# On the Complexity of Extending* Ground Resolution with Symmetry Rules†

Thierry Boy de la Tour Stéphane Demri
LIFIA-IMAG
46, Av. Félix Viallet, 38031 Grenoble Cedex, France
{Thierry.Boy-de-la-Tour,Stephane.Demri}@imag.fr

## Abstract

One important issue of automated theorem proving is the complexity of the inference rules used in theorem provers. If Krishnamurty's general symmetry rule is to be used, then one should provide some way of computing non trivial symmetries. We show that this problem is **NP**-complete. But this general rule can be simplified by restricting it to what we call $S$-symmetries, yielding the well-known symmetry rule. We show that computing $S$-symmetries is in the same complexity class as the graph isomorphism problem, which appears to be neither polynomial nor **NP**-complete. However it is sufficient to compute the set of all $S$-symmetries at the beginning of the proof search, and since it is a permutation group, there exist some efficient techniques from computational group theory to represent this set. We also show how these techniques can be used for applying the $S$-symmetry rule in polynomial time.

## 1 Introduction

Symmetries in theorems have long been recognized to induce symmetries in proofs, and used to avoid the boredom of repetitive arguments. As such, this rule of meta-reasoning can be considered as a useful trick for writing proofs, and hence for discovering them, but by no means is it accredited as a powerful method for gaining deep knowledge of the problem at hand (except when symmetries are the *subject* of the problem, but this is out of the scope of our concern). Indeed, when this argument is invoked (e.g. "case 2 is similar"), the symmetry is supposed to be self-evident, not deserving any proof.

However, the appearance of computers and their use to assist in rigorous proofs shed a new light on the subject. Experience shows that computers *never* get bored, even on repetitive work that could be avoided. But experience also shows that users often get bored when their computer spends more time than expected for solving seemingly trivial problems. Hence the necessity to help computers think twice before doing something, and allow them to conclude: `case 2 is similar`.

One way to do this is to reason in a highly expressive logic, in which symmetry arguments can naturally be expressed. This is usually the case when all non-logical symbols can be quantified. However, such higher order logics are seldom used in Artificial Intelligence. It should also be noted that specific calculi, or strategies, may not be able to *reach* such arguments. Hence the very natural idea to *extend* calculi with symmetry rules, and provide strategies for using them. Indeed, the idea appeared in the literature as early as 1959, in [Gel59][1]. Maybe this was too early: to our knowledge, this paper has had no lineage in the following decades. It is written in a setting that looks weird today, and makes it difficult to read. By the way, [Gel59] is only concerned with symmetries as permutations of first-order variables, and gives a rather complicated algorithm for computing these, which may appear as a waste of computational time.

It is only 26 years later that Krishnamurty, in [Kri85], introduced his symmetry rules (without reference to [Gel59]) to extend propositional resolution, and showed on difficult examples how these rules can be used to reproduce natural arguments. These rules, and the complexity of the problems they involve, are the main concern of the present work, and are defined below. Since 1985, little work has been devoted to this problem (the implementation of [BS94] is discussed below). This should be surprising to anyone sharing Slaney's opinion: "I consider symmetry to be one of the most important topics of current research in ground theorem proving" (see [Sla94]).

In the sequel, we will refer to notions from computational complexity (see [GJ79]), graph and group theory (see [Hof81]), which we will introduce when needed. A *clause* is a set of literals. A set $S$ of clauses is *on* a set of

* Published in the Proceedings of the 14th International Joint Conference on Artificial Intelligence (1995)

† This work has been partly supported by CNRS.

[1] We thank Ricardo Caferra for pointing us to this paper.

literals $L$, or $L$ is *for $S$* iff the literals occurring in $S$ also occur in $L$. For $l \in L$, $l^c$ is the complementary literal of $l$.

Given a resolution derivation of a clause $C$ from a set of clauses $S_1$ on $L_1$ (noted $S_1 \vdash C$), one can obtain a new derivation by substituting new literals to those in $L_1$, under the following conditions: the literals resolved upon should remain complements after the substitution, and they should occur in the clauses, which means that they shouldn't have been resolved away sooner in the resulting proof. Hence the conditions that a substitution $\sigma$, say a function from $L_1$ to $L_2$, should satisfy: first, $\sigma$ should preserve complementarity ($\forall l \in L_1$, if $l^c \in L_1$ then $\sigma(l^c) = \sigma(l)^c$), and second, it should be injective. We note $Subst(L_1, L_2)$ the set of these functions. Hence for all $S_1$ on $L_1$, $\sigma \in Subst(L_1, L_2)$, if $S_1 \vdash C$ then $\sigma(S_1) \vdash \sigma(C)$. While refuting $S_2$, a set of clauses on $L_2$, and a clause $C$ being derived from $S_1 \subset S_2$, it is therefore correct to infer $\sigma(C)$ provided $\sigma(S_1) \subset S_2$. This inference rule depends on the proof of $C$, or more precisely on the hypotheses used in this proof, and can therefore be formalized in sequent style. Let $Sym_{S_2}(S_1) = \{\sigma \in Subst(L_1, L_2) / \sigma(S_1) \subset S_2\}$ be the set of *symmetries of $S_1$ in $S_2$*, the *general symmetry rule* is:

$$\frac{S_1 \vdash C}{S_2 \vdash \sigma(C)} \text{ if } S_1 \subset S_2{}^2 \text{ and } \sigma \in Sym_{S_2}(S_1)$$

To avoid this dependency, which is not quite in the "spirit" of resolution, we consider a restriction of this rule by requiring that $S_1 = S_2$ (and $L_1 = L_2$). Let $Sym_S = \{\sigma \in Subst(L, L) / \sigma(S) \subset S\}$ be the set of *$S$-symmetries*, then the *$S$-symmetry rule* is:

$$\frac{C}{\sigma(C)} \text{ if } \sigma \in Sym_S$$

It is therefore correct to extend resolution with the general symmetry rule, which essentially yields Krishnamurty's SR-II symmetric resolution proof system. It is also correct to extend resolution (on a set of clauses $S$) by the $S$-symmetry rule, resulting in SR-I. It is doubtful whether SR-I can polynomially simulate SR-II, but it has nice properties: the $S$-symmetry rule is simple, $S$-symmetries can be computed before drawing any inference from $S$, and $Sym_S$ forms a *permutation group* on the considered set of literals $L$ for $S$ (the identity is $Id_L$; the *trivial subgroup $\mathcal{I}$* is restricted to the identity). This is obvious since $Sym_S = \{\sigma \in Subst(L, L) / \sigma(S) = S\}$, which comes from the fact that the extension of $\sigma$ to sets of literals (when confusion is possible, this extension will be noted $\widehat{\sigma}$) is injective since $\sigma$ is injective, hence $|\sigma(S)| = |S|$, but $\sigma(S) \subset S$ is finite, hence $\sigma(S) = S$.

[2] As noted by a referee, the condition $S_1 \subset S_2$ is not necessary to the correction of the rule, but elements of $Sym_{S_2}(S_1)$ could not be considered as symmetries without it.

