Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)
Sister Conferences Best Papers Track

A Formal Approach for Cautious Reasoning in Answer Set Programming (Extended Abstract)[*]

Giovanni Amendola[1], **Carmine Dodaro**[1,†] and **Marco Maratea**[2]
[1]DeMaCS, University of Calabria, Italy
[2]DIBRIS, University of Genoa, Italy
{amendola, dodaro}@mat.unical.it, marco@dibris.unige.it

Abstract

The issue of describing in a formal way solving algorithms in various fields such as Propositional Satisfiability (SAT), Quantified SAT, Satisfiability Modulo Theories, Answer Set Programming (ASP), and Constraint ASP, has been relatively recently solved employing abstract solvers. In this paper we deal with cautious reasoning tasks in ASP, and design, implement and test novel abstract solutions, borrowed from backbone computation in SAT. By employing abstract solvers, we also formally show that the algorithms for solving cautious reasoning tasks in ASP are strongly related to those for computing backbones of Boolean formulas. Some of the new solutions have been implemented in the ASP solver WASP, and tested.

1 Introduction

Abstract solvers are a method to formally analyse solving algorithms. In this methodology, the states of a computation are represented as nodes of a graph, the solving techniques as edges between such nodes, the solving process as a path in the graph, and formal properties of the algorithms are reduced to related graph properties. This framework enjoys some advantages w.r.t. traditional ways such as pseudo-code-based descriptions, e.g., being based on formal and well-known, yet simple, mathematical objects like graphs, which helps (i) comparing solving algorithms by means of comparison of their related graphs, (ii) mixing techniques in different algorithms in order to design novel solving solutions, by means of mixing arcs in the related graphs, and (iii) stating and proving formal properties of the solving algorithms, by means of reachability within the related graphs. Abstract solvers methodology was introduced in 2006 for Propositional Satisfiability (SAT) and Satisfiability Modulo Theories (SMT) [Nieuwenhuis *et al.*, 2006], and was the basis for the design and implementation of the BARCELOGIC SMT solver awarded as winner of some tracks at the 2005 SMT Competition. Then, it proved to be useful in various fields such as Quantified SAT [Brochenin and Maratea, 2015b], Answer Set Programming [Lierler, 2011; Lierler and Truszczynski, 2011; Brochenin *et al.*, 2014], and Constraint ASP [Lierler, 2014], for analysing current algorithms, proposing and/or implementing new solvers based on the analysis.

In ASP, abstract solvers have been so far mainly applied to ASP solvers for brave reasoning tasks where, given an input query and a knowledge base, answers are witnessed by some stable model [Gelfond and Lifschitz, 1988], or answer set.

However, in ASP, also cautious reasoning has been deeply studied in the literature: differently from the previous task, answers here must be witnessed by all stable models. This task has found a significant number of interesting applications as well, including consistent query answering [Arenas *et al.*, 2003; Manna *et al.*, 2013], data integration [Eiter, 2005], multi-context systems [Brewka and Eiter, 2007], and ontology-based reasoning [Eiter *et al.*, 2008]. Two of the most well-known ASP solvers, i.e., DLV [Leone *et al.*, 2006] and CLASP [Gebser *et al.*, 2012], have been extended for computing cautious consequences of ASP programs. Several algorithms for cautious reasoning in ASP, including those implemented in DLV and CLASP, borrowed from the backbone computation of Boolean formulas [Janota *et al.*, 2015], have been presented in a unified, high-level view in [Alviano *et al.*, 2014]. Moreover, some of these techniques have been implemented on top of the ASP solver WASP [Alviano *et al.*, 2019], and further strategies have been analyzed through abstract solvers in [Brochenin and Maratea, 2015a].

