Dynamic and Static Symmetry Breaking in Answer Set Programming

Belaïd Benhamou[1,2,⋆]

[1] Aix-Marseille Université
Laboratoire des Sciences de l'Information et des Systèmes (LSIS)
Domaine universitaire de Saint Jérôme, Avenue Escadrille Normandie Niemen, 13397
MARSEILLE Cedex 20
[2] Centre de Recherche en Informatique de Lens (CRIL)
Université d'Artois, Rue Jean Souvraz, SP 18 F 62307 Lens Cedex
belaid.benhamou@univ-amu.fr

Abstract. Many research works had been done in order to define a semantics for logic programs. The well know is the stable model semantics which selects for each program one of its canonical models. The stable models of a logic program are in a certain sens the minimal Herbrand models of its *reduct* programs. On the other hand, the notion of symmetry elimination had been widely studied in constraint programming and shown to be useful to increase the efficiency of the associated solvers. However symmetry in non monotonic reasoning still not well studied in general. For instance Answer Set Programming (ASP) is a very known framework but only few recent works on symmetry breaking are known in this domain. Ignoring symmetry breaking in the answer set systems could make them doing redundant work and lose on their efficiency. Here we study the notion of *local* and *global* symmetry in the framework of answer set programming. We show how *local symmetries* of a logic program can be detected dynamically by means of the automorphisms of its graph representation. We also give some properties that allow to eliminate theses symmetries in SAT-based answer set solvers and show how to integrate this symmetry elimination in these methods in order to enhance their efficiency.

Keywords: symmetry, logic programming, stable model semantics, answer set programming, non-monotonic reasoning.

1 Introduction

The work we propose here to investigate the notion of symmetry in Answer Set Programming (ASP). The (ASP) framework can be considered as a sub-framework of the default logic [37]. One of the main questions in ASP, is to define a semantics to logic programs. A logic program π is a set of first order (formulas) rules of the form $r : concl(r) \leftarrow prem(r)$, where $prem(r)$ is the set of premises of the rule given as a conjunction of literals that could contain negations and negations as failure. The right part $concl(r)$ is the conclusion of the rule r which is generally, a single atom,

⋆ Actually, I am at CRIL for one year CNRS delagtion position.

K. McMillan, A. Middeldorp, and A. Voronkov (Eds.): LPAR-19, LNCS 8312, pp. 112–126, 2013.

or in some cases a disjunction of atoms for logic programs with disjunctions. Some researchers considered $prem(r)$ as the $body$ of the rule r and $concl(r)$ as its $head$ ($r : head(r) \leftarrow body(r)$). Each logic program π is translated into its equivalent ground logic program $ground(\pi)$ by replacing each rule containing variables by all its ground instances, so that each literal in $ground(\pi)$ is ground. This technique is used to eliminate the variables even when the program contains function symbols and its Herbrand universe is infinite. Among the influential semantics that had been given for these logic programs with negation and negation as failure are the completion semantics [15] and the stable model or the answer set semantics [25]. It is well know that each answer set for a logic program is a model of its completion, but the converse, is in general not true. Fages in his paper [21] showed that both semantics are equivalent for free loops logic programs that are called tight programs. A generalization of Fage's results to logic programs with eventual nested expressions in the bodies of their rules was given in [20]. On the other hand Fangzhen Lin and Yutin Zhao proposed in [31] to add what they called *loop formulas* to the completion of a logic program and showed that the set of models of the extended completion is identical to the program's answer sets even when the program is not tight.

On the other hand, symmetry is by definition a multidisciplinary concept. It appears in many fields ranging from mathematics to Artificial Intelligence, chemistry and physics. It reveals different forms and uses, even inside the same field. In general, it returns to a transformation, which leaves invariant (does not modify its fundamental structure and/or its properties) an object (a figure, a molecule, a physical system, a formula or a constraints network...). For instance, rotating a chessboard up to 180 degrees gives a board that is indistinguishable from the original one. Symmetry is a fundamental property that can be used to study these various objects, to finely analyze these complex systems or to reduce the computational complexity when dealing with combinatorial problems.

As far as we know, the principle of symmetry has been first introduced by Krishnamurthy [29] to improve resolution in propositional logic. Symmetries for Boolean constraints are studied in depth in [5,6]. The authors showed how to detect them and proved that their exploitation is a real improvement for several automated deduction algorithms efficiency. Since that, many research works on symmetry appeared. For instance, the static approach used by James Crawford et al. in [16] for propositional logic theories consists in adding constraints expressing global symmetry of the problem. This technique has been improved in [1] and extended to 0-1 Integer Logic Programming in [2]. The notion of interchangeability in Constraint Satisfaction Problems (CSPs) is introduced in [22] and find a good exploitation in [27], and symmetry for CSPs is studied earlier in [36,4].

Within the framework of the Artificial Intelligence, an important paradigm is to take into account incomplete information (uncertain information, revisable information...). Contrary to the mode of reasoning formalized by a conventional or a classical logic, a result deducible from information (from a knowledge, or from beliefs) is not true but only probable in the sense that it can be invalidated further, and can be revised when adding new information.

