Optimised Calculation of Symmetries for State Space Reduction

Harro Wimmel

Universität Rostock, Institut für Informatik

Abstract. Since the state space explosion is a common problem when analysing Petri nets several ways to deal with this problem leading to a smaller – reduced – state space have been invented. One of them is finding symmetries, an equivalence relation on places and transitions of a Petri net, and only evaluating one object from each equivalence class. All other objects in the same class are then known to yield the same information. Finding symmetries by brute force is known to be expensive, it is even unclear if it can be done in polynomial time due to the inclusion of the graph isomorphism problem. While a first few symmetries need to be found by brute force, later ones might also be generated. We show how to generate new symmetries from known ones efficiently, how to tell if the brute force algorithm enters a branch not containing symmetries, and how to reduce the symmetries themselves to move towards orthogonality.

Keywords: Petri Net, State Space, Symmetry.

1 Introduction

Tools for state space exploration like LoLA [Wol10] use many techniques to reduce the state space before exploring it to answer a question. This is due to the well-known problem of state space explosion, where the size of the state space can grow exponentially or worse with the size of the system to be analysed. If a system contains components that are indistinguishable from each other (by the structure of the system and the question asked about the system) it is obviously sufficient to analyse one such component only. The result can just be mapped to an equivalent component then. A mapping maintaining the relation between components is called a symmetry.

With Petri nets as systems, components are just the places and the transitions. A symmetry therefore maps each place to some place (possibly itself) and likewise for transitions, keeping the edges, i.e. a mapped pair of place and transition has an edge between them if and only if the original pair has that edge. A symmetry is thus not more than a structure-preserving permutation on the places and transitions of a net.

Unluckily, the number of symmetries (the size of the *automorphism group*) of a Petri net (or any system) can be much larger than the system itself. It is necessary to find a small set of symmetries (called *generator set*) from which all other symmetries may be derived. If the components are ordered and numbered (say from 1 to n), a generator set consists of n levels and each level i contains one symmetry for each possible image j of the component i with $j \geq i$. Components with a number less than i are mapped to themselves. Take the Petri net from Fig. 1 as an example.

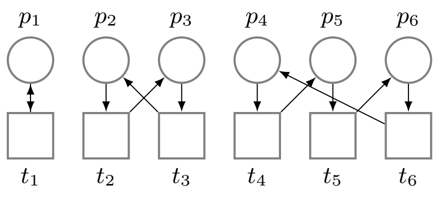

Fig. 1. An example Petri net N

If a place p_i is mapped to p_j then the transition t_i is mapped to t_j due to the structure of the net (each place has only one successor transition). If we thus use just the number of a place/transition instead of its name, we may obtain the following generators: the identity for level 1, (2 3) and the identity for level 2, (4 5 6), (4 6 5) and the identity for level 3. Only components mapped to different components are shown explicitly in this notation, and each one is mapped to the next inside the parentheses (the last to the first). For level 2 there are two replacement candidates for (2 3): (2 3)(4 5 6) and (2 3)(4 6 5). Any one of these three is sufficient. It can be shown (see e.g. [Sch02]) that any symmetry σ can be written as the composition of some generators g_i, one from each level i, by $\sigma = g_1 \circ g_2 \circ \ldots \circ g_n$.

While LoLA seems to implement an algorithm (called *Refine*/Define* in [Sch02]) that runs in polynomial time in practical cases, this cannot be guaranteed due to the inclusion of the graph isomorphism problem for which membership in P is unknown. Therefore, it is important to rely on such an algorithm (that builds generators from scratch) as seldom as possible and use already known generators instead to derive new ones when possible. To a certain extent, LoLA already does this by composing generators with themselves, building powers, and checking whether these powers can fill the gaps where generators are still missing. In the following, we show how this can be improved.

2 Basic Definitions

We assume Petri nets, formally a tuple (P, T, F) with F: $(P \times T) \cup (T \times P) \to \mathbb{N}$, to be known to the reader. We also expect some knowledge about linear algebra, especially the definition of a group, and start now by defining symmetries.

