Acta Informatica 36, 545–590 (2000)

How to calculate symmetries of Petri nets

Karsten Schmidt

Institut für Informatik, Humboldt-Universität zu Berlin, D-10099 Berlin, Germany
(e-mail: kschmidt@informatik.hu-berlin.de, Fax: (+49) (30) 2093 3081)

Received 7 July 1997 / 10 August 1999

Abstract. Symmetric net structure yields symmetric net behaviour. Thus, knowing the symmetries of a net, redundant calculations can be skipped. We present a framework for the calculation of symmetries for several net classes including place/transition nets, timed nets, stochastic nets, self–modifying nets, nets with inhibitor arcs, and many others. Our approach allows the specification of different symmetry groups. Additionally it provides facilities either to calculate symmetries on demand while running the actual analysis algorithm, or to calculate them in advance. For the latter case we define and calculate a *ground set* of symmetries. Such a set has polynomial size and is sufficient for an efficient implementation of the *for all symmetries* loop and the partition of net elements into equivalence classes. These two constructions are the usual way to integrate symmetries into an analysis algorithm.

1 Introduction

Symmetries of Petri nets have been introduced in [HJJJ84] for reducing the size of the reachability graph for coloured Petri nets. In [Sta91], the idea of symmetric reduction was applied to the reachability graph of place/transition nets. Meanwhile, the state space of Petri nets with time restrictions can be reduced as well [RS97]. The application of symmetries is not restricted to the generation of state spaces. The method can be combined with the stubborn set method [Val91] to reduce space and time complexity, the calculation of semi–positive invariants to reduce space complexity [Sch93], the calculation of siphons and traps to reduce space and time complexity [Sch93], the calculation of coverability graphs [Pet90], BDD–based analysis methods

[CEFJ96,Tiu94], and the quantitative analysis of stochastic nets [DH91]. Symmetric reduction can be applied to model–checking large classes of temporal–logic formulae [ES96,CEFJ96]. It is likely that symmetric reductions can be found for methods developed in the future.

All methods require knowledge about the net symmetries. There are three major approaches to provide this knowledge:

1. The user comes up with a description of net symmetries;
2. Symmetries are deduced from the syntax of high level net inscriptions;
3. A tool calculates the symmetries.

The first approach [HJJJ84,Jen96] is based on the assumption that the one who generates the net model is familiar with the modelled system and knows the symmetries. This assumption might be realistic when a net model is drawn directly. However, when a net model is generated by the translation of another description language, or when many people are involved in the generation of the model, it could be difficult to find the symmetries without tool support. Additionally one needs a reasonable way to describe the (up to exponentially many) net symmetries. The notations of [HJJJ84] are acceptable for high level nets, but do not necessarily cover all symmetries of the underlying place/transitions net. Moreover, the notation cannot be transferred to low level net classes.

The second approach [CDFH90,DH91,HIMZ95,GC95,CF90] is closely related to the concept of *well formed coloured nets*, a dialect of high level nets. There, the restriction to only few syntactic constructs for the net inscriptions allows to deduce symmetries directly from the net structure. However, the approach covers mainly the symmetries in the sense of [HJJJ84]. Thus, similar problems appear as for the first approach. Though the user is not forced to describe the symmetries explicitly, profound skills are necessary to ”exhibit” as many as possible net symmetries in the net inscriptions (see [CFG94] for a case study).

We contribute to the third approach. This is the only feasible one for classes other than high level nets, and even for high level nets it could be an interesting option. The first contribution to this approach was [Sta91]. There, the calculation of a generating set of symmetries was based on the calculation on transpositions (exchangings of two elements). This approach does not necessarily cover all symmetry groups, since, for instance, rotation groups cannot be generated by transpositions. To the authors knowledge, two Petri net analysis tools are currently offering a symmetry calculation (plus those contributing to the second approach): INA [RS97], written at Humboldt–Universität zu Berlin, Germany, and PROD [VHHP95], written at Helsinki University of Technology, Finland. Both calculation procedures rely on technical reports [SS91,Sch93] containing earlier, less powerfull versions of the approach presented below.

After introducing the basic notations in Sect. 2, we introduce in Sect. 3 a framework for the specification of symmetry calculation problems, including both the calculation of different symmetry groups as a pre–processing and the calculation of symmetries on demand. The ”on demand” approach is designed for the application of symmetries without pre–processing. Our framework is sufficient for most calculation problems, but for instance not for symmetry groups of self–modifying nets [Val78], and the coverability test for place/transition nets on demand. The symmetry approach to high level nets with *finite* colour domains is covered by considering the corresponding unfolded (place/transition or condition/event) nets. In Sects. 4 and 5 we present the basic calculation algorithm. Sections 6 and 7 are concerned with modifications of the basic algorithm for self–modifying nets and the coverability test. In the remaining sections we study significant speed–ups for the basic procedure. In Sect. 8 we introduce a certain generator set for a symmetry group that we call *ground set*. A ground set is a generator set for all symmetries with polynomially many elements (in the size of the net). We modify the basic calculation procedure for the calculation of ground sets (Sect. 9), and propose an implementation of the *for all symmetries* loop solely based on ground sets (the *for all symmetries* loop forms the kernel of any combination of the symmetry approach as pre–processing with graph based analysis algorithms). In Sect. 10, we study symmetry respecting hash functions to speed up the symmetry on demand approach.

2 Basic notations

We use the usual symbols for the set operations intersection ($\cap$), union ($\cup$), difference ($\setminus$), product ($\times$), the relations inclusion ($\subseteq$), containment ($\in$), the natural numbers ($\mathbb{N}$), the integer numbers ($\mathbb{Z}$), and the empty set ($\emptyset$). For a finite set M, $|M|$ denotes its number of elements. $f : M \longrightarrow N$ denotes that f is a mapping from M into N. $f\mid_{M'}$ denotes the restriction of f to M', i.e. $f\mid_{M'} = \{[x, f(x)] \mid x \in M'\}$.

Nets

In order to cover as many as possible net classes with our approach, we start with a rather general notion of nets. For our purposes, it is sufficient to consider a net to be a bipartite graph with the usual restrictions, where the different elements may be inscribed arbitrarily.

Definition 1 (Net). *A tuple* $[P, T, F, \mathcal{I}, \chi, m_0]$ *is a* net *iff* P *and* T *are finite and disjoint sets (of* places *and* transitions*, respectively),* F *is a subset of* $(P \times T) \cup (T \times P)$ *(the set of* arcs*).* $\chi : P \cup T \cup F \longrightarrow \mathcal{I}$ *assigns* inscriptions

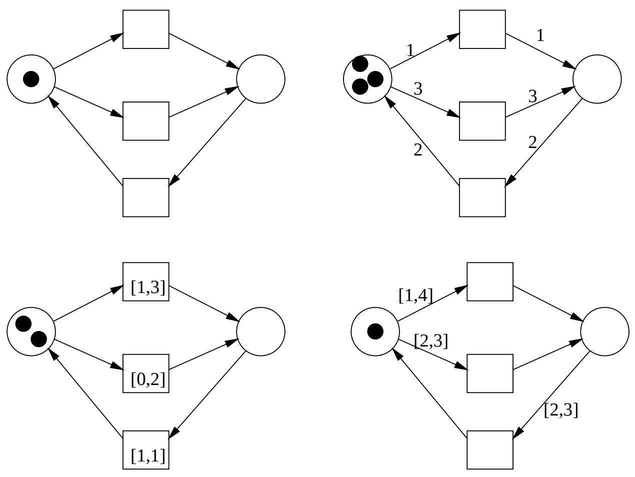

Fig. 1. Inscriptions of condition/event nets (upper left), place/transition (upper right), time (lower left), and arc timed (lower right) Petri nets

to places, transitions and arcs ($\mathcal{I}$ is an arbitrary set) and m_0 is a state (see below), the initial state.

This definition covers condition/event nets (in this case the inscriptions assign equal values to every net element — that is we do not have a distinguishable inscription), place/transition nets (in this case, χ assigns a nonzero natural number, the multiplicity, to every arc), various kinds of nets with time restrictions (the timing parameters are used as inscriptions), stochastic nets (the parameters and kinds of distributions are inscribed), nets with inhibitor arcs or other special kinds of arcs (the arc type can be seen as an inscription of the arc), nets with capacities, self–modifying nets (see Sect. 7) and so on. In the sequel, if no particular net is addressed, we assume N to be an arbitrary, but fixed net.

For every net class, a notion of *states* is introduced that describes the dynamically changing part of a net. For several net classes, the state is associated to places only. But there are other net classes where parts of the state are associated to transitions. For instance, in the case of nets with time constraints, transitions may have local clocks. The time these clocks are showing may influence the future behaviour of the net. Therefore they are part of the state of the net. To the authors knowledge, nets where parts of the states are assigned to arcs have not been considered anywhere. For our purposes the interpretation of states is of minor interest. We consider states to be dynamically changing inscriptions of places and transitions. Therefore, without loss of generality, we assume that values of states are contained in the set $\mathcal{I}$ of possible values for net inscriptions.

Definition 2 (State). *A state of a net $[P, T, F, \mathcal{I}, \chi, m_0]$ is a mapping $m : P \cup T \longrightarrow \mathcal{I}$.*

Symmetry groups

Symmetries, as defined in [Sta91], are bijections on the set $P \cup T$ of nodes of a net that respect node type, arc relation, and net inscriptions.

Definition 3 (Symmetry). *A bijection σ on $P \cup T$ is called* symmetry *of the net $[P, T, F, \mathcal{I}, \chi, m_0]$ iff*

1. *$\sigma(P) = P$, $\sigma(T) = T$ (σ respects the node type);*
2. *for all x, y in $P \cup T$, $[x, y] \in F$ iff $[\sigma(x), \sigma(y)] \in F$ (σ respects the arc relation; in the sequel, we write $\sigma([x, y])$ for $[\sigma(x), \sigma(y)]$);*
3. *for all $x \in P \cup T \cup F$, $\chi(\sigma(x)) = \chi(x)$ (σ respects the net inscriptions).*

We denote the set of all symmetries of a net N by Σ_N. With $\circ$, we denote the composition of two symmetries: $\sigma_1 \circ \sigma_2(x) := \sigma_1(\sigma_2(x))$. $[\Sigma_N, \circ]$ is a group where the identity on $P \cup T$ is the neutral element. We call every subgroup of $[\Sigma_N, \circ]$ a *symmetry group*.

A symmetry group Σ induces equivalence relations on the nodes of the net as well as on the states.

Definition 4 (Equivalence of nodes/states). *Two nodes x and y in $P \cup T$ are equivalent with respect to a symmetry group 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 equivalence class containing some state m according to $\sim_\Sigma$ with $[m]_\Sigma$.

A state m of a net is called *symmetric* with respect to a symmetry group Σ iff $[m]_\Sigma = \{m\}$.

The equivalent to a symmetric state for nodes is called *fixed point*. A fixed point with respect to Σ is an element $x \in P \cup T$ such that $[x]_\Sigma = \{x\}$.

In Fig. 2., the net at the left has three symmetries (the three rotations that map places on places and transitions on transitions). The net at the right has 48 symmetries, since all input transitions of the central place can be mapped to each other while all output transitions can be mapped to each other.

3 Calculation problems

For the generation of symmetrically reduced state spaces, the following proposition plays a key role.

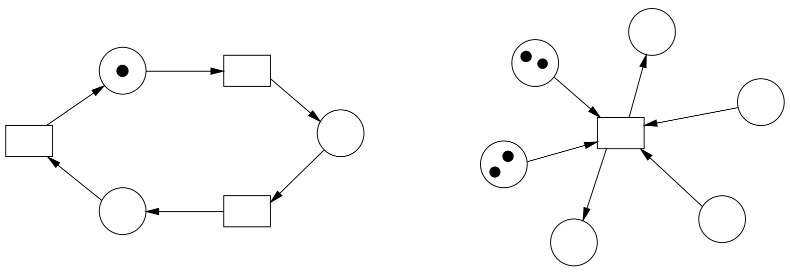

Fig. 2. Net with 3 symmetries (left), net with 48 symmetries (right)

Proposition 1 (Symmetric behaviour). *Let N be a net where places, transitions, and arcs do not appear in net inscriptions, and let σ be one of its symmetries. A transition t can fire in a state m leading to a state m' iff $\sigma(t)$ can fire in $\sigma(m)$ leading to the state $\sigma(m')$ (as a formula: $m \xrightarrow{t} m' \iff \sigma(m) \xrightarrow{\sigma(t)} \sigma(m')$).*

We cannot prove this proposition in general, since the concepts ”a transition can fire” and ”firing leads to a state m'” depend on the particular net class. Once having defined them, the proof of the above proposition is however straightforward. To give an idea, we present the proof for a particular net class.

Proof. (for place/transition nets)

In a place/transition net, the relation $m \xrightarrow{t} m'$ holds iff, for all $p \in P, m(p) \geq \chi([p,t])$, and $m'(p) = m(p) - \chi([p,t]) + \chi([t,p])$. Thereby, define $\chi([x,y]) = 0$ for $[x,y] \notin F$.

Assume $m \xrightarrow{t} m'$. Let $p \in P$. Then, $\sigma(m)(p) = m(\sigma^{-1}(p))$ (by Def. 4). Since $m \xrightarrow{t} m'$, we have $m(\sigma^{-1}(p)) \geq \chi([\sigma^{-1}(p),t])$ and $m'(\sigma^{-1}(p)) = m(\sigma^{-1}(p)) - \chi([\sigma^{-1}(p),t]) + \chi([t,\sigma^{-1}(p)])$. By Def. 3, for arbitrary nodes x,y either both $[x,y]$ and $[\sigma(x),\sigma(y)]$ are not contained in F (thus, both $\chi([x,y]$ and $\chi([\sigma(x),\sigma(y)])$ are equal to 0), or $\chi([x,y]) = \chi([\sigma(x),\sigma(y)])$. This observation leads to $\sigma(m)(p) = m(\sigma^{-1}(p)) \geq \chi([\sigma^{-1}(p),t]) = \chi([p,\sigma(t)])$ and $\sigma(m')(p) = m'(\sigma^{-1}(p)) = m(\sigma^{-1}(p)) - \chi([\sigma^{-1}(p),t]) + \chi([t,\sigma^{-1}(p)]) = \sigma(m)(p) - \chi([p,\sigma(t)]) + \chi([\sigma(t),p])$. Therefore, $\sigma(m) \xrightarrow{\sigma(t)} \sigma(m')$.

Assume, $\sigma(m) \xrightarrow{\sigma(t)} \sigma(m')$. Since σ^{-1} is a symmetry, we can apply the first direction of the current proposition. This leads to $\sigma^{-1}(\sigma(m)) \xrightarrow{\sigma^{-1}(\sigma(t))} \sigma^{-1}(\sigma(m'))$ that is equivalent to $m \xrightarrow{t} m'$. □

Thereby the non–appearance of net elements in the inscriptions is essential. As an example for a net class where places may appear in arc inscriptions, we study self–modifying nets (in Sect. 7). For virtually all other net classes, net inscriptions are independent of the net elements (mostly integer or rational numbers, or boolean values).

