A new approach to conceive ASP solvers

Tarek Khaled
Supervisor : Belaïd Benhamou

Aix Marseille Université, LIS, Marseille, France.
{tarek.khaled,belaid.benhamou}@univ-amu.fr

Abstract. The Answer set programming (ASP) is a non-monotonic declarative programming paradigm that is widely used for the formulation of problems in artificial intelligence. The ASP paradigm provides also a general framework for the resolution of decision and optimization problems. The idea behind ASP is to represent a problem as a logic program and solve that problem by computing stable models. In our work, we propose a new method for searching stable models of logical programs. This method is based on a relatively new semantic that had not been exploited yet. This semantic captures and extends that one of the stable models. The method performs a DPLL enumerative process only on a restricted set of literals called the strong back-door (STB). This method has the advantage to use a Horn clause representation having the same size as the input logic program and has constant spatial complexity. It avoids the heaviness induced by the loop management from which suffer most of the ASP solvers based on the Clark completion.

1 Introduction

The ASP is increasingly used, and this is due to the availability of several efficient ASP solvers like $smodels$ [1], $Clasp$ [2] and those based on SAT solvers like $ASSAT$ [3] and $Cmodels$ [4]. The natural way to use ASP is to express a problem as a logic program with default negation. To get a concise expression of the problem, the logic program is expressed in First Order Logic (FOL). Grounders like $gringo$ [5] and $lparse$ [6] are designed to compute an equivalent propositional logic program called the ground program. ASP solvers search solutions for the original problem by computing models of the ground logic program [7]. The resulting models are referred to as stable or answer set models.

To give a signification for a logic program, several semantics were proposed. Since the Clark completion [8], many other semantics such as the well-founded semantic [9], the stable models [10, 11], the stable class [12] and the stratified default theory [13] have been introduced. Other works established the relationship between the semantics of logic programs and those of non-monotonic logics [14, 15]. All these semantics try particularly to give a sense to the negation as failure (default negation) appearing in the rules of the program.

We based our approach to compute stable models on the semantic introduced in [16]. This semantics offers many advantages, among them the fact that logic programs are represented by a set of Horn clauses that has the same size as the input propositional program. The Horn set representation allows to get a new resolution method

having good complexity proprieties. This representation offers several benefit exploited in practice by our method. This semantic also extends the semantic of the stable models, and allows an easy characterization of the stable models.

The proposed method avoids the heaviness that could be induced by the loop management performed in ASP solvers that are based on the Clark completion [8]. The method is a boolean enumerative process adapted for the ASP paradigm according to the used semantic and its features. It has the advantage to perform the enumerative process only on a restriction of the set of literals called here the strong back-door (STB) [17] of the logic program. This method computes the different extensions of the source logic program from which we can generate all the stable models. In case no stable model is found, our method could also generate extra-models that extend the stable models.

2 State-of-the-art

A logic program π is a set of rules of the form *r:head(r)* $\leftarrow$ *body(r)*. In general, the rules are given in First Order Logic. Grounders are used to transform the initial logic program into a ground (propositional logic) program $Ground(\pi)$. In the following, we write only π to mean $Ground(\pi)$. There are different classes of logic programs. They differ by the presence or the absence of the classical negation and the negation as failure in the rules of the program. A rule in a logic program is of the form : $r = A_0 \leftarrow A_1, A_2, ..., A_m, not\, A_{m+1}, ..., not\, A_n, (0 \leq m < n)$ where $A_{i \in \{0...n\}}$ is an atom and not the symbol expressing the negation as failure. The positive body of r is $body^+(r) = \{A_1, A_2, ..., A_m\}$ and the negative is $body^-(r) = \{A_{m+1}, ..., A_n\}$. The intuitive meaning of the rule r is the following: A_0 must be true if we prove all the atoms of $body^+(r)$ and at the same time no atom of $body^-(r)$ had been proven.