Dealing with complexity, we will refer to the *length* of particular objects (sets of clauses, sequences of permutations...), which means the length of the representation of this object in any reasonable (unspecified) encoding (see [GJ79]). Given two problems $\mathcal{P}$ and $\mathcal{Q}$, we note $\mathcal{P} \propto_P \mathcal{Q}$ (*$\mathcal{P}$ polynomially reduces* to $\mathcal{Q}$) when the existence of a polynomial time algorithm for $\mathcal{Q}$ implies the existence of a polynomial time algorithm for $\mathcal{P}$. $\equiv_P$ is the symmetric closure of $\propto_P$.

## 2 Detecting general symmetries

We address the problem of the amount of computation required in order to apply the general symmetry rule *in a non trivial way.* This means the search for a $\sigma \in Sym_{S_2}(S_1)$ such that $\sigma(C) \neq C$. The input of this problem is $L_1, L_2, S_1, S_2$ and $C$, such that $L_1 \subset L_2, S_1 \subset S_2$, $S_1$ is on $L_1$ and $S_2, C$ are on $L_2$. The output is a sequence of elements of $L_1 \times L_2$ representing such a symmetry $\sigma$. This problem is obviously in **NP**: this sequence, which length is polynomial in $|L_1|$, can be guessed and checked in polynomial time.

In the sequel, we consider the simpler decision problem $\exists \sigma \in Sym_{S_2}(S_1)$ such that $\sigma \neq Id_{L_1}$, which polynomially reduces to the previous one: such a $\sigma$ exists iff $\exists l \in L_1, \exists \sigma' \in Sym_{S_2}(S_1)$ such that $\sigma'(\{l\}) \neq \{l\}$.

### 2.1 The graph restriction

Let $G = (V, E), G' = (V', E')$ be two graphs, a *homomorphism* from $G$ to $G'$ is a function $\alpha$ from $V$ to $V'$, such that $\alpha(E) \subset E'$. Let $Mon(G, G')$ be the set of injective homomorphisms (*monomorphisms*) from $G$ to $G'$. Also, $\alpha$ is an *isomorphism* from $G$ to $G'$ if $\alpha$ is bijective and $\alpha(E) = E'$, and $Aut(G)$ is the set of isomorphisms from $G$ to $G$ (*automorphisms*). $G$ is a *subgraph* of $G'$ if $V \subset V'$ and $E \subset E'$; this is noted $G \subset G'$.

It is clear that a set of edges $E$ can be considered as a set of clauses on $V$, and conversely that any set $S$ of clauses of length 2 (*2-clauses*) on $L$, without negation, can itself be considered as a set of edges on the vertex set $L$, yielding a graph $(L, S)$. Moreover, we have:

LEMMA 2.1 Let $G = (V, E) \subset G' = (V', E')$, then $Sym_{E'}(E) = Mon(G, G')$.

*Proof.* $\sigma \in Mon(G, G')$ iff $\sigma : V \to V'$ is injective and $\sigma(E) \subset E'$, iff $\sigma \in Subst(V, V')$ and $\sigma(E) \subset E'$, iff $\sigma \in Sym_{E'}(E)$. Q.E.D.

Hence, given any problem $\mathcal{P}$ on $Sym_{S'}(S)$, with $L, L', S, S'$ among the input, we can consider its restriction to sets of 2-clauses without negation, and translate it as a problem on $Mon(G, G')$, having as input $G$ and $G'$, and consequent restrictions of the rest of the input of $\mathcal{P}$. This will be called the *graph restriction* of $\mathcal{P}$.

As for sets of clauses, the finite nature of graphs implies that $Mon(G, G) = Aut(G)$, thus lemma 2.1 on $G = G' = (V, E)$ yields $Sym_E = Aut(G)$. Hence graph restriction of problems on $Sym_S$ translate on $Aut(G)$.

## 2.2 Monomorphisms and rigid graphs

The graph restriction of our problem therefore consists in detecting the existence of a $\alpha \in Mon(G, G') \backslash \{Id_V\}$, where $G = (V, E)$ is a subgraph of $G'$. This problem will be simplified by further restricting it. A graph $G$ is *rigid* if $Aut(G) = \mathcal{I}$; it is *connected* if any two vertices are connected by a path (a sequence of adjacent edges). Any graph can be expressed as a disjoint union (noted $+$) of connected graphs, its *connected components*.

Given a homomorphism $\alpha$ from $G$ to $G'$, it is clear that if $G$ is connected, then $\alpha(G)$ is connected, and is a subgraph of a connected component of $G'$. Also, the *valence* of vertices (noted $val_G(v)$, the number of edges of $G$ adjacent to $v$) increases by $\alpha$: $\forall v \in V, val_G(v) \leq val_{G'}(\alpha(v))$. Moreover, if $\alpha$ is an isomorphism, $\forall v \in V$, $val_{G'}(\alpha(v)) = val_G(v)$.

LEMMA 2.2 Let $G = (V, E), G' = (V', E')$ be two disjoint graphs ($V \cap V' = \emptyset$), if $G$ is connected, then $Mon(G, G') = Mon(G, G + G') \backslash Aut(G)$.

*Proof.* Let $\alpha \in Mon(G, G')$, then $\alpha \in Mon(G, G + G')$, and also $\alpha \notin Aut(G)$ since $\alpha$ is into $V'$. Conversely, let $\alpha \in Mon(G, G + G')$ such that $\alpha \notin Aut(G) = Mon(G, G)$, hence $\exists v \in V / \alpha(v) \in V'$. But $\alpha(G)$ is a subgraph of a connected component of $G + G'$, hence of $G'$ alone, since $G$ and $G'$ are disjoint. Hence $\alpha(V) \subset V'$, and $\alpha \in Mon(G, G')$. Q.E.D.

We can now restrict our problem $Mon(G, G') \neq \mathcal{I}$ to the input $G, G + G'$ ($G \subset G + G'$), where $G$ is rigid and connected: this restriction is equivalent to the problem $Mon(G, G') \neq \emptyset$. We call it the Rigid Connected Graph Monomorphism problem (RCGM for short), which we just proved to be polynomially reducible to the search for non trivial symmetries. RCGM is therefore **NP**, and we prove its **NP**-completeness by reducing CLIQUE to it.

## 2.3 Cliques and monomorphisms

The well-known **NP**-complete CLIQUE problem consists in detecting in $G$ a complete subgraph of $k$ vertices (a $k$-*clique*), where $G = (V, E)$, $k \leq |V|$ are given. Since a clique is not rigid, we build the graph $K_k = (\{\langle i, j\rangle / 0 \leq i \leq j \leq k, 1 \leq j\}, KE_k)$, where

$$(\langle i, j\rangle, \langle i', j'\rangle) \in KE_k \quad \Leftrightarrow \quad \begin{array}{l}(i = i' = 0 \wedge j \neq j') \\ \vee (j = j' \wedge i + 1 = i')\end{array}$$

If $1 \leq i < j \leq k$, then $val_{K_k}(\langle i, j\rangle) = 2$ and $val_{K_k}(\langle j, j\rangle) = 1$. The subgraph of $K_k$ generated by $\{\langle 0, 1\rangle, \ldots, \langle 0, k\rangle\}$ is a $k$-clique (see figure 1).