In this paper we design, implement and test novel abstract solutions for cautious reasoning tasks in ASP. We show how to improve the current abstract solvers with further techniques borrowed from backbone computation in SAT, in order to design new solving algorithms. In particular, we import a technique called "chunk", which generalizes previous techniques by testing a set of atoms simultaneously, and core-based algorithms, which can be considered either a solution per se, or a way for pruning the set of atoms to be considered, given that they can not guarantee completeness. By doing so, we also formally show, through a uniform treatment, a strong relation among algorithms for solving cautious reasoning tasks in ASP and those for computing backbones of Boolean formulas. Finally, we implement some of the new solutions in the solver WASP and we show that their performances are comparable to state-of-the-art solutions. This paper is a summary of the best technical ICLP 2019 paper [Amendola *et al.*, 2019].

[*]Summary of the ICLP 2019 paper [Amendola *et al.*, 2019].
[†]Contact Author

2 Preliminaries

Here, we recall basics on Answer Set Programming (ASP); Boolean logic formulas; and abstract solvers framework.

2.1 Boolean Formulas and Answer Set Programs

We define ASP programs and Conjunctive Normal Form (CNF) formulas so as to underline similarities, thus to compare algorithms on CNF formulas and ASP programs.

Syntax. Let Σ be a propositional signature. An element $a \in \Sigma$ is called *atom* or *positive literal*. The negation of an atom a, i.e., $\neg a$, is called *negative literal*. Given a literal l, we define $|l| = a$, if $l = a$ or $l = \neg a$, for some $a \in \Sigma$. For a set of atoms $X \subseteq \Sigma$, a *literal relative to* X is a literal l s.t. $|l| \in X$, and $lit(X)$ is the set of all literals relative to X. We set $\bar{l} = a$, if $l = \neg a$, and $\bar{l} = \neg a$, if $l = a$. A *clause* is a set of literals (seen as a disjunction). A *CNF formula* is a set of clauses (seen as a conjunction). Given a set of literals M, we denote by M^+ (resp., M^-) the set of positive (resp., negative) literals of M, and by $\overline{M}$ the set $\{\bar{l} \mid l \in M\}$. M is *consistent* if $l \in M$ implies $\bar{l} \notin M$. A *rule* is a pair (A, B), written $A \leftarrow B$, where B is a set of literals and A is an atom or $\emptyset$. A *program* is a finite set of rules. Given a set of literals M, a program Π and a CNF formula Φ, we denote by $atoms(M)$, $atoms(\Pi)$ and $atoms(\Phi)$ respectively the set of atoms occurring in M, Π, and Φ. Note that in Boolean logic $\neg$ is classical negation, while in ASP it is *negation by default*.

Semantics. Let X be a set of atoms, Φ a CNF formula, and Π a program. An *assignment* to X is a total mapping from X to $\{\bot, \top\}$. A consistent set M of literals is an assignment to $atoms(M)$ s.t. $a \in M$ iff a is mapped to $\top$, and $\neg a \in M$ iff a is mapped to $\bot$. A *model* of Φ is an assignment M to $atoms(\Phi)$ s.t. for each clause $C \in \Phi$, $M \cap C \neq \emptyset$. A *model* of Π is an assignment M to $atoms(\Pi)$ s.t. for each rule $(A, B) \in \Pi$, $A \cap M \neq \emptyset$ or $B \not\subseteq M$. Let $M(\Phi)$ be the set of all models of Φ. The *reduct* Π^X of Π w.r.t. to X is obtained from Π by deleting each rule $A \leftarrow B^+ \cup B^-$ s.t. $X \cap atoms(B^-) \neq \emptyset$ and replacing each remaining rule with $A \leftarrow B^+$. An *answer set* of Π is a model M s.t. M^+ is minimal among the M_0^+ s.t. M_0 is a model of Π^{M^+}. Let $AS(\Pi)$ be the set of all answer sets of Π. We define $backbone(\Phi) = \bigcap_{M \in M(\Phi)} M^+$and $cautious(\Pi) = \bigcap_{M \in AS(\Pi)} M^+$.

2.2 Abstract Solvers

Abstract solvers are graphs that represent the status of the computation, and how it changes in response to an application of a technique in a search for a solution with certain properties, e.g., the satisfiability of a formula. We present the notion of *states*, *transition rules*, and *abstract solver graphs*.