The reduct of a program π with respect to a given set of atoms X is the positive program π^X obtained from π by deleting each rule containing an atom $not\, A_i$ in its negative body such that $A_i \in X$ and all the atoms $not\, A_j$ such that $A_j \notin X$ in the negative bodies of the other rules. A set X of atoms is a stable model of π iff X is identical to the minimal Herbrand model of the reduct π^X obtained from π when considering the set of atoms X. This model is also called the canonical model of π^X, it is denoted by $Cn(\pi^X)$. Formally, a set X of atoms is a stable model of π if and only if $X = Cn(\pi^X)$.

In practice, several ASP solvers are also based on the Clark completion[8]. It is well known that every stable model of π is a model of its completion but the converse apply only if the program is tight [18] (without loops). In order to establish the equivalence between the semantic of a logic program and its completion, loops formulas have to be added to the completion [3]. But, the number of loop formulas that Clark based solvers have to manage could be exponential in the size of the considered program and then their processing will be impractical[19]. Consequently, the spatial complexity of ASP solvers adopting this approach could vary exponentially in the worst case.

In our study, we use a Horn clausal representation having the same size as the considered logical program. The proposed method operates on this Horn form with a constant spatial complexity. Our method is based on the semantic introduced in [16]. This semantic consists in computing the extensions of the set of Horn clauses representing the given logic program. Intuitively, for a given logic program, the method

computes its extensions by adding to the Horn clause representation maximal consistent sets of literals ($not\, A_i$) of the so called strong back-door set (STB). The stable models of the logic program could be deduced from some extensions that verifying a simple discriminant condition [16]. The set STB is formed by the literals of the form $not\, A_i$ that appear in the input logic program π. Formally, it is defined by $STB = \{not\, A_i : \exists r \in \pi, A_i \in body^-(r)\} \subseteq nV$.

More precisely, the used semantic is based on a classical propositional language L having two types of atoms : a subset of classical atoms $V = \{A_i : A_i \in L\}$ and an other subset $nV = \{not\, A_i : not\, A_i \in L\}$. For each atom $A_i \in V$, there is a corresponding atom $not\, A_i \in nV$ designating the negation as failure of A_i. This semantic provides a connection between these two types of atoms. This connection is expressed by the addition to the propositional language L of an axiom expressing the mutual exclusion between each literal $A_i \in V$ and its corresponding negative literal $not\, A_i \in nV$. A logic program is expressed in the propositional language L by a set of Horn clauses $CR = \{ \bigcup_{r \in \pi} (A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\, A_{m+1}, ..., \neg not\, A_n), 0 \leq m < n\}$ representing all the rules of the logic program to which we add the set of mutual exclusion clauses $ME = \{(\neg A_i \vee \neg not\, A_i) : A_i \in V\}$. The complete representation of the logic program π in the propositional language L is given as follows: $\mathrm{L}(\pi) = \{ \bigcup_{\mathrm{r} \in \pi} (\mathrm{A}_0 \vee \neg \mathrm{A}_1 \vee, ..., \neg \mathrm{A}_\mathrm{m} \vee \neg \mathrm{not}\, \mathrm{A}_{\mathrm{m}+1}, ..., \neg \mathrm{not}\, \mathrm{A}_\mathrm{n}) \bigcup_{\mathrm{A_i} \in \mathrm{V}} (\neg \mathrm{A_i} \vee \neg \mathrm{not}\, \mathrm{A_i})\}$.

Given a program π and its strong back-door set STB. An extension of $L(\pi)$ with respect to the set STB (or simply an extension of the pair $(L(\pi), STB)$) is the set of consistent clauses derived from $L(\pi)$ when adding a maximal set of literals $not\, A_i \in STB$. That is, if any other literal $not\, A_i \in STB$ is added to the extension, the resulting set of clauses becomes inconsistent. Formally:

Definition 1 *Let $L(\pi)$ be the Horn CNF encoding of a logic program π, STB its strong back-door and $S' \subseteq STB$. The set $\mathrm{E} = \mathrm{L}(\pi) \cup \mathrm{S}'$ of clauses is then an extension of $(\mathrm{L}(\pi), \mathrm{STB})$ if the following conditions hold:*