LEMMA 2.3 $\forall k > 2, K_k$ is rigid.

*Proof.* Let $\alpha \in Aut(K_k)$, then $\forall j \in \{1 \ldots k\}$, $val_{K_k}(\langle 0, j\rangle) = k > 2$, hence $\exists j' \in \{1 \ldots k\}, \alpha(\langle 0, j\rangle) = \langle 0, j'\rangle$, and then $\forall i \in \{1 \ldots j\}, \alpha(\langle i, j\rangle) = \langle i, j'\rangle$, but $val_{K_k}(\langle j, j'\rangle) = val_{K_k}(\alpha(\langle j, j\rangle)) = val_{K_k}(\langle j, j\rangle) = 1$, hence $j = j'$. This for all $j$, hence $\alpha$ is the identity on the vertex set of $K_k$. Q.E.D.

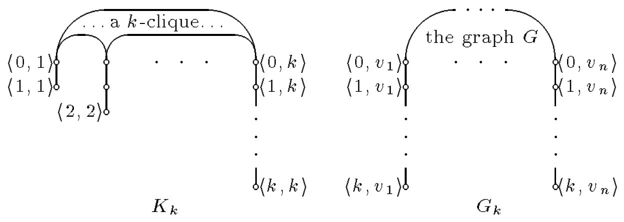


Figure 1: the graphs $K_k$ and $G_k$

We are now going to embed a $k$-clique in a graph $G = (V, E)$ by embedding $K_k$ in an extension $G_k$ of $G$, defined as follows: $G_k = (V_k, E_k)$ where $V_k = \{\langle i, v\rangle / i \in \{0 \ldots k\}, v \in V\}$ and

$$(\langle i, v\rangle, \langle i', v'\rangle) \in E_k \quad \Leftrightarrow \quad \begin{array}{l}(i = i' = 0 \wedge (v, v') \in E) \\ \vee (v = v' \wedge i + 1 = i')\end{array}$$

$\forall v \in V$, $val_{G_k}(\langle k, v\rangle) = 1$ and $\forall i \in \{1 \ldots k - 1\}$, $val_{G_k}(\langle i, v\rangle) = 2$. The subgraph of $G_k$ generated by $\{\langle 0, v\rangle / v \in V\}$ is isomorphic to $G$ (see figure 1).

THEOREM 2.4 Let $G = (V, E)$ be a graph and $k$ such that $2 < k \leq |V|$, $G$ contains a $k$-clique iff $Mon(K_k, G_k) \neq \emptyset$.

*Proof.* If $G$ contains a $k$-clique $v_1 \ldots v_k$, it is trivial to build a monomorphism $\alpha$ from $K_k$ to $G_k$ with $\alpha(\langle 0, j\rangle) = \langle 0, v_j\rangle$. Conversely, let $\alpha \in Mon(K_k, G_k)$, since $\forall j \in \{1 \ldots k\}, val_{G_k}(\alpha(\langle 0, j\rangle)) \geq k > 2$, then $\exists v_j \in V, \alpha(\langle 0, j\rangle) = \langle 0, v_j\rangle$. Let $G'$ be the subgraph of $G$ generated by $\{v_1, \ldots, v_k\}$; since $\alpha$ is injective, it has $k$ distinct vertices, and $\forall i, j \in \{1 \ldots k\}, i \neq j$, we have $(\langle 0, i\rangle, \langle 0, j\rangle) \in KE_k$, hence $(\langle 0, v_i\rangle, \langle 0, v_j\rangle) \in E_k$, and $(v_i, v_j) \in E$, which proves that $G'$ is a $k$-clique. Q.E.D.

COROLLARY 2.5 RCGM, and all the intermediate problems up to the problem of computing a non trivial general symmetry, are **NP**-complete.

# 3 Detecting $S$-symmetries

We now address the problem of the complexity of applying the $S$-symmetry rule in a non trivial way, which is a restriction of the previous one, and is therefore in **NP**. We first consider the simpler problem of detecting non trivial symmetries, i.e. let SYM be the problem $Sym_S \neq \mathcal{I}$, having $L$ and $S$ on $L$ as input. By lemma 2.1, the graph restriction of SYM is the problem of detecting a non trivial automorphism of a graph, known as GA. Hence GA $\propto_P$ SYM, and a polynomial algorithm for SYM would yield one for GA, which is very unlikely (GA is not even known to be in **co-NP**).

Before attempting to prove the converse, we should mention that SYM is not exactly the problem we need to solve: we are more interested in the associated search problem S-SYM (to compute a non trivial symmetry if there is one). But the search problem associated to GA, say S-GA, may not be polynomially equivalent to GA.

It will not be possible to reproduce the argument of the previous section, where the **NP**-completeness of the decision problem made the associated search problem polynomially equivalent to it. Hence we have to carefully distinguish the different problems we consider, beginning with CSYM: given $L$, $S$ on $L$ and a clause $C$ on $L$, is there a $\sigma \in Sym_S$ such that $\sigma(C) \neq C$? The associated search problem S-CSYM is the one needed for non trivial applications of the $S$-symmetry rule. The graph restriction of CSYM will be noted CGA: given $G = (V, E)$ and $V' \subset V$, is there a $\alpha \in Aut(G)$ such that $\alpha(V') \neq V'$?

We will frequently refer to the following problem, noted GI: given two graphs, are they isomorphic? This is a well-known **NP** problem that seems to be *non* **NP**-complete (hence the same holds for GA, since GA $\propto_P$ GI). There are quite a lot of problems polynomially equivalent to GI, called *isomorphism-complete*, among which the associated counting problem (how many isomorphisms between two graphs, see [Mat79]), bringing evidence for the non-**NP**-completeness of GI. For deeper evidence, see [KST92]. Other isomorphism-complete problems of more direct interest to us refer to group-theoretic notions on graph automorphisms, or more precisely on the permutation group $Aut(G)$, notions we therefore have to introduce.

Given a permutation group $\mathcal{G}$ on a permutation domain $X$, the *orbit* of $x \in X$ is $x^{\mathcal{G}} = \{y \in X / \exists \varphi \in \mathcal{G}, x\varphi = y\}$ ($x\varphi$ is standard notation for $\varphi(x)$, as well as $\varphi\psi$ for $\psi \circ \varphi$). The relation $\{\langle x, y\rangle / x \in y^{\mathcal{G}}\}$ is an equivalence relation, and the orbits form a partition of $X$, noted $\mathcal{P}art(\mathcal{G})$. Two literals $l, l'$ are *symmetric* if $l' \in l^{Sym_S}$. To any problem $\mathcal{P}$ for which the set of solution forms a permutation group $\mathcal{G}$, we associate the problem O-$\mathcal{P}$ of computing $\mathcal{P}art(\mathcal{G})$.

