An argumentation workflow for reasoning in Ontology Based Data Access
Bruno Yun, Madalina Croitoru

To cite this version:

HAL Id: lirmm-01401316
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01401316
Submitted on 23 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
An argumentation workflow for reasoning in Ontology Based Data Access

Bruno YUN, Madalina CROITORU

*INRIA Graphik/LIRMM University Montpellier, France*

**Abstract.** In this paper we demonstrate how to benefit from structured argumentation frameworks and their implementations to provide for reasoning capabilities of Ontology Based Data Access systems under inconsistency tolerant semantics. More precisely, given an inconsistent Datalog± knowledge base we instantiate it using the ASPIC+ framework and show that the reasoning provided by ASPIC+ is equivalent to the main inconsistent tolerant semantics in the literature. We provide a workflow that shows the practical interoperability of the logic based frameworks handling Datalog± and ASPIC+.

**Keywords.** Applications and Structured Argumentation and Datalog+/-

**Introduction and Motivation**

Ontology Based Data Access (OBDA) is a popular setting used by many Semantic Web applications that encodes the access to data sources using an ontology (vocabulary) \[17, 18, 9\]. The use of the ontology will help obtain a unified view over heterogeneous data sources. Moreover, the ontology will enable the exploitation of implicit knowledge not explicitly stored in the data sources alone. One of the main difficulties in OBDA consists in dealing with potentially inconsistent union of facts (data sources). Reasoning with inconsistency needs additional mechanisms because classical logic will infer everything out of *falsum*. It is classically assumed (and a hypothesis that we will also follow in this paper) that the inconsistency in OBDA occurs at the fact level and not due to the ontology \[17, 18, 9\]. The facts are error prone due to their unrestrained provenance while ontologies are considered agreed upon as shared conceptualisations.

We consider here two main methods of handling inconsistency. On one hand (and inspired from database research) we consider repair based techniques. A repair is a maximally consistent set of facts. Reasoning with inconsistency using repairs relies on reasoning with repairs and combining the results using various methods (called inconsistency tolerant semantics). \[5, 7, 13\] Despite them being the mainstream techniques for OBDA reasoning, the main drawback of inconsistent tolerant semantics is the lack of implementations excepting a few dedicated approaches to particular semantics \[14, 6\].

A second method consists of using argumentation techniques. A Dung argumentation system \[12\] is a pair \(\mathcal{AS} = (\mathcal{A}, \mathcal{C})\), where \(\mathcal{A}\) is a set of arguments and \(\mathcal{C} \subseteq \mathcal{A} \times \mathcal{A}\) is a binary attack relation on them.

---

1Corresponding Author: E-mail: croitoru@lirmm.fr
In this paper we demonstrate how to benefit from structured argumentation frameworks and their implementations to provide for reasoning capabilities of OBDA systems under inconsistency tolerant semantics. More precisely, given an inconsistent \(Datalog^\pm\) knowledge base we instantiate it using the \(ASPIC^+\) framework \([15]\) and show that the reasoning provided by \(ASPIC^+\) is equivalent to the main inconsistent tolerant semantics in the literature. The significance of this work is a proposed workflow that will enable \(Datalog^\pm\) frameworks to handle inconsistencies in knowledge bases by means of the \(ASPIC^+\) framework. We use two frameworks:

- **Graal**, a Java toolkit dedicated to querying knowledge bases within the framework of \(Datalog^\pm\) and maintained by GraphIK team. Graal takes as input a Dlgp file and a query and answer the query using various means (saturation, query rewriting). This toolkit can be found at [https://graphik-team.github.io/graal/](https://graphik-team.github.io/graal/).

- **ACL’s ASPIC** project that takes as input a query and \(ASPIC^+\) knowledge base, i.e. rules (strict and defeasible), ordinary premises, axioms and preferences. The output is the answer to the query. This inference engine can be found at [http://aspic.cossac.org/components.html](http://aspic.cossac.org/components.html).

We use Graal’s representation of a knowledge base and construct the necessary input for the \(ASPIC^+\) argumentation inference engine. The difficulty of this work resides in the definition of the mapping (the contrariness function and the way facts and rules are handled) that ensures the semantic equivalence proved in the next sections.

