The pyglaf argumentation reasoner

Mario Alviano

Department of Mathematics and Computer Science, University of Calabria, Italy
alviano@mat.unical.it

Abstract. The PYGLAF reasoner takes advantage of circumscription to solve computational problems of abstract argumentation frameworks. In fact, many of these problems are reduced to circumscription by means of linear encodings, and a few others are solved by means of a sequence of calls to an oracle for circumscription. Within PYGLAF, Python is used to build the encodings and to control the execution of the external circumscription solver, which extends the SAT solver GLUCOSE and implements an algorithm based on unsatisfiable core analysis.

1 Introduction

Circumscription [4] is a nonmonotonic logic formalizing common sense reasoning by means of a second order semantics, which essentially enforces to minimize the extension of some predicates. With a little abuse on the definition of circumscription, the minimization can be imposed on a set of literals, so that a set of negative literals can be used to encode a maximization objective function. Since many semantics of abstract argumentation frameworks are based on a preference relation that essentially amount to inclusion relationships, PYGLAF (`http://alviano.com/software/pyglaf/`) uses circumscription as a target language to solve computational problems of abstract argumentation frameworks.

PYGLAF is implemented in Python and uses CIRCUMSCRIPTINO (`http://alviano.com/software/circumscriptino/`), a circumscription solver extending the SAT solver GLUCOSE [1]. Linear reductions are used for all semantics. For the ideal extension, the reduction requires the union of all admissible extensions of the input graph; such a set is computed by means of iterative calls to CIRCUMSCRIPTINO. The communication between PYGLAF and CIRCUMSCRIPTINO is handled in the simplest possible way, that is, via stream processing. This design choice is principally motivated by the fact that the communication is often minimal, limited to a single invocation of the circumscription solver.

2 Circumscription

Let $\mathcal{A}$ be a fixed, countable set of *atoms* including $\bot$. A *literal* is an atom possibly preceded by the connective $\neg$. For a literal ℓ, let $\overline{\ell}$ denote its *complementary literal*, that is, $\overline{p} = \neg p$ and $\overline{\neg p} = p$ for all $p \in \mathcal{A}$; for a set L of literals, let $\overline{L}$ be $\{\overline{\ell} \mid \ell \in L\}$. *Formulas* are defined as usual by combining atoms and the

connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. A *theory* is a set T of formulas including $\neg\bot$; the set of atoms occurring in T is denoted by $atoms(T)$. An *assignment* is a set A of literals such that $A \cap \overline{A} = \emptyset$. An *interpretation* for a theory T is an assignment I such that $(I \cup \overline{I}) \cap \mathcal{A} = atoms(T)$. Relation $\models$ is defined as usual. I is a *model* of a theory T if $I \models T$. Let $models(T)$ denote the set of models of T.

Circumscription applies to a theory T and a set P of literals subject to minimization. Formally, relation $\leq^P$ is defined as follows: for I, J interpretations of T, $I \leq^P J$ if $I \cap P \subseteq J \cap P$. $I \in models(T)$ is a *preferred model* of T with respect to $\leq^P$ if there is no $J \in models(T)$ such that $I \not\leq^P J$ and $J \leq^P I$. Let $CIRC(T, P)$ denote the set of preferred models of T with respect to $\leq^P$.

3 From Argumentation Frameworks to Circumscription

An *abstract argumentation framework* (AF) is a directed graph G whose nodes $arg(G)$ are arguments, and whose arcs $att(G)$ represent an attack relation. An *extension* E is a set of arguments. The *range* of E in G is $E_G^+ := E \cup \{x \mid \exists yx \in att(G) \text{ with } y \in E\}$. In the following, the semantics of ICCMA'17 are characterized by means of circumscription.

For each argument x, an atom a_x is possibly introduced to represent that x is attacked by some argument that belongs to the computed extension E, and an atom r_x is possibly introduced to enforce that x belongs to the range E_G^+:

$$attacked(G) := \left\{ a_x \leftrightarrow \bigvee_{yx \in att(G)} y \;\middle|\; x \in arg(G) \right\} \tag{1}$$

$$range(G) := \left\{ r_x \rightarrow x \vee \bigvee_{yx \in att(G)} y \;\middle|\; x \in arg(G) \right\} \tag{2}$$

The following set of formulas characterize semantics not based on preferences:

$$conflict\text{-}free(G) := \{\neg\bot\} \cup \{\neg x \vee \neg y \mid xy \in att(G)\} \tag{3}$$

$$admissible(G) := conflict\text{-}free(G) \cup attacked(G) \cup \{x \rightarrow a_y \mid yx \in att(G)\} \tag{4}$$

$$complete(G) := admissible(G) \cup \left\{ \left(\bigwedge_{yx \in att(G)} a_y \right) \rightarrow x \;\middle|\; x \in arg(G) \right\} \tag{5}$$

$$stable(G) := complete(G) \cup range(G) \cup \{r_x \mid x \in arg(G)\} \tag{6}$$

Note that in (4) truth of an argument x implies that all arguments attacking x are actually attacked by some true argument. In (5), instead, whenever all attackers of an argument x are attacked by some true argument, argument x is forced to be true. Finally, in (6) all atoms of the form r_x are forced to be true, so that the range of the computed extension has to cover all arguments.