Proposition 1 justifies the generation of a reduced state space. Instead of enumerating all reachable states we enumerate all equivalence classes of reachable states with respect to a symmetry group Σ. This way we obtain a *non–reachability test*: if the equivalence class of a node is not contained in the reduced state space, then the state itself is not reachable. If for a state m, its equivalence class is contained in the reduced state space, then m may or may not be reachable from the initial state. The problem is that Proposition 1 assures only its reachability from some symmetric image of the initial state but not necessarily from the initial state itself. The situation is different when the initial state is symmetric. In this case, a state is reachable iff its equivalence class is contained in the reduced state space. Thus, it can be wise to work with a subgroup of Σ_N where the initial state is symmetric instead of working with Σ_N itself. On the other hand, this subgroup can be significantly smaller than Σ_N and therefore yield a larger reduced state space. Thus, we should allow the user to decide whether to work with the whole Σ_N or an (as large as possible) subgroup where the initial state is symmetric. Accordingly, the reduced state space does not allow to deduce properties related to nodes (for instance, liveness of transitions). Instead, corresponding properties for *equivalence classes* of nodes can be decided (for instance, collective liveness of a class of transitions). In order to be able to decide at least properties related to some important nodes, we should allow the user to work with symmetry groups where some given nodes are fixed points. Thus, our calculation procedure should support different side conditions for the calculation of symmetries.

Once having fixed a symmetry group Σ, the major problem during reduced state space generation is:

> Given a set M of (representatives of already calculated classes of) states, and a (recently generated) state m^*, are there an $m \in M$ and a $\sigma \in \Sigma$ such that $m^* = \sigma(m)$?

There are two principal ways to solve this problem. The first one is to calculate the symmetries before running the state space generation algorithm. In this case the solution would be

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

The test $\sigma(m^*) \in M$ can be implemented by a search in a hash or tree structure for M and is therefore quite efficient. Thus, the running time is mainly determined by $|\Sigma|$.

The other principal solution is

```
**FOR ALL** $m \in M$ **DO**
    **IF** there is a $\sigma \in \Sigma$ such that $\sigma(m) = m^*$ **THEN**
        **RETURN** yes;
    **END IF**
**END FOR**
**RETURN** no;
```

The test in this solution requires a procedure trying to calculate a symmetry that maps a given m to a given m^*. This symmetry is calculated "on demand", that is only when the corresponding state equivalence problem appears. Therefore, the second solution does not require a pre–processing of the symmetries. We compare these two principal solutions later in this article. Meanwhile, we develop a framework that supports both ways.

The key to a uniform handling of all the different calculation problems is their representation using the following concepts.

Definition 5 (Constraint, symmetry specification). *A* constraint *has the form $A \longmapsto B$ where A and B are subsets of $P \cup T$. A bijection π on $P \cup T$ is consistent with $A \longmapsto B$ iff $\pi(A) = B$. (For a set A, $\pi(A) = \{\pi(a) \mid a \in A\}$.)*

A set $\mathcal{C}$ of constraints forms a symmetry specification. *A bijection is consistent with a symmetry specification $\mathcal{C}$ iff it is consistent with every constraint contained in $\mathcal{C}$. For a specification $\mathcal{C}$, $\Pi_\mathcal{C}$ is the set of all* bijections *being consistent with $\mathcal{C}$, and $\Sigma_\mathcal{C}$ is the set of all* symmetries *being consistent with $\mathcal{C}$.*

Example 1. Let $\mathcal{C}_0 = \{P \longmapsto P, T \longmapsto T\}$. Then $\Pi_{\mathcal{C}_0}$ is the set of all bijections respecting the node type, and $\Sigma_{\mathcal{C}_0} = \Sigma_N$. Define $\mathcal{C}_1 = \{\{x \mid x \in P \cup T, \chi(x) = i\} \longmapsto \{x \mid x \in P \cup T, \chi(x) = i\} \mid i \in \mathcal{I}\}$. Since $\Pi_{\mathcal{C}_1}$ is the set of all bijections which respect the inscriptions of nodes, $\Sigma_{\mathcal{C}_1} = \Sigma_N$ as well. For $\mathcal{C}_{m_1,m_2} = \{\{p \mid p \in P, m_1(p) = i\} \longmapsto \{p \mid p \in P, m_2(p) = i\} \mid i \in \mathcal{I}\}$, $\Sigma_{\mathcal{C}_{m_1,m_2}}$ is the set of all symmetries σ such that $\sigma(m_1) = m_2$. For every state m, m is symmetric with respect to $\Sigma_{\mathcal{C}_{m,m}}$. For a node $x \in P \cup T$, x is a fixed point with respect to $\Sigma_{\{\{x\} \longmapsto \{x\}\}}$. For every bijection π on $P \cup T$, the constraint $\mathcal{C}_\pi = \{\{x\} \longmapsto \{\pi(x)\} \mid x \in P \cup T\}$ satisfies $\Pi_{\mathcal{C}_\pi} = \{\pi\}$.

Corollary 1 (Properties of symmetry specifications).

(1) *For two specifications $\mathcal{C}_1$ and $\mathcal{C}_2$, it holds $\Pi_{\mathcal{C}_1 \cup \mathcal{C}_2} = \Pi_{\mathcal{C}_1} \cap \Pi_{\mathcal{C}_2}$ and $\Sigma_{\mathcal{C}_1 \cup \mathcal{C}_2} = \Sigma_{\mathcal{C}_1} \cap \Sigma_{\mathcal{C}_2}$.*

(2) For a specification $\mathcal{C}$, $\Pi_{\mathcal{C}\cup\{A_1\longmapsto B_1, A_2\longmapsto B_2\}} =$ $\Pi_{\mathcal{C}\cup\{A_1\cap A_2\longmapsto B_1\cap B_2, A_1\setminus A_2\longmapsto B_1\setminus B_2, A_2\setminus A_1\longmapsto B_2\setminus B_1\}}$.

According to Part 1 of Corollary 1, different restrictions to symmetries can be combined. For example, the set of all bijections respecting node type, node inscriptions, leaving the initial state symmetric, and having at least the fixed points x and y can be specified by $\mathcal{C}_0 \cup \mathcal{C}_1 \cup \mathcal{C}_{m_0,m_0} \cup \{\{x\} \longmapsto \{x\}, \{y\} \longmapsto \{y\}\}$. With Part 2 of Corollary 1, we can transform any specification into a normal form where all the sets left of $\longmapsto$ form a partition of $P \cup T$, and all the sets right of $\longmapsto$ form a (not necessarily the same) partition of $P \cup T$ as well.

Definition 6 (Normal form). *A symmetry specification $\mathcal{C} = \{A_j \longmapsto B_j \mid 1 \leq j \leq |\mathcal{C}|\}$ is in* normal form *iff $\bigcup_{j=1}^{|\mathcal{C}|} A_j = P \cup T$, $\bigcup_{j=1}^{|\mathcal{C}|} B_j = P \cup T$, and for all $j, k \in \{1, \ldots, |\mathcal{C}|\}, j \neq k$ it holds $A_j \cap A_k = \emptyset$ and $B_j \cap B_k = \emptyset$.*

Every specification can be easily transformed into a normal form without changing $\Pi_{\mathcal{C}}$ and $\Sigma_{\mathcal{C}}$. First, by adding $P \cup T \longmapsto P \cup T$ to $\mathcal{C}$, we obtain a specification where the union of the left sides of constraints as well as the union of right sides of constraints equal $P \cup T$. The addition of this constraint does not change $\Pi_{\mathcal{C}}$. Then, repeatedly every pair $A_1 \longmapsto B_1$ and $A_2 \longmapsto B_2$ where $A_1 \cap A_2 \neq \emptyset$ or $B_1 \cap B_2 \neq \emptyset$, can be replaced according to Corollary 1 (2). The replaced constraints cover all removed nodes, thus the union of left and right sides remains $P \cup T$.

The following theorem shows that any combination of $\mathcal{C}_0$, $\mathcal{C}_1$, $\mathcal{C}_{m_0,m_0}$, and fixed point declarations always specifies a *group*, and not just a set of symmetries.

Theorem 1 (Group specifications). *Let $\mathcal{C}$ be a symmetry specification. $\Sigma_{\mathcal{C}}$ forms a group iff $A \longmapsto B \in \mathcal{C}$ implies $A = B$.*

Proof. "if": For two symmetries σ_1 and σ_2, $\sigma_1(A) = A$ and $\sigma_2(A) = A$ implies $\sigma_1 \circ \sigma_2(A) = A$ and $\sigma_1^{-1}(A) = A$. Therefore, if $A \longmapsto B \in \mathcal{C}$ implies $A = B$, $\Sigma_{\mathcal{C}}$ is closed under composition and inversion. Thus, $\Sigma_{\mathcal{C}}$ forms a group.
"only if": If there is a $A \longmapsto B$ contained in $\mathcal{C}$ such that $A \neq B$, then the identity is not contained in $\Sigma_{\mathcal{C}}$. Consequently $\Sigma_{\mathcal{C}}$ does not form a group. □

Corollary 2 (Maximal symmetry groups). *$\Sigma_{\mathcal{C}_{m_0,m_0}}$ ($\mathcal{C}_{m_0,m_0}$ is defined in Example 1) is the maximal subgroup of Σ_N where the initial state is symmetric. $\Sigma_{\{\{x\}\longmapsto\{x\}\}}$ is the maximal subgroup of Σ_N where x is a fixed point.*

Since the intersection of subgroups of $\varSigma_N$ forms a subgroup of $\varSigma_N$ again, the maximal subgroup satisfying more than one fixed point or initial state condition, can be specified by the union of the specifications mentioned in Corollary 2. Thus, all relevant subgroups of $\varSigma_N$ addressed earlier in this section, including $\varSigma_N$ itself, can be specified. Additionally, for a symmetry group specified by $\mathcal{C}$, the equivalence problem for two states m_1 and m_2 can be traced back to the problem $\varSigma_{\mathcal{C} \cup \mathcal{C}_{m_1,m_2}} \neq^? \emptyset$, and the node equivalence problem for x and y can be traced back to the problem $\varSigma_{\mathcal{C} \cup \{\{x\} \longmapsto \{y\}\}} \neq^? \emptyset$. Thus, the most relevant calculation problems, whether for pre–processing or symmetry on demand calculation, can be expressed in terms of symmetry specifications. For pre–processing, we have to calculate the whole $\varSigma_{\mathcal{C}}$ for a given $\mathcal{C}$. For a symmetry on demand problem, we can return after the first element of $\varSigma_{\mathcal{C}}$ has been found (if it exists). Consequently, the majority of symmetry calculation problems can be traced back to the calculation of $\varSigma_{\mathcal{C}}$ for a given symmetry specification $\mathcal{C}$. Unfortunately, there is one symmetry on demand problem that cannot be specified in terms of specifications as defined above: the coverability problem that is relevant for place/transition nets. We discuss this problem separately in Sect. 6. Meanwhile, in the next two sections, we present the basic calculation procedure.

4 The REFINE transformation

The calculation is based upon the observation that single symmetries can be represented by a specification (remember the $\mathcal{C}_\pi$ in Example 1). Hence, the calculation consists of transformations replacing a specification by more specific ones. Finally, we yield specifications solely consisting of constraints for singleton left and right sides. These terminal specifications turn out to be the symmetries. The task of the first transformation, called REFINE, is to narrow a specification such that the set of symmetries remains invariant while the set of non–symmetries decreases. This is done by exploiting the conditions established in the definition of symmetries. While the conditions "a symmetry respects the node type" and "a symmetry respects the node inscriptions" are completely covered by starting with supersets of $\mathcal{C}_0$ and $\mathcal{C}_1$ (see Example 1), the conditions "a symmetry respects the arc relation" and "a symmetry respects the arc inscriptions" have not been involved yet. This is the task of REFINE. The transformation is based on the observation that for every symmetry σ being consistent with a constraint $A \longmapsto B$, a node x has as many arcs inscribed with a value $i \in \mathcal{I}$ and pointing to (from, resp.) a node in A, as $\sigma(x)$ has arcs inscribed with i and pointing to (from, resp.) a node in B.

For a better readability, we introduce some notations. We write F_i for the set of all arcs inscribed with a value i: for $i \in \mathcal{I}$, let $F_i = \{f \mid f \in F, \chi(f) = i\}$.

Theorem 2 (Arcs and Constraints). *Let $A \longmapsto B$ be a constraint and σ a symmetry being consistent with it. Let $i \in \mathcal{I}$, and $x \in P \cup T$. Then $|F_i \cap (\{x\} \times A)| = |F_i \cap (\{\sigma(x)\} \times B)|$, and $|F_i \cap (A \times \{x\})| = |F_i \cap (B \times \{\sigma(x)\})|$.*

Proof. If a $y \in A$ satisfies $[x, y] \in F_i$ ($[y, x] \in F_i$, resp.) then, according to Definition 3, $[\sigma(x), \sigma(y)]$ ($[\sigma(y), \sigma(x)]$, resp.) is contained in F_i as well. Since σ is consistent with $A \longmapsto B$, $\sigma(y)$ is contained in B. As a symmetry, σ is injective. Thus, $|F_i \cap (\{x\} \times A)| \leq |F_i \cap (\{\sigma(x)\} \times B)|$ and $|F_i \cap (A \times \{x\})| \leq |F_i \cap (B \times \{\sigma(x)\})|$.

If a $z \in B$ satisfies $[\sigma(x), z] \in F_i$ ($[z, \sigma(x)] \in F_i$, resp.) then (remember the "only if" in Definition 3) $[x, \sigma^{-1}(z)]$ ($[\sigma^{-1}(z), x]$, resp.) is contained in F_i, too. Since σ is consistent with $A \longmapsto B$, $\sigma^{-1}(z)$ is contained in A. σ^{-1} is injective as well as σ. Thus, $|F_i \cap (\{x\} \times A)| \geq |F_i \cap (\{\sigma(x)\} \times B)|$ and $|F_i \cap (A \times \{x\})| \geq |F_i \cap (B \times \{\sigma(x)\})|$. These four inequations yield the claimed equations. □

Using Theorem 2, the REFINE transformation splits constraints according to the number of arcs from and to a fixed constraint $A^* \longmapsto B^*$.

For two constraints $A \longmapsto B$ and $A^* \longmapsto B^*$, a value $i \in \mathcal{I}$, and two natural numbers j and k, let

$$A_{A^* \longmapsto B^*, i, j, k} = \{a \mid a \in A, |F_i \cap (\{a\} \times A^*)| = j, |F_i \cap (A^* \times \{a\})| = k\};$$

$$B_{A^* \longmapsto B^*, i, j, k} = \{b \mid b \in B, |F_i \cap (\{b\} \times B^*)| = j, |F_i \cap (B^* \times \{b\})| = k\}.$$

Corollary 3 (Constraint splitting). *Let $i \in \mathcal{I}$, $A^* \longmapsto B^*$ a constraint, and σ a symmetry being consistent with $A^* \longmapsto B^*$. σ is consistent with a(nother) constraint $A \longmapsto B$ iff σ is consistent with every $A_{A^* \longmapsto B^*, i, j, k} \longmapsto B_{A^* \longmapsto B^*, i, j, k}$ for $j, k \in \mathbb{N}$.*

Proof. The "if" follows from the observation that $\bigcup_{j,k \in \mathbb{N}} A_{A^* \longmapsto B^*, i, j, k} = A$ and $\bigcup_{j,k \in \mathbb{N}} B_{A^* \longmapsto B^*, i, j, k} = B$. The "only if" is an immediate consequence of Theorem 2. □

Corollary 3 justifies the following operation.

```
PROCEDURE REFINE($\mathcal{C}$:spec,$A^* \longmapsto B^*$:constraint,$i$:$\mathcal{I}$):spec

BEGIN
    RETURN
```

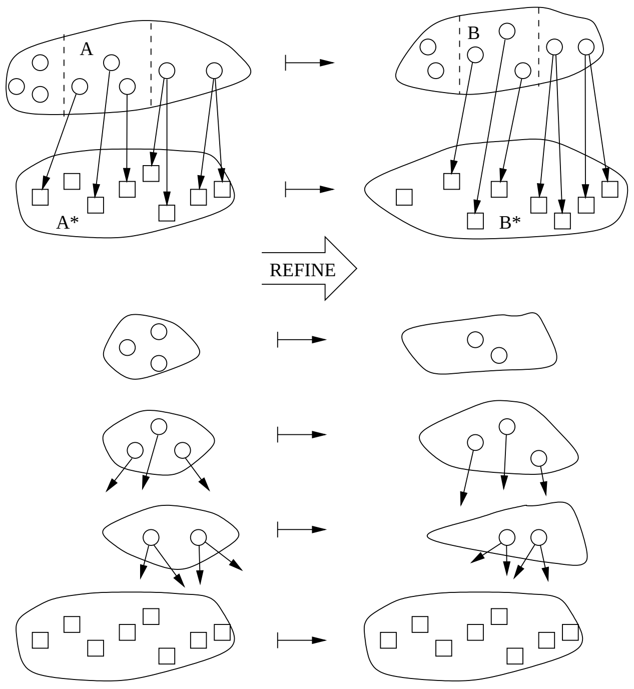

Fig. 3. Sketch of the REFINE transformation

$$\{A_{A^*\longmapsto B^*,i,j,k} \longmapsto B_{A^*\longmapsto B^*,i,j,k} \mid A \longmapsto B \in \mathcal{C}, j,k \in \mathbb{N}\}$$
$$\setminus\{\emptyset \longmapsto \emptyset\};$$

END REFINE.