![Interoperability Workflow of ACL and Graal](https://graphik-team.github.io/graal/papers/datalog+_v2.0_en.pdf)

In Figure 1 the interoperability workflow of the Graal software and the \(ASPIC^+\) implementation are shown. Let us detail here how the workflow functions:

- **Step 1**. The input of the software is a \(Datalog^\pm\) knowledge base obtained from the OBDA setting (that considers several data sources unified under the same ontology). The dlgp file that encodes this knowledge base (a textual format for the existential rule / Datalog framework) is parsed by the Graal framework. Each line in a dlgp file corresponds either to a fact, existential rule, negative constraint or conjunctive query. Please note that a complete grammar of the dlgp format is available here: [https://graphik-team.github.io/graal/papers/datalog+_v2.0_en.pdf](https://graphik-team.github.io/graal/papers/datalog+_v2.0_en.pdf).
• **Step 2 and 3.** The intermediary files are meant to serve as input for the contrariness function computation in Step 3. The contrariness function encodes the conflicts between atoms. This will be formally defined in the next section. Please note that one of the main difficulties of this work is properly defining the contrariness function such that the produced results are sound and complete with respect to inconsistent tolerant semantics.

• **Querying.** The output of this inference engine is the answer to the query w.r.t the inconsistent knowledge base \( \mathcal{K} = (F, \mathcal{R}, N) \). The soundness and completeness of the answer with respect to inconsistency tolerant semantics is ensured by the equivalence results presented in the next section.

**The Logical Language: Datalog±**

In this section we explain the logical language Datalog± used throughout the paper. We define the notion of Datalog± knowledge base, inconsistent knowledge base and explain the three inconsistency tolerant semantics mostly used in the literature. Its language \( \mathcal{L} \) is composed of formulas built with the usual quantifier (\( \exists, \forall \)) and only the connectors implication (\( \rightarrow \)) and conjunction (\( \land \)). We consider first-order vocabularies with constants but no other function symbol. An atomic formula (or atom) is of the form \( p(t_1, \ldots, t_n) \) where \( p \in \mathcal{P} \) is an \( n \)-ary predicate, and \( t_1, \ldots, t_n \) are terms. Classically, a fact is a ground atom. We denote by \( \bar{x} \) a vector of variables. An existential rule (or simply a rule) is a closed formula of the form \( R = \forall \bar{x} \exists \bar{y} (B \rightarrow \exists \bar{z} H) \), where \( B \) and \( H \) are conjuncts, with \( \text{vars}(B) = \bar{x} \cup \bar{y} \) and \( \text{vars}(H) = \bar{x} \cup \bar{z} \). The variables \( \bar{z} \) are called the existential variables of the rule \( R \). \( B \) and \( H \) are respectively called the body and the head of \( R \). We denote them respectively \( \text{body}(R) \) for \( B \) and \( \text{head}(R) \) for \( H \). We may sometimes omit quantifiers and write \( R = B \rightarrow H \). A negative constraint (or simply a constraint) is a rule of the form \( N = \forall \bar{x} (B \rightarrow \bot) \). A rule \( R = B \rightarrow H \) is applicable to a fact \( F \) if there is a homomorphism \( \sigma \) from \( B \) to \( F \). Let \( F' \) be a fact and \( \mathcal{R} \) be a set of rules. A fact \( F'' \) is called an \( \mathcal{R} \)-derivation of \( F \) if there is a finite sequence (called the derivation sequence) \( (F_0 = F, \ldots, F_n = F'') \) such that for all \( 0 \leq i < n \) there is a rule \( R \) which is applicable to \( F_i \) and \( F_{i+1} \) is an immediate derivation from \( F_i \). Given a fact \( F \) and a set of rules \( \mathcal{R} \), the chase (or saturation) procedure starts from \( F \) and performs rule applications in a breadth-first manner. The chase computes the closure of \( F \), i.e. \( CL_{\mathcal{R}}(F) \), which is the smallest set that contains \( F \) and that is closed under \( \mathcal{R} \)-derivation, i.e. for every \( \mathcal{R} \)-derivation \( F' \) of \( F \) we have \( F' \in CL_{\mathcal{R}}(F) \). Given a chase variant \( C \)\(^4\), we call \( C \)-finite the class of set of rules \( \mathcal{R} \), such that the \( C \)-chase halts on any fact \( F \), consequently produces a finite \( CL_{\mathcal{R}}(F) \). We limit our work in this paper to these kind of classes.