1. E *is consistent,*
2. $\forall not\, A_i \in STB - S', E \cup \{not\, A_i\}$ *is inconsistent.*

Example 1 *Consider the logic program :*

$$\pi = \{\, a \leftarrow c, not\, b \qquad b \leftarrow a \qquad c \leftarrow not\, d \qquad a \leftarrow \,\}$$

The Horn clausal representation of the logic program π is formed by the set $L(\pi) = CR \cup ME$ where $CR = \{a \vee \neg c \vee \neg not\, b, b \vee \neg a, c \vee \neg not\, d, a\}$, $ME = \{\neg a \vee \neg not\, a, \neg b \vee \neg not\, b, \neg c \vee \neg not\, c, \neg d \vee \neg not\, d\}$ and its strong back-door is $STB = \{not\, b, not\, d\}$. We can see that $(L(\pi), STB)$ admit one extension $E = L(\pi) \cup \{not\, d\}$. Indeed, E is maximally consistent with respect to the strong back-door set STB. That is, if for instance we add $not\, b$ to the extension E, the resulting set of clauses becomes inconsistent.

It is shown in [16] that each stable model of a logic program π is represented by an extension E of its logic form $L(\pi)$ satisfying the discriminant condition ($\forall \mathrm{A_i} \in \mathrm{V}$, $\mathrm{E} \models \neg \mathrm{not}\, \mathrm{A_i} \Rightarrow \mathrm{E} \models \mathrm{A_i}$). The extensions of $L(\pi)$ that do not satisfy the discriminant condition do not correspond to any stable model. These are what we call extra-extensions, they identify extra-models (or extended models) representing a kind of extension to the semantic of stable models [10]. That is, a program that has no stable

model could have extra-models. Since we have a Horn clause representation, the characterization of the stable models and the verification of the discriminant condition are done by unit resolution. The main theoretical results are given in [16] and recalled in the following:

Theorem 1 *If E is an extension of $(L(\pi), STB)$, that verify the discriminant condition: $\forall A_i \in V, E \models \neg not\, A_i \Rightarrow E \models A_i$, then $\mathrm{X} = \{\mathrm{A_i} : \mathrm{E} \models \mathrm{A_i}\}$ is a stable model of π.*

Example 2 *The extension $E = L(\pi) \cup \{not\, d\}$ found in Example 1 satisfies the discriminant condition. The stable model $M = \{a, b, c\}$ is deduced from E by unit resolution.*

3 Description of the new method

We describe here the new search method for stable models that is based on the semantic summarized previously [16]. For a given logic program π, this method computes all the extensions of $(L(\pi), STB)$ from which the stable models are deduced by unit resolution. Intuitively, the search of the extensions of $(L(\pi), STB)$ is done by the progressive addition of literals $not\, A_i$ of the STB to $L(\pi)$ and checking the consistency of the obtained set at each node. The representation of the ME set in $L(\pi)$ could be omitted, since it can be implemented like an inference rule without the need to memorize it. The method that we propose takes then as input a Horn clause form $L(\pi)$ having the same size as the input program π. If we focus only on stable models, we just have to look after the extensions verifying the discriminant condition. In other words, we make cuts in the search tree to remove the extra-extensions which don't verify that condition.

The enumeration process builds incrementally an extension by alternating in the search tree between deterministic nodes corresponding to the unit propagations and non deterministic nodes that are the choice points. The choice points are defined by the affectation of truth values (true or false) to literals of the strong back-door set STB. In the case of our method, the enumeration is done only on the subset of literals forming the strong back-door $STB = \{not\, A_i : \exists r \in \pi, A_i \in body^-(r)\}$. The non-deterministic treatment of a choice point corresponding to a strong back-door literal $not\, A_j$ is done by first its assignment to the value $true$ to favor the current extension maximality. The exploration of the branch corresponding to the assignment of the truth value $false$ to $not\, A_j$ is necessary only when the first branch produced at least one sub-clause $c_i \in C_{STB}$. $C_{STB} = \{c_i = \neg not\, A_{i_1} \vee, ..., \vee \neg not\, A_{i_k} / \mid c_i \mid \geq 1, \forall j \in \{1..k\}, not\, A_{i_j} \in STB\}$ be the set of all possible negative clauses formed by some literals of the set STB and which have at least one literal. This could avoid to the method to explore redundant and pointless branches. Hence, this property lead to reduce the number of choice point in the search tree. The proposed method is able to compute all the stable models of a given logic program.