States. Let X be a set of atoms. We denote by $\mathcal{A}(X)$ the set of *action relative to* X. A *record* relative to X is a string L from $lit(X)$ without repetitions. We may view a record as the set containing all its elements stripped from their annotations. For example, we may view $\neg ab$ as $\{\neg a, b\}$, and hence as the assignment that maps a to $\bot$ and b to $\top$. Let Π be a program, and $X = atoms(\Pi)$. The set of the *states relative to* Π is the union of: *core* states relative to Π: $L_{O,U,A,P}$ where $O, U \subseteq X$, $A \in \mathcal{A}(X)$, and P is a program depending by Π, O, U, A (e.g., $\neg ab_{\{a,b\},\emptyset,A,P}$, $\emptyset_{\{a\},\{b\},A,P}$, $\neg a \neg b \neg c_{\emptyset,\emptyset,A,P}$); *control* states relative to X: $Cont(O, U)$ where $O, U \subseteq X$ (e.g., $Cont(\{a, b\}, \{a\})$, $Cont(\{a, b, c\}, \emptyset)$, $Cont(\emptyset, \emptyset)$); *terminal* states relative to X: $Ok(W)$ where $W \subseteq X$ (e.g., $Ok(\{a, b, c\})$, $Ok(\emptyset)$). Intuitively, these states represent computation steps of the algorithms that search for assignments with certain properties, in our case backbone or cautious consequences. Core states $L_{O,U,A,P}$ and control states $Cont(O, U)$ represent all the intermediate steps of the computation, where L is the current state of the computation of a model; O is the current over-approximation of the solution; U is the current under-approximation of the solution; and A is the action currently carried out. Intuitively, a core state represents the computation within a call to an ASP oracle, i.e., an ASP solver, while a control state controls the computation between different calls to ASP oracles, depending on over- and under-approximation. Terminal states represent the end of the computation.

Transition Rules. *Transition rules* are of the form:

$$ruleName \quad S \implies S' \quad \text{if} \{ \; conditions$$

where $ruleName$ is the name of the rule; $S \implies S'$ represents a transition from the state S to the state S'; and $conditions$ is a set of conditions for the rule to be applied. We also consider a special transition rule, $Oracle$, reported in Figure 1. Intuitively, it represents an oracle call to an ASP [resp., SAT] solver by providing as result a set of literals L corresponding to the output of an ASP [resp., SAT] solver, i.e., L is an answer set of a program [resp., a model of a formula], if such an answer set [resp., model] exists, and to $lit(atoms(P))$, otherwise. Transition rules will be organized into $Return$ rules, dealing with the outcome of an oracle call, or the application of a given technique, depending on the status of the set of literals L returned; and $Control$ rules, starting from a control state and directing the computation depending on the content of the over- and under-approximation.

Abstract Solver Graphs. Given a program Π, and a set of transition rules T, an *abstract solver graph* $G(\Pi, T) = \langle V_X, E_T \rangle$ is such that V_X is the set of all states relative to $X = atoms(\Pi)$; and $(S, S') \in E_T$ if a transition rule of the form $S \implies S'$ can be applied. Moreover, $S' \in V_X$ is *reachable from* $S \in V_X$ if there is a path from S to S'; there is a state in V_X called *initial state* (from which the computation starts, depending on the specific algorithm); a state reachable from the initial state is called *reachable state* (it represents a possible state of a computation); and *no cycle is reachable* if there is no reachable state which is reachable from itself.

Definition 1. *Given a program* Π *[a CNF formula* Φ*] and a set of transition rules T, we say that an abstract solver graph* $G(\Pi, T)$ *[*$G(\Phi, T)$*] solves cautious reasoning [backbone computation], if* (i) $G(\Pi, T)$ *[resp.,* $G(\Phi, T)$*] is finite and no cycle is reachable; and* (ii) *the unique terminal reachable state is* $Ok(cautious(\Pi))$ *[resp.,* $Ok(backbone(\Pi))$*].*

3 Abstract Solvers for Cautious Consequences

In this section, without loss of generality, we focus on the computation of cautious consequences for an ASP program.