Let \( F \) and \( F'' \) be two facts. \( F \models F'' \) if and only if there is a homomorphism from \( F'' \) to \( F \). Given two facts \( F \) and \( F'' \) and a set of rules \( \mathcal{R} \) we say \( F, \mathcal{R} \models F'' \) if and only if \( CL_{\mathcal{R}}(F) \models F'' \) where \( \models \) is the classical first-order entailment [16].

**Knowledge base and inconsistency** Let us denote by \( \mathcal{L} \) the language described so far. A knowledge base \( \mathcal{K} \) is a finite subset of \( \mathcal{L} \). Precisely, \( \mathcal{K} \) is a tuple \( (F, \mathcal{R}, N) \) of a finite set of facts \( F \), rules \( \mathcal{R} \) and constrains \( N \). Saying that \( \mathcal{K} \models F \) means \( CL_{\mathcal{R}}(F) \models F \). We say a set of facts \( F \) is \( \mathcal{R} \)-inconsistent with respect to a set of constrains \( N \) and rules \( \mathcal{R} \) if and only if there exists \( N \in \mathcal{N} \) such that \( CL_{\mathcal{R}}(F) \models \text{body}(N) \), otherwise \( F \) is \( \mathcal{R} \)-consistent. A knowledge base \( \mathcal{K} = (F, \mathcal{R}, N) \) is said to be inconsistent with respect
to \( R \) and \( N \) (inconsistent for short) if \( F \) is \( R \)-inconsistent. We may use the notation \( CL_R(F) \models \bot \) to mean the same thing.

In the area of inconsistent ontological knowledge base query answering, we usually check what can be inferred from an inconsistent ontology. We usually begin by calculating all maximal consistent subsets of \( K \) called repairs. Given a knowledge base \( K = (F, R, N) \), we call by \( \text{Repairs}(K) \) the set of all repairs defined as:

\[
\text{Repairs}(K) = \{F' \subseteq F | F' \text{ is maximal for } \subseteq \text{ and } R\text{-consistent}\}
\]

Different inconsistency tolerant semantics are used for inconsistent ontology knowledge base query answering (Intersection of All Repairs: IAR, All Repairs: AR, Intersection of Closed Repairs: ICR); these semantics can yield different results.

**Definition 1 [cf [11]].** Let \( K = (F, R, N) \) be a knowledge base and \( \alpha \) be a query.

- \( \alpha \) is AR-entailed from \( K \): \( K \models_{AR} \alpha \) iff for \( \forall r \in \text{Repairs}(K) \), \( CL_R(r) \models \alpha \).
- \( \alpha \) is ICR-entailed from \( K \), written \( K \models_{ICR} \alpha \) iff \( \bigcap_{r\in\text{Repairs}(K)} CL_R(r) \models \alpha \).
- \( \alpha \) is IAR-entailed from \( K \), written \( K \models_{IAR} \alpha \) iff \( CL_R(\bigcap_{r\in\text{Repairs}(K)} r) \models \alpha \).

**Structured Argumentation for Datalog**

In this section we address the problem of how to use structured argumentation for Datalog. We show how the ASPIC+ framework can be instantiated to yield results equivalent to the state of the art in OBDA inconsistency tolerant semantics. We define the first instantiation in the literature of ASPIC+ using Datalog+.