We address the problem of establishing as precisely as possible the complexity classes of the problems associated with SYM. The graph restriction trivially yields S-GA $\propto_P$ S-SYM, O-GA $\propto_P$ O-SYM and CGA $\propto_P$ CSYM. Apart from SYM and S-SYM, we are going to show that GI polynomially reduces to all the other problems associated with SYM. First, it is proved in [Mat79] that GI $\propto_P$ O-GA, hence we directly obtain GI $\propto_P$ O-SYM.

Consider the problem 1R-GA (GA with one restriction): given a graph $G = (V, E)$ and $v \in V$, is there a $\alpha \in Aut(G)$ such that $\alpha(v) \neq v$? It is proved in [Lub81] that GI $\propto_P$ 1R-GA. But it is trivial to see that 1R-GA $\propto_P$ CGA, since 1R-GA is exactly the restriction of CGA to the case where $|V'| = 1$. Hence GI $\propto_P$ CSYM $\propto_P$ S-CSYM.

In order to prove these relations in the reverse direction, we need to establish a kind of converse to lemma 2.1 (restricted to $G = G'$). Let $S$ be a set of clauses, we first consider the set of literals $L_S$ defined as the smallest one for $S$ that is complete for complementarity: $\forall l \in L_S, l^c \in L_S$. Clearly, $L_S$ can be computed in polynomial time in the length of $S$. For any literal $l$, $l^+$ refers to $l$ stripped of its possible negation, and $l^-$ to $(l^+)^c$.

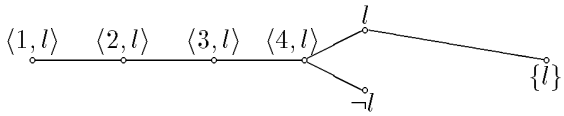


Figure 2: the graph $G_S$ for $S = \{\{l\}\}$

We now consider the graph[3] $G_S = (V_S, E_S)$ such that $V_S = L_S \cup S \cup N$ with $N = \{\langle i, l\rangle / l \in L_S^+, i \in \{1 \dots 4\}\}$ and

$$\begin{aligned}(a, b) \in E_S \quad \Leftrightarrow \quad & (a = \langle i, l\rangle \wedge b = \langle i+1, l\rangle) \\ & \vee(a = \langle 4, l\rangle \wedge b \in \{l, \neg l\}) \\ & \vee a \in b \in S\end{aligned}$$

We have $\forall l \in L_S^+, val_{G_S}(\langle 1, l\rangle) = 1$, $val_{G_S}(\langle 2, l\rangle) = val_{G_S}(\langle 3, l\rangle) = 2$, and $val_{G_S}(\langle 4, l\rangle) = 3$ (see figure 2).

Lemma 3.1 $\forall \alpha \in Aut(G_S), \forall l \in L_S^+$, we have $\alpha(l) \in L_S$ and $\forall i \in \{1 \dots 4\}$, $\alpha(\langle i, l\rangle) = \langle i, \alpha(l)^+\rangle$.

*Proof.* Since $val_{G_S}(\alpha(\langle 1, l\rangle)) = 1$, the first possibility is $\alpha(\langle 1, l\rangle) = l'$ with $l' \in L_S$ and $\forall C \in S, l' \notin C$. Hence $\alpha(\langle 2, l\rangle) = \langle 4, l'^+\rangle$, which is impossible since the valences of these vertices are different. The second possibility is $\alpha(\langle 1, l\rangle) = C = \{l'\} \in S$, hence $\alpha(\langle 2, l\rangle) = l'$, which therefore should not appear in any other clause, and hence $\alpha(\langle 3, l\rangle) = \langle 4, l'^+\rangle$, which is impossible.

The only possibility is therefore $\alpha(\langle 1, l\rangle) = \langle 1, l'\rangle$ (realized by $\alpha = Id_{V_S}$ with $l' = l$), hence $\alpha(\langle i, l\rangle) = \langle i, l'\rangle$ for $i = 1 \dots 4$, and either $\alpha(l) = l'$ or $\alpha(l) = l'^c$. In both cases $l' = \alpha(l)^+$. Q.E.D.

For all $\alpha \in Aut(G_S)$, we note $\alpha^*$ the restriction of $\alpha$ on $L_S$. We then have:

Lemma 3.2 $\forall \alpha \in Aut(G_S)$, then $\forall l \in L_S, \alpha^*(l^c) = \alpha^*(l)^c$ and $\forall C \in S$, $\widehat{\alpha^*}(C) = \alpha(C)$.

*Proof.* $\forall l \in L_S$, let $l' = \alpha(l^+) \in L_S$ from lemma 3.1, and also $\alpha(\langle 4, l^+\rangle) = \langle 4, l'^+\rangle$. Since $\alpha(\langle 3, l^+\rangle) = \langle 3, l'^+\rangle$, and $\alpha$ is bijective, then $\alpha(l^+) \in \{l'^+, l'^-\} = \{l', l'^c\}$, and $\alpha(l^-) \in \{l', l'^c\} \backslash \{\alpha(l^+)\}$. Hence $\alpha^*(l^c) = \alpha(l^c) = \alpha(l)^c = \alpha^*(l)^c$.

Hence $\widehat{\alpha}(N) = N$ (lemma 3.1) and $\widehat{\alpha}(L_S) = L_S$. Hence $\forall C \in S, \alpha(C) \in S$, and $\forall l \in L_S$, $\alpha(l) \in \alpha(C) \Leftrightarrow (\alpha(l), \alpha(C)) \in E_S \Leftrightarrow (l, C) \in E_S \Leftrightarrow l \in C$, hence $\alpha(C) = \{\alpha(l) / l \in C\} = \widehat{\alpha^*}(C)$. Q.E.D.

Theorem 3.3 The function $^*$ is an isomorphism from $Aut(G_S)$ onto $Sym_S$.

*Proof.* By lemma 3.2, $^*$ is into $Sym_S$: $\forall \alpha \in Aut(G_S)$, $\alpha^*$ is a complementarity preserving permutation of $L_S$, and $\widehat{\widehat{\alpha^*}}(S) = \widehat{\alpha}(S) = S$.

$^*$ is onto $Sym_S$: for $\sigma \in Sym_S$, let $\alpha : V_S \to V_S$ such that $\forall l \in L_S^+, \forall i \in \{1 \dots 4\}, \alpha(\langle i, l\rangle) = \langle i, \sigma(l)^+\rangle$, and

[3] We cannot use the graph construction from [Bas94], although it has similarities with ours, since it is unable to take account of the double negation law.

$\forall l \in L_S$, $\alpha(l) = \sigma(l)$ (hence $\alpha^* = \sigma$), and $\forall C \in S$, $\alpha(C) = \widehat{\sigma}(C)$. Of course, $\alpha \in Aut(G_S)$: edges on $N, L_S$ are trivially preserved, and $\forall l \in L_S, \forall C \in S$, $(l, C) \in E_S \Leftrightarrow l \in C \Leftrightarrow \sigma(l) \in \widehat{\sigma}(C) \Leftrightarrow \alpha(l) \in \alpha(C)$ $\Leftrightarrow (\alpha(l), \alpha(C)) \in E_S$.