To manage the problem of exceptions, several logical approaches in Artificial Intelligence had been introduced. Many non-monotonic formalisms were presented since about thirty years. But, the notion of symmetry within this framework was not well studied. The principle of symmetry had been extended recently in [8,9,11] to non-monotonic reasoning. Symmetry had been defined and studied for three known non-monotonic logics: the preferential logic [13,14,12,28], the X-logic [38] and the default logic [38]. More recently, global symmetry had been studied for the Answer Set Programming framework [18,19]. In the same spirit as what it is done in [16,1,2] for the satisfiability problem, the authors of [18,19] showed how to break the global symmetry statically in a pre-processing phase for the ASP system Clasp[24]. They did that by adding symmetry breaking predicates to the considered logic program. They showed that global symmetry elimination in Clasp improves dramatically its efficiency on several problems. In this work, we investigate dynamic *local symmetry* detection and elimination and static *global symmetry* exploitation in SAT-based answer set programming systems. *Local symmetry* is the symmetry that we can discover at each node of the search tree during search. *Global symmetry* is the particular local symmetry corresponding to the root of the search tree (the symmetry of the initial problem). Almost all of the known works on symmetry are on global symmetry. Only few works on local symmetry [5,6,7,10] are known in the literature. Local symmetry breaking remains a big challenge. As far as we know, local symmetry is not studied yet in ASP.

The rest of the paper is structured as follows: in Section 2, we give some necessary background on answer set programming and permutations. We study the notion of symmetry for answer set programming in Section 3. In Section 4 we show how local symmetry can be detected by means of graph automorphism. We show how both global and local symmetry can be eliminated in Section 5. Section 6 shows how local symmetry elimination is implemented in a SAT-based answer set programming Method. Section 7 investigates the first implementation and experiments. We give a conclusion in Section 8.

2 Background

We summarize in this section some background on both the answer set programming framework and permutation theory.

2.1 Answer Set Programming

A ground general logic program π is a set of rules of the form $r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n$, $(0 \leq m < n)$ where L_i $(0 \leq i \leq n)$ are atoms, and not is the symbol expressing negation as failure. The positive body of r is denoted by $body^+(r) = \{L_1, L_2, \ldots, L_m\}$, and the negative body by $body^-(r) = \{L_{m+1}, \ldots, L_n\}$. The word *general* expresses the fact that the rules are more general than Horn clauses, since they contain negations as failure. The sub-rule $r^+ : L_0 \leftarrow L_1, L_2, \ldots, L_m$ expresses the positive projection of the rule r. Intuitively the rule r means "*If we can prove all of* $\{L_1, L_2, \ldots, L_m\}$ *and we can not prove all of* $\{L_{m+1}, \ldots, L_n\}$*, then we deduce* L_0". Given a set of atoms A, we say that a rule r is applicable (active) in A if

$body^+(r) \subseteq A$ and $body^-(r) \cap A = \emptyset$. The reduct of the program π with respect to a given set A of atoms is the positive program π^A where we delete each rule containing an expression $notL_i$ in its negative body such that $L_i \in A$ and where we delete the other expressions $notL_i$ in the bodies of the other rules. More precisely, $\pi^A = \{r^+/r \in \pi, body^-(r) \cap A = \emptyset\}$. The most known semantics for general logic programs is the one of stable models defined in [25] which could be seen as an improvement of the negation as failure of Prolog. A set of atoms A is a stable model (an answer set) of π if and only if A is identical to the minimal Herbrand model of π^A which is called its canonical model (denoted by $CM(\pi^A)$). That is, if only if $A = CM(\pi^A)$. The stable model semantics is based on the closed world assumption, an atom that is not in the stable model A is considered to be false.

An extended logic program is a set of rules as the ones given for general programs which could contain classical negation. The atoms L_i could appear in both positive and negative parity. In other words, the atoms L_i become literals. A logic program is said to be disjunctive when at least one of its rules contains a disjunction of literals in its head part. In the sequel, we will use indifferently the words stable model and answer set to designate a stable model of a general logic program.

2.2 Permutations

Let $\Omega = \{1, 2, \ldots, N\}$ for some integer N, where each integer might represent a propositional variable or an atom. A permutation of Ω is a bijective mapping σ from Ω to Ω that is usually represented as a product of cycles of permutations. We denote by $Perm(\Omega)$ the set of all permutations of Ω and $\circ$ the composition of the permutation of $Perm(\Omega)$. The pair $(Perm(\Omega), \circ)$ forms the permutation group of Ω. That is, $\circ$ is closed and associative, the inverse of a permutation is a permutation and the identity permutation is a neutral element. A pair $(T, \circ)$ forms a sub-group of $(S, \circ)$ iff T is a subset of S and forms a group under the operation $\circ$.

The orbit $\omega^{Perm(\Omega)}$ of an element ω of Ω on which the group $Perm(\Omega)$ acts is $\omega^{Perm(\Omega)}$=$\{\omega^\sigma : \omega^\sigma = \sigma(\omega), \sigma \in Perm(\Omega)\}$.

