Integrating Low Level Symmetries into Reachability Analysis

Karsten Schmidt

Institut für Informatik, Humboldt–Universität zu Berlin
10099 Berlin, Germany
kschmidt@informatik.hu-berlin.de

Abstract. We present three methods for the integration of symmetries into reachability analysis. Two of them lead to maximal reduction but their runtime depends on the symmetry structure. The third one works always fast but does not always yield maximal reduction.

Keywords: symmetries, automorphisms, reachability analysis

1 Introduction

Symmetric structure yields symmetric behavior. Thus, symmetries can be employed to reduce the size of reachability graphs for analyzing particular properties [HJJJ84,Sta91,ID96] or for model checking [ES96,CEFJ96]. Instead of storing all states, only (representatives of) equivalence classes of states are stored. There are two major problems that need to be solved in the context of symmetries. Before starting reachability graph generation, we need to investigate the symmetries of the system. During graph generation, we need to decide repeatedly whether for a (recently generated) state an equivalent one has been explored earlier.

In the context of *high level Petri nets* or structured programming languages, we can use operations and relations on the data domains or replicated components to describe the symmetries symbolically [HJJJ84,Jen92]. Then the user can provide the description of the symmetries together with the system using terms (such as data operations) of the structured language. For instance, having the integer numbers as data type, one can map a state where a certain variable has value i to a state where that variable has value $i + 1$. For some classes of high level nets or other system descriptions, a symbolic description of a set of symmetries can be deduced automatically [CDFH90,Jun98,ID96], though this approach seems to be rather sensitive to the syntax used for system descriptions. Furthermore it is sometimes necessary to model the system very carefully to make the symmetries visible to the deduction tool [Chi98,CFG94]. For the decision of equivalence between states, one uses the symmetries to transform the current state into an equivalent one which is minimal with respect to some, say lexicographical, order. The transformed state is called *canonical representative*, is unique, and is used to represent its equivalence class. Currently, this approach is more or less restricted to symmetry groups that can be composed of full permutation groups and rotation groups over the involved data domains. As an efficient alternative, non–unique representatives have been proposed in [CEFJ96,ID96] which potentially lead to a larger graph and some problems for reachability tests on the reduced graph.

Using low level symmetries [Sta91][1] arbitrary symmetry groups can be handled in a uniform manner as graph automorphisms of the Petri net graph representation. There is no straightforward symbolic description of low level symmetries. Thus, calculation is the only way to get the information about symmetries. However, the algorithm proposed in [Sch97] and implemented in INA [RS97] is able to calculate polynomially large generating sets of (in worst case exponentially large) maximal symmetry groups in reasonable space and time. For instance, the symmetry group calculation for a net with 10000 vertices requires usually less than 5 minutes. For nets of that size, reachability analysis is faced with a lot of other challenges than symmetry calculation.

This paper surveys three solutions of the problem of using low level symmetries in reduced graph generation and reports experimental results.

With the present low level symmetry technology, we fill the following gaps left by the symbolic high level calculus:

– We can deal with systems where a structured representation is not available (for instance, translations from other formalisms into low level nets);
– We offer a fully automated approach to high level (structured) systems with small or medium size data domains (prototypes of larger systems) independently of the net inscription syntax;
– We can handle small or medium size systems having non–standard symmetry groups

2 Petri Nets

For the purpose of simplicity (in particular due to the immediate correspondence between net symmetries and graph automorphisms) , we present the approach for place/transition nets. However, it should not be difficult to transfer the results to other formalisms, whether Petri net based or not.

Definition 1 (Petri net). *A tuple $N = [P, T, F, W, m_0]$ is a Petri net iff P and T are finite, nonempty, and disjoint sets (of* places *and* transitions*), $F \subseteq (P \times T) \cup (T \times P)$ (the set of* arcs*), $W : (P \times T) \cup (T \times P) \longrightarrow \mathbb{N}$ such that $W([x, y]) > 0$ iff $[x, y] \in F$ (the arc* multiplicities*), and m_0 is a* state*, i.e. a mapping $m_0 : P \longrightarrow \mathbb{N}$.*

[1] throughout the paper, we use the term *low level* for the symmetry approach to place/transition nets, i.e. Petri nets with unstructured, uniform tokens. In contrast, high level symmetries rely on the ability to use data specific terms for symbolic description of symmetries.

Throughout the paper, we assume that places and transitions are totally ordered.

