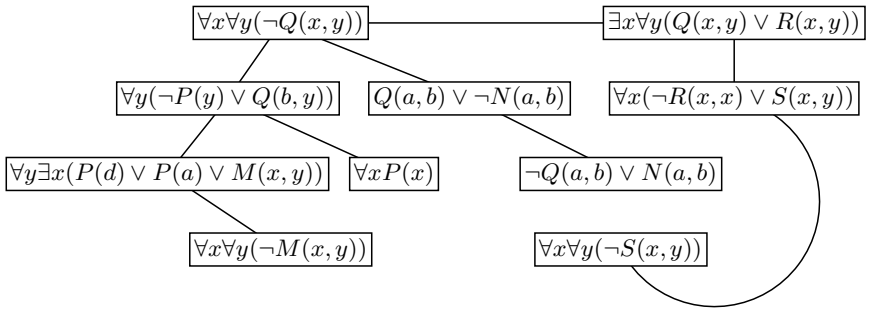


Fig. 1. The query graph of $\alpha = \exists x \exists y(Q(x,y))$ in Δ. The negation of the claim $\neg\alpha = \forall x \forall y(\neg Q(x,y))$ on the top left of the graph is the starting point for the search for arguments for α.

the graph are generated with respect to the conditions of definitions 11, 12, 14 and 15. Every time a clause ϕ on the graph is visited, a node q for an assignment tree is created with $e(q) = \phi$. For this node, an existential-free instance of $e(q)$ is generated by substituting each of its existentially quantified variables with an arbitrary constant that does not appear anywhere in $\Delta \cup \{\neg\alpha\}$ or in the instances already created during the search. This instantiation initializes the value $g(q)$ and sets the value $f(q)$ for q: $f(q) = \mathsf{Assign}(e(q), g(q))$. The value of $h(q)$ is also initialized at this stage to be equal to $f(q)$. After node q has been initialized as an assignment tree node, another instantiation process follows, which is based on unifying the atoms of the contradictory units in $h(q)$ and its parent with their most general unifier. This updates values $g(q)$ and $h(q)$. Let p be the parent of q in an assignment tree. If $\theta \subset \wp(\mathcal{B})$ is the the most general unifier of the atoms of a pair of contradictory units from $h(q)$ and $h(p)$, then $g(q) = g(q) \cup \theta$ and $h(q) = \mathsf{Assign}(h(q), g(q))$. Apart from node q, the values for g and h for any other node in the assignment tree are also updated. Every time such a unification binding is retrieved, its values are forwarded to the rest of the nodes in the assignment tree. These values are assigned to any of the corresponding clauses that can be associated through a sequence of arcs in the assignment tree to the variables of $e(q)$ and can therefore be affected by the bindings in θ.

Figure 2 represents the result of searching for arguments for $\alpha = \exists x \exists y(Q(x,y))$ using the query graph of figure 1. The result of the first branch of the search tree (at the leaf) is a complete consistent assignment tree which by substituting variable y in disjuncts $M(f,y)$ and $\neg M(f,y)$ by the same arbitrary constant gives a complete grounded assignment tree. The leaf of the second branch corresponds to a complete grounded assignment tree while the third branch is rejected because for node p with $e(p) = f(p) = h(p) = Q(a,b) \vee \neg N(a,b)$ there is only one arc in the graph that connects $e(p)$ with a clause that contains a complement of $\neg N(a,b)$. This is clause $\neg Q(a,b) \vee N(a,b)$ but a child q of p with