A generating set of the group $Perm(\Omega)$ is a subset Gen of $Perm(\Omega)$ such that each element of $Perm(\Omega)$ can be written as a composition of elements of Gen. We write $Perm(\Omega)$=$< Gen >$. An element of Gen is called a generator. The orbit of $\omega \in \Omega$ can be computed by using only the set of generators Gen.

3 Symmetry in Logic Programs

Since Krishnamurthy's [29] symmetry definition and the one given in [5,6] in propositional logic, several other definitions are given in the CP community.

We will define in the following both semantic and syntactic symmetries in answer set programming and show their relationship. In the sequel π could be the logic program or its completion [15] $Comp(\pi)$, the symmetry definitions and properties remain valuable.

Definition 1. *(semantic symmetry of the logic program) Let π be a logic program and L_π its complete [1] set of literals. A semantic symmetry of π is a permutation σ defined on L_π such that π and $\sigma(\pi)$ have the same answer sets.*

Definition 2. *(semantic symmetry of the completion) Let $Comp(\pi)$ be the Clark completion of a logic program π and $L_{Comp(\pi)}$ its complete [2] set of literals. A semantic symmetry of $Comp(\pi)$ is a permutation σ defined on $L_{Comp(\pi)}$ such that $Comp(\pi)$ and $\sigma(Comp(\pi))$ have the same answer sets.*

In other words a semantic symmetry is a literal permutation that conserves the set of answer sets of the logic program π. We adapt in the following the definition of syntactic symmetry given in [5,6] for satisfiability to logic programs.

Definition 3. *(syntactic symmetry of the logic program) Let π be a logic program and L_π its complete set of literals. A syntactic symmetry of π is a permutation σ defined on L_π such that the following conditions hold:*

1. $\forall \ell \in L_\pi, \sigma(\neg\ell) = \neg\sigma(\ell)$,
2. $\forall \ell \in L_\pi, \sigma(not\ell) = not\{\sigma(\ell)\}$,
3. $\sigma(\pi) = \pi$

Definition 4. *(syntactic symmetry of the completion) Let $Comp(\pi)$ be a logic program and $L_{Comp(\pi)}$ its complete set of literals. A syntactic symmetry of $Comp(\pi)$ is a permutation σ defined on $L_{Comp(\pi)}$ such that the following conditions hold:*

1. $\forall \ell \in L_\pi, \sigma(\neg\ell) = \neg\sigma(\ell)$,
2. $\sigma(Comp(\pi)) = Comp(\pi)$

In other words, a syntactical symmetry of a logic program or its completion is a literal permutation that leaves the logic program or the completion invariant. If we denote by $Perm(L_\pi)$ the group of permutations of L_π and by $Sym(L_\pi) \subset Perm(L_\pi)$ the subset of permutations of L_π that are the syntactic symmetries of π, then $Sym(L_\pi)$ is trivially a sub-group of $Perm(L_\pi)$.

Theorem 1. *Each syntactical symmetry of a logic program π is a semantic symmetry of π.*

Proof. It is trivial to see that a syntactic symmetry of a logic program π is always a semantic symmetry of π. Indeed, if σ is a syntactic symmetry of π, then $\sigma(\pi) = \pi$, thus it results that π and $\sigma(\pi)$ have the same set of answer sets.

In a similar way, we can prove the following theorem :

Theorem 2. *Each syntactical symmetry of the completion $Comp(\pi)$ is a semantic symmetry of $Comp(\pi)$.*

[1] The set of literals containing each literal of π and its negation as failure.
[2] The set of literals containing each literal of $Comp(\pi)$ and its negation.

Example 1. consider the logic program $\pi = \{d \leftarrow; c \leftarrow; b \leftarrow c, nota; a \leftarrow d, notb\}$ and the permutation σ=$(a,b)(c,d)(nota, notb)$ defined on the complete set L_π of literals occurring in π. We can see that σ is a syntactic symmetry of π ($\sigma(\pi)$=π).

Remark 1. The converse of each of the previous theorems is not true. That is, it is not true that a semantic symmetry is always a syntactical symmetry.

Now, we give an important property which establishes a relationship between the symmetries of a logic program and its completion.

Proposition 1. *Each syntactical symmetry of a logic program π is a semantic symmetry of its completion $Comp(\pi)$.*

Proof. Let σ be a syntactical symmetry of the program π and I a model of $Comp(\pi)$ which is an answer set of π. We have to prove that $\sigma(I)$ is also a model of $Comp(\pi)$ which is an answer set of π. The permutation σ is a syntactical symmetry of π, thus by Theorem 1 we deduce that σ is also a semantic symmetry of π. It results that $\sigma(I)$ is also an answer set of π. Since each model of a logic program π is also a model of its Clark completion, it follows that $\sigma(I)$ is a model of $Comp(\pi)$ which is in fact an answer set of π.