ASPIC+ [15] is a framework for obtaining logical based argumentation system using any logical language \( L \). It is meant to generate an abstract argumentation framework and was created because abstract argumentation does not specify the structure of arguments and the nature of attacks. ASPIC+ is meant to provide guidance to those aspects without losing a large range of instantiating logics. Before going any further we will provide a few basic abstract argumentation notions needed later in this section. We consider an argumentation \( AS = (A, C) \), where \( A \) is a set of arguments and \( C \subseteq A \times A \) is a binary attack relation on them. We say that the argument \( a \in A \) is acceptable w.r.t a set of arguments \( \varepsilon \subseteq A \) iff \( \forall b \in A \) such that \( (b, a) \in C \), \( \exists c \in \varepsilon \) such that \( (c, b) \in C \). \( \varepsilon \) is conflict-free iff \( \exists a, b \in \varepsilon \) such that \( (a, b) \in C \). \( \varepsilon \) is admissible iff \( \varepsilon \) is conflict-free and all arguments of \( \varepsilon \) are acceptable w.r.t. \( \varepsilon \). \( \varepsilon \) is preferred iff it is minimal (for set inclusion) and admissible. \( \varepsilon \) is stable iff it is conflict-free and \( \forall a \in A \setminus \varepsilon , \exists b \in \varepsilon \) such that \( (b, a) \in C \). \( \varepsilon \) is complete iff it contains all arguments that are acceptable w.r.t. \( \varepsilon \). \( \varepsilon \) is grounded iff it is minimal (for set inclusion) and complete. Reasoning takes place on the various \( \varepsilon \) (also called extensions). Following [15], to use ASPIC+ , we need to choose a logical language \( L \) closed under negation (\( \neg \)), provide a set of rules \( R = R_d \cup R_s \) composed of defeasible rules and strict rules with \( R_d \cap R_s = \emptyset \), specify a contrariness function \( cf : L \rightarrow 2^L \) and a partial naming function \( n : R_d \rightarrow L \) that associates a well-formed formulas of \( L \) to a defeasible rule. The function \( n \) will not be used in this instantiation. In ASPIC+ an argumentation system is a triple \( AS = (L, R, n) \) and a knowledge base is \( K \subseteq L \) consisting of two disjoint subsets \( K_n \) (the axioms) and \( K_p \) (the ordinary premises).
To instantiate $\text{ASPIC}^+$ for $\text{Datalog}^\pm$, we define $\mathcal{L}$ as $\text{Datalog}^\pm$, rules in definition 2, and the contrariness function in definition 3. Please note that definitions 4 and 7 are new w.r.t. state of art regarding $\text{Datalog}^\pm$ instantiations conform [15].

**Definition 2** Strict rules (resp. defeasible rules) are of the form $\forall \vec{x} \neg \exists \vec{y}(B \rightarrow \exists \vec{z}H)$ (resp. $\forall \vec{x} \neg \exists \vec{y}(B \Rightarrow \exists \vec{z}H)$) with $B$, the body and $H$, the head are atoms or conjunction of atoms with $\text{vars}(B) = \vec{x} \cup \vec{y}$, and $\text{vars}(H) = \vec{x} \cup \vec{z}$.

**Definition 3** [15]. The function $cf$ is a function from $\mathcal{L}$ to $2^\mathcal{L}$ such that:

- $\varphi$ is the contrary of $\psi$ if $\varphi \in cf(\psi), \psi \notin cf(\varphi)$
- $\varphi$ is the contradictory of $\psi$ if $\varphi \in cf(\psi), \psi \in cf(\varphi)$
- Each $\varphi \in \mathcal{L}$ has at least one contradictory.

We define our own contrariness function to instantiate $\text{ASPIC}^+$ for $\text{Datalog}^\pm$ ($\mathcal{L} = \text{Datalog}^\pm$). This contrariness function is necessary because it is used in the attack relation. It is worth noting that the idea that we want to capture (as also defined in [1]) is that $x$ is the contrary of $y$ iff they cannot be both true but they can be both false. They are contradictory if the truth of one implies the falsity of the other and vise versa.

**Definition 4** ($\text{Datalog}^\pm$ contrariness function) Let $a \in \mathcal{L}$ and $b$ be a conjunction of atoms, $b \in cf(a)$ iff $\exists \psi$ an atom such that $a \models \psi$ and $\{b, \psi\}$ is $R$-inconsistent.

Here we recall that an $\text{ASPIC}^+$ argument can be built from axioms and ordinary premises or from rules and other arguments. The arguments are built once $R_d, R_s, cf$ and $K$ are known.