	$Oracle$	$\emptyset_{O,U,A,P}$	$\Longrightarrow$	$L_{O,U,A,P}$	if $\{$ $L \in AS(P)$, or $AS(P) = \emptyset$ and $L = lit(atoms(P))$
Return rules	$Fail_{over}$	$L_{O,U,over,P}$	$\Longrightarrow$	$Cont(O, O)$	if $\{$ L is inconsistent
	$Fail_{under}$	$L_{O,U,under_S,P}$	$\Longrightarrow$	$Cont(O, U \cup S)$	if $\{$ L is inconsistent, and $S = \emptyset$ or $S = \{a\}$
	$Fail_{chunk}$	$L_{O,U,chunk_N,P}$	$\Longrightarrow$	$Cont(O, U \cup N)$	if $\{$ L is inconsistent
	$Find$	$L_{O,U,A,P}$	$\Longrightarrow$	$Cont(O \cap L, U)$	if $\{$ L is consistent and $L \neq \emptyset$
Control rules	$Terminal$	$Cont(O, U)$	$\Longrightarrow$	$Ok(O)$	if $\{$ $O = U$
	$OverApprox$	$Cont(O, U)$	$\Longrightarrow$	$\emptyset_{O,U,over,P}$	if $\{$ $O \neq U$
	$UnderApprox$	$Cont(O, U)$	$\Longrightarrow$	$\emptyset_{O,U,under_{\{a\}},P}$	if $\{$ $a \in O \setminus U$
	$Chunk$	$Cont(O, U)$	$\Longrightarrow$	$\emptyset_{O,U,chunk_N,P}$	if $\{$ $N \subseteq O \setminus U$ and $N \neq \emptyset$

Figure 1: Transition rules for over-approximation, under-approximation, and chunking.

In the following we assume that all abstract solvers share the same general structure. In particular, given a program Π, over-approximation O is set to $atoms(\Pi)$, while the under-approximation U is empty. Note that $U \subseteq cautious(\Pi) \subseteq O$. Iteratively either under-approximation or over-approximation are applied. When $U = O$, the set of cautious consequences, O, has been found and the computation terminates. It means that the state $Ok(O)$ is a reachable state. Hence, the full extent of states relative to X becomes useful. The unique terminal state is $Ok(W)$, where $W = cautious(\Pi)$.

3.1 Solving Techniques

In this section, we report three techniques, namely over-approximation, under-approximation, and chunking.

Over-approximation. We set $\mathcal{A}(atoms(\Pi)) = \{over\}$, and $P = \Pi \cup \{\leftarrow O\}$. The initial state is $\emptyset_{atoms(\Pi),\emptyset,over,P}$. We set $ov = \{Fail_{over}, Find, Terminal, OverApprox\}$ (see Figure 1), and $OS(\Pi) = (V_{atoms(\Pi)}, \{Oracle\} \cup ov)$. Intuitively, $Fail_{over}$ means that a call to an oracle did not find an answer set, so O is the solution. If $Find$ is triggered, instead, we go to a control state where O is updated according to the answer set found: then, if $O = U$ a solution is found through $Terminal$, otherwise the search is restarted ($L = \emptyset$) in an oracle state with $OverApprox$. Thus, in $OS(\Pi)$, the oracle is called to find answer sets that reduce the over-approximation O in the $over$ action, unless no answer set exists. If an answer set M is found, then $M \cap \overline{O} \neq \emptyset$, as $P = \Pi \cup \{\leftarrow O\}$.