Remark 2. The previous proposition allows to use the syntactical symmetries of a logic program π in its Clark completion $Comp(\pi)$ in order to detect symmetrical answer sets of π. This gives an important alternative for symmetry detection in SAT-based ASP systems that use the the Clark completion. Indeed, we can just calculate the symmetries of the logic program π instead of calculating those of its completion. This could accelerate the symmetry detection as the size of the program π is generally substantially smaller than the size of its completion.

In the sequel we give some symmetry properties only in the case of logic programs π, but the considered properties are also valid in the case of the completion $Comp(\pi)$.

Definition 5. *Two literals ℓ and ℓ' of a logic π are symmetrical if there exists a symmetry σ of π such that $\sigma(\ell) = \ell'$.*

Definition 6. *Let π be a logic program, the orbit of a literal $\ell \in L_\pi$ on which the group of symmetries $Sym(L_\pi)$ acts is $\ell^{Sym(L_\pi)}$=$\{\sigma(\ell) : \sigma \in Sym(L_\pi)\}$*

Remark 3. All the literals in the orbit of a literal ℓ are symmetrical two by two.

Example 2. In Example 1, the orbit of the literal a is $a^{Sym(L_\pi)}$= $\{a, b\}$, the orbit of the literal c is $c^{Sym(L_\pi)}$= $\{c, d\}$ and the one of the literal $nota$ is $nota^{Sym(L_\pi)}$= $\{nota, notb\}$ All the literals of a same orbit are all symmetrical.

If *I* is an answer set of π and σ a syntactic symmetry, we can get another answer set of π by applying σ on the literals which appear in *I*. Formally we get the following property.

Proposition 2. *I is an answer set of π iff $\sigma(I)$ is an answer set of π .*

Proof. Suppose that I is an answer set of π, then I is a minimal Herbrand model of the reduct π^I. It follows that $\sigma(I)$ is a minimal model of $\sigma(\pi)^{\sigma(I)}$. We can then deduce that $\sigma(I)$ is a minimal model of $\pi^{\sigma(I)}$ since π is invariant under σ. We conclude that $\sigma(I)$ is an answer set of π. The converse can be shown by considering the converse permutation of σ.

For instance, in Example 1 there are two symmetrical answer sets for the logic program π. The fist one is $I = \{d, c, a\}$ and the second is $\sigma(I) = \{d, c, b\}$. These are what we call symmetrical answer sets of π. A symmetry σ transforms each answer set into an answer set and each no-good (not an answer set) into a no-good.

Theorem 3. *Let ℓ and ℓ' be two literals of π that are in the same orbit with respect to the symmetry group $Sym(L_\pi)$, then ℓ participates in an answer set of π iff ℓ' participates in an answer set of π.*

Proof. If ℓ is in the same orbit as ℓ' then it is symmetrical with ℓ' in π. Thus, there exists a symmetry σ of π such that $\sigma(\ell) = \ell'$. If I is an answer set of π then $\sigma(I)$ is also an answer set of $\sigma(\pi) = \pi$, besides if $\ell \in I$ then $\ell' \in \sigma(I)$ which is also an answer set of π. For the converse, consider $\ell = \sigma^{-1}(\ell')$, and make a similar proof.

Corollary 1. *Let ℓ be a literal of π, if ℓ does not participate in any answer set of π, then each literal $\ell' \in orbit^\ell = \ell^{Sym(L_\pi)}$ does not participate in any answer set of π.*

Proof. The proof is a direct consequence of Theorem 3

Corollary 1 expresses an important property that we will use to break local symmetry at each node of the search tree of a SAT-based answer set procedure. That is, if a no-good is detected after assigning the value True to the current literal ℓ, then we compute the orbit of ℓ and assign the value false to each literal in it, since by symmetry the value true will not lead to any answer set of the logic program.

For instance, consider the program of Example 1, and the partial interpretation $I = \{a, b, c\}$ where c is the current literal under assignation. It is trivial that I is not a stable model of the program. By corollary 1, we can deduce that the set $I' = \{a, b, d\}$ is not a stable model of the program too. Indeed, I' is obtained by replacing the current literal c in I by its symmetrical literal d. I is a no-good and by symmetry (without duplication of effort) we infer that I' is a no-good.

4 Symmetry Detection

The most known technique to detect syntactic symmetries for CNF formulas in satisfiability is the one consisting in reducing the considered formula into a graph [16,3,2] whose the automorphism group is identical to the symmetry group of the original formula. We adapt the same approach here to detect the syntactic symmetries of the completion of a program π. That is, we represent the CNF formula corresponding to the completion ($Compl(\pi)$) of the logic program π by a graph G_π that we use to compute the symmetry group of π by means of its automorphism group. When this graph is built, we use a graph automorphism tool like Saucy [3], Nauty [32], AUTOM [35] or the one

described in [33] to compute its automorphism group which gives the symmetry group of $Comp(\pi)$. Following the technique used in [16,3,2] to represent CNF formulas, we summarize bellow the construction of the graph which represent the completion of the logic program π. Here we focus on the case of general logic programs, but the technique could be generalized to other classes of logic programs like extended logic programs or disjunctive logic programs. Given the completion of a general logic program π, the associated colored graph $G_\pi(V, E)$ of its completion is defined as follows:

- Each positive literal ℓ_i of $Compl(\pi)$ is represented by a vertex $\ell_i \in V$ of the color 1 in G_π. The negative literal $not\ell_i$ associated with ℓ_i is represented by a vertex $not\ell_i$ of color 1 in G_π. These two literal vertices are connected by an edge of E in the graph G_π.
- Each clause c_i of $Compl(\pi)$ is represented by a vertex $c_i \in V$ (a clause vertex) of color 2 in G_π. An edge connects this vertex c_i to each vertex representing one of its literals.

