ASPARTIX-V19 - System Description for ICCMA'19

Wolfgang Dvořák, Anna Rapberger, Johannes P. Wallner, and Stefan Woltran

Institute of Logic and Computation, TU Wien, Vienna, Austria
{dvorak,arapberg,wallner,woltran}@dbai.tuwien.ac.at

Abstract

In this solver description we present ASPARTIX-V, in its 2019 edition, which participates in the International Competition on Computational Models of Argumentation (ICCMA) 2019. ASPARTIX-V is capable of solving all classical (static) reasoning tasks part of ICCMA'19 and extends the ASPARTIX suite of systems by incorporation of recent ASP language constructs (e.g. conditional literals), domain heuristics within ASP, and multi-shot methods. In this light we partially deviate from an earlier focus on monolithic approaches (i.e., one-shot solving via a single ASP encoding) to further enhance performance.

1 Solver Description

In this paper we describe ASPARTIX-V (Answer Set Programming Argumentation Reasoning Tool - Vienna) in its 2019 edition. ASPARTIX-V19 solves several reasoning tasks on argumentation frameworks (AFs) [1] and is based on earlier versions of ASPARTIX and its derivatives [4, 2, 3, 6, 10]. Given an AF as input, in the format of `apx`, ASPARTIX-V delegates the main reasoning to an answer set programming (ASP) solver (e.g. [8]), with answer set programs encoding the argumentation semantics and reasoning tasks. The basic workflow is shown in Figure 1, i.e., the AF is given in `apx` format (facts in the ASP language), and the AF semantics and reasoning tasks are encoded via ASP rules, possibly utilizing further ASP language constructs. In the next section we highlight specifics of the current version and in particular differences to prior versions. ASPARTIX, and its derivatives, are available online under

https://www.dbai.tuwien.ac.at/research/argumentation/aspartix/

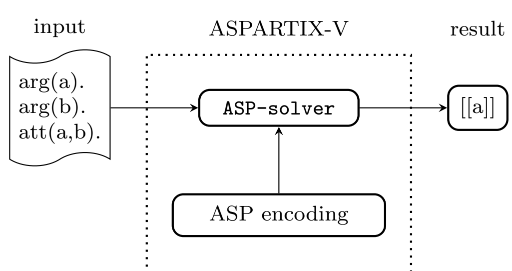

Figure 1: Basic workflow of ASPARTIX-V

2 Differences to earlier Versions

In this competition version of the ASPARTIX system we deviate from classical ASPARTIX design virtues. First, while traditional ASPARTIX encodings are modular in the sense that fixed encodings for semantics can be combined with the generic encodings of reasoning tasks, we use semantics encodings specific to a reasoning task. Second, when appropriate we apply multi-shot methods for reasoning which is in contrast to the earlier focus on so-called monolithic encodings, where one uses a single ASP-encoding and runs the solver only once (as illustrated in Figure 1). Third we make use of advanced features of the ASP-language, and utilize clingo v5.3.0 and v4.4.0 [1][8].

Next, we list and briefly discuss some of the ASP-techniques novel to the ASPARTIX system. First, we exploit the concept of conditional literals [7, Section 3.1.11], which has first been applied for ASP-encodings of argumentation semantics in [6]. For example we simplified the encoding of grounded semantics (cf. Listing 1). Moreover, conditional literals enable us to give ASPARTIX style encodings of the translations from AF semantics to ASP semantics provided in [11]. Second, we exploit clingo domain heuristics [9] (see also [7, Chapter 10]), in order to compute subset-maximal extensions while only specifying constraints for the base semantics [5].

Listing 1: Encoding for grounded semantics (using conditional literals)

in(X) ← **arg**(X), **defeated**(Y) : **att**(Y,X).
defeated(X) ← **arg**(X), **in**(Y), **att**(Y,X).