Under-approximation. Let $\mathcal{A}(atoms(\Pi)) = \{under_{\emptyset}\} \cup \{under_{\{a\}} \mid a \in atoms(\Pi)\}$, let $P = \Pi \cup \{\leftarrow a\}$, if $A = under_{\{a\}}$, and $P = \Pi$, if $A = under_{\emptyset}$. The initial state is $\emptyset_{atoms(\Pi),\emptyset,under_{\emptyset},P}$. We set $un = \{Fail_{under}, Find, Terminal, UnderApprox\}$ (see Figure 1), and $US(\Pi) = (V_{atoms(\Pi)}, \{Oracle\} \cup un)$. Intuitively, $Fail_{under}$ updates over- and under-approximations in case a test on the atom a failed, and leads to a control state, while $UnderApprox$ restarts a new test if $Find$ is not applicable. In $US(\Pi)$, again, a first oracle call takes place with the action $under_{\emptyset}$, which provides first over-approximation, then calls with actions $under_{\{a\}}$, where a is the tested atom.

Chunking. In [Janota *et al.*, 2015] a more general technique for under-approximation that allows to test multiple literals at once is presented. Let $\mathcal{A}(atoms(\Pi)) = \{chunk_N \mid N \subseteq atoms(\Pi)\}$, and $P = \Pi \cup \{\leftarrow N\}$, if $A = chunk_N$. The initial state is $\emptyset_{atoms(\Pi),\emptyset,chunk_{\emptyset},P}$. Let $ch = \{Fail_{chunk}, Find, Terminal, Chunk\}$ (see Figure 1) and $CS(\Pi) = (V_{atoms(\Pi)}, \{Oracle\} \cup ch)$. In particular, $Fail_{chunk}$ updates the over- and under-approximations accordingly in case the test on the set N fails (the ASP oracle call failed, thus all literals in N must be cautious consequences), and goes to a control state. Meanwhile, $Chunk$ restarts a new ASP oracle call with a new (nonempty) set N such that $N \subseteq O \setminus U$ in case the computation must continue.

3.2 Designing New Abstract Solvers

The composition of techniques described so far can be applied to computing cautious consequences of a program, but actually is not included in any solver. This outlines another important feature of the abstract solvers methodology, i.e., its capability to design new solutions by combining techniques implemented in different solvers. We can mix techniques from Section 3.1 to compute either cautious consequences or backbones.

Theorem 1. *Let Π be a program, and $S \subseteq \{ov, un, ch\}$ s.t. $S \neq \emptyset$. Then, $(V_{atoms(\Pi)}, \{Oracle\} \cup \bigcup_{x \in S} x)$ solves cautious reasoning and backbone computation.*

Core-based Methods. We now model core-based algorithms from [Janota *et al.*, 2015] in terms of abstract solvers, in particular Algorithm 6, and apply it to the computation of cautious consequences of ASP programs. Let $\mathcal{A}(atoms(\Pi)) = \{core_N \mid N \subseteq lit(atoms(\Pi))\}$, and $P = \Pi \cup \{\leftarrow \bar{l} | l \in N\}$, if $A = core_N$. The initial state is $\emptyset_{atoms(\Pi),\emptyset,core_{\overline{atoms(\Pi)}},P}$. Moreover, given a logic program Π, we say that a set $C \subseteq lit(atoms(\Pi))$ is *a core* of Π, if $\Pi \cup \{\leftarrow \bar{l} | l \in C\}$ is incoherent. By $cores(\Pi)$ we denote the set of all cores of Π. Cores have two important properties: (1) if C is a core, then all of its supersets are also cores; (2) $p \in atoms(\Pi)$ is a cautious consequence of Π iff $\{\neg p\}$ is a core. To define an abstract solver graph, we start with the rule

$$CoreOracle \quad \emptyset_{O,U,core_N,P} \Longrightarrow L_{O,U,core_N,P}$$
$$\text{if } \{L \in AS(P), \text{ or } L \in cores(P) \text{ and } L \subseteq N.$$

It represents an oracle call to compute a set of literals L, which is a core of P and a subset of N, whenever P is incoherent; and an answer set of P, otherwise. Then, we introduce two intermediate control states, namely Pre_N and $Eval$.