This technique could be extended to extended and disjunctive logic programs in a natural way.

This is different from the approach which uses a body-atom graph [18]. Since our study is oriented to SAT-based ASP using the completion, we do not need to manage an oriented body-atom graph.

An important property of the graph G_π is that it preserves the syntactic group of symmetries of $Compl(\pi)$. That is, the syntactic symmetry group of the logic program $Compl(\pi)$ is identical to the automorphism group of its graph representation G_π, thus we could use a graph automorphism system like Saucy on G_π to detect the syntactic symmetry group of $Comp(\pi)$. The graph automorphism system returns a set of generators Gen of the symmetry group from which we can deduce each symmetry of $Compl(\pi)$.

5 Symmetry Elimination

There are two ways to break symmetry. The first one is to deal with the global symmetry which is present in the formulation of the given problem. Global symmetry can be eliminated in a static way in a pre-processing phase of an answer set solver by just adding the symmetry predicates. For instance, a method for global symmetry elimination is introduced in [18] for the Clasp ASP system [24]. The second way is the elimination of local symmetry that could appear in the sub-problems corresponding to the different nodes of the search tree of an answer set solver. Global symmetry can be considered as the local symmetry corresponding to the root of the search tree.

Local symmetries have to be detected and eliminated dynamically at some decision node of the search tree. Dynamic symmetry detection in satisfiability had been studied in [5,6] where a local syntactic symmetry search method had been given. However, this method is not complete, it detects only one symmetry σ at each node of the search tree when failing in the assignment of the current literal ℓ. As an alternative to this incomplete symmetry search method, a complete method which uses the tool Saucy [3] had been introduced in [10] to detect and break all the syntactic local symmetries of a

constraint satisfaction problem (CSP) [34] during search and local symmetry had been detected and eliminated dynamically in a SAT solver [7].

Consider the completion $Compl(\pi)$ of a logic program π, and a partial assignment I of a SAT-based answer set solver applied to $Compl(\pi)$. Suppose that ℓ is the current literal under assignment. The assignment I simplifies $Compl(\pi)$ into a sub-completion $Compl(\pi)_I$ which defines a state in the search space corresponding to the current node n_I of the search tree. The main idea is to maintain dynamically the graph G_{π_I} of the sub-completion $Compl(\pi)_I$ corresponding to the current node n_I, then color the graph G_{π_I} as shown in the previous section and compute its automorphism group $Aut(\pi_I)$. The sub-completion $Compl(\pi)_I$ can be viewed as the remaining sub-problem corresponding to the unsolved part. By applying an automorphism tool on this colored graph we can get the generator set Gen of the symmetry sub-group existing between literals from which we can compute the orbit of the current literal ℓ that we will use to make the symmetry cut.

After this, we use Corollary 1 to break dynamically the local symmetry and then prune search spaces of tree search answer set methods. Indeed, if the assignment of the current literal ℓ defined at a given node n_I of the search tree is shown to be a failure, then by symmetry, the assignment of each literal in the orbit of ℓ will result in a failure too. Therefore, the negated literal of each literal in the orbit of ℓ has to be assigned in the partial assignment I. Thus, we prune in the search tree, the sub-space which corresponds to the assignment of the literals of the orbit of ℓ. That is what we call the local symmetry cut.

6 Local Symmetry Exploitation in SAT-Based ASP Solvers

The solver ASSAT [31] has some drawbacks: it can compute only one answer set and the formula could blow-up in space. Taking into account these disadvantages of ASSAT and the fact that each answer set of a program π is a model of its completion $Compl(\pi)$, Guinchiglia et al. in [26] do not use SAT solvers as black boxes, but implemented a method which is based on the DLL [17] procedure and where they include a function which checks if a generated model is an answer set or not. This method had been implemented in the Cmodels-2 system [30] and has the following advantages: it performs the search on $Compl(\pi)$ without introducing any extra variable except those used by the clause transformation of $Compl(\pi)$, deals with tight and not tight programs, and works in a polynomial space. Global symmetry breaking do not need any extra-implementation, a SAT-based answer set solver is used as a black box on the completion of the logic program and the generated symmetry breaking predicates. More recently the ASP solvers like the conflict-driven Clasp solver [24] include some materials of modern SAT solvers such as: conflict analysis via the First UIP scheme, no-good recording and deletion, backjumping, restarts, conflict-driven decision heuristics, unit propagation via watched literals, equivalence reasoning and resolution-based pre-processing [23] have shown dramatic improvements in their efficiency and compete with the best SAT solvers.