Definition 1 (Symmetry). *Given a net* (P, T, F)*, a* symmetry σ *is a map* σ: $P \cup T \to P \cup T$ *with* $\sigma(P) = P$, $\sigma(T) = T$, *and* $F(\sigma(x), \sigma(y)) = F(x, y)$ *for all* $x, y \in P \cup T$.

We might also be interested in a (initial or final) marking m, in which case $m(\sigma(p)) = m(p)$ must hold additionally for all $p \in P$. A symmetry σ is written in the style $(a_{11} \dots a_{1j_1}) \dots (a_{n1} \dots a_{nj_n})$ where $\sigma(a_{im}) = a_{i,(m \bmod j_i)+1}$.

In the following we assume a fixed Petri net $N = (P, T, F)$, a fixed bijection b: $P \cup T \to \{i \in \mathbb{N} \,|\, 1 \le i \le |P \cup T|\}$ and identify the places and transitions of N with their images under b.

Definition 2 (Generator, level, orbit). *A symmetry g* for level *$lev(g) = i$ ($1 \le i \le |P \cup T|$) is a symmetry with $g(k) = k$ for all $k < i$. The number $g(i)$ is called the* orbit *of g. Each level i also has* orbits*, numbered from i to $|P \cup T|$. An orbit k of level i is* consistent *if there is a symmetry g for level $lev(g) = i$ with an orbit $g(i) = k$, otherwise the orbit is called* inconsistent *or* empty*. Define G_i to be the* group of all symmetries for level *i (with $G_i \supseteq G_{i+1}$). On the other hand, a* generator set *G consists of one symmetry, called* generator*, g_{ij} for each level i and each consistent orbit j on that level.*

Corollary 1 ([Sch02]). *Let G be a generator set. Each symmetry g for level i can be expressed as a consecutive composition of one generator $g_j \in G$ from each level $j \ge i$.*

Corollary 2 ([Sch02]). *There is an algorithm* Refine*/Define *taking a level i and an orbit k as input and producing a symmetry g for level i with $g(i) = k$ if such a symmetry exists. Otherwise, the algorithm terminates with the result "inconsistent".*

3 Inheriting Inconsistency

Inconsistencies are obviously the worst result that can be obtained from the *Refine*/Define* algorithm. They waste time and do not even produce a generator. Once we know of an inconsistent orbit for some level, we may use this information to find other inconsistent orbits without the *Refine*/Define* algorithm. An equivalence relation for places/transitions can be helpful here.

Definition 3 (Equivalence of components). *Let G be a generator set and i some level. For $x, y \in P \cup T$ we define an equivalence relation $x \equiv_i y \iff \exists g \in G_i$: $g(x) = y$.*

Note that $\equiv_i$ being an equivalence relation follows from the fact that G_i is a group with identity, an inverse, and composition as group operation.

Lemma 1 (Inheritance). *For level i, let k be an inconsistent orbit and n be a consistent orbit. Then, $n \not\equiv_i k$.*

Proof. Assume $n \equiv_i k$, then there is $g \in G_i$ with $g(n) = k$. Let $g' \in G$ be the generator for level i with $g'(i) = n$. Then, $g(g'(i)) = k$ and $g' \circ g \in G_i$ has the orbit k. A contradiction, as no symmetry g'' for level i with $g''(i) = k$ exists.

Looking from the other side, this means every orbit m with $m \equiv_i k$ must be inconsistent. It is therefore unnecessary to call the *Refine*/Define* algorithm for orbits equivalent to k. In our example from the introduction, all orbits $k > 1$ on the first level are inconsistent, since p_1 can only be mapped to itself by any symmetry. If we know that the orbits 2 and 4 are inconsistent, we conclude from the generators (2 3) and (4 5 6) (from levels 2 and 3) that $2 \equiv_1 3$ and $4 \equiv_1 5 \equiv_1 6$. We save three calls to *Refine*/Define*.