3 Capabilities of the solver

ASPARTIX-V19 supports all the standard tasks of ICCMA 2019 but does not support the dynamic settings of the special track. That is, ASPARTIX-V19 supports complete (CO), preferred (PR), stable (ST), semi-stable (SST), stage (STG), grounded (GR), and ideal (ID) semantics. For each of the semantics it supports the following reasoning tasks.

- Some Extension (SE): Given an AFs, determine some extension
- Enumerate Extensions (EE): Given an AFs, determine all extensions
- Decide Credulous Acceptance (DC): Given an AFs and some argument, decide whether the given argument is credulously inferred
- Decide Skeptical Acceptance (DS): Given an AFs and some argument, decide whether the given argument is skeptically inferred

The docker of the competition version ASPARTIX-V19 is available at the following link:

`https://hub.docker.com/r/aspartix19/aspartix19-repo`

4 Implementation Details

When not stated otherwise, for a supported semantics we provide an ASP-encoding such that when combined with an AF in the `apx` format the answer-sets of the program are in a one-to-one correspondence with the extensions of the AF. Given an answer-set of such an encoding the corresponding extension is given by the `in(·)` predicate, i.e., an argument `a` is in the extensions iff `in(a)` is in the answer-set. With such an encoding we can exploit a standard ASP-solver to: compute some extension (SE) by computing an answer-set; enumerate all extensions (EE)

[1] `https://potassco.org/`

by enumerating all answer-sets; decide credulous acceptance (DC) of an argument a by adding the constraint `:- in(a)` to the program and testing whether the program is satisfiable, i.e., a is credulously accepted if there is at least one answer set; and decide skeptical acceptance (DS) of an argument a by adding the constraint `:- not in(a)` to the program and testing whether the program is unsatisfiable, i.e., a is skeptically accepted if there is no answer set;.

For the implementation of some semantics and reasoning task we deviate from the above described standard way of ASPARTIX. In the following we briefly describe these modifications:

For credulous and skeptical semantics with complete, preferred, grounded, and ideal semantics we do not need to consider the whole framework but only those arguments that have a directed path to the query argument (notice that this does not hold true for stable, semi-stable and stage semantics). That is, we perform pre-processing on the given AF that removes arguments without a directed path to the queried argument before starting the reasoning with ASP-solver. We do so for the reasoning tasks DC-CO (DC-PR), DC-ID, and DS-PR.

For computing the ideal extension (SE-ID, EE-ID) we follow a two-shot strategy. That is, we first use an encoding for complete semantics and the brave reasoning mode of clingo to compute all arguments that are credulously accepted/attacked w.r.t. preferred semantics. Second, we use the outcome of the first call together with an encoding that computes a fixed-point which corresponds to the ideal extension. For reasoning with ideal semantics (DC-ID, DS-ID) we use an encoding for ideal sets and perform credulous reasoning on ideal sets as described in the first paragraph of this section.

Clingo provides an option to add user-specific domain heuristic to the ASP program which in particular allow to select the answer-sets that are subset-maximal/minimal w.r.t. a specified predicate. We use such heuristics for preferred semantics (EE-PR, SE-PR) by using an encoding for complete semantics and then identifying the subset-maximal answer-sets w.r.t. the `in(·)` predicate. Moreover, we use domain heuristics to compute the subset-maximal ranges[2] of complete and conflict-free sets, which we exploited for computing some semi-stable (SE-SST) or stage extension (SE-STG). However, the domain heuristics only return one witnessing answer-set for each maxima and thus this technique is not directly applicable to the corresponding enumerations tasks (we would miss some extensions if several extensions have the same range).

Semi-stable extensions correspond to those complete labellings for which the set of undecided arguments is subset-minimal. For enumerating semi-stable extensions (EE-SST), multiple answer-sets possessing the same subset-minimal set of undecided arguments can exist. In our approach, we utilize an encoding for complete semantics extended by an `undec(·)` predicate and process the answer-sets. We check whether models without `undec(·)` predicate have been computed; in that case, semi-stable extensions coincide with stable extensions. In the other case, we compute all subset-minimal sets among all undecided sets using the set class in python and return the corresponding models.