We give in the following a DLL-based answer set method in which we implement dynamic local symmetry breaking. We used as a baseline method the DLL-based answer

set procedure given in [26] to show the implementation of local symmetry eliminations (local symmetry cuts).

If I is an inconsistent partial interpretation in which the assignment of the value $true$ to the current literal ℓ is shown to be a no-good, then, all the literals in the orbit of ℓ computed by using the group $Sym(\pi_I)$ returned by the graph automorphism tool are symmetrical to ℓ. Thus, we assign the value false to each literal in $\ell^{Sym(L_\pi)}$ since the value true is shown to be contradictory, and then we prune the sub-space which corresponds to the value true assignments. The other case of local symmetry cut happen when the assignment I is shown to be a model of $Compl(\pi)$, but is not an answer set of π. In this case, the algorithm makes a backtracking on the last decision literal ℓ in I, then according to corollary 1 assigns the value false to each literal in the orbit $\ell^{Sym(L_\pi)}$ since the value true does not lead to an answer set of π. If $\Gamma = Compl(\pi)$, then the resulting procedure called DLLAnswerSet, is given in Figure 1.

Procedure DLLAnswerSet(Γ, I);
begin
  if $\Gamma = \emptyset$ **then return** $AnswerSetCheck(I, \pi)$
    else return False
  else if Γ contains the empty clause, **then return** False
    else
      if there exists a mono-literal or a monotone literal ℓ **then**
        return DLLAnswerSet($\Gamma_\ell, I \cup \{\ell\}$)
      begin
        Choose an unsigned literal ℓ of Γ
        Gen=AutomorphismTool(Γ_I);
        $\ell^{Sym(L_{\pi_I})}$=orbit(ℓ,Gen)=$\{\ell_1, \ell_2, ..., \ell_n\}$;
        return DLLAnswerSet($\Gamma_\ell, I \cup \{\ell\}$) **or**
        DLLAnswerSet($\Gamma_{\neg\ell \wedge \neg\ell_1 \wedge \neg\ell_2 \wedge ... \wedge \neg\ell_n}$,
                $I \cup \{\neg\ell, \neg\ell_1, \ldots, \neg\ell_n\}$)
      end
end

Fig. 1. The DLL-based answer set procedure with local symmetry elimination

The function AutomorphismTool(π_I) is a call to the automorphism tool which return the set of generators in the variable GEN. The function $orbit(\ell, Gen)$ is elementary, it computes the orbit (the symmetrical literals) of the literal ℓ from the set of generators Gen returned by AutomorphismTool(π_I). The set Γ_ℓ is the set of clauses obtained from Γ by removing the clauses to which ℓ belongs, and by removing $\neg\ell$ from the other clauses of Γ.

The function $AnswerSetCheck(I, \pi)$ is also elementary:

- it computes the set $A = I \cap \{head(r) : r \in \pi\}$ of positive literals (atoms) in I and returns $True$ if A is an answer set or π, and
- return $False$, otherwise.

7 Experiments

Now we shall investigate the performances of our search techniques by experimental analysis. We choose for this first implementation the graph coloring problem to show the local symmetry behavior on answer sets search vs the global symmetry. Graph coloring problem is expressed naturally as a set of rules of a general problem. For more details, the reader can refer to the Lparse user's manual given on line on the Cmodels site (http://www.cs.utexas.edu/ tag/cmodels/). Here, we tested and compared on some random graph coloring instances two methods:

1. **Global-sym:** search with global symmetry breaking. This method uses in a preprocessing phase the program SHATTER [1,2] that detects and eliminates the global symmetries of the considered instance by adding to it symmetry breaking clauses, then apply the SAT based answer set solver defined in [26] to the resulting instance. The CPU time of *Global-sym* includes the time that SHATTER spends to compute the global symmetry. A disadvantage of this method is that it could significantly increase the size of the considered instance. Its advantage is that its implementation requires no modification of the solver.
2. **Local-sym:** search with local symmetry breaking. This method implements in the SAT based answer set solver defined in [26] the dynamic local symmetry detection and elimination strategy described in this work. The resulting method is depicted in figure 1 (the DLLAnswerSet procedure). The CPU time of *Local-sym* includes local symmetry search time. A disadvantage of this method is that it could significantly increase the time of execution in the case of instances which contain few local symmetries. Its advantage is that its application does not require any increase in the size of the instance, changing the solver is simple and it detects more symmetries.

The common baseline answer set search method for both previous methods is the one given in [26]. The complexity indicators are the number of nodes of the search tree and the CPU time. Both the time needed for computing local symmetry and global symmetry are added to the total CPU time of search. The source codes are written in C and compiled on a Pentium 4, 2.8 GHZ and 1 Gb of RAM.

7.1 The Results on the Graph Coloring Instances

Random graph coloring problems are generated with respect to the following parameters: (1) n : the number of vertices, (2) $Colors$: the number of colors and (3) d: the density which is a number between 0 and 1 expressed by the ratio : the number of constraints (the number of edges in the graph) to the number of all possible constraints (the number of possible edges in the graph). For each test corresponding to some fixed values of the parameters n, $Colors$ and d, a sample of 100 instances are randomly generated and the measures (CPU time, nodes) are taken on the average.