For a place or transition x, ${}^\bullet x = \{y \mid [y, x] \in F\}$ denotes the *pre–set* of x, and $x^\bullet = \{y \mid [x, y] \in F\}$ denotes its *post–set.*

Definition 2 (Transition relation). *We say that t can fire at a state m yielding state m' (written: $m@ > t >> m'$) iff for all $p \in P$, $m(p) \geq W([p, t])$ and $m'(p) = m(p) - W([p, t]) + W([t, p])$.*

If, for a given state m and transition t there exists a state m' such that $m@ > t >> m'$, then we say that t is enabled at m. We extend the transition relation to sequences of transitions. Define $m@ > e >> m$ for an arbitrary m and the empty sequence e, and $m@ > wt >> m'$ (w being a transition sequence and t a transition) iff there is an m^* such that $m@ > w >> m^*$ and $m^*@ > t >> m'$. If there is a transition sequence w such that $m@ > w >> m'$, we write $m@ > * >> m'$.

Definition 3 (Reachability graph). *A directed labeled graph is the reachability graph of a Petri net $N = [P, T, F, W, m_0]$ iff its set of vertices is the set of all reachable states, i.e. $\{m \mid m_0@ > * >> m\}$, and $[m, m']$ is an edge labeled with t iff $m@ > t >> m'$.*

3 Graph Automorphisms

Consider a directed graph $[V, E]$ (with a finite set V of vertices and a set $E \subseteq V \times V$ of edges) together with a mapping $\phi : V \cup E \longrightarrow C$ that assigns a *color* of a set C to every graph element.

Definition 4 (Graph automorphism). *A graph automorphism is a bijection $\sigma : V \to V$ of the vertices of a directed colored graph that respects adjacency and coloring, i.e.*

- *$e = [x, y] \in E$ iff $\sigma(e) = [\sigma(x), \sigma(y)] \in E$;*
- *$\phi(\sigma(z)) = \phi(z)$ for $z \in V \cup E$.*

The set of all automorphisms of a graph forms a group under composition and inversion. The identity is always an automorphism and serves as neutral element of the group. For the remainder of this section, consider an arbitrary graph and its automorphism group. There can be exponentially many automorphisms. However, there is always a generating set of at most $\frac{|V| \cdot (|V|-1)}{2}$ elements for the whole automorphism group. In the sequel we consider a rather well formed generating set that enjoys a regular structure though it is not necessarily of minimal size. The algorithm proposed in [Sch97] returns a generating set as described below. Assume a total ordering of V, i.e. $V = \{v_1, \ldots, v_n\}$. A well formed generating set G for all graph automorphisms consists of $|V|$ families $G_1, \ldots, G_{|V|}$. If, for $i \in \{1, \ldots, |V|\}$ and $j \in \{i, \ldots, |V|\}$, there exist automorphisms σ such that $\sigma(v_1) = v_1, \ldots, \sigma(v_{i-1}) = v_{i-1}$ and $\sigma(v_i) = v_j$ then family G_i contains exactly one of them. In other words, the elements of family G_i are equal to the identity on all vertices smaller than v_i and cover all possible images of v_i.

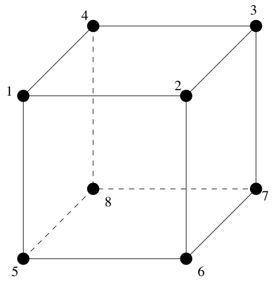

Fig. 1. 3 dimensional grid

Example 1. Consider the graph depicted in Fig. 1. Every drawn line between two vertices x and y corresponds to a pair $[x, y]$ and $[y, x]$ of edges. Table 1 lists a generating set for the 48 graph automorphisms that fits to the rules above. The generating set is not minimal. For instance, $\sigma_2 = \sigma_1 \circ \sigma_1$.

Table 1. Generating set of the grid automorphisms

Argument	Images												
	Family 1								Family 2			Family 3	
1	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	1	1	1	1	1
2	2	3	4	1	1	5	8	7	**2**	**4**	**5**	2	2
3	3	4	1	2	4	8	5	6	3	8	6	**3**	**6**
4	4	1	2	3	8	7	6	5	4	5	2	4	5
5	5	6	7	8	6	2	3	4	5	2	4	5	4
6	6	7	8	5	2	1	4	3	6	3	8	6	3
7	7	8	5	6	3	4	1	2	7	7	7	7	7
8	8	5	6	7	7	3	2	1	8	6	3	8	8
	id	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	id	σ_8	σ_9	id	σ_{10}

Proposition 1 (Generating set). *Every set of automorphisms that fits to the rules described above is a generating set for all automorphisms. In particular, every automorphism can be represented as the composition* $\sigma_1 \circ \cdots \circ \sigma_{|V|}$ *where* σ_i *belongs to family* G_i *of the generating set.*

Proposition 2 (Non–repetition). *If for all* i, σ_i *and* τ_i *are members of family* G_i *of a generating set as described above, then* $\sigma_1 \circ \cdots \circ \sigma_n = \tau_1 \circ \cdots \circ \tau_n$ *implies* $\sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n$.