4 Building Products

LoLA so far takes a newly acquired generator g for level i and calculates the powers $g^2 = g \circ g$, g^3, g^4, and so on, until $g^n(i) = i$ holds. If one of the powers has an orbit for which no symmetry has been found so far, the power is saved as the new gnerator for that orbit. Further powers do not yield anything new as g^{n+1} has the same orbit as $g^1 = g$.

Instead, we propose building compositions of any new generator with any generator found so far until no new generators are derived anymore. This looks like a losing approach at first as there are by a linear factor more such products than powers. But note that in the powers approach $O(n)$ (with $n = |P \cup T|$) powers must be calculated until $g^n(i) = i$ holds and each composition done also needs $O(n)$ (the size of the map). The powers approach therefore looks quadratic.

Lemma 2 (Complexity of product testing). *Let $g \in G_i$ and $g' \in G_j$ with $j \geq i$. A test if the composition $g \circ g'$ leads to an orbit for which no generator has been found so far can be done in $O(1)$.*

Proof. We check if there is a generator for level i with orbit $g'(g(i))$. This takes $O(1)$ time. If there is none, $g \circ g'$ will be the new generator for level i and orbit $g'(g(i))$.

Note that this simple test is useless for the powers. We would test and then calculate the composition anyway, independently of the test's result, when we need the next, higher power.

What is important here is that a newly found generator for level i is composed only with generators of a higher level. To guarantee this, the levels have to be filled with generators from highest to lowest. This is the way it is done in LoLA anyway: LoLA uses a recursion from easier to harder problems, and higher levels represent the easier problems (as more elements are mapped to themselves).

If we try to calculate a complexity for our approach and (falsely) assume that the size of a generator set G is roughly equal to $|P \cup T|$ we obtain $O(n^2)$ tests (for pairs of generators in G) with complexity $O(1)$ each and $O(n)$ compositions with complexity $O(n)$ each. This would suggest a quadratic complexity just like for the powers' calculation. There are examples where the size of the generator set is much higher as well as those where it is much lower than $|P \cup T|$, so the real complexity comparison is much more difficult.

In general, the product approach will produce more new generators in a single call than the powers approach, but this depends on the structure of the automorphism group. If the *order* of generators (the lowest power yielding the identity) is lower than the number of orbits on some level, it is impossible to fill all orbits in a single call of the powers approach. Since products are iterated, from them the whole subgroup spanned by all known generators could be computed. This can mean an exponential gain compared to the powers, e.g. in groups with p^n elements (p prime) where g^p is the identity for all symmetries g. By powers at most $p-1$ new generators can be built from each call to *Refine*/Define*, while with products the group spanned from the known generators increases by a factor of p for each call.

5 Minimizing the Carrier

Our last optimisation does not deal with the finding of generators but with the size of their representation. While a smaller size might reduce execution times there may be other benefits. Let two generators g, g' be *orthogonal* if $g(i) \neq i \Longrightarrow g'(i) = i$, then g and g' can be composed without effort. The set $\{i \mid g(i) \neq i\}$ is called the *carrier* of g. Conclusions drawn about the carrier of g cannot be influenced by g' and are thus valid for $g \circ g'$. While orthogonality is unreachable in general, we may still try to minimize the carrier of any generator g. Candidates that may have a smaller carrier are easy to find:

Corollary 3. *If g is a generator for level i and orbit k and $n > 0$ is the smallest integer with $g^n(i) = i$, then, from all powers of g, exactly the g^{jn+1} (for $j \in \mathbb{N}$) are generators for level i and orbit k.*

Take e.g. the cycle representation of one of the generators for the Petri net from Fig. 1: $g = (2\ 3)(4\ 5\ 6)$. If we take g as generator for level 2 and orbit 3 then $(2\ 3)$ is the *orbit cycle* of g (it contains level and orbit). Its length is $n = 2$, i.e. $g^2(2) = 2$. Thus, reduction candidates are $g^{2+1} = (2\ 3)$, $g^{4+1} = (2\ 3)(4\ 6\ 5)$, $g^{6+1} = g^1$ and so on. Since $2 + 1 = 3$ is divisible by the length 3 of the second cycle $(4\ 5\ 6)$, this cycle is eliminated in g^3 ($g^3(i) = i$ for $i = 4, 5, 6$ and identities are not shown in cycle representation). In general, the following holds.