We reported in Figure 2 the practical results of the methods: *Global-sym*, and *Local-sym*, on the random graph coloring problem where the number of variables is $n = 30$ and where the density is ($d = 0.5$). The curves give the number of nodes respectively the CPU time with respect to the number of colors for each search method.

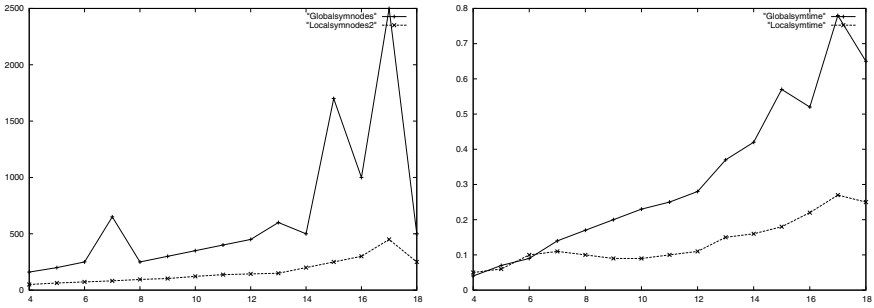

Fig. 2. Node and Time curves of the two symmetry methods on random graph coloring where $n = 30$ and $d = 0.5$

We can see on the node curves (the curves on the left of the figure) that *Local-sym* detects and eliminates more symmetries than the *Global-sym* method and *Global-sym* is not stable for graph coloring. From the CPU time curves (the curves on the right of the figure), we can see that *Local-sym* is in average faster than *Global-sym* even that Saucy is run at each contradictory decision node. Local symmetry elimination is profitable for solving random graph coloring instances and outperforms dramatically global symmetry breaking on these problems.

These are just our first results, our implementation and experiments are still in progress, we need to experiment much more and greater size instances than the ones presented here in order to further confirm the advantage of local symmetry breaking.

8 Conclusion

We studied in this work the notions of global and local symmetry for logic programs in the answer set programing framework . We showed how a logic program or its completion is represented by a colored graph that can be used to compute symmetries. The syntactic symmetry group of the completion is identical to the automorphism group of the corresponding graph. Graph automorphism tools like SAUCY can be naturally used on the obtained graph to detect the syntactic symmetries. Global symmetry is eliminated statically by adding in pre-processing phase the well known lex order symmetry breaking predicates to the program completion and applying as a black box a SAT-based answer set solver on this resulting encoding. We showed how local symmetry can be detected and eliminated dynamically during search. That is, the symmetries of each sub-problem defined at a given contradictory decision node of the search tree and which is derived from the initial problem by considering the partial assignment corresponding to that node. We showed that graph automorphism tools can be adapted to compute this local symmetry by maintaining dynamically the graph of the sub- program or the sub-completion defined at each node of the search tree. We proved some properties that allow us to make symmetry cuts that prune the search tree of a SAT-based answer set method. Finally, we showed how to implement these local symmetry cuts in a DLL-based answer set method.

The proposed local symmetry detection method is implemented and exploited in the tree search method $DLLAnswerSet$ to improve its efficiency. The first experimental results confirmed that local symmetry breaking is profitable for answer set solving and improves global symmetry breaking on the considered problems.

As a future work, we are looking to experiment other problems and combine both the global symmetry and local symmetry eliminations in a DLL-based answer set solver and compare the performances of the obtained methods to existing methods.

Another alternative of symmetry detection that we want to do in the future is to detect symmetries of the logic program by means of a body-atom graph, instead of those of its completion, then use Proposition 1 to make cuts in the search tree of the considered ASP solver. This could accelerated the symmetry detection then get a fastest solver.

We studied the notion of symmetry for the general logic programs, but the study could naturally be generalized for extended logic programs, disjunctive logic programs or other extensions. This is another important point that we are looking to investigate in future.