These two propositions state that every automorphism can be generated in exactly one way as the composition of one member per family of the generating set. As a corollary, the product of the sizes of the families yields the size of the automorphism group. For instance, our grid has $8 \cdot 3 \cdot 2 = 48$ automorphisms.

Rotation groups (automorphism groups of ring–like graphs) have a generating set consisting of one family, permutation groups (symmetry groups of clique–like graphs) have generating sets where the size of subsequent families of the generating set decreases by one.

4 Petri Net Symmetries

A Petri net $N = [P, T, F, W, m_0]$ can be regarded as colored graph with $V = P \cup T$, $E = F$, $\phi(p) =$ `place` for $p \in P$, $\phi(t) =$ `transition` for $t \in T$, and $\phi(f) = W(f)$ for $f \in F$.

Definition 5 (Symmetry). *A Petri net symmetry is an automorphism of the underlying colored graph.*

We denote the symmetry group of a Petri net by Σ. A symmetry group Σ induces equivalence relations on the Petri net vertices as well as on the states.

Definition 6 (Equivalence of vertices/states). *Two vertices x and y in $P \cup T$ are equivalent with respect to Σ ($x \sim_\Sigma y$) iff there is a $\sigma \in \Sigma$ such that $\sigma(x) = y$. For a state m, let $\sigma(m)$ be the state satisfying, for all $x \in P \cup T$, $\sigma(m)(\sigma(x)) = m(x)$. A state m_1 is* equivalent *to a state m_2 with respect to Σ ($m_1 \sim_\Sigma m_2$) iff there is a $\sigma \in \Sigma$ such that $\sigma(m_1) = m_2$.*

$\sim_\Sigma$ is an equivalence relation. We denote the $\sim_\Sigma$ equivalence class containing some state m by $[m]_\Sigma$.

5 The Integration Problem

During generation of the reduced reachability graph, we want to merge equivalent states. As soon as a new state m is generated, we compare it against the set of already existing states M. If, among those states, there is one equivalent to m, we do not store m. Instead, we redirect all arcs to m to the equivalent state. Thus, the problem we are confronted with (and which we call the integration problem) is:

> *Given a symmetry group Σ, a set M of states, and a state m, decide whether there* exist *an $m' \in M$ and a $\sigma \in \Sigma$ such that $\sigma(m) = m'$; in the case that such an m' exists, return m'.*

Here, Σ is given by a well formed generating set. M is the set of already calculated states and is usually organized as a search structure (tree, hash table or whatsoever).

We study three approaches to the integration problem. The first two procedures implement one of the existential quantifiers appearing in the integration problem as a loop. The third method implements the canonical representative method to low level Petri nets.

6 Iterating the Symmetries

The integration problem quantifies a symmetry existentially. If we implement this quantification as a loop on all symmetries, we obtain the following brute force solution:

```
FOR ALL $\sigma \in \Sigma$ DO
    IF $\sigma^{-1}(m) \in M$ THEN
        RETURN yes;
    END;
END;
RETURN no;
```

The test $\sigma^{-1}(m) \in R$ is a standard operation on sets of states and can be efficiently implemented. So the costs of the procedure depend on the number of iterations of the outer loop which is, in the worst case (the case where m is not in M), equal to the size of the symmetry group. Due to the up to exponential size of symmetry groups, this approach does not work well. However, using decision trees for storing the states in M enables a nice reduction concerning the number of loops.

A decision tree treats states as strings and merges common prefixes. Fig. 2 depicts a decision tree storing the set $\{(1, 0, 0), (1, 0, 1), (1, 2, 2), (1, 3, 1)\}$.