Corollary 4 (Correctness of REFINE). *Let $\mathcal{C}$ be a symmetry specification, $A^* \longmapsto B^*$ be contained in $\mathcal{C}$, and $i \in \mathcal{I}$. Let $\mathcal{C}' = REFINE(\mathcal{C}, A^* \longmapsto B^*, i)$. Then it holds $\Sigma_{\mathcal{C}'} = \Sigma_{\mathcal{C}}$.*

Proof. Apply Corollary 3 to every $A \longmapsto B$ in $\mathcal{C}$. □

Observe that REFINE, applied to a specification in normal form, returns a specification that is again in normal form. Observe further, that usually constraints consist either only of places or only of transitions (due to $\mathcal{C}_0$). Arcs from places lead always to transitions and vice versa. Thus, if the REFINE transformation is applied to a constraint for places, only transition constraints are splitted while an application to a constraint for transitions splits only place constraints. That is, in practical cases a splitting of the argument constraint $A^* \longmapsto B^*$ does not occur. This might be an advantage for the implementation.

While the REFINE transformation leaves the set of consistent symmetries invariant, the number of consistent bijections decreases when a constraint is replaced by a set of more specific ones. We can interpret the REFINE transformation as the removal of bijections which do not respect the arc relation and arc inscriptions. Of course, we want to apply REFINE transformations as often as possible, i.e. until the specification remains unchanged by any application of REFINE. The following procedure implements this ”REFINE as much as possible”.

```
**PROCEDURE** REFINE*($\mathcal{C}$: spec):spec

**VAR**  $Unprocessed, \mathcal{C}'$ : spec;
     $i : \mathcal{I}$

**BEGIN**
  $Unprocessed := \mathcal{C}$;
  **WHILE** $Unprocessed \neq \emptyset$ **DO**
    Select $A^* \longmapsto B^*$ from $Unprocessed$;
    **FOR ALL** $i \in \chi(F)$ **DO**
      $\mathcal{C}'$ := REFINE($\mathcal{C}, A^* \longmapsto B^*, i$);
      $Unprocessed := (Unprocessed \cup (\mathcal{C}' \setminus \mathcal{C})) \cap \mathcal{C}'$;
      $\mathcal{C} := \mathcal{C}'$;
    **END FOR**;
    $Unprocessed := Unprocessed \setminus \{A^* \longmapsto B^*\}$;
  **END WHILE**;
  **RETURN** $\mathcal{C}$;
**END** REFINE*;
```

In this procedure, $Unprocessed$ consists of all constraints which have not yet been the argument of the REFINE procedure. Thus, as soon as $Unprocessed$ is empty, no REFINE transformation can change the specification any more. Nevertheless the procedure terminates. The specification can change only finitely often, since a single REFINE step replaces always constraints by more specific ones (or leaves them unchanged). When, during a single REFINE step, it holds $\mathcal{C}' = \mathcal{C}$, $Unprocessed$ remains unchanged in the body of the **FOR** loop. When $\mathcal{C}$ remains unchanged during a whole loop, $Unprocessed$ decreases properly in the last statement of the **WHILE** loop.

Since REFINE* consists of a repeated application of REFINE, Corollary 4 holds as well for REFINE*. Additionally, REFINE* serves as symmetry check:

Theorem 3 (Symmetry check). *Let $\mathcal{C}$ be a specification in normal form which is the result of an application of REFINE* and where every $A \longrightarrow B$ in $\mathcal{C}$ satisfies $|A| = |B| = 1$. Then the unique σ in $\Pi_{\mathcal{C}}$ respects the arc relation and all arc inscriptions.*

Proof. First, it is clear that there is a unique bijection consistent with a specification where all left and right sides of constraints are singletons. Let σ be this bijection. We have to show that for arbitrary nodes x, y and every $i \in \mathcal{I}$ there is an arc $[x, y]$ with $\chi([x, y]) = i$ iff there is an arc $[\sigma(x), \sigma(y)]$ with $\chi([\sigma(x), \sigma(y)]) = i$.

"only if": Let $[x, y]$ be an arc with $\chi([x, y]) = i$. Since $\mathcal{C}$ is in normal form, there are constraints $A_x \longmapsto B_x$ and $A_y \longmapsto B_y$ in $\mathcal{C}$ such that $x \in A_x$ and $y \in B_y$. First we show that both constraints have been arguments to a REFINE call when running the REFINE* procedure. Both constraints were once elements of the $Unprocessed$ set (either they were contained in the argument specification, then the initial statement put them into $Unprocessed$, or they are result of splitting a larger constraint by a REFINE call, then they are contained in the result of the call but not in its argument, and consequently put into $Unprocessed$). Without being an argument to a REFINE call, they cannot be removed from $Unprocessed$. Thus, both constraints have been arguments to a REFINE call.

Therefore one of them was an argument (among others together with the value i) when the other constraint was already contained in the current specification. This call did not split the other constraint. Thus, according to the definition of the split operations for REFINE, there are as many arcs from A_x to A_y with inscription i as there are arcs from B_x to B_y with inscription i. Hence, (B_x contains just $\sigma(x)$ and B_y contains $\sigma(y)$) the arc $[\sigma(x), \sigma(y)]$ exists and is inscribed with i.

"if" This part of the proof relies on the same arguments as the "only if" part. The only difference is that we have to conclude from the existence of an arc between B_x and B_y to the arc between A_x and A_y. □

Since REFINE always replaces one specification by one other, it cannot be sufficient to compute all symmetries of a non trivial symmetry group. For this purpose we need a rule which replaces a specification by more than one other. This is the task of the DEFINE transformation which shall be considered in the next section.

5 The basic calculation procedure

Consider a constraint $A \longmapsto B$ and a node $x \in A$. According to the definitions a consistent symmetry σ assigns a value in B to x. Thus, σ is consistent with one of the pairs of constraints $\{\{x\} \longmapsto \{y\}, A \setminus \{x\} \longmapsto B \setminus \{y\}\}$ for $y \in B$. We implement this observation within the DEFINE transformation.

```
**PROCEDURE** DEFINE($\mathcal{C}$ :spec;$A \longmapsto B$ :constr.;$x, y$ : $node$):spec
**BEGIN**
    **RETURN** $(\mathcal{C} \setminus \{A \longmapsto B\})$
        $\cup\{\{x\} \longmapsto \{y\}, A \setminus \{x\} \longmapsto B \setminus \{y\}\}$;
**END** DEFINE;
```

It is easy to see the correctness of this procedure:

Theorem 4 (Correctness of DEFINE). *Let $\mathcal{C}$ be a symmetry specification, $A \longmapsto B \in \mathcal{C}$, and $x \in A$. Then it holds*

$$\Pi_{\mathcal{C}} = \bigcup_{y \in B} \Pi_{DEFINE(\mathcal{C}, A \longmapsto B, x, y)}$$

and therefore $\Sigma_{\mathcal{C}} = \bigcup_{y \in B} \Sigma_{DEFINE(\mathcal{C}, A \longmapsto B, x, y)}$. □

Splitting a constraint by DEFINE may allow us to split constraints further using REFINE*. Thus, the basic calculation procedure consists of alternating calls of DEFINE and REFINE.

```
**PROCEDURE** ComputeSymmetries($\mathcal{C}$ : spec) :SetOfSymmetries
**VAR**   $\Sigma$: SetOfSymmetries;
        $\mathcal{C}'$: spec;
**BEGIN**
    $\mathcal{C}$ := REFINE*($\mathcal{C}$);
    **IF** exists $A \longmapsto B$ in $\mathcal{C}$ such that $|A| \neq |B|$ **THEN**
        **RETURN** $\emptyset$;
    **END IF**;
    **IF** for all $A \longmapsto B$ in $\mathcal{C}$ it holds $|A| = |B| = 1$ **THEN**
        **RETURN** $\Pi_{\mathcal{C}}$;
    **END IF**;
    Select a $A \longmapsto B$ in $\mathcal{C}$ such that $|A| > 1$;
    Select an $x$ in $A$;
    $\Sigma$ := $\emptyset$;
    **FOR** all $y$ in $B$ **DO**
        $\mathcal{C}'$ := DEFINE($\mathcal{C}, A \longmapsto B, x, y$);
        $\Sigma$ := $\Sigma \cup$ ComputeSymmetries($\mathcal{C}'$);
    **END FOR**;
    **RETURN** $\Sigma$;
**END** ComputeSymmetries.
```

Theorem 5 (Correctness of ComputeSymmetris).
ComputeSymmetries($\mathcal{C}$) calculates $\Sigma_{\mathcal{C}}$.

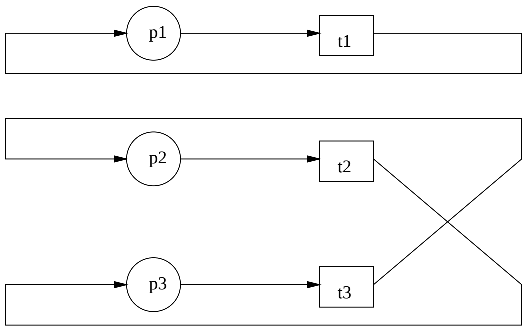

Fig. 4. A Petri net causing dead branches

Proof. The "**RETURN** $\emptyset$" inside the first **IF** statement is correct since no bijection can be consistent with such a constraint. The correctness of "**RETURN** Π_C" is captured by Theorem 3. The correctness of the remaining part of the algorithm relies on Theorem 4 and Corollary 4. □

For a "symmetry on demand" problem, a modification of this procedure can be implemented. Whenever a first symmetry has been found, the procedure can return immediately, since only the existence of a symmetry is of any interest.

Example 2. Consider Fig. 4.. Assume that all arcs are inscribed with the same value. We run ComputeSymmetries($\{\{p1, p2, p3\} \longmapsto \{p1, p2, p3\}, \{t1, t2, t3\} \longmapsto \{t1, t2, t3\}\}$). Figure 5. gives an overview on the calculation. In the sequel we comment the different steps in detail.

All transitions have one arc from a place and one arc to a place. Consequently no REFINE transformation changes anything on the initial specification. We select the constraint containing the places, choose $x = p1$ and apply DEFINE. The first application yields $\{\{p1\} \longmapsto \{p1\}, \{p2, p3\} \longmapsto \{p2, p3\}, \{t1, t2, t3\} \longmapsto \{t1, t2, t3\}\}$. Then a REFINE using $\{p1\} \longmapsto \{p1\}$ as argument splits the transition constraint into $\{t1\} \longmapsto \{t1\}$ and $\{t2, t3\} \longmapsto \{t2, t3\}$. Further REFINE calls do not change anything. Then a second DEFINE splits $\{p2, p3\} \longmapsto \{p2, p3\}$ into $\{p2\} \longmapsto \{p2\}$ and $\{p3\} \longmapsto \{p3\}$. The REFINE* on the next level of recursion yields the specification $\{\{p1\} \longmapsto \{p1\}, \{p2\} \longmapsto \{p2\}, \{p3\} \longmapsto \{p3\}, \{t1\} \longmapsto \{t1\}, \{t2\} \longmapsto \{t2\}, \{t3\} \longmapsto \{t3\}\}$. This specification specifies the identity which is always a symmetry. The procedure backtracks to the last DEFINE and splits $\{p2, p3\} \longmapsto \{p2, p3\}$ into $\{p2\} \longmapsto \{p3\}$ and $\{p3\} \longmapsto \{p2\}$. The next REFINE* splits the transition constraints accordingly, leading to the first non–trivial symmetry.

Since all values for $p2$ in the last DEFINE have been checked, the procedure returns to the first DEFINE, splitting the constraint for all three places into $\{p1\} \longmapsto \{p2\}$ and $\{p2, p3\} \longmapsto \{p1, p3\}$. Assume that the first sub-

$$\begin{tikzcd}
& \begin{array}{l}\{p1,p2,p3\} \longmapsto \{p1,p2,p3\}\\ \{t1,t2,t3\} \longmapsto \{t1,t2,t3\}\end{array} \arrow[d, "\text{REFINE*}"] & \\
& \begin{array}{l}\{p1,p2,p3\} \longmapsto \{p1,p2,p3\}\\ \{t1,t2,t3\} \longmapsto \{t1,t2,t3\}\end{array} \arrow[dl] \arrow[d] \arrow[dr, "\text{DEFINE}"] & \\
\begin{array}{l}\{p1\} \longmapsto \{p1\}\\ \{p2,p3\} \longmapsto \{p2,p3\}\\ \{t1,t2,t3\} \longmapsto \{t1,t2,t3\}\end{array} \arrow[d, "\text{REFINE*}"] & \begin{array}{l}\{p1\} \longmapsto \{p2\}\\ \{p2,p3\} \longmapsto \{p1,p3\}\\ \{t1,t2,t3\} \longmapsto \{t1,t2,t3\}\end{array} \arrow[d, "\text{REFINE*}"] & \begin{array}{l}\{p1\} \longmapsto \{p3\}\\ \{p2,p3\} \longmapsto \{p1,p2\}\\ \{t1,t2,t3\} \longmapsto \{t1,t2,t3\}\end{array} \arrow[d, "\text{REFINE*}"] \\
\begin{array}{l}\{p1\} \longmapsto \{p1\}\\ \{p2,p3\} \longmapsto \{p2,p3\}\\ \{t1\} \longmapsto \{t1\}\\ \{t2,t3\} \longmapsto \{t2,t3\}\end{array} \arrow[d] \arrow[dr, "\text{DEFINE}"] & \emptyset & \emptyset \\
\begin{array}{l}\{p1\} \longmapsto \{p1\}\\ \{p2\} \longmapsto \{p2\}\\ \{p3\} \longmapsto \{p3\}\\ \{t1\} \longmapsto \{t1\}\\ \{t2,t3\} \longmapsto \{t2,t3\}\end{array} \arrow[d, "\text{REFINE*}"] & \begin{array}{l}\{p1\} \longmapsto \{p1\}\\ \{p2\} \longmapsto \{p3\}\\ \{p3\} \longmapsto \{p2\}\\ \{t1\} \longmapsto \{t1\}\\ \{t2,t3\} \longmapsto \{t2,t3\}\end{array} \arrow[d, "\text{REFINE*}"] & \\
\begin{array}{l}\{p1\} \longmapsto \{p1\}\\ \{p2\} \longmapsto \{p2\}\\ \{p3\} \longmapsto \{p3\}\\ \{t1\} \longmapsto \{t1\}\\ \{t2\} \longmapsto \{t2\}\\ \{t3\} \longmapsto \{t3\}\end{array} & \begin{array}{l}\{p1\} \longmapsto \{p1\}\\ \{p2\} \longmapsto \{p3\}\\ \{p3\} \longmapsto \{p2\}\\ \{t1\} \longmapsto \{t1\}\\ \{t2\} \longmapsto \{t3\}\\ \{t3\} \longmapsto \{t2\}\end{array} &
\end{tikzcd}$$