In the following, we give an overview on the new search algorithm for stable models. Its enumerative process explores a boolean tree search. It is similar to that one of a DPLL [20], except that the procedure is adapted to the ASP framework and to the used semantic [16]. We implemented a set of inferences rules to boost the method. The pseudo-code of the general schema of the method is given in Algorithm 1. Throughout

the two alternate phases, the algorithm affect truth values to literals and develops a similar tree search as the one of a DPLL procedure. If a conflict is encountered during the search, then the algorithm explore the second branch corresponding to the second truth value of the literal representing the current choice point only if a clause $c_i \in C_{STB}$ is produced. otherwise a backtrack is done.

Algorithm 1 The general schema of the new search method

Require: The clausal form $L(\pi)$ of a logic program π
Ensure: The set S of all the stable models of π

$S = \emptyset$
repeat
  while $STB \neq \emptyset$ and 'no conflict' **do**
    while $Lmonos \neq \emptyset$ or $Lpure \neq \emptyset$ **do**
      unit-propagation($L(\pi)$,$Lmonos$,I);
      inference($L(\pi)$,$Lpure$,I);
      clause-production($L(\pi)$);
    end while
    literal choice (STB);
  end while
  if no conflict **then**
    $E = L(\pi)_I$ is an extension candidate;
    E= complete(E);
    if Conditions(E) **then**
      M=PositiveAtoms(E);
      $S = S \cup M$;
    end if
  else
    backtrack
  end if
until All the search space is explored

The algorithm starts by a first call to the unit-propagation procedure to propagate all the mono-literals until the list of mono-literals $Lmono$ becomes empty. Then it deals with the pure literals which also could induce mono-literals. When there is no mono-literal and no pure literals to assign, the algorithm try to produce a clause $c_i \in C_{STB}$. If we produce a clause $c_i \in C_{STB}$, then the second branch of the current choice point will be explored. Otherwise, if no clause was produced and all the mono-literals and the pure literals are treated, then the second branch of the choice point literal become useless. The enumeration continues by choosing in STB the next literal to assign. This process is repeated either until the satisfaction of all the clauses, or until the assignation of all the literals of STB without the appearance of the empty clause.

An extension candidate is founded either when all the clauses are satisfied, or when all the literals of STB are affected without falsifying any clause. In both cases, the algorithm execute a completing phase that consists in completing the current interpretation by assigning the value true to all the remaining literals $not\, A_i$ of STB and by assigning the value false to all the others literals $A_i \in V$ not assigned yet according to the closed world assumption. An extension candidate $E = L(\pi)_I$ is obtained when we reach this state and the completing phase is performed to get a kind of minimal model. After the verification of the maximality and the discriminant conditions on E, a stable model M consisting of the positive atoms A_i of E is extracted and added to the set S.

The algorithm complexity

If n is the number of variables of the clausal form $L(\pi)$ of the program π, k the cardinal of the set STB and m the number of clauses of $L(\pi)$, then the algorithm time complexity in the worst case is approximately O($knm2^k$). We can notice that the exponential factor of the complexity function depends on the number k representing the size of the strong back-door set and does not depend on the number of variables n as in the other ASP solvers. The value of k is generally smaller than that one of n, hence a better time complexity.

Unlike the majority of ASP solvers using the Clark completion with loop management and which have an exponential spatial complexity in the worst case, our method works with a constant space. Indeed, the method uses as input the Horn clausal form $L(\pi)$ whose size is identical to that one of the initial program π and it does not vary during the executions. The spatial complexity is constant, it is of order $O(\mid L(\pi) \mid) = O(\mid \pi \mid)$ in the worst case. This algorithm can be used for non tight logic programs and allows to compute all the stable models of any given general program.