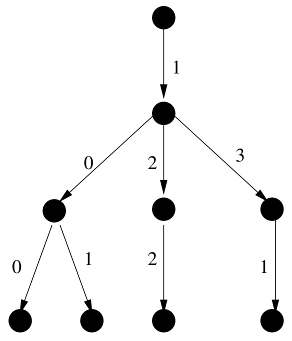

Fig. 2. Decision tree

If we find $\sigma^{-1}(m)$ in the decision tree, we exit the loop. Otherwise, there is some prefix of $\sigma^{-1}(m)$ that is not contained in the tree and the length i of this prefix is available through the search process. This means that any other state with the same prefix is not contained in M as well. However, the σ^{-1}–image of m on the first i places is determined by the generators of the first i families of the generating set. That is, every symmetry that is composed of the same elements of the first i families as σ, but different elements of the last $|P| - i$ families of the generating set, yields the same i–prefix as $\sigma^{-1}(m)$ and is consequently not contained in M. Using this observation, we can skip all such combinations of generators and continue with one where at least one of the first i ones is different. This method reduces significantly the number of iterations of the outer loop in the above code fragment.

Having implemented this iteration, we found out that the loop reduction was powerful enough to make reduced graph generation faster than complete iteration even if complete iteration runs on an explicitly stored list of *all* symmetries (i.e. re–initialization of the loop is just a jump to the next list element instead of composing a symmetry from generators). The current version of INA uses the generating set and the described loop reduction. Older versions of INA maintain an explicit list of all symmetries. The algorithm works quite well if the symmetry group is sparse. Unfortunately, larger symmetry groups require still too many iterations of the symmetry loop and graph generation tends to fail. Therefore, we need other integration technologies.

7 Iterating the States

In this section, we develop another solution of the integration problem where the outer loop iterates on states. The principal solution is:

```
FOR ALL $m' \in M$ DO
    IF $\exists\sigma : \sigma(m) = m'$ THEN
        RETURN yes;
    END;
END;
RETURN no;
```

For the test $\exists\sigma : \sigma(m) = m'$ we do not need to iterate the symmetries. The algorithm in [Sch97], otherwise used to compute the generating set, can as well be customized such that it computes a symmetry mapping m to m' (or recognize that such a symmetry does not exist).

Of course, iterating all states in the outer loop is unacceptable, since there are usually too many of them. Fortunately, the number of iterations can be reduced significantly when *symmetry respecting hash functions* are used (the use of hash functions has already been studied in [HJJJ84]). A hash function assigns an index $h(m)$ from a finite index set I to every state m. This way, the set of states M can be partitioned into $card(I)$ classes where two states are in the same class iff they have the same hash value. Simple hash techniques store each hash class separately such that other search techniques need to be applied only to the (hopefully small) hash classes rather than to the complete set of states.

Symmetry respecting hash functions assign equal hash values to equivalent states. Using such a function h, it is sufficient to iterate the subset of M corresponding to $h(m)$ rather than the whole M.

As a simple way to get a symmetry respecting hash function, one can use a weighted sum of component values. For this purpose, one assigns a number c_p to each place p and sets $h(m) = \sum_{p \in P} c_p \cdot m(p) \quad mod \quad k$ where k is the size of the hash table. In order to make this function symmetry respecting, one guarantees $c_p = c_{p'}$ for all equivalent places. The equivalence relation for places can be efficiently deduced from the generating set without iterating all symmetries. In fact, it is sufficient to start with a family of singleton sets $\{\{p\} \mid p \in P\}$ and then to build repeatedly the union of the set containing p with the set containing $\sigma(p)$ (for all $p \in P$ and all σ *of the generating set*). This task can be efficiently done by TARJAN's union/find algorithm [Tar75] and produces directly the equivalence classes. The implementation used for computing the examples at the end of the paper uses weighted component sums as hash function where the c_p are otherwise randomly chosen. Random choice appears to spread hash values well enough.

The proposed method depends on the number of states in M that share the same hash value. Besides the general uncertainties of hash functions, there are general problems with symmetry respecting hash functions for sparse symmetry groups. Consider Fig. 3 and assume that (maybe by some additional hidden net structure) the rotations along the ring are the only symmetries.

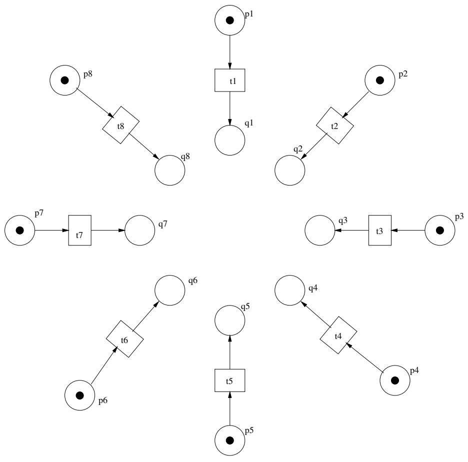

Fig. 3. Difficult situation for symmetry respecting hash functions

In this net, for every pair of outer places there is a symmetry mapping one to another. The same holds for inner places. However, there is no symmetry mapping an outer place to an inner place. Therefore, a hash function following our proposal assigns some weight c_o to all outer places and some weight c_i to all inner places. Consequently, the best symmetry respecting hash function would produce at most 9 different used hash values for the 36 states of the symmetrically reduced reachability graph of this net (the hash value depends only on *how many* inner places are marked, but equivalence depends on the distances between two marked inner places). In a ring of the same kind but length 20, the 52488 states of the reduced graph would share only 21 different values of any symmetry respecting hash function (in general, at least $\frac{2^n}{n}$ states share at most $n + 1$ hash values).