**Definition 5** (Argument cf [15]) Arguments in $\text{ASPIC}^+$ can be in two forms:

- $\emptyset \Rightarrow \varphi$ (resp. $\emptyset \Rightarrow \varphi$) with $\varphi \in K_\emptyset$ (resp. $\varphi \in K_\emptyset$ or $\emptyset \Rightarrow \varphi \in R_d$) such that $\text{Prem}(A) = \{\varphi\}, \text{Conc}(A) = \varphi, \text{Sub}(A) = \{A\}$ with $\text{Prem}$ returns premisses of $A$ and $\text{Conc}$ returns its conclusion.
- $\text{DefRules}(A) = \emptyset$.

- $A_1, \ldots, A_m$ $\Rightarrow \varphi$ (resp. $A_1, \ldots, A_m$ $\Rightarrow \varphi$), such that there exists a strict (resp. defeasible) rule $r = B \Rightarrow H$ (resp. $r = B \Rightarrow H$) and a homomorphism $\sigma$ from $B$ to $X = \text{Conc}(A_1) \wedge \text{Conc}(A_2) \wedge \cdots \wedge \text{Conc}(A_m)$.
  - $\text{Prem}(A) = \text{Prem}(A_1) \cup \cdots \cup \text{Prem}(A_m)$,
  - $\text{Conc}(A) = \alpha(X, r, \sigma)$,
  - $\text{Sub}(A) = \text{Sub}(A_1) \cup \cdots \cup \text{Sub}(A_m) \cup \{A\}$,
  - $\text{TopRule}(A) = r = B \Rightarrow H$ (resp. $r = B \Rightarrow H$), such that there exists an homomorphism $\sigma$ from $B$ to $X$.
  - $\text{DefRules}(A) = \text{DefRules}(A_1) \cup \cdots \cup \text{DefRules}(A_m)$ (resp. $\text{DefRules}(A) = \text{DefRules}(A_1) \cup \cdots \cup \text{DefRules}(A_m) \cup \{\text{TopRule}(A)\}$).

Attacks in $\text{ASPIC}^+$ are based on three notions (undercutting, undermining and rebutting). Each of those notions are useful as they capture different aspects of conflicts. In short, arguments can be attacked on a conclusion of a defeasible inference (rebutting attack), on a defeasible inference step itself (undercutting attack), or on an ordinary premise (undermining attack).
Definition 6 [cf [15]]. Let a and b be arguments, we say that a attacks b iff a undercuts, undermines or rebuts b, where:

- a undercuts argument b (on b′) iff Conc(a) ∈ cf(n(r)) for some b′ ∈ Sub(b) such that b′’s top rule r is defeasible.
- a rebuts argument b (on b′) iff Conc(a) ∈ cf(ψ) for some b′ ∈ Sub(b) of the form b′0, . . . , b′n ⇒ ψ.
- a undermines b (on ψ) iff Conc(a) ∈ cf(ψ) for an ordinary premise ψ of b.

We are now ready to define the mapping that allows the instantiation of ASPIC+ with Datalog++. The mapping will consider each fact as a defeasible rule because the inconsistency in the OBDA setting is assumed to come from the facts level. Therefore the only attack we consider in this instantiation is the undermine attack because we have simple defeasible rules. The rules of the ontology become strict rules.

Definition 7 (Mapping For ASPIC+ Instantiation of Datalog++) We denote by S the set of all possible inconsistent knowledge bases of the form K = (F, R, N) and G the set of all ASPIC+ instantiation using Datalog++ language. The mapping τ : S → G is defined as follows:

1. The mapping τ associates every R-consistent subset F_i ⊆ F to its defeasible rule ∅ ⇒ conjunct(F_i) where conjunct(F_i) denotes the conjunction of facts contained in F_i.
2. The mapping τ associates every rules r_i ∈ R to the same rule r_i ∈ R_s.

We will considerate that if ∅ ⇒ c, then c is an ordinary premise (c ∈ K_p).

In order to give properties of the ASPIC+ instantiation presented in this paper we remind few notions. ε is admissible iff ε is conflict-free and all arguments of ε are acceptable w.r.t. ε. ε is preferred iff it is maximal (for set inclusion) and admissible. ε is stable iff it is conflict-free and ∀a ∈ A|ε, ∃b ∈ ε such that (b, a) ∈ C.