Fig. 5. Calculation of symmetries for Fig. 4.

sequent REFINE is called with argument $\{p1\} \longmapsto \{p2\}$. Then REFINE splits $\{t1, t2, t3\} \longmapsto \{t1, t2, t3\}$ into $\{t1\} \longmapsto \emptyset$ (since only $t1$ has an arc from and an arc to $p1$ while no node has an arc from and to $p2$), $\emptyset \longmapsto \{t2\}$ (since no node has an arc to, but no arc from $p1$ while $t2$ has an arc to but no arc from $p2$), $\emptyset \longmapsto \{t3\}$ (no transition has an arc from, but no arc to $p1$; $t3$ has an arc from but no arc to $p2$), and $\{t2, t3\} \longmapsto \{t1\}$ ($t2$ and $t3$ are not at all connected to $p1$ while $t1$ is not connected at all to $p2$). Actually we can stop the REFINE procedure at this stage since it is clear that no symmetry is consistent with the current specification. However, further REFINE calls would keep constraints with different left hand and right hand sides inside the specification. Therefore, they would not cause any harm, except wasting

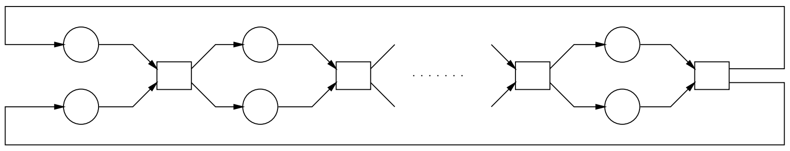

Fig. 6. A net with exponentially many symmetries

run–time. After finishing REFINE*, we immediately return the empty set of symmetries to the caller's level.

The last loop for the first DEFINE, with $\{p1\} \longmapsto \{p3\}$ and $\{p2, p3\} \longmapsto \{p1, p2\}$ runs into a similar dead branch as the previous one. Thus, the calculated symmetry group consists of two elements: the identity, and the mapping that exchanges $p2$ with $p3$ and $t2$ with $t3$.

Figure 5. suggests to regard a symmetry calculation as exploration of a tree. The DEFINE operations correspond to branching nodes in the tree while REFINE* operations just narrow the search space for the subsequent DEFINE. When the calculation starts with a specification in normal form, the depth of the tree is at most linear in the number of nodes of the net (since every DEFINE definitely splits a constraint into two, REFINE does not unify constraints, both operations preserve normal form, and a specification in normal form cannot contain more constraints than nodes). The nodes where the tree branches are those corresponding to DEFINE operations. The branching factor depends on the size of the selected constraint for DEFINE, but cannot be larger than the number of nodes. For the procedures REFINE, REFINE*, and DEFINE, we have polynomial implementations in INA [RS97]. Thus, the overall run–time of the algorithm is in worst case exponential in the size of the net. This is however no surprise, since there are nets with exponentially many symmetries.

Example 3. Consider Fig. 6.. It depicts a ring of, say n, transitions. Every rotation of this ring is a symmetry. This yields n symmetries. Additionally, for every pair of parallel places between two transitions, the bijection which exchanges these two places and is the identity on all other nodes, is a symmetry as well. There are n pairs of places which can be exchanged this way. Every rotation can be composed with arbitrary many of these exchangings, resulting in $n \cdot 2^n$ different symmetries.

Consequently, an algorithm that computes all symmetries can never be polynomial in the size of the net. It arises the question, if it is at least polynomial in the size of the output. Since both REFINE* and DEFINE can be implemented polynomially, the algorithm would be polynomial in the size of the output if we could exclude dead branches (branches of the calculation tree resulting in the empty set of symmetries). The above example shows that

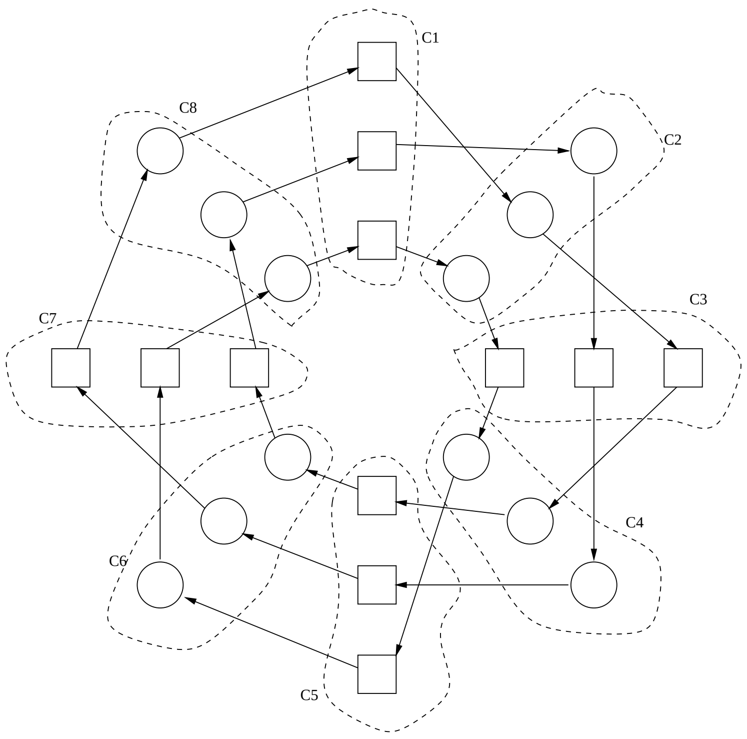

Fig. 7. Local information is not sufficient to avoid dead branches

dead branches may appear in the present algorithm. The following example shows that dead branches cannot be avoided, if at all, without an expensive pre–processing.

Example 4. Consider a net like in Fig. 7. It consists of a ring where alternatingly *triples* of places and *triples* of transitions are placed. From every triple of nodes there are three arcs pointing to the neighbour triple to the right in such a manner that one arc is leaving from every node and one arc is arriving at every node. The net in Fig. 4. is also of this kind (having two triples). Let n be the number of triples and let every triple form a constraint (mapping to itself). Depending on the particular kind of connections between adjacent triples, the net can consist of one ring with $3n$ nodes, two rings (one with $2n$ and one with n nodes — this is the case in the examples), or of 3 rings of length n. In the first case, the net has 3 symmetries, in the second case 2 and in the last case 6 symmetries. Every of the three situations can be transformed into each other by changing the connections between only one pair of adjacent triples. Hence, in order to find out which mappings lead to symmetries it is necessary to consider the *whole* ring. Therefore, algorithms that use only informations about the local environment of nodes (up to whatever horizont) cannot distinguish correct from incorrect mappings in the described situation.

The example shows that an expensive pre–processing, including at least the investigation of the transitive closure of the arc relation would be necessary for avoiding dead branches. On the other hand, experience with running several examples (example nets taken from [Rei85, Sta90, Jen92, WWV$^+$97] and about 20 nets sent to the developpers of the INA tool [RS97] by users of the tool) taught us that dead branches appear rather seldom (we found exactly one case). In the overwhelming majority of cases where we run the procedure on a net without any non–trivial symmetry, the initial REFINE* returns the specification for the identity which is always a symmetry. Likewise, in most cases where REFINE* returns a non–singleton constraint $A \longmapsto B$, a node $x \in A$ may really have every node B as its image. Thus, to our experience the procedure behaves "almost" polynomial in the size of the output when the average run–time is considered rather than the worst case. For further reduction of run–time, we choose another way. Instead of calculating all symmetries, we calculate only a *generating set* which turns out to be polynomial in the size of the net. This generating set can be calculated using a small modification of the present algorithm. This way we can reduce the run–time for the pre–processing of symmetries to an average which is "almost" polynomial in the size of the net, whatever "almost" means. We return to this topic in Sects. 8 and 9. Meanwhile, we propose modifications of the basic calculation procedure which are able to solve calculation problems which are currently not covered by our general approach, in particular the coverability problem on demand and the calculation of symmetries for self–modifying nets.

6 The coverability problem

Consider the coverability problem for place/transition nets. A state of a place/transition net is a mapping $m : P \longrightarrow \mathbb{N}$. A generalised state, as it appears in coverability graphs, is a mapping $m : P \longrightarrow \mathbb{N} \cup \{\omega\}$, where ω is an additional element which satisfies for all $x \in \mathbb{N} \cup \{\omega\}$, $x \leq \omega$. $m(p)$ is usually interpreted as "the number of tokens on place p". ω stands for an unbounded number of tokens. The symmetric coverability problem consists of deciding whether there is a symmetry such that the symmetric image of one given (generalised) state is covered by another (generalised) state. When the "symmetry on demand" concept (see Sect. 3) is used, the symmetric coverability problem appears during the generation of coverability graphs. It appears as well when the generation of reachability graphs is augmented with a boundedness check. A place/transition net is unbounded (i.e. the reachability graph is infinite) iff the reachability graph contains a node from which a strictly covering node is reachable. The clean formulation of the symmetric coverability problem on demand is as follows: given a place/transition net N, a specification $\mathcal{C}$ of a symmetry group Σ and two states m_1 and m_2, decide, whether there is a symmetry σ in Σ such that for all places p of N, $\sigma(m_1)(p) \leq m_2(p)$.

Throughout this section we fix the input parameters $m_1, m_2, \mathcal{C}$ (as specification of a symmetry group) and N.

Note, that the symmetric coverability problem does not apply when symmetries are pre–processed. In this case, the incorporation of symmetries into the coverability graph would be similar to the incorporation into the reachability graph (see Sect. 3 for a comparison of pre–processing and symmetry on demand approaches).

Example 5. Assume, N has 3 places and consider the states $m_1 = (4, 3, 5)$ and $m_2 = (5, \omega, 4)$ (the components stand for the values on the places $p1$, $p2$, and $p3$, respectively). There are exactly four bijections which are candidates for solutions of the coverability problem:

1. $p1 \longmapsto p1$, $p2 \longmapsto p3$, $p3 \longmapsto p2$;
2. $p1 \longmapsto p2$, $p2 \longmapsto p3$, $p3 \longmapsto p1$;
3. $p1 \longmapsto p3$, $p2 \longmapsto p1$, $p3 \longmapsto p2$;
4. $p1 \longmapsto p3$, $p2 \longmapsto p2$, $p3 \longmapsto p1$.

Assume, there is a specification that specifies exactly this set of bijections. Then there is a specification in normal form, too. In a specification in normal form, every node appears only in the left side of one constraint. Let $A \longmapsto B$ be the unique constraint with $p1 \in A$. Then B must consist of all three places, since all places are possible images for $p1$. Consequently A must consist of all three places as well, since otherwise the empty set of bijections would be specified. Unfortunately, the resulting specification $\{\{p1, p2, p3\} \longmapsto \{p1, p2, p3\}\}$ specifies not only the four listed symmetries, but, for instance, also the identity that is not a candidate to solve the given coverability problem.

Hence, with the current framework of specifications we are not able to deal with the symmetric coverability problem. On the other hand, parts of the specification, namely those specifying the used symmetry group (symmetric initial state, fixed points, etc.) can be well specified within the framework of the previous sections. Consequently, we do not throw away the major concepts of symmetry calculation. Instead, we augment constraints by an additional feature and modify the REFINE and DEFINE procedures accordingly. In particular, we append a list to every constraint, specifying the set of possible images for the nodes appearing in the left hand side of a constraint.

Definition 7 (Modified constraints). *Let $N = [P, T, F, \mathcal{I}, \chi, m_0]$ be a place/transition net. A structure $A \longmapsto B; \Gamma$ is a constraint iff $A \subseteq P \cup T$, $B \subseteq P \cup T$, and $\Gamma : A \longrightarrow \wp(B)$. A symmetry σ is* consistent *with a constraint $A \longmapsto B; \Gamma$ iff $\sigma(A) = B$ and for all $x \in A$, $\sigma(x) \in \Gamma(x)$. A (modified) specification is a set of constraints.*

As in Definition 5, $\Sigma_{\mathcal{C}}$ denotes the set of symmetries being consistent with all constraints in $\mathcal{C}$, and $\Pi_{\mathcal{C}}$ the set of all bijections being consistent with $\mathcal{C}$.

Remark 1. A constraint $A \longmapsto B$ according to Definition 5 corresponds exactly to the modified constraint $A \longmapsto B; \Gamma$ where for all $x \in A$, $\Gamma(x) = B$.

Every coverability problem on demand can be specified using modified constraints. Thereby the information about coverability is coded in the additional parts of the constraints corresponding to places. For transitions, the additional features of constraints are not very interesting.

Definition 8 (Specification of coverability problems). *Let $\mathcal{C}$ be the specification of the involved symmetry group (using old constraints). Let m_1 and m_2 be states. Define $\mathcal{C}_{m_1 \leq m_2}$ by:*

$A \longmapsto B; \Gamma \in \mathcal{C}_{m_1 \leq m_2}$ iff

$A \longmapsto B \in \mathcal{C}$,
for all $p \in A \cap P$: $\Gamma(p) = \{p' \mid p' \in B \cap P, m_1(p) \leq m_2(p')\}$, and
for all $t \in A \cap T$: $\Gamma(t) = B \cap T$.

Example 6. Consider the two states of Example 5, and let $\mathcal{C} = \{\{p1, p2, p3\} \longmapsto \{p1, p2, p3\}, \{t1, t2\} \longmapsto \{t1, t2\}\}$. Then

$$\begin{aligned}\mathcal{C}_{(4,3,5)\leq(5,\omega,4)} = \{&\{p1, p2, p3\} \longmapsto \{p1, p2, p3\}; \Gamma_1,\\ &\{t1, t2\} \longmapsto \{t1, t2\}; \Gamma_2\},\end{aligned}$$

where $\Gamma_1(p1) = \Gamma_1(p2) = \{p1, p2, p3\}$, $\Gamma_1(p3) = \{p1, p2\}$, and $\Gamma_2(t1) = \Gamma_2(t2) = \{t1, t2\}$.

The following lemmas list some properties of the specification $\mathcal{C}_{m_1 \leq m_2}$ and clarify the relation between modified constraints.

Lemma 1 (Coverability specification). *Let $\mathcal{C}$ be the specification of a symmetry group (without modification) where for all $p \in P$ there is a constraint $A \longmapsto B$ in $\mathcal{C}$ where $p \in A$ (the left sides of $\mathcal{C}$ cover all P). Then $\Sigma_{\mathcal{C}_{m_1 \leq m_2}}$ is the set of all symmetries σ in $\Sigma_{\mathcal{C}}$ which satisfy $\sigma(m_1) \leq m_2$.*

Proof. It follows directly from the definition of modified constraints that $\Sigma_{\mathcal{C}_{m_1 \leq m_2}} \subseteq \Sigma_{\mathcal{C}}$. Let $p \in P$. By assumption, there is a constraint $A \longmapsto B; \Gamma$ such that $p \in A$. If σ is consistent with $\mathcal{C}_{m_1 \leq m_2}$, then it holds $m_1(p) \leq m_2(\sigma(p))$ (due to the definition of Γ in $\mathcal{C}_{m_1 \leq m_2}$). Consequently, we obtain $\sigma(m_1) \leq m_2$.