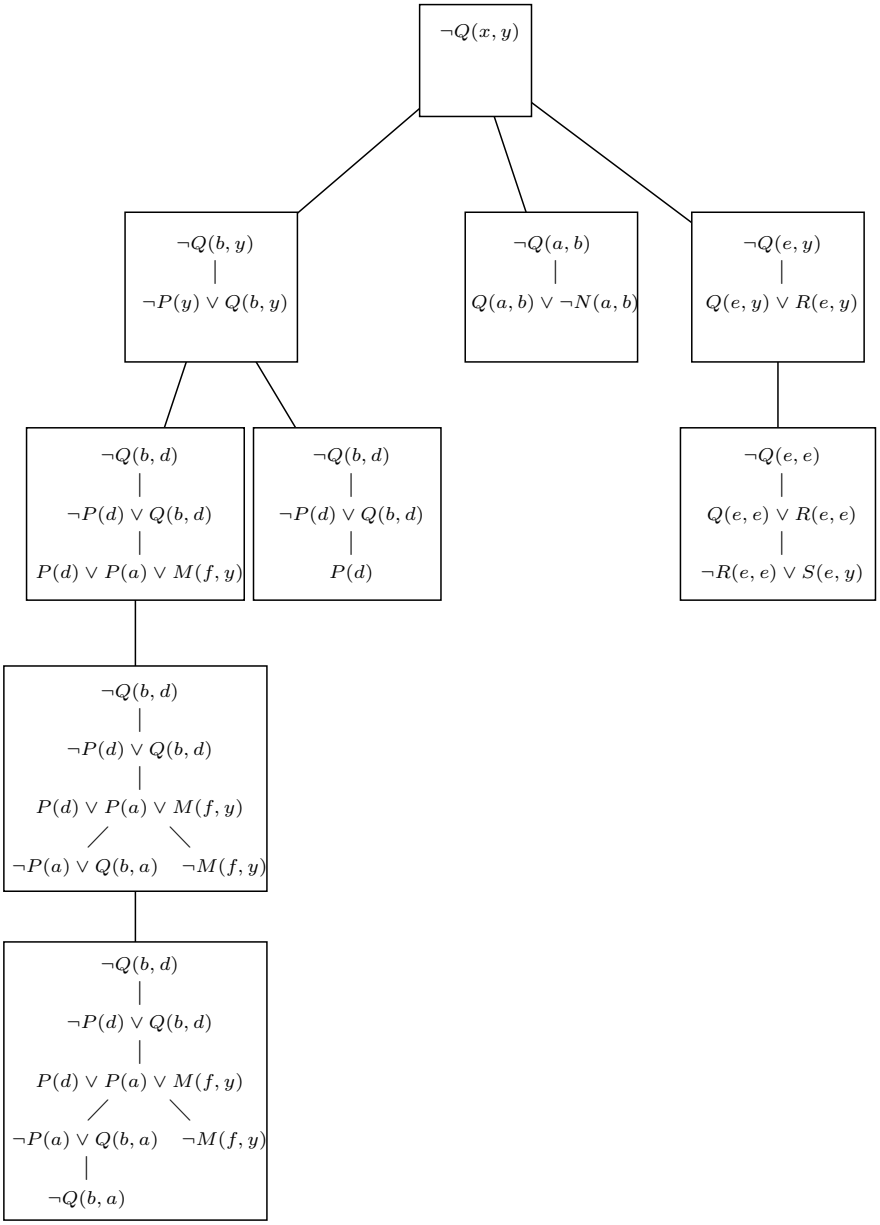


Fig. 2. A search tree generated using algorithm 1 by exploring the graph in figure 1. Each node of this search tree represents an assignment tree which extends the assignment tree contained in its parent node by one level. For this the algorithm adds clauses each of which preattacks their parent clause on a different unit. The atoms of the contradictory units between a parent and a child clause are unified and the assignments of the unification are passed on to any other clauses that can be affected in the assignment tree of this node.

$e(q) = \neg Q(a,b) \vee N(a,b)$ cannot be created because there is no assignment $g(q)$ for which $\mathsf{Attacks}(h(q), h(p)) \neq null$. The last branch of the search tree is rejected because adding node s with $e(s) = \forall x(\neg R(x,x) \vee S(x,y))$ as a child of r with $h(r) = Q(e,y) \vee R(e,y)$ requires unifying $R(x,x)$ with $R(e,y)$ which updates the value of $g(r)$ to $g(r) = \{x/e, y/e\} \in \mathsf{Prohibited}(e(r))$ and so condition 4 of the definition for an assignment tree is violated.

6 Discussion

Classical first-order logic has many advantages for representing and reasoning with knowledge. However, in general it is computationally challenging to generate arguments from a knowledgebase using classical logic. In this paper we propose a method for retrieving arguments in a rich first-order language. We have provided a theoretical framework, algorithms and theoretical results for this proposal.

References

1. L. Amgoud and C. Cayrol. A model of reasoning based on the production of acceptable arguments. *Annals of Math. and A.I.*, 34:197–216, 2002.
2. S. Benferhat, D. Dubois, and H. Prade. Argumentative inference in uncertain and inconsistent knowledge bases. In *Proceedings of the 9th Annual Conference on Uncertainty in Artificial Intelligence (UAI 1993)*, pages 1449–1445, 1993.
3. Ph. Besnard and A. Hunter. A logic-based theory of deductive arguments. *Artificial Intelligence*, 128:203–235, 2001.
4. Ph. Besnard and A. Hunter. Practical first-order argumentation. In *Proceedings of the 20th American National Conference on Artificial Intelligence (AAAI'2005)*, pages 590–595. MIT Press, 2005.
5. Ph. Besnard and A. Hunter. *Elements of Argumentation*. MIT Press, 2008.
6. C. Chesñevar, A. Maguitman, and R. Loui. Logical models of argument. *ACM Computing Surveys*, 32:337–383, 2000.
7. P. Dung, R. Kowalski, and F. Toni. Dialectical proof procedures for assumption-based admissible argumentation. *Artificial Intelligence*, 170:114–159, 2006.
8. V. Efstathiou and A. Hunter. Algorithms for effective argumentation in classical propositional logic: A connection graph approach. In *FoIKS*, pages 272–290. Springer, 2008.
9. M. Elvang-Gøransson, P. Krause, and J. Fox. Dialectic reasoning with classically inconsistent information. In *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence (UAI 1993)*, pages 114–121. Morgan Kaufmann, 1993.
10. A. García and G. Simari. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming*, 4(1):95–138, 2004.
11. R. Kowalski. A proof procedure using connection graphs. *Journal of the ACM*, 22:572–595, 1975.
12. R. Kowalski. *Logic for problem solving*. North-Holland Publishing, 1979.
13. H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, 7:25–75, 1997.
14. H. Prakken and G. Vreeswijk. Logical systems for defeasible argumentation. In D. Gabbay, editor, *Handbook of Philosophical Logic*. Kluwer, 2000.
15. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.