Moreover, we introduce the following set of return rules:

$$\begin{array}{lll} \mathit{Fail}^1_{pre} & L_{O,U,core_N,P} \Longrightarrow \mathit{Pre}_{N \setminus L}(O, U \cup \bar{L}) \\ & \text{if} \{ \ AS(P) = \emptyset \text{ and } |L| = 1 \\ \mathit{Fail}^2_{pre} & L_{O,U,core_N,P} \Longrightarrow \mathit{Pre}_{N \setminus L}(O, U) \\ & \text{if} \{ \ AS(P) = \emptyset \text{ and } |L| > 1 \\ \mathit{Find}_{pre} & L_{O,U,core_N,P} \Longrightarrow \mathit{Eval}(O \cap L, U) \\ & \text{if} \{ \ AS(P) \neq \emptyset \text{ and } \ L \neq \emptyset \end{array}$$

and the following set of control rules:

$$\begin{array}{lllll} \mathit{Main} & \mathit{Pre}_N(O,U) & \Longrightarrow \mathit{Cont}(O,U) & \text{if} \{ & N = \emptyset \\ \mathit{Continue} & \mathit{Pre}_N(O,U) & \Longrightarrow \emptyset_{O,U,core_N,P} & \text{if} \{ & N \neq \emptyset \\ \mathit{NewSet} & \mathit{Eval}(O,U) & \Longrightarrow \emptyset_{O,U,core_{\overline{O}},P} & \text{if} \{ & O \neq U \\ \mathit{Final} & \mathit{Eval}(O,U) & \Longrightarrow \mathit{Ok}(O) & \text{if} \{ & O = U \end{array}$$

Let in be the set of rules defined above, and let $FS(\Pi) = (V_{atoms(\Pi)}, in)$, which represents Algorithm 6 in [Janota *et al.*, 2015]. In particular, Pre_N is reached in case of inconsistency, where N is the set of literals that may be used for the potential upcoming *core* action; and *Eval* is reached in case of consistency. From an outermost state *Eval*, a new *core* is started with *NewSet*, whenever there is a gap between over- and under-approximation; otherwise, *Final* leads to the terminal state. Fail^1_{pre} and Fail^2_{pre} lead to the intermediate type of control state, Pre_N, that can either restart a *core* action with *Continue*, or continue with the *Main* rule.

Theorem 2. *Let* Π *be a program. Then, the only reachable terminal states are either* $\mathit{Cont}(O,U)$ *or* $\mathit{Ok}(O)$*, for some* $O, U \subseteq atoms(\Pi)$*:* (i) *if* $\mathit{Ok}(O)$ *is reachable, then* $FS(\Pi)$ *solves cautious reasoning;* (ii) *if* $\mathit{Cont}(O,U)$ *is reachable, then* $U \subseteq cautious(\Pi) \subseteq O$.

Chunking and core-based methods can be combined using our methodology to abstract Algorithm 7 from [Janota *et al.*, 2015]. Such a combination will be used in the experiments.

4 Experimental Analysis

The abstract solvers reported in this paper have been used for implementing several algorithms in WASP [Alviano *et al.*, 2015; Alviano *et al.*, 2019], resulting in two new versions of WASP, namely WASP-CHUNK, i.e., WASP running the algorithm based on chunking, and WASP-CB, i.e., WASP running the algorithm based on cores. Both versions can limit the size of the chunk, reported in the following using the suffixes -2 and -20% representing that the size of the chunk is set to 2 and to the 20% of the number of atoms, respectively.

The performance of these versions of WASP was measured on the benchmarks considered in [Alviano *et al.*, 2018], which includes *(i)* all the 193 instances from the latest ASP Competitions [Gebser *et al.*, 2017] involving non-ground queries; *(ii)* 115 instances of ASP Competitions classified as *easy*, that is, those for which a stable model is found within 20 seconds of computation by mainstream ASP systems.