For enumerating stage extensions (EE-STG) we use a multi-shot strategy. First we use domain heuristic to compute the maximal ranges w.r.t. naive semantics[3]. Second, for each of the maximal ranges we start another ASP-encoding which computes conflict-free sets with exactly that range (this is equivalent to computing stable extension of a restricted framework). Each of these extensions corresponds to a different stage extension of the AF.

For reasoning with semi-stable and stage semantics (DC-SST, DS-SST, DC-STG, DS-STG) we use a multi-shot strategy similar to that for enumerating the stage extensions. First we use domain heuristics to compute the maximal ranges w.r.t. complete and naive semantics.

[2]The range of a set of arguments S is the set of arguments that are either contained in S or attacked by an argument in S.

[3]Naive sets are subset-maximal conflict-free sets and as each range maximal conflict-free set is also subset-maximal it is sufficient to only consider naive sets.

In the second step we iterate over these ranges and perform skeptical, credulous respectively, reasoning over complete extensions, conflict-free sets respectively, with the given range. For skeptical acceptance, we answer negatively as soon as a counterexample to a positive answer is found when iterating the extensions; otherwise, after processing all maximal ranges we answer with YES. Analogously, for credulous acceptance, we check in each iteration whether we can report a positive answer; otherwise, after processing all maximal ranges, we return NO.

Acknowledgments

This work has been funded by the Austrian Science Fund (FWF): P30168-N31, W1255-N23, and I2854.

References

[1] Phan Minh Dung. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence*, 77(2):321–358, 1995.

[2] Wolfgang Dvořák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran. Making use of advances in answer-set programming for abstract argumentation systems. In Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu Umeda, and Armin Wolf, editors, *Proc. INAP, Revised Selected Papers*, volume 7773 of *Lecture Notes in Artificial Intelligence*, pages 114–133. Springer, 2013.

[3] Wolfgang Dvořák, Sarah Alice Gaggl, Thomas Linsbichler, and Johannes Peter Wallner. Reduction-based approaches to implement Modgil's extended argumentation frameworks. In Thomas Eiter, Hannes Strass, Miroslaw Truszczynski, and Stefan Woltran, editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2015.

[4] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, 1(2):147–177, 2010.

[5] Wolfgang Faber, Mauro Vallati, Federico Cerutti, and Massimiliano Giacomin. Enumerating preferred extensions using ASP domain heuristics: The asprmin solver. In Sanjay Modgil, Katarzyna Budzynska, and John Lawrence, editors, *Proc. COMMA*, volume 305 of *Frontiers in Artificial Intelligence and Applications*, pages 459–460. IOS Press, 2018.

[6] Sarah Alice Gaggl, Norbert Manthey, Alessandro Ronca, Johannes Peter Wallner, and Stefan Woltran. Improved answer-set programming encodings for abstract argumentation. *Theory and Practice of Logic Programming*, 15(4-5):434–448, 2015.

[7] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Torsten Schaub, and Philipp Wanko Sven Thiele. Potassco guide version 2.2.0. https://github.com/potassco/guide/releases/tag/v2.2.0, 2019.

[8] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.

[9] Martin Gebser, Benjamin Kaufmann, Javier Romero, Ramón Otero, Torsten Schaub, and Philipp Wanko. Domain-specific heuristics in answer set programming. In Marie desJardins and Michael L. Littman, editors, *Proc. AAAI*. AAAI Press, 2013.

[10] Alessandro Ronca, Johannes Peter Wallner, and Stefan Woltran. ASPARTIX-V: utilizing improved ASP encodings. http://argumentationcompetition.org/2015/pdf/paper_11.pdf, 2015.

[11] Chiaki Sakama and Tjitze Rienstra. Representing argumentation frameworks in answer set programming. *Fundamenta Informaticae*, 155(3):261–292, 2017.