References

1. Aloul, F.A., Ramani, A., Markov, I.L., Sakallak, K.A.: Solving difficult sat instances in the presence of symmetry. In: DAC, pp. 1117–1137 (2003)
2. Aloul, F.A., Ramani, A., Markov, I.L., Sakallak, K.A.: Symmetry breaking for pseudo-boolean satisfiability. In: ASPDAC 2004, pp. 884–887 (2004)
3. Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Solving difficult SAT instances in the presence of symmetry. In: The Proceedings of the 39th Design Automation Conference (DAC 2002), pp. 731–736. ACM Press (2002)
4. Benhamou, B.: Study of symmetry in constraint satisfaction problems. In: Borning, A. (ed.) PPCP 1994. LNCS, vol. 874, pp. 246–254. Springer, Heidelberg (1994)
5. Benhamou, B., Sais, L.: Theoretical study of symmetries in propositional calculus and application. In: Kapur, D. (ed.) CADE 1992. LNCS, vol. 607, pp. 281–294. Springer, Heidelberg (1992)
6. Benhamou, B., Sais, L.: Tractability through symmetries in propositional calculus. The Journal of Automated Reasoning 12, 89–102 (1994)
7. Benhamou, B., Nabhani, T., Ostrowski, R., Saïdi, M.R.: Dynamic symmetry detection and elimination in the satisfiability problem. In: Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR-16, Dakar, Senegal, April 25-May 1 (2010)
8. Benhamou, B., Nabhani, T., Siegel, P.: Reasoning by symmetry in non-monotonic inference. In: The Proceedings of the International Conference on Machine and Web Intelligence (ICMWI 2010), Algiers, Algeria, pp. 264–269 (October 3, 2010)
9. Benhamou, B., Nabhani, T., Siegel, P.: Reasoning by symmetry in non-monotonic logics. In: 13th International Workshop on Non-Monotonic Reasoning (NMR 2010) (May 14, 2010)
10. Benhamou, B., Saïdi, M.R.: Local symmetry breaking during search in cSPs. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 195–209. Springer, Heidelberg (2007)
11. Benhamou, B., Siegel, P.: Symmetry and non-monotonic inference. In: The Proceedings of Symcon 2008, Sydney, Australia (September 2008)

12. Besnard, P., Siegel, P.: The preferential-models approach in nonmonotonic logics - in nonstandard logic for automated reasoning. In: Smets, P. (ed.) Academic Press, pp. 137–156 (1988)
13. Bossu, G., Siegel, P.: Nonmonotonic reasoning and databases. In: Advances in Database Theory, pp. 239–284 (1982)
14. Bossu, G., Siegel, P.: Saturation, nonmonotonic reasoning and the closed-world assumption. Artif. Intell. 25(1), 13–63 (1985)
15. Clark, K.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and data bases, pp. 293–322 (1978)
16. Crawford, J., Ginsberg, M.L., Luck, E., Roy, A.: Symmetry-breaking predicates for search problems. In: KR 1996, pp. 148–159 (1996)
17. Davis, M., Logemann, G.W., Loveland, D.W.: A machine program for theorem proving. Journal of Communications of The ACM - CACM 5(7), 394–397 (1962)
18. Drescher, C., Tifrea, O., Walsh, T.: Symmetry-breaking answer set solving. AI Commun. 24(2), 177–194 (2011)
19. Drescher, C., Tifrea, O., Walsh, T.: Symmetry-breaking in answer set solving. In: ICLP 2010 Workshop ASPOCP 2010 (2010)
20. Erdem, E., Lifschitz, V.: Tight logic programs. Thoery and Practice of Logic Programming 3, 499–518 (2003)
21. Fages, F.: Consistency of Clark's completion and existence of stable models. Journal of Methods of Logic Programming in Computer Sciences 1, 51–60 (1994)
22. Freuder, E.: Eliminating interchangeable values in constraints satisfaction problems. In: AAAI 1991, pp. 227–233 (1991)
23. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Advanced pre-processing for answer set solving. In: Proceedings of the 18th European Conference on Artificial Intelligence, pp. 15–19. IOS Press, Amsterdam (2008)
24. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: *clasp*: A conflict-driven answer set solver. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 260–265. Springer, Heidelberg (2007)
25. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kawalski, R., Bowen, K. (eds.) Logic Programming: Fifth Int'l Conf. and Symp., pp. 1070–1080 (1988)
26. Giunchiglia, E., Lierler, Y., Maratea, M.: Sat-based answer set programming. In: 19th National Conference on Artificial Intelligence, July 25-29. AAAI, San Jose (2004)
27. Haselbck, A.: Exploiting interchangeabilities in constraint satisfaction problems. IJCAI 93, 282–289 (1993)
28. Kraus, S., Lehmann, D.J., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence 44(1-2), 167–207 (1990)
29. Krishnamurty, B.: Short proofs for tricky formulas. Acta Inf. (22), 253–275 (1985)
30. Lierler, Y., Maratea, M.: Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 346–350. Springer, Heidelberg (2003)
31. Lin, F., Zhao, Y.: Assat: Computing answer sets of a logic program by sat solver. In: Proceedings of AAAI 2002 (2002)
32. McKay, B.: Practical graph isomorphism. Congr. Numer. 30, 45–87 (1981)
33. Mears, C., de la Banda, M.G., Wallace, M.: On implementing symmetry detection. In: Proceedings of SymCon 2006, pp. 1–8 (2006)
34. Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. Information Science 7, 95–132 (1974)

35. Puget, J.-F.: Automatic detection of variable and value symmetries. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 475–489. Springer, Heidelberg (2005)
36. Puget, J.F.: On the satisfiability of symmetrical constrained satisfaction problems. In: Komorowski, J., Raś, Z.W. (eds.) ISMIS 1993. LNCS (LNAI), vol. 689, Springer, Heidelberg (1993)
37. Reiter, R.: A logic for default reasoning. Artificial Intelligence 13, 81–132 (1980)
38. Siegel, P., Forget, L., Risch, V.: Preferential logics are x-logics. Journal of Logic and Computation 11(1), 71–83 (2001)