As a reference to the state of the art, we report the performance of CLASP [Gebser *et al.*, 2012], which implements algorithm OR (i.e., *over-approximation*), and the best performing algorithms implemented by WASP [Alviano *et al.*, 2014; Alviano *et al.*, 2018], namely OR, ICT (i.e., *under-approximation*), OPT, and CM. We refer the reader to [Alviano *et al.*, 2014; Alviano *et al.*, 2018; Di Rosa *et al.*, 2008], for a detailed description of such algorithms.

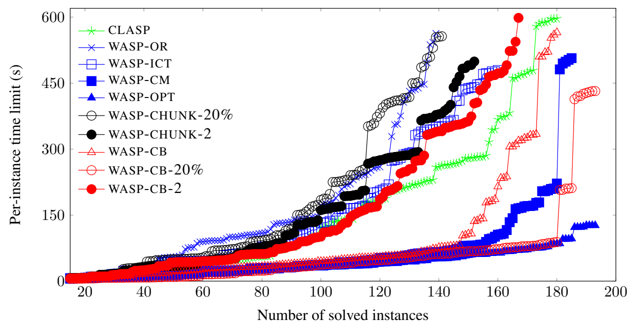

Figure 2: Benchmark *(i)*: Performance comparison on non-ground queries in ASP Competitions.

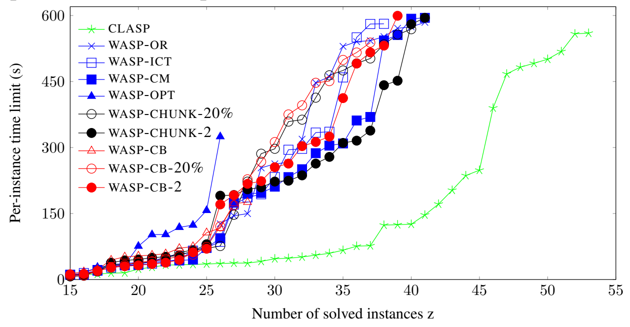

Figure 3: Benchmark *(ii)*: Performance comparison on computation of cautious consequences for *easy* instances of ASP Competitions.

Results are shown in the cactus plots of Figure 2 and Figure 3, where for each algorithm the number of solved instances in a given time is reported, producing an aggregated view of its overall performance.

Concerning benchmark *(i)*, we observe that WASP cannot reach the performance of CLASP on the execution of algorithm OR, and indeed CLASP solved 41 instances more than WASP-OR. However, a better performance is obtained by WASP-CB-20% and by WASP-OPT, which actually solve 13 instances more than CLASP. Moreover, we observe that even a small size of the chunk may influence the performance of the algorithms. Indeed, WASP-CB solves 13 instances more than WASP-CB-2. Finally, we observe that WASP-CHUNK-20% and WASP-CHUNK-2 are not competitive to algorithms based on cores.

Concerning benchmark *(ii)*, we observe that CLASP is the best performing solver solving 53 instances. If we focus on WASP, the best performance is obtained by WASP-CHUNK-2, WASP-OR, WASP-CM, and WASP-CHUNK-20% which are able to solve 41, 41, 41, and 40 instances, respectively. Moreover, WASP-CB cannot reach the same performance on this benchmark, solving only 25 instances, whereas WASP-CB-20% and WASP-CB-2 solve 37 and 39 instances, respectively.

5 Conclusion

In this paper we modeled through abstract solvers advanced techniques for solving cautious reasoning tasks in ASP. We also designed new solving procedures, and implemented them in WASP. The results of the experiments showed that our implementation is competitive to state-of-the-art approaches.

References

[Alviano *et al.*, 2014] Mario Alviano, Carmine Dodaro, and Francesco Ricca. Anytime computation of cautious consequences in answer set programming. *Theory Pract. Log. Program.*, 14(4-5):755–770, 2014.

[Alviano *et al.*, 2015] Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca. Advances in WASP. In *Proc. of LPNMR*, volume 9345 of *Lecture Notes in Computer Science*, pages 40–54. Springer, 2015.