$^*$ is an homomorphism: $\forall \alpha, \beta \in Aut(G_S), \forall l \in L_S$, we have $l(\alpha\beta)^* = l\alpha\beta = l\alpha^*\beta = l\alpha^*\beta^*$ (since $l\alpha^* \in L_S$), hence $(\alpha\beta)^* = \alpha^*\beta^*$.

$^*$ is an isomorphism: let $\alpha \in Aut(G_S)$ such that $\alpha^* = Id_{L_S}$, then $\forall \langle i, l\rangle \in V_S, \alpha(\langle i, l\rangle) = \langle i, \alpha(l)\rangle = \langle i, l\rangle$, and $\forall C \in S$, by lemma 3.2, we have $\alpha(C) = \widehat{\alpha^*}(C) = C$, hence $\alpha = Id_{V_S}$. This proves $Ker^* = \mathcal{I}$. Q.E.D.

## 4 Generating $S$-symmetries

An immediate consequence of theorem 3.3 is SYM $\propto_P$ GA, since the graph $G_S$ can be constructed in polynomial time in the length of $S$, and $Sym_S$ is trivial iff $Aut(G_S)$ is trivial. Hence SYM $\equiv_P$ GA. Furthermore, we also have S-SYM $\propto_P$ S-GA since for all $\alpha \in Aut(G_S) \backslash \mathcal{I}$, the non trivial $S$-symmetry $\alpha^*$ can be computed in polynomial time. Hence S-SYM $\equiv_P$ S-GA.

Similarly, we could prove O-SYM $\propto_P$ O-GA, and use O-GA $\propto_P$ GI from [Mat79]. However, the reduction in [Mat79] involves $\mathcal{O}(n^2)$ calls to GI (where $n$ is the number of vertices), and therefore does not yield a tractable algorithm for O-SYM. In this section we focus our attention on algorithmic issues, in order to provide realistic reductions from problems for which realistic algorithms exist. We will especially consider the problem of generating the group $Sym_S$, in a particularly useful way to be defined below.

### 4.1 Elements of computational group theory

The material in this section is mainly taken from [Hof81], with almost the same notations. Let $\mathcal{G}$ be a group with identity $e$, and $\mathcal{H}$ a subgroup of $\mathcal{G}$, noted $\mathcal{H} < \mathcal{G}$. For $a \in \mathcal{G}$, $\mathcal{H}a = \{ha / h \in \mathcal{H}\}$ is a *right coset* of $\mathcal{H}$ in $\mathcal{G}$. It is known that $|\mathcal{H}a| = |\mathcal{H}|$, and that the right cosets of $\mathcal{H}$ in $\mathcal{G}$ form a partition of $\mathcal{G}$, which yields Lagrange's theorem: if $\mathcal{G}$ is finite, then $\exists k \in I\!N, |\mathcal{G}| = k|\mathcal{H}|$. Let $a_1 \ldots a_k \in \mathcal{G}$ such that $\{\mathcal{H}a_i / i = 1 \ldots k\}$ is the partition of right cosets of $\mathcal{H}$ in $\mathcal{G}$ and $i \neq j \Rightarrow \mathcal{H}a_i \neq \mathcal{H}a_j$, then $\{a_1 \ldots a_k\}$ is a *complete right transversal* of $\mathcal{H}$ in $\mathcal{G}$. Since $\mathcal{H} = \mathcal{H}e$ is a right coset, then $\exists i / \mathcal{H}a_i = \mathcal{H}$, and $\{e\} \cup \{a_1 \ldots a_k\} \backslash \{a_i\}$ is also a complete right transversal of $\mathcal{H}$ in $\mathcal{G}$: we can always choose to include the identity in complete right transversals.

Given a subset $A \subset \mathcal{G}$, the subgroup *generated* by $A$, noted $<A>$, is the smallest subgroup of $\mathcal{G}$ containing $A$; then $A$ is a *generator set* for $<A>$. A *matriochka* in $\mathcal{G}$ is a finite sequence of subgroups $\mathcal{G}_r < \ldots < \mathcal{G}_1 < \mathcal{G}_0$ such that $\mathcal{G}_r = \mathcal{I}$ and $\mathcal{G}_0 = \mathcal{G}$. For $i \in \{1 \ldots r\}$, if $U_i$ is a complete right transversal of $\mathcal{G}_i$ in $\mathcal{G}_{i-1}$, then $\forall a \in \mathcal{G}_{i-1}, \exists! b_i \in U_i$ such that $a \in \mathcal{G}_i b_i$, hence such that $ab_i^{-1} \in \mathcal{G}_i$. By induction we see that $\forall a \in \mathcal{G}$, $\exists! b_1 \in U_1$, $\ldots, \exists! b_r \in U_r$ such that $ab_1^{-1} \ldots b_r^{-1} = e$, hence $a = b_r \ldots b_1$. $\bigcup_{i=1}^r U_i$ is therefore a computationally useful generator set for $\mathcal{G}$, provided $r$ and the $U_i$'s are small.

If $\mathcal{G}$ is a permutation group on $X$ finite, and $Y \subset X$, the *pointwise stabilizer* of $Y$ in $\mathcal{G}$ is $\mathcal{G}_{[Y]} = \{\varphi \in \mathcal{G} / \forall y \in Y,\ y\varphi = y\}$. $Y$ is a *base* of $\mathcal{G}$ if $\mathcal{G}_{[Y]} = \mathcal{I}$. It is not difficult to show that $\forall Y' \subset Y, \mathcal{G}_{[Y]} < \mathcal{G}_{[Y']}, \mathcal{G}_{[\emptyset]} = \mathcal{G}$ and $\mathcal{G}_{[X]} = \mathcal{I}$. Hence to any linear ordering $(x_1, \ldots, x_n)$ of the elements of $X$ we can associate the matriochka $\mathcal{G}^{(n)} < \ldots < \mathcal{G}^{(0)}$, where $\mathcal{G}^{(i)} = \mathcal{G}_{[\{x_1 \ldots x_i\}]}$. Let $U_i$ be a complete right transversal of $\mathcal{G}^{(i)}$ in $\mathcal{G}^{(i-1)}$ for $i = 1 \ldots n$.

$\forall \varphi, \psi \in \mathcal{G}^{(i-1)}$, we have $x_i\varphi = x_i\psi \Leftrightarrow x_i\varphi\psi^{-1} = x_i$ $\Leftrightarrow \varphi\psi^{-1} \in \mathcal{G}^{(i)} \Leftrightarrow \varphi \in \mathcal{G}^{(i)}\psi$. Hence if $\psi, \psi' \in U_i$, then $\psi \neq \psi' \Leftrightarrow x_i\psi \neq x_i\psi'$. Since $x_i\psi \in \{x_i \ldots x_n\}$, there are at most $n - i + 1$ possibilities, hence $|U_i| \leq n - i + 1$, and $|\bigcup_{i=1}^n U_i| \leq \frac{n(n+1)}{2}$; we have at most $\mathcal{O}(n^2)$ generators for $\mathcal{G}$, and elements of $\mathcal{G}$ can be uniquely expressed as compositions of at most $n$ generators.