A situation as described above is typical for all rotation–like symmetry groups. Fortunately, this effect is due to the low number of symmetries (where the integration method of the previous section behaves well). If we use all permutations between the p_i (and corresponding mappings on the t_i and q_i) as symmetry group of the system depicted in Fig. 3, the problem does not appear since those states that share the same hash value become equivalent. Thus, the reduced reachability graph contains only one of them. Assigning, for example, weight 1 to the p_i and weight 2 to the q_i, we get 9 different hash values for the 9 states of the reduced reachability graph (in general, n values for n states). Thus, iterating the states of a hash class is a method for integrating symmetries that displays its best behavior with large symmetry groups. This complements the behavior of the method discussed in the previous section.

8 Canonical Representatives

This version of symmetry integration is inspired by [HJJJ84,ID96,CEFJ96]. The vector representation of states induces a (lexicographical) order between states: Let $m < m'$ iff there is a place p such that for for all places q ($q < p$), one has $m(q) = m'(q)$ while $m(p) < m'(p)$. Given this order, every equivalence class of states with respect to the symmetry has a least element, called the *canonical representative*. In the version of the integration problem we want to discuss now, only canonical representatives are added to M. Then, deciding the integration problem consists of transforming the new state m into its canonical representative $\overline{m}$ and checking whether $\overline{m}$ is contained in M. So, the transformation of arbitrary states into canonical representatives is the bottleneck of this method. Efficient solutions of the problem are restricted to few classes of symmetry groups (such as rotation or permutation groups).

The brute force implementation in absence of a symbolic description of the symmetries would be to iterate all symmetries and to compare their images of the given m. This is, however, not appropriate here since this method would not work at all for larger groups. Therefore, we try once more to benefit from the structure of the generating set introduced in Sect. 3, in particular from the fact that elements of larger families do not change values on the first places. Thus, we could try the following procedure to transform states (σ_{ij} is the j–th member of family G_i of the generating set, $\#_i$ is the size of family G_i):

```
**PROCEDURE** transform(m:state): state:
**VAR** globalmin,localmin: state;
    $i$,$j$:Nat;
**BEGIN**
    globalmin := $m$;
    **FOR** $i$ := 1 **TO** $n$ **DO** (* for all families *)
        localmin := globalmin;
        **FOR** $j$ := 1 **TO** $n$ **DO**
            **IF** $\sigma_{ij}$(globalmin) < localmin **THEN**
                localmin := $\sigma_{ij}$(globalmin);
            **END**
        **END**
        globalmin := localmin;
    **END**
    **RETURN** globalmin;
**END**.
```

This procedure has the advantage that it works in polynomial time in the size of the net. It has, however, one big disadvantage: it does not always produce canonical representatives! For an example, consider the grid example of Sect. 3. Assume for the purpose of simplicity that all vertices of the grid are places and transform the state $m = (1, 2, 2, 3, 3, 2, 1, 3)$. In the first family, only the identity and σ_6 produce 1 on the first place. $\sigma_6(m) = (1, 3, 3, 2, 2, 3, 1, 2)$. Thus, the minimal state produced by the first family is the state itself. In the second family, both σ_8 and σ_9 would produce 3 on the third place, thus m itself remains the minimal state. In the third family, σ_{10} does not change the state. Therefore the transform procedure would return m itself. However, applying σ_9 to $\sigma_6(m)$ leads to $(1, 2, 2, 2, 3, 3, 1, 3)$ which is smaller than m. Thus, the procedure does not always return canonical representatives. The way to the global minimum does not always lead via the minima at the intermediate steps of the algorithm.

The problem is limited to symmetry groups other than rotation and permutation groups. For rotation groups, we have a generating set consisting of a single family containing all symmetries. Thus, no symmetry is forgotten. For full permutation groups, the transform procedure performs a real sorting on the state. Thus, the global minimum is returned.