[Alviano *et al.*, 2018] Mario Alviano, Carmine Dodaro, Matti Järvisalo, Marco Maratea, and Alessandro Previti. Cautious reasoning in ASP via minimal models and unsatisfiable cores. *Theory Pract. Log. Program.*, 18(3-4):319–336, 2018.

[Alviano *et al.*, 2019] Mario Alviano, Giovanni Amendola, Carmine Dodaro, Nicola Leone, Marco Maratea, and Francesco Ricca. Evaluation of disjunctive programs in WASP. In *Proc. of LPNMR*, volume 11481 of *Lecture Notes in Computer Science*, pages 241–255. Springer, 2019.

[Amendola *et al.*, 2019] Giovanni Amendola, Carmine Dodaro, and Marco Maratea. Abstract solvers for computing cautious consequences of ASP programs. *Theory Pract. Log. Program.*, 19(5-6):740–756, 2019.

[Arenas *et al.*, 2003] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Answer sets for consistent query answering in inconsistent databases. *Theory Pract. Log. Program.*, 3(4-5):393–424, 2003.

[Brewka and Eiter, 2007] Gerhard Brewka and Thomas Eiter. Equilibria in heterogeneous nonmonotonic multi-context systems. In *Proc. of AAAI*, pages 385–390. AAAI Press, 2007.

[Brochenin and Maratea, 2015a] Rémi Brochenin and Marco Maratea. Abstract answer set solvers for cautious reasoning. In *Proc. of ICLP*, volume 1433 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

[Brochenin and Maratea, 2015b] Rémi Brochenin and Marco Maratea. Abstract solvers for quantified boolean formulas and their applications. In *Proc. of AI*IA*, volume 9336 of *Lecture Notes in Computer Science*, pages 205–217. Springer, 2015.

[Brochenin *et al.*, 2014] Remì Brochenin, Yuliya Lierler, and Marco Maratea. Abstract disjunctive answer set solvers. In *Proceedings of ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 165–170. IOS Press, 2014.

[Di Rosa *et al.*, 2008] Emanuele Di Rosa, Enrico Giunchiglia, and Marco Maratea. A new approach for solving satisfiability problems with qualitative preferences. In *Proc. of ECAI*, volume 178 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2008.

[Eiter *et al.*, 2008] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. *Artif. Intell.*, 172(12-13):1495–1539, 2008.

[Eiter, 2005] Thomas Eiter. Data integration and answer set programming. In *Proc. of LPNMR*, volume 3662 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2005.

[Gebser *et al.*, 2012] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187:52–89, 2012.

[Gebser *et al.*, 2017] Martin Gebser, Marco Maratea, and Francesco Ricca. The sixth answer set programming competition. *Journal of Artificial Intelligence Research*, 60:41–95, 2017.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of ICLP/SLP*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

[Janota *et al.*, 2015] Mikolás Janota, Inês Lynce, and Joao Marques-Silva. Algorithms for computing backbones of propositional formulae. *AI Comm.*, 28(2):161–177, 2015.

[Leone *et al.*, 2006] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM TOCL*, 7(3):499–562, 2006.

[Lierler and Truszczynski, 2011] Yuliya Lierler and Miroslaw Truszczynski. Transition systems for model generators — a unifying approach. *Theory Pract. Log. Program.*, 11(4-5):629–646, 2011.

[Lierler, 2011] Yuliya Lierler. Abstract answer set solvers with backjumping and learning. *Theory Pract. Log. Program.*, 11:135–169, 2011.

[Lierler, 2014] Yuliya Lierler. Relating constraint answer set programming languages and algorithms. *Artif. Intell.*, 207:1–22, 2014.

[Manna *et al.*, 2013] Marco Manna, Francesco Ricca, and Giorgio Terracina. Consistent query answering via ASP from different perspectives: Theory and practice. *Theory Pract. Log. Program.*, 13(2):227–252, 2013.

[Nieuwenhuis *et al.*, 2006] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.