3.1 Experimental results

In the literature there are two main approaches to conceive ASP systems. The first one deals directly with the considered semantics and its properties to implement the system. The second one, computes first a Clark completion of the given logic program then applies an SAT solver as a black-box on the resulting formula to which are added some formulas that are used to manage the loops. Our method adopts the policy of the first approach. Its implementation is totally based on the semantic previously presented. We implemented a first version of a new ASP solver that we denote here by $HC - asp$ to mean Horn Clause ASP. The solver is implemented in C++ and we used $Gringo$ [5] as a grounder. The output of gringo feeds the input of our solver.

To show the efficiency of the solver $HC - asp$, we compared it to other existing ASP systems. We considered in the comparison the solver $Cmodels$ (version 3.86 with $zChaff$ as a SAT solver). We also considered two other known ASP solvers that are $Smodels$ (version 2.34) and $Clasp$(version 3.3.3). We experimented different highly combinatorial problems. For each of them, we gradually increased its size and studied the behavior of each system when it is applied for its resolution. The benchmarks are: the Reachability problem [1-4], the consistent Pigeon Hole problem [10-17], the Ramsey problem[5-8], the n-queen problem [18-25]. We precise that we used for each benchmark the same encoding for the all the experimented solvers.

In general, in non-monotonic reasoning, we are interested in all possible extensions and then make some preferences on them. Naturally, in answer set programming, it is important to enumerate all the stable models of a logic program. That's why we preferred to enumerate all models rather than just checking for a model. We chose the cited benchmarks because of their important number of stable models. They are very appropriate to study the behavior of each of the solvers when the number of stable models and the size of the problem increase. Almost all of the benchmarks are available on the web site (https://asparagus.cs.uni-potsdam.de). We can say from the results presented in Table 1 that our approach is a good alternative for answer set programming.

Table 1. The results obtained on the Reachability, Ramesey, Pigeon Hole and n-queen problems

N°	#Size	#Stable Models	#Time(sec)			
			HC-asp	Clasp	Smodels	Cmodels
1	R_2	1	0.0005	0.0001	0.0002	0.0003
2	R_3	18	0.0015	0.001	0.0005	0.015
3	R_4	1606	0.030	0.070	0.022	0.091
4	R_5	565080	7.12	12.59	7.62	9991.28
5	R_4_5_5	957	0.011	0.010	0.014	0.049
6	R_4_5_6	27454	0.2	0.5	0.33	18.62
7	R_4_5_7	1452289	12.64	28.76	18.00	•
8	R_4_5_8	137578233	1625.14	3329.66	2219.19	•
10	pi4/4	24	0.007	0.009	0.002	0.003
11	pi5/5	120	0.02	0.02	0.008	0.01
12	pi6/6	720	0.06	0.06	0.04	0.07
13	pi7/7	5040	0.23	0.55	0.30	0.87
14	pi8/8	40320	1.65	4.01	2.5	52.34
15	pi9/9	362880	21.63	47.11	34.60	4591.87
16	pi10/10	3628800	210.14	494.40	369.80	•
17	pi11/11	39916800	2728.53	6936.96	4247.58	•
18	q_10	724	0.68	0.23	0.66	0.27
19	q_11	2680	2.79	1.02	2.95	2.48
20	q_12	14200	12.2	8.75	15.19	41.44
21	q_13	73712	79.55	122.91	87.69	1642.93
22	q_14	365596	371.73	2631.83	496.98	•
23	q_15	2279184	2797.02	34337.21	3352.37	•
24	q_16	14772512	12087.40	•	23134.22	•
25	q_17	95815104	87088.00	•	•	•

Indeed, our method shows better performances. The gain realized with our system increases when the size of the problem increases. This advantage comes from both its constant spatial complexity and the fact that it performs enumeration on a subset of literals representing the strong back door of the logic program. We precise that our implementation does not include for the moment any optimization (like restarts, watched literals, clause learning...). Better results are expected in future when all these techniques will be implemented. Our approach looks to be a good alternative that the community could use to implement ASP solvers.