There are two ways out of the problem of non–minimality. The first one would be to replace the above transform procedure by one returning the actual canonical representative. However, the complexity of this problem is closely related to deciding graph isomorphism [CEFJ96,ES96] and therefore it is unlikely that there is a polynomial solution at all. Therefore we tried the other way which is to study which problems arise from non–unique representatives. The first observation is that a reduced graph using non–unique representatives can contain more than one element of some equivalence classes. Thus, the reduced graph is larger than a perfectly reduced one (i.e. one that contains exactly one representative per reachable class). On the other hand, most global properties do not depend on perfect reduction. In particular, the proof rules described in [Jen95] for boundedness, liveness, reversibility and other properties as well as model checking using formula respecting symmetries [ES96] are still valid. On the other hand, checking reachability of a given state m^* against a reduced reachability graph might be dangerous. It can happen that the transformed image of m^* is different from all representatives of its equivalence class in the reduced state space. Thus, not finding the image of m^* in M does not yet mean that the equivalence class of m^* is not represented in M. The problem can be circumvented in the following way: use the canonical representative method for generating the graph, but use one of the other methods to check reachability. This compromise should work well as long as only some states are to be tested for reachability.

9 Experiments

We have implemented all three methods in a new reachability oriented Petri net tool called LoLA (Low Level Analyzer). So reduced and full graph generation share the same code for all tasks not depending on the integration method.

ItSymm corresponds to the iteration of symmetries (Sect. 6), *ItState* to the iteration of states (Sect.7), and *CanRep* to the canonical representative method of Sect. 8. Times have been taken on a LINUX PC with 64MB RAM and a 400MHz Pentium processor. The times are written as <minutes>:<seconds>. We used a hash table of size 65536.

The first three tables show the behavior of symmetric reduction for different kinds of symmetry groups. We demonstrate the reduction in space, compared with full graph generation and give an impression concerning the running time. Furthermore we provide experimental evidence for the run time predictions we derived in the theoretical sections. There, we expected ItSymm to be fast on sparse groups but slow on large groups, we expected ItStates to be slow on sparse, but acceptable on large groups, and we expected CanRep to be fast anyway but to produce larger graphs. For these goals, small examples should be sufficient.

The first table concerns a deadlocking version of the n dining philosophers example. This is an example of a ring–like agents network.

Table 2. Results for the dining philosophers

	number of phil.		
	5	10	13
Places	25	50	65
Transitions	20	40	52
States in full graph	242	59048	$3^{13} - 1$
Edges in full graph	805	393650	?
Hash entries in full graph	242	38048	?
Time for full graph	0:00.085	0:05.9	?
Symmetries	5	10	13
States in red. graph	50	5933	122642
Edges in red. graph	165	39550	1062893
nonempty hash classes in red. graph	20	65	104
time for ItSymm	0:00.077	0:02	1:38
time for ItState	0:00.22	11:31	?
time for CanRep	0:00.077	0:01.1	0:33.5

The symmetry group is a rotation–like group. Thus, we would store all symmetries explicitly as our generating set. They are organized in a single family. Therefore, the canonical representative method *must* yield full reduction. The times of iteration of states and canonical representatives are acceptable. The long time for the iteration of states is due to the hashing problem discussed in Sect. 7 and makes the method unacceptable for this kind of net. Though the times for the reduced graphs include the calculation of the symmetries, they are smaller than the corresponding full graph generation. Note that the number of instances of the integration problem to be solved corresponds to the number of edges of the reduced graph.

The second example is a data access scheme where r readers can access some data concurrently while w writers access it in mutual exclusion (and excluded from the readers). The readers are not distinguishable from each other and the writers are not distinguishable from each other. Thus, the symmetry group is close to full permutation.

Table 3. Results for the data access scheme

	readers / writers			
	5/5	13/13	20/20	40/40
Places	31	79	121	241
Transitions	25	65	100	200
States in full graph	217	114857	$21 \cdot (2^{20} + 20)$	$41 \cdot (2^{40} + 40)$
Edges in full graph	770	905554	?	?
Hash entries in full graph	217	54952	?	?
Time for full graph	0:00.1	2:54	?	?
Generators of Symm.	20	156	380	1560
Families of gen. set	8	24	38	78
Symmetries	14400	$13! \cdot 13!$	$20! \cdot 20!$	$40! \cdot 40!$
States in red. graph	14	30	44	84
Edges in red. graph	82	470	1072	4142
nonempty hash classes in red. graph	14	30	44	84
time for ItSymm	0:00.64	> 60:00	?	?
time for ItState	0:00.2	0:05.8	0:33	9:44
time for CanRep	0:00.084	0:00.4	0:01.7	0:34
States generated by CanRep	17	41	62	122
Edges generated by CanRep	85	481	1090	4180

