DAGGER: Datalog+/--Argumentation Graph
GEneRator
Bruno Yun, Madalina Croitoru, Srdjan Vesic, Pierre Bisquert

To cite this version:
Bruno Yun, Madalina Croitoru, Srdjan Vesic, Pierre Bisquert. DAGGER: Datalog+/--Argumentation
Graph GEneRator. AAMAS: Autonomous Agents and Multiagent Systems, Jul 2018, Stockholm,
Sweden. pp.1841-1843. lirmm-01892588

HAL Id: lirmm-01892588
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01892588
Submitted on 10 Oct 2018

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.
DAGGER: Datalog+/- Argumentation Graph GEnErator

Demonstration

Bruno Yun
INRIA GraphIK, Université de Montpellier, France
yun@lirmm.fr

Srdjan Vesic
CRIL - CNRS, Université d’Artois, France
vesic@cril.fr

Madalina Croitoru
INRIA GraphIK, Université de Montpellier, France
croitoru@lirmm.fr

Pierre Bisquert
INRIA GraphIK, INRA, France
pierre.bisquert@inra.fr

ABSTRACT
We introduce DAGGER: a generator for logic based argumentation frameworks instantiated from inconsistent knowledge bases expressed using Datalog+/- . The tool allows to import a knowledge base in DLGP format and the generation and visualisation of the corresponding argumentation graph. Furthermore, the argumentation framework can also be exported in the Aspartix format.

ACM Reference Format:

1 THE DAGGER’S TIMELINESS
This demonstration paper will present DAGGER, a Datalog+/- [9] Argumentation Graph GEnErator. We place ourselves in the context of multi agent argumentation systems [12], and, more precisely, logic-based argumentation systems, i.e. argumentation systems that employ arguments built over a logical knowledge base (KB). When reasoning about inconsistent logical KBs, one has to deploy reasoning mechanisms that are not following the classical logical inference. This is due to the fact that, in classical logic, falsum implies everything. Alternative reasoning techniques are therefore needed in order to make sense of such KBs. Argumentation is one reasoning under inconsistency technique, that allows to build arguments and attacks over an inconsistent data. The arguments represent the various logical consequences one can draw from consistent subsets of the KB. The attacks capture the inconsistency between the different pieces of knowledge. The set of arguments and the corresponding set of attacks is referred to as an argumentation framework (AF). AFs are visually represented using a directed graph where the nodes represent the arguments and the directed edges the attacks between the arguments [12].

Classically, reasoning with argumentation systems consists of finding the maximal sets of arguments that (1) are not attacking each other and (2) defend themselves (as a group) from all incoming attacks. Such sets are called extensions. An argument is skeptically accepted if it is in all extensions and credulously accepted if it is in only one extension. In the graph theory, such reasoning task is translated into finding and intersecting all the graph’s stables.

While a lot of theoretical work in the past 23 years has focused, amongst others, on optimising the extension finding procedures [13, 15], on the investigation of various extension notions [6] or on the investigation of desirable properties of logic based instantiations [1, 17], there are few tools that allow to generate an AF from a given KB [20]. Furthermore, the few available tools for reasoning using argumentation over inconsistent KBs do not allow for further interoperability (allowing their output to be used by other tools).

A workflow that will allow a knowledge engineer to (1) input a KB in a commonly used format and then (2) generate, (3) visualise or (4) export the argumentation graph are very useful for practical argumentation. Such scenario could be used when a non expert wants to reason, using argumentation, over a KB in a particular domain ([4, 18, 19], etc.). It could also be useful for investigating the theoretical properties of the generated AF. Given the fact that certain graph theoretical properties could radically improve the extension computation efficiency [23] such visualisation could be a useful tool for argumentation specialists. Last, please note that, even when the KB is modestly large, the corresponding argumentation graph can become truly immense [23]. In this case, allowing tool interoperability that will directly load a logically generated argumentation graph into efficient solvers [15, 20] can make the difference between time out errors and obtaining a result.

2 USING THE DAGGER
We propose the DAGGER tool that assists domain experts and argumentation developers in the specification, visualisation and/or export of logic based AFs built over the Datalog+/- language.

2.1 Agent Techniques: Logic Argumentation
Let us first make a note about the logical language used in this paper. Existential rules (whom computationally decidable subclasses are referred to as Datalog+/-) have been intensively investigated on the Semantic Web for their generalisation w.r.t. Description Logic fragments [21]. It has been shown that using argumentation techniques over inconsistent existential rules KBs yields extensions logically equivalent to the maximally consistent subsets [11] of the KB (called repairs [16]). Using argumentation over existential rules has been shown to be of practical interest over existing repair based approaches [14]. Argumentation for handling inconsistency tolerant semantics enhance the human interaction [4], are used in food science applications [3, 4] or allow for alternative computation methods [22]. Such techniques have been shown to have further implications w.r.t. human reasoning and bias detection [8].
An existential rule \( KB \mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N}) \) is composed of a finite set of facts \( \mathcal{F} \) (such as \( \{\text{packaging}(A)\} \) representing the fact that the individual \( A \) is a packaging), a set of rules \( \mathcal{R} \) (such as \( \{\forall x(\text{packaging}(x) \land \text{has}(x, \text{PlasticFilm}) \rightarrow \text{pollute}(x))\} \) representing the knowledge that a packaging that has a plastic film is polluting the environment) and a set of negative constraints \( \mathcal{N} \) (such as \( \{\forall x(\text{pollute}(x) \land \text{protectEnv}(x) \rightarrow \bot)\} \) representing the impossibility to both protect the environment and pollute it). The constraints are used to express negative knowledge about the world. In the considered setting, rules and constraints act as an ontology used to “access” different data sources. Therefore, we suppose that the set of naive extensions is different without computer expertise exists [10].