Lemma 3. *Let g be a generator with an orbit cycle $o = (o_0\ \dots o_{i-1})$ and another cycle $c = (c_0, \dots, c_{m-1})$. Let k be the greatest divisor of m such that i and k have a greatest common divisor (gcd) of one. Then, in g^{k+qm} (for $q \in \mathbb{N}$) the cycle c is replaced by k cycles of length m/k and there is no power of g having the same orbit cycle and shorter replacement cycles for c.*

Proof. Just note that $g^k(c_{(n+\ell k) \bmod m}) = c_{(n+(\ell+1)k) \bmod m}$ and for $\ell = \frac{m}{k} - 1$ for the first time $g^k(c_{(n+\ell k) \bmod m}) = g^k(c_{(n+m-k) \bmod m}) = c_{n+m \bmod m} = c_n$ holds. If we choose k a greater divisor of m, there can be no j such that k divides $ji+1$, since the *gcd* of k and i also divides ji (and not $ji+1$). Since a divisor of k also divides $k + qm$, $k + qm = ji + 1$ is impossible, i.e. if $gcd(i, k) > 1$, g^{k+qm} will not preserve the orbit cycle.

Thus, for $k + qm = ji + 1$ the cycle c is reduced as far as possible and at the same time the orbit cycle stays intact. It is unnecessary though to solve this equation as the proof of lemma 3 already tells us how the cycle c will be modified.

6 Experimental Results

The approach of building powers of generators can already be optimal, as it happens e.g. with the dining philosophers and reader/writer systems. Experiments for these examples show a worst case slow down of $1 - 2\%$ in execution times from the powers approach to our new optimisations. This supports our earlier assumption that building products is not (much) slower than building powers.

A good example to show the difference between old and new approach is the hypercube ECHO [Rei98], a grid-like network with d dimensions and n communicating agents per dimension. The value of detecting inconsistencies can be shown with the nets N_i, where $N_3 = N$ from Fig. 1, and each higher index adds a new ring with i places and transitions each. Execution times for computing the symmetries are shown in Table 1.

ECHO	#symm	#gen	Old	New
3/3	48	10	0.2s	<0.1s
3/5	48	10	4.6s	2.9s
3/7	48	10	29s	16s
4/3	384	21	3.5s	1.6s
4/5	384	21	244s	96s
5/3	3840	41	82s	23s

N_i	#gen	Old	New
N_{10}	45	<0.1s	<0.1s
N_{15}	105	0.4s	0.2s
N_{20}	190	2.4s	1.3s
N_{25}	300	8.2s	4.4s
N_{30}	435	25s	13s
N_{35}	595	66s	34s

Table 1. Results for ECHO d/n with numbers of symmetries and computed generators as well as execution times for old and new approach and nets N_i consisting of i cycles of lengths from 1 to i

7 Conclusion

Theoretical observations and experimental results have shown that finding and using symmetries can be optimised beyond the current state of affairs represented by LoLA. Using products and an equivalence to detect inconsistencies leads to a clear speed up in not already optimal cases.

References

[Sch02] K. Schmidt: *Explicit State Space Verification*, Habilitation Thesis, Humboldt Universität zu Berlin, 2002.

[Wol10] K. Wolf: *LoLA – A low level analyzer*, http://www.informatik.uni-rostock.de/ ˜ nl/wiki/tools/lola, 2010.

[Rei98] W. Reisig: *Elements of Distributed Algorithms*, Springer Verlag, 1998.