A *representation matrix* is a $n \times n$ matrix $\mathcal{M}$ such that $\mathcal{M}[i,j]$ either is $\mathsf{empty}$, or if $i < j$, $\mathcal{M}[i,j]$ is a permutation of $X$ such that $\forall i' < i, x_{i'}\mathcal{M}[i,j] = x_{i'}$ and $x_i\mathcal{M}[i,j] = x_j$, or if $i = j$, $\mathcal{M}[i,j]$ is $Id_X$. Note that $\mathcal{M}$ is triangular. $\mathcal{M}$ then *represents* the set $T_1^{\mathcal{M}}$, where $T_k^{\mathcal{M}} = \{\psi\mathcal{M}[k,j] / \psi \in T_{k+1}^{\mathcal{M}}, j \in \{k \ldots n\}, \mathcal{M}[k,j] \neq \mathsf{empty}\}$ and $T_n^{\mathcal{M}} = \{\mathcal{M}[n,n]\} = \mathcal{I}$; permutations are composed from the $n^{th}$ line to the first.

For any permutation $\varphi$ of $X$, $\varphi \in T_1^{\mathcal{M}} \Leftrightarrow \exists \psi \in T_2^{\mathcal{M}}$, $\exists j \in \{1 \ldots n\}$, $\varphi = \psi\mathcal{M}[1,j]$. But $x_1\varphi = x_1\psi\mathcal{M}[1,j] = x_1\mathcal{M}[1,j] = x_j$, hence $\varphi \in T_1^{\mathcal{M}} \Leftrightarrow x_1\varphi = x_j, \mathcal{M}[1,j] \neq \mathsf{empty}$ and $\varphi\mathcal{M}[1,j]^{-1} \in T_2^{\mathcal{M}}$. Since $\varphi\mathcal{M}[1,j]^{-1}$ can be computed in time $\mathcal{O}(n)$, then by iterating from $T_1^{\mathcal{M}}$ to $T_n^{\mathcal{M}}$ we have a membership test in time $\mathcal{O}(n^2)$ for the set of permutations represented by $\mathcal{M}$.

Coming back to the matriochka $\mathcal{G}^{(i)}$ and the right transversals $U_i$, we build the following $n \times n$ matrix $\mathcal{M}^{\mathcal{G}}$:

$$\mathcal{M}^{\mathcal{G}}[i,j] = \begin{cases} \psi & \text{if } \psi \in U_i \text{ and } i\psi = j > i \\ Id_X & \text{if } i = j \\ \mathsf{empty} & \text{otherwise} \end{cases}$$

From what precedes, one easily proves that $\mathcal{M}^{\mathcal{G}}$ is a representation matrix, and that it represents $\mathcal{G}$. Remember that $\mathcal{M}^{\mathcal{G}}$ depends on the order $x_1 \ldots x_n$ of $X$, and on the $U_i$'s. It can also be proved that such a matrix can be computed in polynomial time from any generator set for $\mathcal{G}$ (see [Hof81]).

### 4.2 Representation matrix for $Sym_S$

We now consider the problem MG-SYM: given $S$, compute a representation matrix for $Sym_S$. From the previous remark, such a matrix can be computed in polynomial time once a generator set for $Sym_S$ is available, if its length is polynomial in the length of $S$. Hence MG-SYM $\propto_P$ G-SYM, where G-SYM is the problem of computing such a polynomial length generator set for $Sym_S$. The graph restriction G-GA of G-SYM is well-known: it is proved in [Mat79] that G-GA $\propto_P$ GI, yielding a generator set of $Aut((V,E))$ of length $\mathcal{O}(|V|^3)$. Hence

COROLLARY 4.1 G-SYM $\propto_P$ G-GA

*Proof.* Given a set of clauses $S$ of length $n$, we build the graph $G_S = (V_S, E_S)$ in time $\mathcal{O}(n)$. G-GA yields a generator set $A$ of $Aut(G_S)$ of length $\mathcal{O}(|V_S|^3)$, hence $A^*$ has length $\mathcal{O}(n^3)$ and can be computed in time $\mathcal{O}(n^3)$. Since $^*$ is an isomorphism from $Aut(G_S)$ onto $Sym_S$ (theorem 3.3), it is easily shown that $A^*$ is a generator set for $Sym_S$, hence G-SYM is solved in polynomial time with one call to G-GA. Q.E.D.

As a consequence, we have MG-SYM $\propto_P$ GI. In order to prove that O-SYM and S-CSYM are isomorphism-complete, there remains to show that O-SYM $\propto_P$ MG-SYM and S-CSYM $\propto_P$ MG-SYM, which will emphasize the usefulness of representation matrices.

### 4.3 Using the matrix

For sake of generality, we show how to compute $\mathcal{P}art(\mathcal{G})$ for any permutation group $\mathcal{G}$ on $X = \{x_1 \ldots x_n\}$, from a representation matrix $\mathcal{M}^{\mathcal{G}}$ (corresponding to the matriochka $\mathcal{G}^{(i)}$). The algorithm is given in 4 steps; the input is $X, \mathcal{M}^{\mathcal{G}}$, the output is a partition $\mathcal{P}$ of $X$.

1. initialize a $n \times n$ boolean matrix $\mathcal{B}$ to $\mathsf{false}$
2. $\forall i \in \{1 \ldots n\}, \forall j, p \in \{i \ldots n\}$,
   if $x_q = x_p \mathcal{M}^{\mathcal{G}}[i,j]$ then $\mathcal{B}[p,q] \leftarrow \mathsf{true}$
3. $\mathcal{P} \leftarrow \{\{x_1\} \ldots \{x_n\}\}$
4. $\forall p, q \in \{1 \ldots n\}$, if $\mathcal{B}[p,q]$, let $O, O' \in \mathcal{P}$ such that $x_p \in O, x_q \in O'$ in $\mathcal{P} \leftarrow \mathcal{P} \backslash \{O, O'\} \cup \{O \cup O'\}$

THEOREM 4.2 After step 4, $\mathcal{P} = \mathcal{P}art(\mathcal{G})$.

*Proof.* Notice that after step 2, $\forall p, q \in \{1 \ldots n\}$, $\mathcal{B}[p,q] \Leftrightarrow \exists i, j \in \{1 \ldots n\}$ such that $x_p \mathcal{M}^{\mathcal{G}}[i,j] = x_q$ (which implies that $x_p \in x_q^{\mathcal{G}}$).

We prove inductively that $\mathcal{P}$ is a refinement of $\mathcal{P}art(\mathcal{G})$: this is true for the initial value of $\mathcal{P}$ in step 3, and, in step 4, if $\exists x, x' \in X/O \subset x^{\mathcal{G}}, O' \subset x'^{\mathcal{G}}$ (induction hypothesis), then $O \cup O' \subset x^{\mathcal{G}} = x'^{\mathcal{G}}$ since $x_p \in x^{\mathcal{G}}, x_q \in x'^{\mathcal{G}}$ and $\mathcal{B}[p,q]$ holds.