The ideal semantic is defined as follows (Proposition 3.6 by [2]): Let X be the set of admissible extensions of G that are not attacked by any admissible extensions, that is, $X := \{E \in models(admissible(G)) \mid \nexists E' \in models(admissible(G))$

Algorithm 1: Compute the union of all admissible extensions of an AF G

```
1 T := admissible(G);      U := ∅;
2 repeat
3 |   Compute I ∈ CIRC(T, overline(arg(G) \ U));      // prefer arguments not in U
4 |   U' := U;      U := U ∪ (I ∩ arg(G));      // possibly extend the union
5 until U = U';                // terminate when no argument is added to U
```

such that $yx \in att(G), x \in E, y \in E'\}$. E is the ideal extension of G if $E \in X$, and there is no $E' \in X$ such that $E' \supseteq E$.

All semantics of ICCMA'17 are characterized in circumscription as follows:

$$co(G) := CIRC(complete(G), \emptyset) \tag{7}$$
$$st(G) := CIRC(stable(G), \emptyset) \tag{8}$$
$$gr(G) := CIRC(complete(G), arg(G)) \tag{9}$$
$$pr(G) := CIRC(complete(G), \overline{arg(G)}) \tag{10}$$
$$sst(G) := CIRC(complete(G) \cup range(G), \{\neg r_x \mid x \in arg(G)\}) \tag{11}$$
$$stg(G) := CIRC(conflict\text{-}free(G) \cup range(G), \{\neg r_x \mid x \in arg(G)\}) \tag{12}$$
$$id(G, U) := CIRC(admissible(G) \cup \overline{arg(G) \setminus Y},\ \overline{Y}) \tag{13}$$

where in (13) U is the union of all admissible extensions of G, and Y is $U \setminus \{x \mid \exists yx \in att(G),\ y \in U\}$.

4 Implementation

Abstract argumentation frameworks can be encoded in trivial graph format (TGF) as well as in aspartix format (APX). The following data structures are populated during the parsing of the input graph G: a list `arg` of the arguments in $arg(G)$; a dictionary `argToIdx`, mapping each argument x to its position in `arg`; a dictionary `att`, mapping each argument x to the set $\{y \mid xy \in att(G)\}$; a dictionary `attR`, mapping each argument x to the set $\{y \mid yx \in att(G)\}$. Within these data structures, theories (7)–(13) are constructed in amortized linear time.

Ideal Extension. The union U of all admissible extensions is computed by Algorithm 1. Initially, U is empty, and CIRCUMSCRIPTINO is iteratively asked to compute an admissible extension that maximize the accepted arguments not already in U, so to expand U as much as possible at each iteration.

Credulous and Skeptical Acceptance. For complete, stable, and preferred extensions, credulous acceptance is addressed by checking consistency of the theory extended with the query argument. Similarly, skeptical acceptance is addressed by adding the complement of the query argument for complete, and stable extensions. Grounded and ideal extensions are unique, and therefore credulous

Algorithm 2: Compute grounded, stable and preferred extensions of G

```
1 Compute I_gr ∈ co(G);      stable := ∅;      preferred := ∅;
2 T := complete(G) ∪ {x ∈ arg(G) | x ∈ I_gr};      P := ¬arg(G);
3 for I ∈ CIRC(T, P) do                    // enumerate preferred extensions
4     preferred := preferred ∪ {I};          // found new preferred extension
5     if for all x ∈ arg(G) \ I there is yx ∈ att(G) such that y ∈ I then
6         stable := stable ∪ {I};   // the preferred extension is also stable
7 return (I_gr, stable, preferred);
```

acceptance is addressed by checking the presence of the query argument in the computed extension. Actually, for the ideal extension, a negative answer is possibly produced already if the query argument is not part of the union of all admissible extensions. The remaining acceptance problems are addressed naively by extension enumeration; we plan to extend CIRCUMSCRIPTINO with query answering, so to improve the implementation of these acceptance problems.

Dung's Triathlon. The triathlon is addressed by Algorithm 2 based on the following observations: the grounded extension is contained in every preferred extension (Theorem 25 by [3]), and every stable extension is a preferred extension (Lemma 15 by [3]). Accordingly, the algorithm starts by computing the unique grounded extension I_{gr} of the input graph. After that, a theory whose models are complete extensions is built, and simplified by enforcing truth of all arguments in I_{gr}. The objective literals are the negation of all arguments, so that preferred extensions will be computed by CIRCUMSCRIPTINO. Every preferred extension returned by CIRCUMSCRIPTINO is finally checked for stability by means of a linear time Python function.

References

1. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Boutilier, C. (ed.) IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. pp. 399–404 (2009), http://ijcai.org/Proceedings/09/Papers/074.pdf
2. Caminada, M.: A labelling approach for ideal and stage semantics. Argument & Computation 2(1), 1–21 (2011), http://dx.doi.org/10.1080/19462166.2010.515036
3. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. 77(2), 321–358 (1995), http://dx.doi.org/10.1016/0004-3702(94)00041-X
4. McCarthy, J.: Circumscription - A form of non-monotonic reasoning. Artif. Intell. 13(1-2), 27–39 (1980), http://dx.doi.org/10.1016/0004-3702(80)90011-9