Starting from an inconsistent \( KB \) composed of a set of facts and an ontology containing positive and negative rules), using the AF of [11] we generate the arguments and the attacks corresponding to the KB. An argument [11] in Datalog+/– is composed of a minimal (w.r.t. set inclusion) set of facts called support and a set of facts entailed from the support called the conclusion. The Skolem chase coupled with the use of decidable classes of Datalog+/– ensures the finiteness of the AF proposed (following from [5]). The attack considered is the undermining [11]: \( a \) attacks \( b \) iff the union of the conclusion of \( a \) and an element of the support of \( b \) entails a negative constraint. Please note that the attack is not symmetric which ensures the fact that the set of naive extensions is different from the sets of stable and preferred extensions.

The AF above outputs a set of extensions equivalent to the repairs [7, 16] of the KB (i.e. the maximum consistent set of facts).

### 2.2 DAGGER Architecture

The layered architecture of the DAGGER tool is shown in Figure 1 and it is detailed as follows:

- **High level**: This layer is mainly composed of the graphical user interface (GUI) that is used for the different interactions. It has a text area that allows to enter a KB expressed in the DGLP format (i.e. the formal for expressing existential rules).
- **Mid level**: This layer is composed of the logical model: KBs and AFs.
- **Low level**: This layer is composed of the computational tools that allow the computation of the arguments, the attacks (via the Graal library) and the repairs (i.e. the extensions). The information flow passes from the high level to the low level through the intermediate level using the different communication channels between modules.

### 2.3 Usability Scenarios

We consider three usability scenarios of DAGGER. All of these scenarios are easily employed using DAgGER.

**Scenario 1.** First, we consider the task of a non computer science specialist inputting an inconsistent KB of his expertise and wanting to find the maximally consistent point of views one can consider. For instance, let us consider the KB of Example 2.1. Please note that tools for assisting non domain experts in building such KBs without computer expertise exists [10].

**Example 2.1.** Let \( \mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N}) \) be a KB with:

\[
\mathcal{F} = \{\text{packaging}(A), \text{has}(A, \text{PlasticFilm}), \text{protectEnv}(A)\}
\]

\[
\mathcal{R} = \{\forall x(\text{packaging}(x) \land \text{has}(x, \text{PlasticFilm}) \rightarrow \text{pollute}(x))\}
\]

\[
\mathcal{N} = \{\forall x(\text{pollute}(x) \land \text{protectEnv}(x) \rightarrow \bot)\}
\]

In this KB, a packaging \( A \) with a plastic film is said to pollute the environment. However, since the possession of a plastic film leads to pollution, this KB is thus inconsistent. Finding maximally consistent point of views (or repairs) consists in computing all maximal subsets of \( \mathcal{F} \) that do not trigger a negative constraint of \( \mathcal{F} \). Here, we have three repairs: \( \{\text{packaging}(A), \text{has}(A, \text{PlasticFilm})\} \), \( \{\text{packaging}(A), \text{protectEnv}(A)\} \) and \( \{\text{has}(A, \text{PlasticFilm}), \text{protectEnv}(A)\} \).

**Scenario 2.** Second, we consider an argumentation specialist looking for graph based structural properties of argumentation graphs instantiated with particular KBs. For instance, let \( \mathcal{K} \) be a KB with three facts \( a(m), b(m), c(m) \), no rules and only containing one binary negative constraint \( \forall x(b(x) \land c(x) \rightarrow \bot) \). By generating the graph representation one might observe that the graph is symmetrical thus satisfying certain restrictions over its extensions [2]. However, if one considers a ternary negative constraint that is added to the KB (i.e. \( \forall x(a(x) \land b(x) \land c(x) \rightarrow \bot) \)), one can observe that the structure of the graph changes and it is no longer symmetric (and thus the properties of [2] do not hold anymore).

**Scenario 3.** Third, we consider a KB composed of 7 facts, 2 rules and 1 negative constraints. Generating the graph over such a KB will yield a graph of 383 arguments and 32768 attacks. Non optimised tools will not be able to handle these large graphs for a computationally expensive operation such as finding all its stable. However, ASPARTIX solvers based on SAT [15] will generate all extensions in less than 1 seconds. This is why one can use the export feature for such computations.