If, the other way round, σ is in $\Sigma_{\mathcal{C}}$ and satisfies $\sigma(m_1) \leq m_2$, then σ is obviously consistent with all modified constraints in $\mathcal{C}_{m_1 \leq m_2}$, in particular with the settings of Γ. □

Lemma 2 (The $\Gamma(p)$ include each other). *Let $A \longmapsto B; \Gamma \in \mathcal{C}_{m_1 \leq m_2}$. For all $p_1, p_2 \in A \cap P$ it holds $m_1(p_1) \leq m_2(p_2)$ iff $\Gamma(p_1) \supseteq \Gamma(p_2)$.*

Proof. Nodes which cover p_2, cover all nodes with less tokens as well. Formally, the claim follows from the transitivity of $\leq$. □

Lemma 3 (Constraint splitting). *Let $A \longmapsto B; \Gamma$ be a modified constraint. Let $A' \subseteq A$ and $B' \subseteq B$. Then a symmetry σ is consistent with $A \longmapsto B; \Gamma$ and (the unmodified constraint) $A' \longmapsto B'$ iff σ is consistent with $A' \longmapsto B'; \Gamma_1$ and $A \setminus A' \longmapsto B \setminus B'; \Gamma_2$ where $\Gamma_1(x) = \Gamma(x) \cap B'$ and $\Gamma_2(x) = \Gamma(x) \setminus B'$.*

Proof. Follows directly from the definitions. □

Lemma 3 can be used to adapt the REFINE and DEFINE procedures of the basic calculation procedure to modified constraints. REFINE and DEFINE can be done just by ignoring the augmented part of the involved constraints. All we have to do is to update the Γ after splitting a constraint. Additionally, we may restrict the calls of DEFINE($\mathcal{C}, x, y$) to those y which appear in $\Gamma(x)$ (where Γ is part of the currently considered constraint). However, with the modified REFINE and DEFINE procedures we do not sufficiently involve the Γ-part of constraints into the refinement process. Therefore we need an additional refinement operation which is justified by the following lemma.

Lemma 4 (Additional refinement). *Let $A \longmapsto B; \Gamma$ appear in a specification $\mathcal{C}_{m_1 \leq m_2}$.*

1. *Assume, there is a $x \in A \cap P$ such that $|\{y \mid y \in A \cap P, m_1(x) \leq m_1(y)\}| = |\Gamma(x)|$. Then every symmetry which is consistent with $\mathcal{C}_{m_1 \leq m_2}$ is consistent with the (unmodified) constraint $\{y \mid y \in A \cap P, m_1(x) \leq m_1(y)\} \longmapsto \Gamma(x)$.*
2. *Assume, there is a $x \in A \cap P$ such that $|\{y \mid y \in A \cap P, m_1(x) \leq m_1(y)\}| > |\Gamma(x)|$. Then no symmetry is consistent with $\mathcal{C}_{m_1 \leq m_2}$.*

Proof. Let σ be a symmetry that is consistent with $\mathcal{C}_{m_1 \leq m_2}$. By definition it holds $\sigma(x) \in \Gamma(x)$ for all $x \in A$. With Lemma 2, $\sigma(\{y \mid y \in A \cap P, m_1(x) \leq m_1(y)\}) \subseteq \Gamma(x)$.

Ad 1. Since σ is a bijection and both sides of the inclusion have same cardinality, the "$\subseteq$" can be replaced by "$=$".

Ad 2. This assumption and the above inclusion cannot be satisfied simultaneously by any bijection. □

Part 2 of Lemma 4 can be used to stop the further consideration of inconsistent specifications. With Part 1, we can deduce further restrictions for a specification. This way, we can transfer step by step the Γ–part of a constraint to the traditional parts of the constraint. This transfer is finished before the specification consists of singleton constraints. For a singleton constraint $\{p\} \longmapsto \{p'\}; \Gamma$ there are only two possible values for $\Gamma(p)$: $\emptyset$ or $\{p\}$. In the first case Part 2 of the above lemma states that no symmetry is consistent with $\{p\} \longmapsto \{p'\}$ while in the latter case the Γ–part does not restrict the symmetries more than $\{p\} \longmapsto \{p'\}$ itself. Thus, if a modified algorithm consisting of REFINE, DEFINE (both modified for updating Γ–parts), and the additional refinement operation according to Lemma 4 stops on a specification of singleton constraints, and neither REFINE nor additional refinement operations can change this specification anymore, then the unique bijection that is consistent with this specification is indeed a symmetry witnessing the symmetric covering of the two input states. If the procedure detects an inconsistency (according to Part 2 of Lemma 4, or according to unbalanced cardinalities of the traditional part of any constraint), then the procedure has to backtrack and to try another choice of the last recent DEFINE operation. When all choices of DEFINE operations fail to produce a symmetry, we can be sure (using the above set of lemmas) that the given states are not symmetrically coverable.

Example 7. Consider the net of Fig. 4. with the maximal symmetry group, i.e. the one specified by $\{p_1, p_2, p_3\} \longmapsto \{p_1, p_2, p_3\}, \{t_1, t_2, t_3\} \longmapsto \{t_1, t_2, t_3\}\}$. We want to decide if the state $(4, 3, 5)$ is symmetrically covered by $(5, \omega, 4)$. That is, we enter the calculation procedure with the above specification, where the place constraint is augmented by $\Gamma(p_1) = \{p_1, p_2, p_3\}$, $\Gamma(p_2) = \{p_1, p_2, p_3\}$, and $\Gamma(p_3) = \{p_1, p_2\}$. (In the sequel, we report only the value $\Gamma(p_3)$, since Γ–values which cover the whole right side of the associated constraint, do not establish any restriction to the set of consistent symmetries.)

For this specification, neither a REFINE transformation can be performed, nor can we apply Lemma 4. Thus, we have to do a DEFINE, say for place p_2. As the first branch we consider $\{\{p_2\} \longmapsto \{p_2\}, \{p_1, p_3\} \longmapsto \{p_1, p_3\}, \{t_1, t_2, t_3\} \longmapsto \{t_1, t_2, t_3\}\}$. The Γ–value of p_3 changes to $\{p1\}$, since p_2 is no longer contained in the right side of $\{p_1, p_3\} \longmapsto \{p_1, p_3\}$.

Now the application condition of Lemma 4 is satisfied, and we can introduce the constraint $\{p_3\} \longmapsto \{p_1\}$. Consequently, we obtain $\{p_1\} \longmapsto \{p_3\}$ as well (in order to keep the specification in normal form). Now a REFINE can be applied. We choose $A^* = \{p_3\}$, $B^* = \{p_1\}$, and $i = 1$. REFINE produces, among others, the constraint $\{t_1, t_2, t_3\}_{\{p_3\} \longmapsto \{p_1\},1,1,1} \longmapsto \{t_1, t_2, t_3\}_{\{p_3\} \longmapsto \{p_1\},1,1,1} = \emptyset \longmapsto \{t_1\}$. There is no symmetry which can be consistent with this constraint, thus we have to continue with the next branch of DEFINE.

Consider next $\{\{p_2\} \longmapsto \{p_3\}, \{p_1, p_3\} \longmapsto \{p_1, p_2\}, \{t_1, t_2, t_3\} \longmapsto \{t_1, t_2, t_3\}\}$. This yields $\Gamma(p_3) = \{p_1, p_2\}$. Therefore even the last Γ–value does not express any restriction. We can continue with REFINE*. This leads us after two steps to the symmetry $\{p_1 \longmapsto p_1, p_2 \longmapsto p_3, p_3 \longmapsto p_2, t_1 \longmapsto t_1, t_2 \longmapsto t_3, t_3 \longmapsto t_2\}$. This symmetry indeed witnesses the symmetric covering of the two given states.

Of course, we could come to the same result by testing both symmetries, as calculated in the previous section. But remember that we tried to solve the coverability problem ”on demand”; the general purpose of the ”on demand” concept is to decide problems without pre–processing symmetries!

7 Symmetry groups for self–modifying nets

In a self–modifying net [Val78], the inscription of an arc is a formal sum of places.

Definition 9 (Self–modifying net). *A net $SN = [P, T, F, \mathcal{I}, \chi, m_0]$ is a* self–modifying net *iff for all $x \in P \cup T$, $\chi(x) =$* **nil** *(places and transitions are not inscribed at all), $\chi\mid_F$: $F \longrightarrow (P \longrightarrow \mathbb{Z})$ (every arc is inscribed with a formal sum of places, i.e. a mapping that assigns an integer value to every place), and $m_0 : P \longrightarrow \mathbb{N}$.*

Given a state m, the number of tokens to be produced or consumed via an arc is calculated by replacing the places in the corresponding sum by the current number of tokens on it. Therefore, the multiplicity of arcs may change according to the change of states.

Definition 10 (Transition rule). *Let $SN = [P, T, F, \mathcal{I}, \chi, m_0]$ be a self–modifying net, and m, m' states, i.e. $m, m' : P \longrightarrow \mathbb{N}$. A transition $t \in T$ is enabled at m iff for all $p \in P$, $\sum_{p' \in P} \chi([p,t])(p') \cdot m(p') \leq m(p)$. The relation $m \xrightarrow{t} m'$ holds iff t is enabled at m and for all $p \in P$, $m'(p) = m(p) - max\{0, \sum_{p' \in P} \chi([p,t])(p') \cdot m(p')\} + max\{0, \sum_{p' \in P} \chi([t,p])(p') \cdot m(p')\}$ (negative values of arc inscriptions are treated as zero).*

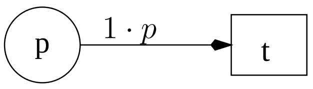

Fig. 8. Reset arc with self-modifying nets

A popular application of self-modification is the concept of so-called ”reset arcs” which set the number of tokens on a place to zero, whatever the current state is. (see Fig. 8.).

In order to get the relation ”$m \xrightarrow{t} m'$ iff $\sigma(m) \xrightarrow{\sigma(t)} \sigma(m')$” which is the key to any state space reduction, it is necessary to change the definition of symmetries for self–modifying nets. Instead of ”σ maps an arc to an arc with the *same* inscription” we have to require ”σ maps an arc to an arc with *symmetric* inscription”. This is due to the fact that net elements appear in the net inscription. Therefore, self–modifying nets do not fall under the general assumption of Proposition 1.

Definition 11 (Symmetries of self–modifying nets).
Let $SN = [P,T,F,\mathcal{I},\chi,m_0]$ be a self–modifying net. A bijection $\sigma : P \cup T \longrightarrow P \cup T$ is a symmetry *of SN iff*

σ respects the node type, i.e. $\sigma(P) = P$, $\sigma(T) = T$;
σ respects the arc relation, i.e. $[x,y] \in F$ iff $[\sigma(x),\sigma(y)] \in F$; and
σ respects the arc inscriptions, i.e. for all $[x,y] \in F$ and all $p \in P$, $\chi([x,y])(p) = \chi([\sigma(x),\sigma(y)])(\sigma(p))$.

Observe the application of p to the left side and $\sigma(p)$ to the right side of the equation appearing in the third item!

As usual, the set of all symmetries of a self–modifying net forms a group under composition, with the identity as its neutral element. Furthermore, the relation between symmetries and transition occurrences holds.

Theorem 6 (Transition rule respects symmetries). *Let $SN = [P,T,F,\mathcal{I},\chi,m_0]$ be a self–modifying net and σ one of its symmetries. Then it holds for every pair m,m' of states and every $t \in T$: $m \xrightarrow{t} m'$ iff $\sigma(m) \xrightarrow{\sigma(t)} \sigma(m')$.*

Proof. Let $m \xrightarrow{t} m'$. First, we show that $\sigma(t)$ is enabled at $\sigma(m)$. It holds for all $p \in P$

$$\begin{array}{rll} & \sigma(m)(p) & \\ = & m(\sigma^{-1}(p)) & \text{(by Def. 4)} \\ \geq & \sum_{p' \in P} \chi([\sigma^{-1}(p),t])(p') \cdot m(p') & \text{(by } m \xrightarrow{t} m'\text{)} \\ = & \sum_{p' \in P} \chi([p,\sigma(t)])(\sigma(p')) \cdot m(p') & \text{(by Def. 11)} \\ = & \sum_{p' \in P} \chi([p,\sigma(t)])(\sigma(p')) \cdot \sigma(m)(\sigma(p')) & \text{(by Def. 4)} \\ = & \sum_{p' \in P} \chi([p,\sigma(t)])(p') \cdot \sigma(m)(p') & (\sigma \text{ is bijection}) \end{array}$$

The derived inequation is just the enabling condition for $\sigma(t)$ at $\sigma(m)$. In the same manner, we can calculate that the result of firing $\sigma(t)$ at $\sigma(m)$ is just $\sigma(m')$. The reverse direction can be deduced from the above direction by applying σ^{-1} to $\sigma(m) \xrightarrow{\sigma(t)} \sigma(m')$. □

It is obvious that the basic calculation procedure cannot be applied to calculate symmetries according to the modified concept. In particular, the REFINE operation would produce wrong results. There, we split left and right sides of a constraint according to equal arc inscriptions. For the modified concept, it would be necessary to put those nodes into the right side of a constraint which have arcs inscribed with the *symmetric image* of the nodes in the corresponding left side. Unfortunately, we do not know yet the symmetries, hence this approach does not work.

The idea to solve this problem is to find an abstraction of the arc inscription which is common for all elements of the same constraint. With this abstraction, it shall be possible to split constraints properly. For this purpose, consider a specification $\mathcal{C}$ and a mapping $\phi : P \longrightarrow \mathbb{Z}$ (that is, an arbitrary arc inscription).

