LamatzSolver-v0.1: A grounded extension finder based on the Java-Collection-Framework

Nico Lamatz

FernUniversität in Hagen, 58084 Hagen, Germany
Department of Knowledge-Based Systems, University of Hagen
http://www.fernuni-hagen.de/wbs/

Abstract. This paper describes the system architecture of LamatzSolver-v0.1, a solver for extension generation of Dung's abstract framework [4] in the contest of International Competition on Computational Models of Argumentation (ICCMA'15). The solver is implemented in Java and determines all extensions of grounded semantics. The algorithm used is oriented on the characteristic function [1, 4] and based on the Java-Collections-Framework.

Keywords: ArgumentContainer, Characteristic function, Grounded semantics, HashMap, Java-Collection-Framework

1 System architecture

LamatzSolver-v0.1 uses the system architecture summarized in Figure 1.

The class LamatzSolver inherits all necessary methods from AbstractSolver of probo [3] for the command line interface which represents the unique possibility for the user to communicate with the system. LamatzSolver hands the input over to TgfParserAAS. Tgf is the only accepted data format. In comparison with the TgfParser from probo TgfParserAAS changed the representation of Dung's abstract framework. Each argument is stored by its name in a class called ArgumentContainer. Furthermore this class stores each *attack from* and *attack on* in a HashMap. All ArgumentContainer are separately stored in a HashMap. The class AdvanceAAS investigates each argument about the attacks. If the HashMap "attacksFrom" is empty, the argument is also stored in another HashMap called "typeZero". There are two more types of HashMaps "typeOne" and "typeTwo" beside the specified one. As explained, the system only generates the grounded extension but AdvanceAAS provides with these two additional HashMaps the opportunity to advance the system. HashMap "typeOne" stores arguments which can defend themself and HashMap "typeTwo" stores arguments that only can be defended by other arguments.

After preparation phase the class LamatzSolver gets a reference to AdvanceAAS and overhands this to class GroundedExtensionFinder. GroundedExtensionFinder

is now able to get all important information for generating the grounded extension. After determination GroundedExtensionFinder returns the HashMap "grounded". Then the class LamatzSolver brings the grounded extension into the required form for probo.

Every return command is implemented with a public method. The determination of the HashMaps of AdvanceAAS as well as GroundedExtensionFinder are private and consequently not visible from the outside.

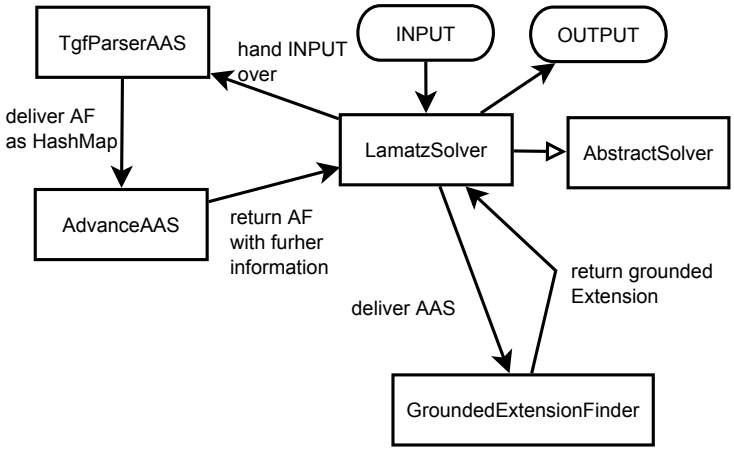

Fig. 1. System Architecture of LamatzSolver-v0.1

2 How the problem is solved

Finding the grounded extension is elementary for determination of other semantics. In fact it is really important getting a solution quickly. The algorithm of LamatzSolver-v0.1 can be described in five steps as follows:

1. Checks if HashMap "typeZero" is empty. If the answer is "yes" than a empty HashMap called "grounded" is being returned, otherwise the build() method will be called.
2. In the build() method each argument of HashMap "typeZero" is copied to the HashMap "grounded".
3. Then the size of HashMap "grounded" is stored in a parameter "prev" and for each argument the defended arguments will be determined and added to the HashMap "grounded". According to this each ArgumentContainer has a reference of the attacks on other arguments. These arguments are stored in a HashMap "out" [2] and the attacks of these arguments are candidates for the grounded extension. Then the method checks if all attackers of a candidate are defeated.

4. Step 3 is being repeated for HashMap "grounded" until the HashMap does not grow anymore. This is realized with a comparison of the current size of HashMap "grounded" and the parameter "prev".
5. HashMap "grounded" will be returned.

In a nutshell GroundedExtensionFinder is an implementation of the characteristic function. The runtime depends on the problem. For example the problem "real_4" from iccma2015[1] delivers the following runtime[2] result:

```
Start:
Attacks: 146530
Arguments: 100000
Size of grounded: 9568
Time elapsed: 0.41 Seconds
```

Annotation: This result is a Java console output from a separate class which is not part of LamatzSolver-v0.1.

3 Design choices and gained experience

There are two design choices made in LamatzSolver-v0.1. First is the implementation of sets in a mathematical sense and second the relation between arguments. Java provides for sets the Collections-Framework. Thus important operations for sets like *intersect, minus, union* and *complement* are possible. For a relation the Collection-Framework can be used too, but it is circuitous without a separate class. LamatzSolver-v0.1 uses a container class ArgumentContainer which stores the name of argument and the attacks from and on arguments. Sets like attacks are implemented as a HashMap. The advantage of a HashMap in comparison to other data structures as HashSet or Set is the access to the ArgumentContainer. HashSets or Sets have to be searched by an iterator from the beginning or the last element of the iterator. In a HashMap the key is sufficient for getting access to the object. In LamatzSolver-v0.1 HashMaps are organized with the key of the argument name and the value ArgumentContainer. This choice is based upon the fact that all information about an argument should be available at any point of runtime. Resources should simply be used for extension generation.

References

1. Christoph Beierle, Gabriele Kern-Isberne: Methoden wissensbasierter Systeme, 5. Auflage, Hagen und Dortmund 2014

[1] http://argumentationcompetition.org/2015/rules.html

[2] The system is tested on a machine with a Intel(R) Core(TM) i5-4570 CPU @ 3.20 GHz processor and 8 GB RAM with 1333 MHZ.

2. Martin Caminada: A Gentle Introduction to Argumentation Semantics, 2008
3. Federico Cerutti, Nir Oren, Hannes Straß, Matthias Thimm, Mauro Vallati: The First International Competition on Computational Models of Argumentation (ICCMA'15)
4. Phan Minh Dung: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Artificial Intelligence 77 (1995) 321-357