SAT solvers

Computational Models of Argumentation

Christophe Yang
Quentin Januel
Sylvain Declercq

2022

1 Introduction

Argumentation frameworks are widely used in artificial intelligence since they can model formalized arguments and draw conclusions from them.
Formally, an argumentation framework is an oriented graph where the nodes represent the arguments and the edges represent the attacks. To decide if an argument is accepted, we can compute an extension (with regards to a semantics) of the graph, which is a subset of its nodes, and see if the argument belongs to it.
In this paper, we will focus on the following semantics: **complete**, **stable**, **grounded** and **preferred** [1].
We will refer to them as CO, ST, GR and PR from now on. Except for the grounded one, finding extensions of these semantics is a hard problem (NP). That's why the **ICCMA**[2] has been organized since 2015 to find the best algorithms.

We will use SAT to compute the extensions, and study the impact of the choice of the SAT solver.

2 Tasks

For each semantics, there are four different tasks we are interested in:
- **SE**: find one extension, no matter which one
- **EE**: find all the extensions
- **DC**: decide if a given argument is credulously accepted
- **DS**: decide if a given argument is skeptically accepted

An argument is said to be credulously accepted if it is accepted in at least one of the extensions, and skeptically accepted if it is accepted in all of them.
That leaves us with the 16 following problems: SE-CO, SE-ST, SE-GR, SE-PR, EE-CO, EE-ST, EE-GR, EE-PR, DC-CO, DC-ST, DC-GR, DC-PR, DS-CO, DS-ST, DS-GR, DS-PR.

3 SAT

We can convert each semantics to a SAT instance. Finding one extension is then equivalent to finding a model of the SAT instance, and finding all the extensions is equivalent to enumerating all its models.
For the DC task, we add a clause forcing the argument to be accepted, and then check if the model is satisfiable.
For the DS task, we add a clause forcing the argument to be rejected, and then check if the model is unsatisfiable.

We have the following CNF:

$$\begin{aligned}\text{ST}: &\bigwedge_{a_i \in A} \left(a_i \vee \bigvee_{a_j \in A, (a_j, a_i) \in R} a_j \right), \\ &\bigwedge_{a_i \in A} \left[\bigwedge_{a_j \in A, (a_j, a_i) \in R} (\neg a_i \vee \neg a_j) \right]\end{aligned} \tag{1}$$

$$
\begin{aligned}
\mathrm{CO}: & \bigwedge_{a_i \in A} \left(\neg a_i \vee \neg P_{a_i}\right), \\
& \bigwedge_{a_i \in A} \left(a_i \vee \bigvee_{a_j \in A, (a_j, a_i) \in R} \neg P_{a_j} \right), \\
& \bigwedge_{a_i \in A} \bigwedge_{a_j \in A, (a_j, a_i) \in R} \left(\neg a_i \vee P_{a_j}\right), \\
& \bigwedge_{a_i \in A} \left(\neg P_{a_i} \vee \bigvee_{a_j \in A, (a_j, a_i) \in R} a_j \right), \\
& \bigwedge_{a_i \in A} {}_{a_j \in A, (a_j, a_i) \in R} \left(P_{a_i} \vee \neg a_j\right)
\end{aligned}
\tag{2}
$$

where A is the set of arguments and R the set of attacks[3].
The grounded semantics is trivial: it is always unique and we are guaranteed to find it by doing a unit propagation, hence its linear time complexity.
The preferred semantics is more complex. There are two different approaches we will compare in this paper.
The first one is to compute all the complete extensions and
- for SE, return the longest one
- for EE, filter the complete extensions that are subsets of other complete extensions
- for DC, same as EE but then check if the argument belongs to one preferred extension
- for DS, same as EE but then check if the argument belongs to all preferred extensions

The second one is to find one complete extension and to try to add arguments to it until not possible.

4 Building the solver

The solver had to be fast so we chose Rust as the language. We first made a library[4] to use famous SAT solvers we will then compare. It allows for minisat, manysat and glucose, and also for portfolio that combines these solvers in parallel. Then we made the argumentation solver itself[5].

5 Observations

References

[1] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artificial intelligence*, vol. 77, no. 2, pp. 321–357, 1995.

[2] "The international competition on computational models of argumentation." http://argumentationcompetition.org/.

[3] J.-M. Lagniez, E. Lonca, and J.-G. Mailly, "Coquiaas: A constraint-based quick abstract argumentation solver," in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 928–935, IEEE, 2015.

[4] "Out rust sat solver library." https://github.com/QuentinJanuel/sat-portfolio.

[5] "Out argumentation solver." https://github.com/QuentinJanuel/TER.