Define two mappings $\phi_{\mathcal{C},l}$ and $\phi_{\mathcal{C},r}$ ($\phi_{\mathcal{C},l}, \phi_{\mathcal{C},r} : \mathcal{C} \longrightarrow \mathbb{Z}$) by

$$\phi_{\mathcal{C},l}(A \longmapsto B) = \sum_{a \in A \cap P} \phi(a)$$
$$\phi_{\mathcal{C},r}(A \longmapsto B) = \sum_{b \in B \cap P} \phi(b)$$

Corollary 5 (Symmetries respect abstract arcs). *Let $[x,y]$ be an arc of a self–modifying net SN, and $\mathcal{C}$ a symmetry specification. For all symmetries σ of SN which are consistent with $\mathcal{C}$ it holds*

$$\chi([x,y])_{\mathcal{C},l} = \chi([\sigma(x),\sigma(y)])_{\mathcal{C},r}$$

Proof. Let $A \longmapsto B$ be an element of $\mathcal{C}$. Then it holds

$$\begin{array}{lll} & \chi([x,y])_{\mathcal{C},l}(A \longmapsto B) & \\ = & \sum_{x \in A \cap P} \chi([x,y])(p) & \text{(by Def. } \phi_{\mathcal{C},l}) \\ = & \sum_{x \in A \cap P} \chi([\sigma(x),\sigma(y)])(\sigma(p)) & \text{(by Def. 11)} \\ = & \sum_{x \in B \cap P} \chi([\sigma(x),\sigma(y)])(p) & \text{(by Def. 5)} \\ = & \chi([\sigma(x),\sigma(y)])_{\mathcal{C},r} & \text{(by Def. } \phi_{\mathcal{C},r}) \end{array}$$

□

The proof, in particular the third step, shows how the more abstract view on arc inscriptions becomes independent of a particular symmetry. In fact, for an arbitrary arc inscription $\chi([x,y])$, the values $\chi([x,y])_{\mathcal{C},l}$ and $\chi([x,y])_{\mathcal{C},r}$ can be calculated using $\mathcal{C}$. Moreover, the relation established in Corollary 5 can be used to make the REFINE transformation fit to self–modifying nets. Instead of sorting left and right sides of a constraint according to arc inscriptions, we sort the left sides according to the $\chi_{\mathcal{C},l}$, and the right sides according to the $\chi_{\mathcal{C},r}$. Applying this idea, the theorem corresponding to Theorem 2 (the crucial theorem for the correctness of REFINE in the general case) can be expressed as follows:

Theorem 7 (Modified REFINE). *Let $A \longmapsto B$ be a constraint, $\mathcal{C}$ a symmetry specification, and σ a symmetry of a self–modifying net being consistent with both. Let $\phi : \mathcal{C} \longrightarrow \mathbb{Z}$ be a mapping, and $x \in P \cup T$. Then $|\{[x,y] \mid y \in A, \chi([x,y])_{\mathcal{C},l} \equiv \phi\}| = |\{[\sigma(x), z] \mid z \in B, \chi([\sigma(x), z])_{\mathcal{C},r} \equiv \phi\}|$, and $|\{[y,x] \mid y \in A, \chi([y,x])_{\mathcal{C},l} \equiv \phi\}| = |\{[z,\sigma(x)] \mid z \in B, \chi([z,\sigma(x)])_{\mathcal{C},r} \equiv \phi\}|$.*

Proof. Repeat the proof of Theorem 2, but replace references to Definition 3 by references to Corollary 5. Distinguish further arcs between left sides of constraints (use $\chi([x,y])_{\mathcal{C},l}$) and arcs between right sides of constraints (use $\chi([x,y])_{\mathcal{C},r}$). $\square$

Now, similar to the original REFINE transformation, the modified REFINE transformation can be defined as splitting constraints $A \longmapsto B$ into constraints $A_{A^* \longmapsto B^*,\phi,j,k} \longmapsto B_{A^* \longmapsto B^*,\phi,j,k}$, according to the number of arcs from and to a fixed constraint $A^* \longmapsto B^*$.

For two constraints $A \longmapsto B$ and $A^* \longmapsto B^*$, both contained in a specification $\mathcal{C}$, a mapping $\phi : \mathcal{C} \longrightarrow \mathbb{Z}$, and two natural numbers j and k, let

$$\begin{aligned} A_{A^* \longmapsto B^*,\phi,j,k} = \{a \mid a \in A, &|\{[a,y] \mid y \in A^*, \chi([a,y])_{\mathcal{C},l} \equiv \phi\}| = j, \\ &|\{[z,a] \mid z \in A^*, \chi([z,a])_{\mathcal{C},l} \equiv \phi\}| = k\}\,; \\ B_{A^* \longmapsto B^*,\phi,j,k} = \{b \mid b \in B, &|\{[b,y] \mid y \in B^*, \chi([b,y])_{\mathcal{C},r} \equiv \phi\}| = j, \\ &|\{[z,b] \mid z \in B^*, \chi([z,b])_{\mathcal{C},r} \equiv \phi\}| = k\}\,. \end{aligned}$$

Theorem 7 guarantees that this transformation keeps the set of consistent symmetries (of the self–modifying net) invariant. It is straightforward now to implement a modified calculation procedure by combining the modified REFINE transformation with the original DEFINE operation (which is obviously applicable to the new application area without any changes). Therefore it remains to consider the final output of the procedure, i.e. the specifications where no modified REFINE transformation would yield any changes, and where all constraints are built upon singletons.

Let $\mathcal{C}$ be such a specification in normal form, and consider an arc $[x,y]$. Clearly, there are x' and y' such that $\{x\} \longmapsto \{x'\} \in \mathcal{C}$ and $\{y\} \longmapsto \{y'\} \in \mathcal{C}$. Thus, for the unique consistent bijection σ it holds $\sigma(x) = x'$ and $\sigma(y) = y'$. By assumption, no modified REFINE transformation

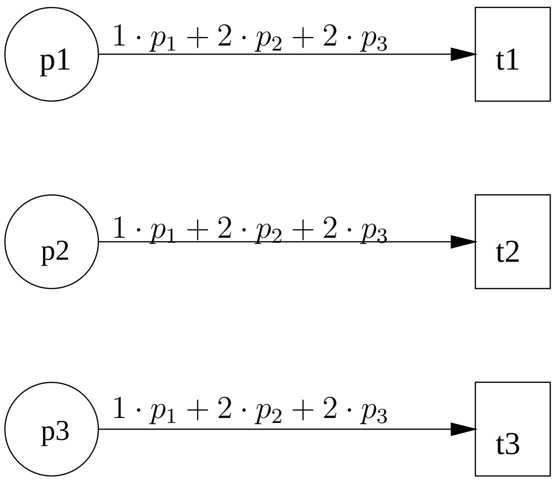
p1
$1 \cdot p_1 + 2 \cdot p_2 + 2 \cdot p_3$
t1
p2
$1 \cdot p_1 + 2 \cdot p_2 + 2 \cdot p_3$
t2
p3
$1 \cdot p_1 + 2 \cdot p_2 + 2 \cdot p_3$
t3

Fig. 9. A self–modifying net

changes anything. Therefore, there is an arc $[x', y'] = [\sigma(x), \sigma(y)]$ such that $\chi([x, y])_{\mathcal{C},l} \equiv \chi([\sigma(x), \sigma(y)])_{\mathcal{C},r}$. Applying the definition of $\phi_{\mathcal{C},l/r}$, and taking into consideration that $\mathcal{C}$ consists exactly of the constraints $\{z\} \longmapsto \{\sigma(z)\}$ (this is just the definition of the only consistent bijection σ), we find that for all places $p \in P$ it holds $\chi([x, y])(p) = \chi([\sigma(x), \sigma(y)])(\sigma(p))$. But this is exactly the requirement to symmetries of self–modifying nets! Thus, the modified procedure, applied to a symmetry specification $\mathcal{C}$, calculates exactly the set of symmetries being consistent with $\mathcal{C}$. Thereby, $\mathcal{C}$ is a normal symmetry specification as considered for the general case. Thus, we are able to apply the full range of specification power, including fixed points, symmetric initial state, respecting of the node type, and the "on demand" specifications to self–modifying nets.

Example 8. Consider the net in Fig. 9.. We want to compute the symmetries without any further restrictions, that is we start with the constraint $\mathcal{C}_0 = \{\{p_1,\, p_2, p_3\} \longmapsto \{p_1, p_2, p_3\}, \{t_1, t_2, t_3\} \longmapsto \{t_1, t_2, t_3\}\}$. For every arc f, it holds $\chi(f)_{\mathcal{C}_0,l}(P \longmapsto P) = \chi(f)_{\mathcal{C}\prime,r}(P \longmapsto P) = 5$, and $\chi(f)_{\mathcal{C}_0,l}(T \longmapsto T) = \chi(f)_{\mathcal{C}_0,r}(T \longmapsto T) = 0$ (values of transition constraints are always equal to zero and can be ignored). So we cannot perform successfully any REFINE transformation. Therefore we apply DEFINE to p_1 and continue with $\mathcal{C}_1 = \{\{p_1\} \longmapsto \{p_1\}, \{p_2, p_3\} \longmapsto \{p_2, p_3\}, \{t_1, t_2, t_3\} \longmapsto \{t_1, t_2, t_3\}\}$.

Now all of $\chi([p_1, t_1])_{\mathcal{C}_1,l}$, $\chi([p_1, t_1])_{\mathcal{C}_1,r}$, $\chi([p_2, t_2])_{\mathcal{C}_1,l}$, and $\chi([p_2, t_2])_{\mathcal{C}_1,r}$ map $\{p_1\} \longmapsto \{p_1\}$ to 2 and the other place constraint to 3, while $\chi([p_3, t_3])_{\mathcal{C}_1,l}$ and $\chi([p_3, t_3])_{\mathcal{C}_1,r}$ map these to constraints to 1 and 4, respectively. We choose ϕ such that it maps the place constraints to 2 and 3, respectively, and perform a REFINE transformation with

$A^* \longmapsto B^* = T \longmapsto T$. This yields $\mathcal{C}_2 = \{\{p_1\} \longmapsto \{p_1\}, \{p_2\} \longmapsto \{p_2\}, \{p_3\} \longmapsto \{p_3\}, \{t_1, t_2, t_3\} \longmapsto \{t_1, t_2, t_3\}\}$ which can be further refined to the specification for the identity. Observe, that during the descent to the identity the l- and r–indexed mappings are still equal. This situation changes as soon as we step into other branches.

Trying the next branch of the DEFINE operation, we assign p_2 to p_1 and obtain $\mathcal{C}_3 = \{\{p_1\} \longmapsto \{p_2\}, \{p_2, p_3\} \longmapsto \{p_1, p_3\}, \{t_1, t_2, t_3\} \longmapsto \{t_1, t_2, t_3\}\}$. We obtain

	$\{p_1\} \longmapsto \{p_2\}$	$\{p_2, p_3\} \longmapsto \{p_1, p_3\}$
$\chi([p_1, t_1])_{\mathcal{C}_3,l}$	2	3
$\chi([p_1, t_1])_{\mathcal{C}_3,r}$	1	4
$\chi([p_2, t_2])_{\mathcal{C}_3,l}$	2	3
$\chi([p_2, t_2])_{\mathcal{C}_3,r}$	2	3
$\chi([p_3, t_3])_{\mathcal{C}_3,l}$	1	4
$\chi([p_3, t_3])_{\mathcal{C}_3,r}$	2	3

Next, we perform a REFINE with arguments $T \longmapsto T$ and ϕ mapping $\{p_1\} \longmapsto \{p_2\}$ to 2, and the other place constraint to 3. This yields $\{p_2\} \longmapsto \{p_3\}$ (since p_2 but not p_3 has an arc where the l–values in the above table are 2 and 3, while p_3 but not p_1 has an arc with r–values 2 and 3). Accordingly we get $\{p_3\} \longmapsto \{p_1\}$. This specification refines without further complications to a specification of singleton constraints, resulting in the symmetry $\sigma = \{p_1 \longmapsto p_2, p_2 \longmapsto p_3, p_3 \longmapsto p_1, t_1 \longmapsto t_2, t_2 \longmapsto t_3, t_3 \longmapsto t_1\}$. This is indeed a symmetry of the self–modifying net in Fig. 9.

The remaining symmetry — σ^2 — is the result of the last DEFINE branch where the value of p_1 is set to p_3. The reader might verify that other bijections are not symmetries of the depicted net.

The example illustrates how the abstract arc inscriptions converge to the original arc inscriptions while descending recursively to singleton specifications. We start with image vector $[5]$, continue with $[2, 3]$ or $[1, 4]$, respectively, and end up in $[2, 2, 1]$. Thereby the different paths (via $[2, 3]$, or via $[1, 4]$) provide the necessary information to split the constraints by the REFINE transformation. This converging sequence of abstract arc inscriptions is the way how we, similar to our solution for the coverability problem, transfer little by little the problem specification to the calculation procedure. In both cases we weren't able to express the problem specification directly within the general symmetry calculus.

8 Ground sets of symmetries

Now, we return to the general concept of pre–processing symmetries. We develop a polynomial generating set for the set of (up to exponentially many) symmetries. We show that it is sufficient to store the generating set permanently, even when loops over all symmetries are required, as in the application to reachability analysis. The following code fragment sketches the search whether, for some node, an equivalent one has already been computed.

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

The outer loop iterates on all symmetries. Since in most cases the set of symmetries is small, compared with the size of a (reduced) reachability graph, this kind of nesting the loops yields a faster solution than using the ”on demand” concept. The drawback thereby is, that an expensive pre–processing is necessary to run the basic calculation procedure, which is in worst case exponential in the size of the net (see Example 3). Moreover, we have to store somehow the up to exponentially large set of symmetries.

The purpose of this and the next section is to improve upon that. Our generating set shall have the following properties:

- it can be calculated easily by another modification of the basic calculation procedure;
- it has polynomial size (in the size of the net);
- it allows an efficient and repetition free implementation of the ”for all symmetries” loop in the above procedure.

Particularly the last item forces us to use a specific generating set rather than an arbitrary one. In fact it shall turn out that we are able to calculate generating sets of symmetries with a modified calculation procedure, even a smaller one than the one proposed below. Furthermore, arbitrary generating sets can be used to enumerate the whole symmetry group by an exhaustive composition of the generators. However, a blind enumeration is usually not repetition free. Thus, we would run the body of the loop much more frequent than necessary. The generating set introduced in the sequel does not starve from this problem.

Let $N = [P, T, F, \mathcal{I}, \chi, m_0]$ be an arbitrary net. Assume that there is a linear ordering on $P \cup T$. That is, without loss of generality we may assume $P \cup T = \{x_1, \ldots, x_{|P \cup T|}\}$. We fix a symmetry group Σ for N.

Consider now the following subsets of Σ: For j, k in $\{1, \ldots, |P \cup T|\}$, let $S_{jk} = \{\sigma \mid \sigma \in \Sigma, \sigma\mid_{\{x_1,\ldots,x_{j-1}\}}= id, \sigma(x_j) = x_k\}$ (id is the identity). The first index marks that σ is equal to the identity on all nodes with smaller position than j, the second one the image of the j–th node.

Observe some properties of the S_{jk}.

Corollary 6 (Properties of S_{jk}).

(1) If $k_1 \neq k_2$, then for all $j \in \{1, \ldots, |P \cup T|\}$, $S_{jk_1} \cap S_{jk_2} = \emptyset$;
(2) If $k < j$, then $S_{jk} = \emptyset$;
(3) For all $j \in \{1, \ldots, |P \cup T| - 1\}$, $S_{jj} = \bigcup_{k=j+1}^{|P \cup T|} S_{j+1k}$;
(4) $S_{|P \cup T|\,|P \cup T|} = \{id\}$;
(5) $\bigcup_{k=1}^{|P \cup T|} S_{1k} = \Sigma$.

The following property is not as easy to see, but crucial for our approach.

Lemma 5 (Generation of S_{jk}). *Let $j, k \in \{1, \ldots, |P \cup T|\}$ such that $S_{jk} \neq \emptyset$. Let $\sigma^* \in S_{jk}$. Then for all $\sigma \in \Sigma$ holds: $\sigma \in S_{jk}$ iff there is a $\sigma' \in S_{jj}$ such that $\sigma = \sigma^* \circ \sigma'$.*

Proof. "if": Since $\sigma' \in S_{jj}$, it holds $\sigma'(x_i) = x_i$ for $1 \leq i \leq j$. Since $\sigma^* \in S_{jk}$, it holds for all $1 \leq i \leq j - 1$:

$$\sigma(x_i) = \sigma^*(\sigma'(x_i)) = \sigma^*(x_i) = x_i$$

and for x_j:

$$\sigma(x_j) = \sigma^*(\sigma'(x_j)) = \sigma^*(x_j) = x_k$$

Thus, $\sigma \in S_{jk}$.
"only if": Since $\sigma \in S_{jk}$ and $\sigma^* \in S_{jk}$, we get $\sigma^*(x_i) = \sigma(x_i)$ for all $1 \leq i \leq j$ and consequently $\sigma^{*-1}(\sigma(x_i)) = x_i$ for those i. Let $\sigma' = (\sigma^*)^{-1} \circ \sigma$. Thus, $\sigma' \in S_{jj}$ and $\sigma = \sigma^* \circ (\sigma^*)^{-1} \circ \sigma = \sigma^* \circ \sigma'$. □

Lemma 5 states that every S_{jk} can be generated from S_{jj} and one element of S_{jk} itself. Taking into consideration the properties of Corollary 6, we are ready to define a generating set for Σ.