For this system, iteration of symmetries fails due to the huge number of symmetries even in cases where the full graph can be generated without problems. Iteration of states works for this system though it cannot reach the speed of iterating the symmetries for a rotation symmetry group. However, compared to the time needed for the complete graph we can be satisfied with the result. Observe that the number of used hash entries shows that every instance of the integration problem involves at most one attempt to calculate a symmetry. The fastest method is once more the canonical representative approach. However, the example shows that it produces slightly larger graphs. This is due to the interlacing of two permutation groups. This problem could be avoided by rearranging the order of places which would require knowing the symmetries in advance; but that is not the intended application area of the low level Petri net method. However, the advantage with respect to time could justify a few more states. Again, one of the perfectly reducing methods and the canonical representative method were able to outperform the full graph generation.

The third example is designed to study the behavior of our procedures on grid–like networks. For this purpose we arranged several identical agents in a grid network. The agents have two states. The second state can only be entered in mutual exclusion to the direct neighbors, controlled by a set of semaphores. We vary the number of dimensions of the grid and the number of agents per row (our running example would have dimension 3 and two agents per row).

Table 4. Results for the grid network

	dimensions / agents per row			
	2/5	3/3	4/2	5/2
Places	75	81	48	96
Transitions	50	54	32	64
States of full graph	55447	70633	734	?
Edges of full graph	688478	897594	5664	?
Time for full graph	0: 10	0:15	0:00.18	?
Generators	4	10	21	41
Nontrivial families	2	3	4	5
Symmetries	8	48	384	3840
States of red. graph	7615	2352	21	332
Edges of red. graph	94850	29912	172	4937
nonempty hash entries of red. graph	192	106	9	17
Time for ItSymm	0:03.8	0:03.7	0:00.18	0:26.6
Time for ItState	12:29	4:38	0:00.72	1:20
Time for CanRep	0:04.7	0:04.9	0:00.15	0:27
States gen. by CanRep	15138	10832	29	3032
Edges gen. by CanRep	188706	137345	234	44650

For this example, the a lot of symmetries can be skipped during the iteration of symmetries in ItSymm, so ItSymm is even able to outperform the canonical representative method (which has, in addition, the disadvantage of generating more states). The grid symmetries can be considered as a sparse symmetry group, which explains the bad behavior of ItState. The reason for the slow ItState can be found in the small number of used hash classes compared with the large number of states.

The grid example demonstrates the benefits of using low level net symmetries on non–standard symmetry groups. In the 3–dimensional case we have 48 symmetries. The size of the reduced graph is approximately 30 times smaller than the full one. Since every equivalence class of reachable states should appear in the reduced graph, and the size of an equivalence class is bounded by the number of symmetries, a reduced graph can never be smaller than the size of the full graph divided by the number of symmetries. As outlined in [CFG94,CDFH90], the symbolic symmetry approach based on rotation and permutation groups considers only a rotation (sub-)group of grid structures that consists only of four symmetries. Thus, instead of a reduction factor 30, a reduction factor smaller than 4 is achieved.

The symmetry approach has been criticized for its lack of practical feasibility. Extrapolation of the previous tables to larger examples supports this criticism. In particular, sparse symmetry groups do not yield sufficient reduction to keep pace with the growth of the reachability graph. However, things look different if we take into consideration that symmetries can be applied jointly with several other reduction techniques, for instance the stubborn set method [Val88]. Benefits of combined application have been reported earlier [Val91,DBDS93,EJP97]. We report results for our particular symmetry approach. Applying symmetries alone leads to memory overflow on all reported instances, not to mention full graphs.

Table 5. Stubborn sets versus stubborn sets with symmetries (for the dining philosophers)

	number of phil.			
	100	200	300	900
Places	500	1000	1500	4500
Transitions	400	800	1200	3200
States in full graph	$3^{100}-1$	$3^{200}-1$	$3^{300}-1$	$3^{900}-1$
Symmetries	100	200	300	900
States in symm./stubb. red. graph	299	599	899	2699
Edges in symm./stubb. red. graph	496	996	1496	4496
time for ItSymm+ stubborn	0:02	0:10	0:26	7: 00
States in stubb. red. graph	29702	119402	overflow	-
Edges in stubb. red. graph	39700	159400	-	-
Time for stubb. red. graph	0:05	1:08	-	-

In the grid example, stubborn sets do not reduce the number of states. Concerning the data access scheme (40 readers/40 writers), stubborn sets produce 121 states while combined application of stubborn sets and symmetries leads to 4 states, independently of the number of readers and writers.