4 Conclusion

In this paper, we provided a new method to compute stable models that is based on a relatively new semantic introduced in [16]. This method has the advantage to use a Horn clausal logic form whose size is identical to that one of the source ground logic program. The proposed method has a constant spatial complexity and the semantic on which it is based prevent it from the heaviness induced by the addition of loop formulas that is performed in almost all the known ASP solvers that use the Clark completion. The other benefit of our approach is the simplified enumerative process which is done only on a subset of the literals representing the strong back-door of the source logic program. This lead to a considerable gain in the time complexity. We also proposed and implemented some inference rules which are used in practice to reduce the size of the search tree. We experimented the proposed method on a variety of known combinatorial problems and the obtained results showed that our approach is a good alternative to implement

ASP solvers. Indeed, with a non optimized implementation we outperformed several efficient ASP solvers like $Clasp$, $Smodels$ and $Cmodels$.

As a future work, we look to enhance our implementation by concepts used in modern SAT solvers. The idea is to incorporate techniques such as watched literals, lazy structures, clause learning and restart. Another point is to investigate some extensions of our approach to others classes of logic programming or to pieces of more general non-monotonic logics.

References

1. Simons, P., Nimelä, I., Soininen, T.: Extending and implementing the stable model semantic. Artificial Intelligence **138** (2002) 181–234
2. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. IJCAI **7** (2007) 386–392
3. Lin, F., Zhao, Y.: Assat: Computing answer sets of a logic program by sat solvers. Artificial Intelligence (2004) 115–137
4. Lierler, Y., Maratea, M.: Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. Logic Programming and Nonmonotonic Reasoning (2004) 346350
5. Gebser, M., Schaub, T., Thiele, S.: Gringo: A new grounder for answer set programming. International Conference on Logic Programming and Nonmonotonic Reasoning **7** (2007) 266–271
6. Nimelä, I., Simons, P., Syrjanen, T.: Smodels: A system for answer set programming. Proceedings of the 8th International Workshop on Non-Monotonic Reasoning (2000)
7. Kaufmann, B., Leone, N., Perri, S., Schaub, T.: Grounding and solving in answer set programming. AI Magazine **37** (2016) 25–32
8. Clark, K.L.: Negation as failure. Logic and data bases (1978) 293–322
9. Gelder, A.V., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. Journal of the ACM (JACM) **38** (1991) 619–649
10. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. ICLP/SLP **50** (1988) 1070–1080
11. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New generation computing **9** (1991) 365–385
12. Baral, C.R., Subrahmanian, V.: Stable and extension class theory for logic programs and default logics. Journal of Automated Reasoning **8**(3) (1992) 345–366
13. Bidoit, N., Froidevaux, C.: More on stratified default theories. In: Proceedings of the 8th European Conference on Artificial Intelligence, Pitman Publishing (1988) 492–494
14. Przymusinski, T.C.: Three-valued formalizations of non-monotonic reasoning and logic programming. In: KR. (1989) 341–348
15. Marek, V.W., Truszczynski, M.: Relating autoepistemic and default logics. In: KR. (1989) 276–288
16. Benhamou, B., Siegel, P.: A new semantics for logic programs capturing and extending the stable model semantics. Tools with Artificial Intelligence (ICTAI) (2012) 25–32
17. Williams, R., Gomes, C.P., Selman, B.: Backdoors to typical case complexity. International joint conference on artificial intelligence **18** (2003) 1173–1178
18. Fages, F.: Consistency of clark's completion and existence of stable models. Methods of Logic in Computer Science **1** (1994) 51–60
19. Lifschitz, V., Razborov, A.: Why are there so many loop formulas? ACM Transactions on Computational Logic (TOCL) **7** (2006) 261–268
20. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Communications of the ACM **5** (1962) 394397