Conversely, $\mathcal{P}art(\mathcal{G})$ is a refinement of $\mathcal{P}$: $\forall x \in X$, let $O \in \mathcal{P}$ such that $x \in O$. $\forall y \in x^{\mathcal{G}}$, $\exists \varphi \in \mathcal{G}/ y\varphi = x$, and then $\exists! j_1 \ldots j_n \in \{1 \ldots n\}$ such that $\varphi = \mathcal{M}^{\mathcal{G}}[n, j_n] \ldots \mathcal{M}^{\mathcal{G}}[1, j_1]$. Let $y_n = y$ and $y_{i-1} = y_i \mathcal{M}^{\mathcal{G}}[i, j_i]$ for $i = n \ldots 1$. Let $O_i \in \mathcal{P}$ such that $y_i \in O_i$ for $i = n \ldots 0$. Then $\forall i \in \{1 \ldots n\}$, if $y_i = x_p, y_{i-1} = x_q$ then $\mathcal{B}[p,q]$ holds, hence $O_i = O_{i-1}$ (by step 4). But $y_0 = x$, hence $O_n = O$, and $y \in O$. We have proved that $x^{\mathcal{G}} \subset O$. Q.E.D.

It is easily seen that the algorithm is in $\mathcal{O}(n^3)$, hence we can conclude that O-SYM $\propto_P$ MG-SYM. We now give an algorithm for S-CSYM, in the same general setting: given $X$ and $\mathcal{M}^{\mathcal{G}}$ as above, and $Y \subset X$, it computes a $\psi \in \mathcal{G}$ such that $Y\psi \neq Y$, if there is one.

1. $b \leftarrow \mathsf{false}$
2. $\forall i < j \in \{1 \ldots n\}$,
   if $Y\mathcal{M}^{\mathcal{G}}[i,j] \neq Y$ then $b \leftarrow \mathsf{true}; \psi \leftarrow \mathcal{M}^{\mathcal{G}}[i,j]$

THEOREM 4.3 After step 2, if $b$ then $\psi \in \mathcal{G} \wedge Y\psi \neq Y$, else $\forall \varphi \in \mathcal{G}$, $Y\varphi = Y$.

*Proof.* The case $b = \mathsf{true}$ is trivial. If $b = \mathsf{false}$, then $\forall i < j \in \{1 \ldots n\}$, $Y\mathcal{M}^{\mathcal{G}}[i,j] = Y$, and we easily obtain $\forall \varphi \in \mathcal{G}, Y\varphi = Y$. Q.E.D.

This algorithm is also in $\mathcal{O}(n^3)$, hence S-CSYM $\propto_P$ MG-SYM. As a consequence, the problems C-SYM, S-CSYM, O-SYM and MG-SYM are isomorphism-complete. Remember that SYM $\equiv_P$ GA and S-SYM $\equiv_P$ S-GA, and of course GA $\propto_P$ S-GA $\propto_P$ GI. In the sequel, we give some hints on solving MG-SYM as efficiently as possible, and then analyse the use of these different problems in automated deduction.

### 4.4 Algorithmic aspects

As shown previously, a representation matrix for $Sym_S$ can be computed from a generator set for $Aut(G_S)$. It is therefore possible to use known algorithms for general graphs, some of which can be found in [Hof81]. More precisely, [Hof81] contains an algorithm for generating the automorphisms of *labelled* graphs: i.e. graphs with labels attached to vertices, and only label-preserving automorphisms are considered. Clearly, the labels of a labelled graph can be given as a partition of the vertices of an unlabelled graph. The orbit partition of label-preserving automorphisms is a refinement of this partition. Given any graph $G = (V, E)$, and a partition $\mathcal{P}$ of $V$ such that $\mathcal{P}art(Aut(G))$ is a refinement of $\mathcal{P}$ (or, say, $\mathcal{P}$ is *compatible* with $G$), then any $\alpha \in Aut(G)$ is label-preserving (or, say, $\mathcal{P}$-preserving), and we can therefore use the algorithm in [Hof81] to generate $Aut(G)$.

A possible value for $\mathcal{P}$ is $\{V\}$. For $G_S$, we can take $\mathcal{P} = \{N, L_S, S\}$. The interest of providing a $\mathcal{P}$ as fine as possible is to reduce the search space. Indeed, the algorithm in [Hof81] has worst case complexity $\mathcal{O}(n^6(k!)^6(n+k^2))$, where $n = |V|$ and $k = max\{|O|/O \in \mathcal{P}\}$, producing a generator set $K$ such that $|K| \leq n^2(k!)^2$, which can be transformed in a representation matrix in time $\mathcal{O}(|K|n^2 + n^6)$. The number of lines of this matrix can be reduced by considering that $L_S^+$ is a base of $Sym_S$, hence any ordering of $L_S$ where positive literals precede negative ones leads to a matrix where the lines corresponding to negative literals only contain $Id_{L_S}$, and can therefore be removed.

It should be mentioned that using labels makes it possible to simplify $G_S$. Consider the initial partition $\mathcal{P}_S = \{L_S, S\}$ and the graph $G'_S = (V'_S, E'_S)$ defined as $V'_S = L_S \cup S$ and

$$(a,b) \in E'_S \quad \Leftrightarrow \quad (a, b \in L_S \wedge a = b^c) \\ \vee a \in b \in S$$

Then $^*$ is an isomorphism from the group of $\mathcal{P}_S$-preserving automorphisms of $G'_S$ onto $Sym_S$ (we leave this to the reader). We will therefore say that a partition $\mathcal{P}'$ is $\mathcal{P}$-*compatible* with $G$ if the partition orbit of the group of $\mathcal{P}$-preserving automorphisms of $G$ is a refinement of $\mathcal{P}'$.

One natural way of partitioning vertices is based on valences: since $\forall \alpha \in Aut(G), \forall v \in V, val_G(\alpha(v)) = val_G(v)$, the valences *are* labels, and any partition $\mathcal{P}$ can be refined by splitting any $O \in \mathcal{P}$ into $O_1 \ldots O_p$ such that $\forall v \in O_j, val_G(v) = j$, resulting in a partition $\mathcal{P}$-compatible with $G$.

This way of combining vertex partitioning principles can be defined in a more formal way. A *vertex classification* into a set $A$ is a function $\iota$ mapping any graph $G = (V, E)$ to a function $\iota_G : V \to A$. The partition of $V$ *induced* by $\iota_G$ is $\{\iota_G^{-1}(a)/a \in A\} \backslash \{\emptyset\}$; if this partition is compatible with $G$, for any graph $G$, then $\iota$ is a *v-invariant* (from [CK80]). Hence two v-invariants $\iota, \nu$ on $A, B$ can be combined by defining $\iota \times \nu$ as $\iota_G \times \nu_G$ for any $G = (V, E)$ and $\forall v \in V, (\iota_G \times \nu_G)(v) = \langle \iota_G(v), \nu_G(v) \rangle$; the result is a v-invariant into $A \times B$, and the corresponding partition is a refinement of both partitions for $\iota$ and $\nu$.