Definition 12 (Ground set of symmetries). *A subset S_0 of Σ is called a* ground set *of Σ iff for all nonempty S_{jk} ($1 \leq j \leq k \leq | P \cup T|$) there is exactly one element of S_{jk} contained in S_0.*

A ground set selects one element from every nonempty S_{jk}. For the S_{jj}, the selected element is necessarily the identity (follows from Items 3 and 4 of Corollary 6). For the other elements of S_0, the first node not mapping to itself determines to which unique S_{jk} this node belongs. Thus, we may write σ_{jk} for the unique node in S_0 which is contained in S_{jk}.

Lemma 6 (S_0 is generating set). *Every ground set S_0 of Σ is a generating set for Σ.*

Proof. By Corollary 6 (Item 4), every element of $S_{|P\cup T|,|P\cup T|}$ can be generated from S_0, since the identity is alway contained in S_0. If, for some j, S_{jj} can be generated, then by Lemma 5, every S_{jk} can be generated as well, since by definition one element of every nonempty S_{jk} is already contained in S_0. By Part 3 of Corollary 6 this is sufficient to generate $S_{j-1,j-1}$. By induction, S_{11} and consequently all S_{1k} can be generated. Thus, by Part 5 of Corollary 6 every element of Σ can be generated. Since S_0 is a subset of Σ and Σ is a group, S_0 cannot generate more elements than those in Σ. □

Lemma 7 (Size of S_0).
A ground set contains at most $\frac{|P\cup T|\cdot(|P\cup T|-1)}{2}+1$ *elements.*

Proof. One element is always the identity, contained in all the S_{jj}. By Corollary 6 (Part 2), the worst case is that additionally all S_{jk} with $j<k$ are nonempty. The number of these sets is $\frac{|P\cup T|\cdot(|P\cup T|-1)}{2}$. □

Up to now, we have shown that ground sets are generating sets of polynomial size. Next we show that ground sets allow us to implement a repetition free enumeration of the whole Σ.

In general, the task is to find an implementation of the code fragment

```
FOR ALL $\sigma \in \Sigma$ DO
    $f(\sigma)$;
END FOR
```

Thereby f is an arbitrary procedure. Given a ground set S_0, we implement this loop recursively.

```
PROCEDURE APPLY($f$:proc.,$\sigma$: symm.,$j : \{0,\dots,|P\cup T|\}$);

VAR $k : \{1,\dots,|P\cup T|\}$;

BEGIN
    IF $j = |P\cup T|$ THEN
        $f(\sigma)$;
    ELSE
        FOR $k := j+1$ TO $|P\cup T|$ DO
            IF $S_{j+1,k} \neq \emptyset$ THEN
                APPLY($f$,$\sigma \circ \sigma_{j+1,k}$,$j+1$);
            END IF
        END FOR
    END IF
END APPLY.
```

Note, that the non–emptiness of a S_{jk} is equal to the existence of an element σ in S_0 with $\sigma\mid_{\{x_1,\ldots,x_{j-1}\}}= id$ and $\sigma(x_j) = x_k$.

Lemma 8 (Correctness of APPLY). *For all j ($1 \leq j \leq |P \cup T|$) and all $\sigma \in \Sigma$, the following holds:*

(1) Calling APPLY(f,σ,j), f is applied to all $\sigma \circ \sigma'$ with $\sigma' \in S_{jj}$;
(2) During a call of APPLY(f,σ,j), f is never called twice for the same symmetry;
(3) APPLY(f,id,0) applies f a single time to every element of Σ.

Proof. We prove the first two claims by induction on descending values of j: If $j = |P\cup T|$, APPLY runs into the **THEN** branch. That is, f is called once, namely for σ itself. Since $S_{|P\cup T|,|P\cup T|} = \{id\}$ (see Part 5 of Corollary 6), the first two claims of the lemma are satisfied in this case.

Let $j \in \{1, \ldots, |P \cup T| - 1\}$. In this case, APPLY runs into the **ELSE** branch. APPLY(f,$\sigma \circ \sigma_{j+1,k}$, $j + 1$) is called for every existing $\sigma_{j+1,k}$. By induction assumption, APPLY(f,$\sigma\circ\sigma_{j+1,k}$, $j+1$) calls f repetition free for every $\sigma\circ\sigma_{j+1,k}\circ\sigma'$ with $\sigma' \in S_{j+1,j+1}$. By Lemma 5, APPLY(f,$\sigma\circ\sigma_{j+1,k}$, $j+1$) calls f repetition free for every $\sigma\circ\sigma''$ with $\sigma'' \in S_{j+1,k}$. By Corollary 6 (Part 3), f is called during the **FOR** statement for all $\sigma \circ \sigma'$ with $\sigma' \in S_{jj}$. This proves the first claim. For proving the second claim consider calls of f during different iterations of a **FOR** statement, say for k_1 and k_2 ($k_1 \neq k_2$). By induction assumption, the first iteration calls f for symmetries of the form $\sigma \circ \sigma_{j+1,k_1} \circ \sigma'$ while the second iteration calls f for symmetries of the form $\sigma\circ\sigma_{j+1,k_2}\circ\sigma''$, where σ' and σ'' are elements of $S_{j+1,j+1}$. Thus, $\sigma'(x_{j+1}) = \sigma''(x_{j+1}) = x_{j+1}$. Consequently, $\sigma_{j+1,k_1}(\sigma'(x_{j+1})) = k_1 \neq k_2 = \sigma_{j+1,k_2}(\sigma''(x_{j+1}))$. This means that $\sigma\circ\sigma_{j+1,k_1}\circ\sigma'$ and $\sigma\circ\sigma_{j+1,k_2}\circ\sigma''$ are always different symmetries. This proves the second claim.

For the third claim, apply the arguments of the induction step once more, but use Part 5 of Corollary 6 instead of Part 3. □

Using APPLY, every iteration of the **FOR ALL** $\sigma \in \Sigma$ loop requires up to $|P\cup T|$ compositions of ground set symmetries. In practise (see next section), the necessary recursion depth is often much lower than $|P \cup T|$. Composing symmetries is of course slower than jumping from pointer to pointer in a completely calculated list of symmetries, but still efficient enough to be a fair price for the reduction of the space for storing the symmetries (polynomial instead of up to exponential). As an additional advantage of a ground set S_0, we shall show that the size of Σ can be calculated from S_0. With this value we can decide whether to use the ground set approach with the APPLY implementation of the **FOR ALL** $\sigma \in \Sigma$ loop (preferably for large $|\Sigma|$), or whether to switch to the full calculation of Σ (for small $|\Sigma|$). The generation of the full Σ can be implemented through a single call of APPLY where f

appends its argument to an initially empty list. The generation of the full Σ by the two steps *Compute a ground set* S_0 and *Generate* Σ *from* S_0 *using APPLY* is much faster than the generation of Σ by the basic calculation procedure (for the net in Fig. 6. with 10 transitions in our test implementation about 10 times faster for the generation of the 10240 symmetries). Therefore the roundabout way to the full Σ via S_0 and APPLY is anyway the suitable one.

The calculation of $|\Sigma|$ from S_0 is considerably simple. The size of Σ is just the product over all j over the number of nonempty S_{jk}.

Lemma 9 (Size of Σ). *Let* S_0 *be a ground set of* Σ. *Then*

$$|\Sigma| = \prod_{j=1}^{|P \cup T|} |\{\sigma_{jk} \mid \sigma_{jk} \in S_0, j \leq k \leq |P \cup T|\}|$$

Proof. Count the number of iterations in the APPLY procedure. □

9 Calculation of ground sets

We modify the basic calculation procedure for the calculation of a ground set. The modification is based on two observations. The first observation shows that the S_{jk} correspond to certain branches of the basic calculation procedure.

Lemma 10 (S_{jk} and ComputeSymmetries). *Consider a specification $\mathcal{C}$ of a symmetry group Σ. Let $\mathcal{C}$ be in normal form. Let $\mathcal{C}' =$ REFINE*($\mathcal{C}$). Let the nodes of the involved net be ordered, and x_j be the least node such that the unique $A \longmapsto B$ with $x_j \in A$ satisfies $|A| > 1$. Then the following holds for $\Sigma = \Sigma_{\mathcal{C}}$:*

(1) $S_{j-1,j-1} = \Sigma$ (define $S_{00} := \Sigma$);
(2) For all k,
  (a) If $x_k \in B$, then $S_{jk} =$ ComputeSymmetries(DEFINE($\mathcal{C}'$, $A \longmapsto B$, x_j, x_k));
  (b) If $x_k \notin B$, then $S_{jk} = \emptyset$.

Proof. Ad 1. By Corollary 4, $\Sigma = \Sigma_{\mathcal{C}} = \Sigma_{\mathcal{C}'}$. Since $\mathcal{C}'$ specifies a group, all constraints in $\mathcal{C}'$ have equal left and right sides. Furthermore ($\mathcal{C}'$ is in normal form!) for all $x_{j'}$ with $j' < j$, there is a unique constraint containing $x_{j'}$ which consists by assumption of a singleton left side. Thus, the constraint containing $x_{j'}$ is $\{x_{j'}\} \longmapsto \{x_{j'}\}$. This means that every symmetry in $\Sigma = \Sigma_{\mathcal{C}'}$ is the identity on its first $j - 1$ nodes. This implies $\Sigma = S_{j-1,j-1}$.

Ad 2. The DEFINE operation includes the constraint $\{x_j\} \longmapsto \{x_k\}$. With Part 1 of this Lemma, this means that ComputeSymmetries(DEFINE($\mathcal{C}'$,$A \longmapsto B$, x_j, x_k)) $\subseteq S_{jk}$. The equality and the emptiness of the remaining S_{jk} can be easily deduced from Theorem 4, (1), and Corollary 6 (Parts 1 and 3). □

Obviously, the conditions of Lemma 10 describe the situation for the initial call of ComputeSymmetries in the basic calculation procedure. Therefore, the subsequent calls of ComputeSymmetries calculate all nonempty sets among $S_{jj}, \ldots, S_{j|P \cup T|}$. Lemma 10 in connection with Corollary 6 assures further that all $S_{j'k'}$ for $j' < j$ and $j' \neq k'$ are empty. Thus, if we want to calculate a single element of a S_{jk} $(j \neq k)$ there is nothing left to do than to return immediately from deeper recursion levels as soon as one symmetry has been found there.

It remains to calculate the $S_{j'k}$ for $j' > j$. For this purpose, observe that S_{jj} is a subgroup of Σ. It is just the one specified by DEFINE($\mathcal{C}'$,$A \longmapsto B$; x_j,x_j) (see Lemma 10). Furthermore the following Lemma holds.

Lemma 11 (Relation between Σ and S_{jj}).
For all $j, j' \in \{1, \ldots, |P \cup T|\}$ it holds: If $j < j'$, then for all $k \in \{1, \ldots, |P \cup T|\}$, $S_{j'k} = (S_{jj})_{j'k}$.

Proof. Follows directly from the definition of the S_{jk}. □

Observe that whole j–levels of S_{jk} can be empty. In APPLY, the corresponding recursion levels can be skipped totally. Thus, the actual recursion depth of APPLY is usually much smaller than $|P \cup T|$.

With Lemma 11, we see that we can calculate a complete ground set by a recursive call of a modified ComputeSymmetries where we collect all symmetries on the path leading to the identity mapping while we narrow the calculation to one symmetry on the remaining branches. Before presenting pseudo–code for the modified procedure, we discuss another observation which allows us to speed up the calculation.

Lemma 12 (Composition of Symmetries). *If σ is contained in S_{jk} and $\sigma^n(x_j) = x_{k'}$, then σ^n is contained in $S_{jk'}$.*

Proof. This is obvious. □

With Lemma 12, we can remove whole branches in our computation. As soon as an element of S_{jk} has been calculated, we can compose this symmetry repeatedly with itself. If we obtain an element of $S_{jk'}$ this way, we can skip the branch of the calculation where the image of x_j is set to $x_{k'}$. Of course, composition is much faster than the recursive call of ComputeSymmetries (remember the danger to run into dead branches).

Using the two above observations, we are ready to present the procedure for the calculation of a ground set of symmetries. We assume that $\mathcal{C}$ is in normal form. The boolean parameter $ToId$ is TRUE iff we are on the descent to the identity. That means, if $ToId$ is FALSE, we may return after a symmetry has been found. Initially, we have to call the procedure with $ToId =$ TRUE.

```
PROCEDURE ComputeGroundSet($\mathcal{C}$ : spec,$ToId$: BOOL)
    :SetOfSymmetries
VAR   $\Sigma$, $\Sigma'$: SetOfSymmetries;
        $\sigma, \sigma'$; Symmetry;
        $\mathcal{C}'$: spec;
        $U$: SetOfNodes;

BEGIN
    $\mathcal{C} :=$ REFINE*($\mathcal{C}$);
    IF exists $A \longmapsto B$ in $\mathcal{C}$ such that $|A| \neq |B|$ THEN
        RETURN $\emptyset$;
    END IF;
    IF for all $A \longmapsto B$ in $\mathcal{C}$ it holds $|A| = |B| = 1$ THEN
        RETURN $\Pi_{\mathcal{C}}$;
    END IF;
    $A \longmapsto B :=$ select among the non–singleton constraints
    of $\mathcal{C}$ the one which contains the least node $x$;
    IF ToId THEN
        (* First, we descend further to calculate $\sigma_{jk}$ for larger $j$ *)
        $\mathcal{C}' :=$ DEFINE($\mathcal{C}, A \longmapsto B, x, x$);
        $\Sigma :=$ ComputeGroundSet($\mathcal{C}', TRUE$);
        (* For the remaining branches, we may calculate
        or compose the elements *)
        $U := B \setminus \{x\}$;
        WHILE $U \neq \emptyset$ DO
            (* Calculate a symmetry *)
            Select $y \in U$;
            $U := U \setminus \{y\}$;
            $\mathcal{C}' :=$ DEFINE($\mathcal{C}, A \longmapsto B, x, y$);
            $\Sigma' :=$ ComputeGroundSet($\mathcal{C}'$,$FALSE$);
            IF $\Sigma' \neq \emptyset$ THEN (* $\Sigma'$ contains at most 1 element *)
                $\Sigma := \Sigma \cup \Sigma'$;
                $\sigma :=$ the unique element in $\Sigma'$;
                (* Compose symmetries *)
                $\sigma' := \sigma \circ \sigma$;
                WHILE $\sigma' \neq id$ DO
                    IF $\sigma'(x) \in U$ THEN
                        $\Sigma := \Sigma \cup \{\sigma'\}$;
                        $U := U \setminus \sigma'(x)$;
                    END IF
                    $\sigma' := \sigma' \circ \sigma$;
                END WHILE
            END IF
        END WHILE
        RETURN $\Sigma$;
    ELSE
        $(*$ Here we are in a branch not leading to the identity, thus
        we have to return as soon as the first symmetry is found $*)$
        FOR all $y$ in $B$ DO
            $\mathcal{C}' :=$ DEFINE$(\mathcal{C}, A \longmapsto B, x, y)$;
            $\Sigma :=$ ComputeGroundSet$(\mathcal{C}', FALSE)$;
            IF $\Sigma \neq \emptyset$ THEN
                RETURN $\Sigma$;
            END IF
        END FOR;
        RETURN $\emptyset$;
    END IF;
END ComputeGroundSet.
```

Theorem 8 (Correctness of ComputeGroundSet).
Let $\mathcal{C}$ be the specification of a symmetry group. The procedure ComputeGroundSet($\mathcal{C}$,$TRUE$) calculates a ground set for $\Sigma_{\mathcal{C}}$.