We denote by AF^+_R the ASPIC+ argumentation framework constructed from K using the mapping of definition 7. We restate that attacks in AF^+_R are composed only of undermining because we only have simple defeasible rules of the form ∅ ⇒ c. The following lemma shows that stable extensions are closed under sub-arguments in the Datalog++ instantiation of ASPIC+.

Lemma 1 Let ε be an ASPIC+ stable extension and A ∈ ε an argument contained in ε. Then Sub(A) ⊆ ε.

Notation Let c = α_1 ∧ α_2 ∧ ⋯ ∧ α_n be a conjunction of facts. Elimination(c) = {α_1, α_2, . . . , α_n} is the set resulting from eliminating the conjunction of c. Let S be a set of facts. We denote by P(S) the superset of S which correspond to all subsets of S.

We can now define the set of arguments constructed on a consistent set of facts.

Definition 8 Let K = (F, R, N) be a knowledge base and AF^+_R be the corresponding ASPIC+ instantiation and S ⊆ F a R-consistent subset of F. We denote by Arg^+_R(S) the set of arguments such that their premises are contained in S. Formally:
Arg^A(S) = \{ASPIC^+ \text{ argument } a | \bigcup_{c \in \text{Prem}(a)} \text{Elimination}(c) \subseteq P(S)\}

The main result shows that the set of stable extension coincides with the set of preferred one and it is obtained from the arguments built on repairs.

**Theorem 1 (Repair Equivalence for ASPIC^+ Instantiation)** Let \( \mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N}) \) be a knowledge base, \( AF^A_{\mathcal{K}} \) be the corresponding ASPIC^+ instantiation and \( \sigma \in \{\text{preferred, stable}\} \). Then:

\[
\{\text{Arg}^A(R)|R \in \text{Repair}(\mathcal{K})\} = \text{Ext}_\sigma(AF^A_{\mathcal{K}})
\]

The state of the art can also provide a structured argumentation framework of Datalog± [11,10]. Let \( \mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N}) \) be a knowledge base, we denote by \( AF^M_{\mathcal{K}} \) the instantiated logical argumentation framework \((A, C)\) with \( A = \text{Arg}(\mathcal{F}) \) and \( C \) defined in [11,10]. According to [11] the arguments constructed on the set of repairs coincide with the arguments in the stable and preferred extension: \( \{\text{Arg}^M(R)|R \in \text{Repair}(\mathcal{K})\} = \text{Ext}_\sigma(AF^M_{\mathcal{K}}) \). We can thus conclude that the preferred/stable extensions in the two instantiated frameworks are the same and that for each stable/preferred extension of one framework, there is a corresponding stable/preferred extension in the other and vice-versa. This is formalized in the theorem below.

**Theorem 2 (Instantiations Equivalence)** Let \( \mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N}) \) be a knowledge base, \( AF^M_{\mathcal{K}} \) and \( AF^A_{\mathcal{K}} \) be the two argumentation framework instantiations. Then if \( \sigma \in \{\text{preferred, stable}\} \), \( |\text{Ext}_\sigma(AF^M_{\mathcal{K}})| = |\text{Ext}_\sigma(AF^A_{\mathcal{K}})| \) and for each extension under semantics \( \sigma, \varepsilon \in \text{Ext}_\sigma(AF^M_{\mathcal{K}}) \), there is a corresponding extension \( \varepsilon_2 \in \text{Ext}_\sigma(AF^A_{\mathcal{K}}) \) and vice-versa (the corresponding extension can be found via repairs).

**Conclusions**

In this paper we demonstrated how to benefit from structured argumentation frameworks and their implementations to provide for reasoning capabilities of OBDA systems under inconsistency tolerant semantics. More precisely, given an inconsistent Datalog± knowledge base we instantiated it using the ASPIC^+ framework and showed that the reasoning provided by ASPIC^+ is equivalent to the main inconsistent tolerant semantics in the literature. A workflow of interoperability between ASPIC^+ ACL framework and Graal Datalog± framework was thus formally underpinned. In future work we are interested in exploiting this workflow for the explanation capabilities of inconsistent tolerant semantics [23].

**Acknowledgments**

The authors acknowledge the support of ANR grant QUALINCA (ANR-12-0012).
References