10 Conclusion

We have presented three solutions of the integration problem (deciding whether for the current state there is a symmetrical one in the set of already computed states). All methods have their advantages and disadvantages. The advantage of the iteration of symmetries and iteration of states is that they yield a completely reduced graph. Unfortunately they work well only on sparse (iteration of symmetries) or large (iteration of states) symmetry groups. The canonical representative method is the fastest method that works for all kind of symmetries, but it often produces more than one representative for some equivalence classes of states. The canonical representative method cannot be used for testing the reachability of states. This problem can be repaired by using one of the other method for testing the reachability on a graph produced by the canonical representative method.

With the set of methods presented in this paper, low level symmetries are no longer a restricting factor for reduced reachability graph generation. In most cases, the remaining size of the reduced graph limited the analysis. This in turn is due to the limitation of the symmetry approach *as it* and not due to its *low level* version (low level symmetries do not produce larger graphs than high level (symbolic) symmetries).

In connection with other reduction techniques (such as stubborn sets), much larger systems can be analyzed than using either method in isolation. The easy use of maximal symmetry groups other than rotation and permutation groups is another argument in favor of the low level symmetry approach.

References

CDFH90. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well–formed colored nets and their symbolic reachability graph. *Proc. of the 11th Int. Conf. on Application and Theory of Petri Nets*, pages 387–410, 1990. 315, 328

CEFJ96. E.M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design 9*, pages 77–104, 1996. 315, 316, 323, 324

CFG94. G. Chiola, G. Franceschinis, and R. Gaeta. Modelling symmetric computer architectures by swn's. *Proc. 15th Int. Conf. on Application and Theory of Petri Nets, LNCS 815*, pages 139–158, 1994. 315, 328

Chi98. G. Chiola. Manual and automatic exploitation of symmetries in spn models. *Proc. 19th International Conference on Application and Theory of Petri nets, LNCS 1420*, pages 28–43, 1998. 315

DBDS93. S. Duri, U. Buy, R. Devarapalli, and S. Shatz. Using state space methods for deadlock analysis in ada tasking. *ACM Proc. Int. Symp. on Software Testing and Analysis*, pages 51–60, 1993. 328

EJP97. E.A. Emerson, S. Jha, and D. Peled. Combining partial order and symmetry reductions. *Proc. TACAS '97, LNCS 1217*, pages 19–34, 1997. 328

ES96. E.A. Emerson and A.P. Sistla. Symmetry and model checking. *Formal Methods in System Design 9*, pages 105–131, 1996. 315, 324

HJJJ84. Huber, A. Jensen, Jepsen, and K. Jensen. Towards reachability trees for high–level petri nets. In *Advances in Petri Nets 1984, Lecture Notes on Computer Science 188*, pages 215–233, 1984. 315, 321, 323

ID96. C.N. Ip and D.L. Dill. Better verification through symmetry. *Formal Methods in System Design 9*, pages 41–75, 1996. 315, 316, 323

Jen92. K. Jensen. *Coloured Petri Nets*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1992. 315

Jen95. K. Jensen. *Coloured Petri Nets Vol. 2: Analysis Methods.* Springer, 1995. 324

Jun98. T. Junttila. Towards well-formed algebraic system nets. *Workshop CSP'98 Berlin, Technical Report 110, Humboldt–University Berlin*, pages 116–127, 1998. 315

RS97. S. Roch and P. Starke. *INA – Integrierter Netz–Analysator Version 1.7. Handbuch.* Humboldt–University Berlin, Institute of Computer Science, 1997. 316

Sch97. K. Schmidt. How to calculate symmetries of petri nets. To appear in Acta Informatica. 316, 317, 321

Sta91. P. Starke. Reachability analysis of petri nets using symmetries. *J. Syst. Anal. Model. Simul.*, 8:294–303, 1991. 315, 316

Tar75. R.E. Tarjan. Efficiency ofa good but not linear set union algorithm. *Journal of the ACM 22(2)*, pages 215–225, 1975. 321

Val88. A. Valmari. Error detection by reduced reachability graph generation. *Proc. of the 9th European Workshop on Application and Theory of Petri Nets, Venice*, 1988. 328

Val91. A. Valmari. Stubborn sets for coloured petri nets. In *The Proceedings of the 12th International Conference on Application and Theory of Petri Nets*, pages 102–121, 1991. 328