It is also possible to build a new v-invariant from any v-invariant $\iota$ into $A$, by counting how many adjacent vertices have the same value by $\iota_G$: more formally, $\iota^\circ$ is defined by: $\forall v \in V, \iota^\circ_G(v) = \lambda a \in A. |\iota_G^{-1}(v) \cap V_v|$, where $V_v$ is the set of vertices adjacent to $v$ in $G$. $\iota^\circ$ is then a v-invariant into $A \to \mathbb{N}$. The corresponding partition may not be a refinement of the partition for $\iota$, but one can combine $\iota \times \iota^\circ$, $\iota \times \iota^\circ \times \iota^{\circ\circ}$, $\iota \times \iota^\circ \times (\iota \times \iota^\circ)^\circ$, etc. There are many other v-invariants (see [CK80]), related to distance information, connectivity, or number of small cycles (small cliques) through a vertex (see also the star partition in [Wei71]). It is sometimes meaningful to combine v-invariants on a graph and on its complement.

As an example, *val* is a v-invariant, and using $val^\circ_{G'_S}$ (see [RC77]) corresponds to stating that two literals can only be symmetric if they appear an equal number of times in clauses of equal length, *and* that two clauses are in the same orbit only if they contain equal number of literals appearing in equal number of different clauses. This is a refinement of $val_{G'_S}$.

### 4.5 Symmetries in automated deduction

In this section, we will frequently refer to [BS94], which describes the only known implementation of Krishnamurty's rule. Although this is only a restriction of the $S$-symmetry rule (both on $C$ and $\sigma$), experimentations display drastic improvements on some examples.

This is not the place to describe precisely the proving methods used in [BS94]. First, we stress the fact that only symmetric literals are considered, which means that the $S$-symmetry rule is used with unit clauses. The advantage is that symmetry on literals can be easily tested once the orbit partition is known. However, the symmetry test performed in [BS94] only corresponds to $\mathcal{P}art(<\sigma>)$, (since only one $\sigma \in Sym_S$ is computed), which is a refinement of $\mathcal{P}art(Sym_S)$; hence this test is correct but not complete. This is the second restriction of the $S$-symmetry rule (on $\sigma$); a useless one, as shown below.

It may be argued that computing a single non trivial symmetry is simpler than computing $\mathcal{P}art(Sym_S)$, but we show on an example that this is not relevant. Consider the famous pigeonhole problem with $n$ pigeons ($n \geq 2$), it can be formalized with the following set of clauses $PH_n$:

- for $i \in \{1 \ldots n\}, P_{i,1} \vee \ldots \vee P_{i,n-1}$
- for $k \in \{1 \ldots n-1\}$, for $1 \leq i < j \leq n$, $\neg P_{i,k} \vee \neg P_{j,k}$

$P_{i,j}$ means "the pigeon $i$ is in the hole $j$". It is easy to show that $PH_n$-symmetries correspond to permutations on pigeons and on holes (independently). Hence $|Sym_{PH_n}| = n!(n-1)!$, which shows that $PH_n$ is highly symmetric, and is therefore a good example to illustrate the relevance of symmetry rules. Consequently, $Sym_{PH_n}$ has only two orbits: the set of positive literals, and the set of negative ones.

The strategy in [BS94] consists in choosing a clause as long as possible and using symmetries inside that clause. In $PH_n$, the longest clauses are the positive ones (if $n \geq 3$). If the symmetry $\sigma$ obtained by S-SYM is a permutation on pigeons, leaving holes unchanged, then no two literals in a positive clause are symmetric according to $\mathcal{P}art(<\sigma>)$, and there will be no application of the symmetry rule in this method (leading to an exponential proof!).

Hence one has to search for a symmetry satisfying a particular property, depending on the strategy for the symmetry rule. For example, one property required by the symmetry searching algorithm in [BS94] is that a cycle of the symmetry should contain a given literal, and be as long as possible. It is easy to show that solving such a problem allows to solve 1R-GA (by graph restriction), and is therefore at least as difficult as GI (moreover, restrictions on $\sigma$ easily lead to **NP**-complete problems, see [Lub81]). Hence the problem of maximizing the length of a cycle is at least as difficult as O-SYM, but yields only a refinement of $\mathcal{P}art(Sym_S)$. It should be mentioned that the algorithm in [BS94] uses a technique for reducing the search space which corresponds to the v-invariant $val_{G'_S}$. But the reason of its apparent efficiency is that it is allowed to backtrack only a fixed amount of time; it therefore has a polynomial time complexity (see [RC77]), and hence cannot be expected to solve S-SYM.

O-SYM is only one example of relevant problems for implementing symmetry rules, that is solvable in polynomial time from a representation matrix $\mathcal{M}$ for $Sym_S$.

There is also S-CSYM, and others may exist. Hence representation matrices are not only useful for proving complexity results, as done above, but also for implementing extensions of resolution with the symmetry rule. Other symmetric resolution methods, less restricted than the one in [BS94], may be designed, and probably may involve problems which can efficiently be solved if the representation matrix is available, hence making good use of the nice group structure of $Sym_S$.

## References


[Bas94] David Basin. A term equality problem equivalent to graph isomorphism. *Information Processing Letters*, 51:61–66, 1994.

[BS94] B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning*, 12:89–102, 1994.

[CK80] D. Corneil and D. Kirkpatrick. A theoretical analysis of various heuristics for the graph isomorphism problem. *SIAM Journal of Computing*, 9(2):281–297, May 1980.

[Gel59] H. Gelernter. A note on syntactic symmetry and the manipulation of formal systems by machine. *Information and Control*, 2:80–89, 1959.

[GJ79] M. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of* **NP**-*completeness*. Freeman, San Francisco, California, 1979.

[Hof81] C. Hoffmann. *Group theoretic algorithms and graph isomorphism*. LNCS 136, Springer-Verlag, 1981.

[Kri85] B. Krishnamurty. Short proofs for tricky formulas. *Acta Informatica*, 22:253–275, 1985.

[KST92] J. Köbler, U. Schöning, and J. Torán. Graph isomorphism is low for PP*. In A. Finkel and M. Jantzen, editors, *9th Annual Symposium on Theoretical Aspects of Computer Science*, pages 401–411. LNCS 577, Springer-Verlag, February 1992.

[Lub81] A. Lubiw. Some **np**-complete problems similar to graph isomorphism. *SIAM Journal of Computing*, 10(1):11–21, February 1981.

[Mat79] R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–132, March 1979.

[RC77] R. Read and D. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1:339–363, 1977.

[Sla94] John Slaney. The crisis in finite mathematics: automated reasoning as cause and cure. In A. Bundy, editor, *CADE-12, Nancy*, pages 1–13. Springer Verlag, LNAI 814, July 1994.

[Wei71] P. Weichsel. A note on assymmetric graphs. *Isreal Journal of Mathematics*, 10:234–243, 1971.