Proof. Let $\Sigma = \Sigma_{\mathcal{C}}$. Comparing for any $\mathcal{C}'$ the call ComputeSymmetries($\mathcal{C}'$) with the call ComputeGroundSet($\mathcal{C}', FALSE$), it is appearant by Theorem 5 that ComputeGroundSet($\mathcal{C}', FALSE$) calculates one symmetry of $\Sigma_{\mathcal{C}'}$ if this set is non–empty, and the empty set otherwise.

For ComputeGroundSet($\mathcal{C}', TRUE$), correctness follows by induction on the recursion depth of those calls. First, the deepest recursion of ComputeGroundSet($\mathcal{C}', TRUE$) cannot calculate anything but the identity that is the unique element of $S_{|P \cup T|,|P \cup T|}$.

Consider now an arbitrary nonterminal recursion level of the procedure ComputeGroundSet($\mathcal{C}', TRUE$). The unique subsequent call of ComputeGroundSet($\mathcal{C}', TRUE$) calculates by induction assumption a ground set for S_{jj} (where j is the position of x in the assumed linear ordering of the nodes). By Lemma 11, this set contains a unique element of every non–empty $S_{j'k}$ ($j' > j$). By Lemma 10 and Lemma 12, the "**WHILE** $U \neq \emptyset$" loop calculates a unique element of every non–empty S_{jk} ($j \neq k$).

By Lemma 10, all $S_{j'k}$ ($j' < j, j' \neq k$) are empty. Thus, the calculated set is a ground set of $\Sigma = \Sigma_{\mathcal{C}}$.

□

Remark 2. The same modification can be done for the calculation of ground sets of symmetry groups for self–modifying nets.

We can see that the calculation of ground sets consists of a narrowed calculation of the whole symmetry group. Branches neither leading to the identity nor to the first symmetry of a certain set are cut as well as branches leading to symmetries which can be composed from existing ones. Thus, the running time for the calculation of a ground set of size k is roughly the same as the running time for the calculation of a full symmetry group of size k. Taking into consideration that symmetry composition is much faster than symmetry generation by DEFINE/REFINE, we strongly recommend not to use the basic calculation procedure, even if the calculation of the full group is intended. Instead, calculate a ground set and generate the group using the APPLY procedure described earlier.

Many considerations so far concerned the state equivalence problem. For the node equivalence problem, the calculation of any generating set of symmetries is sufficient. By Definition 4, the equivalence relation $\sim_\Sigma$ on nodes equals $\{[x, \sigma(x)] \mid \sigma \in \Sigma, x \in P \cup T\}$. If the equation $\sigma = \sigma_1 \circ \sigma_2$ is satisfied, σ requires exactly the pairs $\{[x, \sigma_1(\sigma_2(x))] \mid x \in P \cup T\}$ to be contained in $\sim_\Sigma$. These equivalences follow by transitivity from $[x, \sigma_2(x)]$ and $[\sigma_2(x), \sigma_1(\sigma_2(x))]$. Thus, $\sim_\Sigma$ can be calculated as the (reflexive, symmetric and) transitive closure of $\{[x, \sigma(x)] \mid \sigma \in S_0, x \in P \cup T\}$ where S_0 is an arbitrary generator set of Σ, for instance a ground set. For the generation of the reflexive, symmetric and transitive closure of a binary relation, there exists the famous Union/Find algorithm [Tar79, Tar83] working in almost linear time.

10 Symmetry–respecting hash functions

In this section we take up (for the last time) the ”symmetry on demand” concept. The major advantage of this concept is that no — in worst case exponential — pre–processing of the symmetries is necessary, and no space is required for storing that many symmetries. We were able to demonstrate that all relevant problems occurring in the context of symmetries can be solved ”on demand” (if necessary, by small modifications of the basic calculation procedure, see Sect. 6). If we focus on algorithms similar to reachability graph generation (i.e. coverability graph, stubborn reduced graph, etc.), the symmetry on demand concept has a major drawback, concerning the order in which two loops are nested. As already reported, symmetries are involved when we want to check for a new node m^*, whether a symmetric node is already contained in the graph. Using symmetries on demand, this step is implemented (as already presented earlier) as

```
**FOR ALL** $m \in M$ **DO**
    **IF** there is a $\sigma \in \Sigma$ such that $\sigma(m) = m^*$ **THEN**
        **RETURN** yes;
    **END IF**
**END FOR**
**RETURN** no;
```

The condition of the **IF** statement is exactly a state equivalence on demand problem. It is however rarely acceptable to run in worst case $|M|$ iterations of the outer loop, since reachability graphs contain usually a huge number of nodes. Thus, in order to defend the ”on demand” concept, we have to find a solution that allows us to run significantly less iterations of the outer loop than $|M|$. For this purpose, we propose symmetry respecting hash functions.

A hash function[1] for a state space $\mathcal{I}^{P \cup T}$ is a mapping $h : \mathcal{I}^{P \cup T} \longrightarrow I$, where I is a finite set of indices. The idea is to have an array A of sets — indexed by I — and to store an element m always in $A[h(m)]$. For the decision whether an element m is already contained in M, it is sufficient to search in $A[i]$. The implementation of the sets in $A[i]$ can be any suitable structure, e.g. a list, a tree, or even again a hash structure. When the hash function is well chosen (which means that all hash entries contain roughly the same number of elements), the access time to some element of a set with n elements decreases from $f(n)$ (if $f(n)$ is the access time for a monolithic set implementation) to $f(\frac{n}{|I|})$.

In the sequel we demonstrate that hash functions h can be generated in such a way that, given a state m^*, all states that are symmetric to m^* are guaranteed to be contained in $A[h(m^*)]$ (if they are contained in M at all). This way it is possible to replace M in the outer loop of the above procedure by $A[h(m^*)]$. In average, this improves significantly the running time of the graph generation procedure. A hash function that is suitable for our purposes must satisfy the following property.

Definition 13 (Symmetry respecting hash function). *A hash function h is* symmetry respecting *with respect to a symmetry group Σ iff for all states m and all $\sigma \in \Sigma$, $h(m) = h(\sigma(m))$.*

Example 9. Assume, $\mathcal{I} = \mathbb{Z}$. Then $h(m) = \sum_{x \in P \cup T} m(x)$ mod k is symmetry respecting for any array size k and any symmetry group Σ. (Symmetries permute values of m, but do not change them).

[1] For the sake of simplicity we do not present hash concepts in their full generality.

The hash function in Example 9 does not only respect the symmetries, but any bijection. Thus, the hash classes $A[i]$ become much larger than actually necessary. For better functions, it would be wise to apply knowledge about the involved symmetry group Σ. Since Σ itself is not available due to the "on demand" approach, the only available information about Σ is its specification, say $\mathcal{C}$. We may assume that $\mathcal{C}$ is totally refined (i.e., $\mathcal{C} = REFINE*(\mathcal{C})$). This pre–processing does not hurt, since it is polynomial in time and space. Furthermore, $\mathcal{C}$ is involved in every equivalence on demand specification. Therefore the pre–processing is anyway welcome. We assume further that $\mathcal{C}$ is in normal form. Since $\mathcal{C}$ specifies a group, the left and right sides of constraints in $\mathcal{C}$ are equal (see Theorem 1) . Hence, $\mathcal{C}$ has the form $\{A_i \longmapsto A_i \mid 1 \leq i \leq |\mathcal{C}|\}$. This means, that for every symmetry σ and all i, images of nodes in A_i are again contained in A_i.

With this knowledge, it is easy to construct symmetry respecting hash functions. All we have to do is to assure, that the value of the function does not change when arguments belonging to the same A_i are exchanged.

Example 10. For instance, we can introduce weights for every node and build the weighted sum of all state components. In order to be symmetry respecting, the weights must be equal for nodes belonging to the same constraint.

Assume again, $\mathcal{I} = \mathbb{Z}$. Let $\mathcal{C} = \{A_i \longmapsto A_i \mid 1 \leq i \leq |\mathcal{C}|\}$. Let $k, k_1, \ldots, k_{|\mathcal{C}|}$ be integer numbers. Then the mapping h defined by

$$h(m) = \left(\sum_{i=1}^{|\mathcal{C}|} k_i \cdot \sum_{x \in A_i} m(x) \right) \bmod k$$

is symmetry respecting, since for all symmetries $\sigma \in \Sigma_{\mathcal{C}}$,

$$\begin{aligned}
& h(\sigma(m)) \\
&= \left(\textstyle\sum_{i=1}^{|\mathcal{C}|} k_i \cdot \sum_{x\in A_i} \sigma(m)(x)\right) \bmod k && \text{(Def. of } h\text{)} \\
&= \left(\textstyle\sum_{i=1}^{|\mathcal{C}|} k_i \cdot \sum_{x\in A_i} m(\sigma^{-1}(x))\right) \bmod k && \text{(Def. of } \sigma(m)\text{)} \\
&= \left(\textstyle\sum_{i=1}^{|\mathcal{C}|} k_i \cdot \sum_{y\in \sigma^{-1}(A_i)} m(y)\right) \bmod k && (\sigma \text{ Bijection}) \\
&= \left(\textstyle\sum_{i=1}^{|\mathcal{C}|} k_i \cdot \sum_{y\in A_i} m(y)\right) \bmod k && (\sigma \text{ cons. with } A_i \longmapsto A_i) \\
&= h(m)
\end{aligned}$$

The hash function of Example 10 has another advantage. Let h' be equal to h, but with positive k_i and without the "mod k". It respects even the partial order between states. Formally, if there is a symmetry $\sigma \in \Sigma$ such that $\sigma(m_1) \leq m_2$, then $h'(m_1) \leq h'(m_2)$. For the proof of this claim, observe that $h'(m_1) = h'(\sigma(m_1))$ (analogous to Example 10). Furthermore

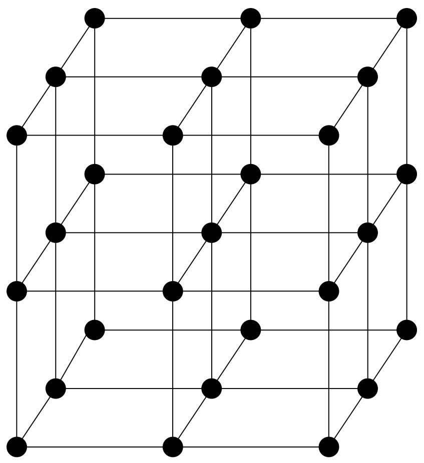

Fig. 10. 3–dimensional grid network with 3 agents per dimension

h' is monotonous, i.e. $m_1 \leq m_2$ implies $h'(m_1) \leq h'(m_2)$. Though h' itself is unbounded, we can easily transform it into a hash function, by partitioning its values into a finite number of intervals where the last one is infinitely large. Thus, with some additional efforts to cope with ω, we are even able to speed up the generation of symmetrically reduced coverability graphs.

11 Experiments

In this section, we report some results achieved with our test implementation. The times are taken on a LINUX system running on a 400 MHz Pentium II PC with 64 MB RAM.

DIJKSTRA's n *dining philosophers* problem (PHIL n) is a small mutual exclusion protocol for n agents arranged as a ring. The net model is taken from [Val91]. The symmetries correspond to the n rotations of the ring.

The echo algorithm [Cha82] implements a protocol for propagation of information with feedback in a network. The Petri net model has been taken from [KRVW97]. ECHO d/l refers to grid–like networks with d dimensions and l agents per dimension (i.e. the whole network consists of l^d agents). The agent that initiates the process is in the centre of the structure.

The same underlying network structure had been used for a consensus protocol in [Rei98] (CONS d/l) and a simple protocol where every agent switches between two states in mutual exclusion from its neighbours (SIMP d/l). The number of symmetries for distributed algorithms in a grid like network depends on the number of dimensions. For $d = 2$ we have 8 symmetries, for $d = 3$ there are 48 symmetries, 384 symmetries for $d = 4$ and 3840 for $d = 5$.

Table 1. Dimensions of the examples

Net	$\|P\|$	$\|T\|$	$\|\Sigma\|$	$\|S\|$
PHIL 13	65	52	13	12
PHIL 1000	5000	4000	1000	999
ECHO 3/3	1593	1512	48	10
ECHO 2/5	1375	1300	8	4
CONS 3/3	1512	2214	48	10
SIMP 4/4	768	512	384	21
SIMP 5/2	96	64	3840	41
DATA 40/40	241	200	$40! \cdot 40!$	1560
DATA 50/70	361	310	$50! \cdot 70!$	3640

Table 2. Times for the generation of symmetries

Net	Time for ground set	Time for ground set + APPLY	Time for basic proc.
PHIL 13	0:00,01	0:00,01	0:00,01
PHIL 1000	3:02	3:06	3:02
ECHO 3/3	1:30	1:36	7:40
ECHO 2/5	0:40	0:41	1:20
CONS 3/3	1:29	1:32	5:31
SIMP 4/4	0:20	0:21	6:05
SIMP 5/2	0:00,8	0:00,86	1:14
DATA 40/40	0:11	–	–
DATA 50/70	0:45	–	–

In the DATA r/w nets, a semaphore controls access to some database. r reading processes are allowed to read concurrently while w writing processes get exclusive access. The nets have $r! \cdot w!$ symmetries.

The dimensions for the studied examples are listed in Table 1.. Σ is the set of all symmetries, S is the ground set.

Table 2. contains times for the calculation of ground sets, the calculation of ground sets with subsequent generation of the full group, and extrapolated times for the generation of all symmetries (based on the times for the ground sets). All times include reading the net and are given as $< minutes >$:$<$ $seconds >$.

Next, we consider the generation of reduced reachability graphs. We use the PHIL example for the preprocessing concept (APPLY) and the DATA example for the symmetry on demand concept.

The full graph for PHIL 10 has 59048 nodes and 393650 edges. It can be generated in 0:05. The reduced graph has 5933 nodes, 39550 edges, and is generated in 0:02 (all times include reading the net, generating the symmetries — if necessary — and generating the graph). The reduced graph of PHIL 13 has 122642 nodes, 1062893 edges, and is generated in 1:38. In this example, the physical memory is exceeded. Swapping the virtual memory consumes much of the time. The full graph is about 13 times as large and does not fit into the available memory.

The full graph of DATA 13/13 has 114857 nodes and 905554 edges. The generation takes 2:54. The reduced graph has 30 nodes, 470 edges, and it takes 0:00,5 to calculate it. For DATA 40/40, the full graph has $41 \cdot (2^{40} + 40)$ states and cannot be calculated. The reduced graph has 84 nodes, 4142 edges and was generated in 9:44.

12 Conclusion

We have demonstrated that the concept of symmetry specifications is sufficient to specify a wide range of problems usually appearing in the context of symmetry application. We are able to specify symmetry groups, important constraints to these groups, and even the most relevant decision problems appearing in the context of symmetries. The mechanism to calculate the specified symmetries is — with only few exceptions — independent of the involved net class.

Our methods allow three different ways for the integration of symmetries into an analysis algorithm. They differ in the amount of necessary pre–processing, the number of symmetries to be stored permanently, and the time to handle the symmetries during the run of the host algorithm. In particular, we have studied the use of symmetries on demand, the ground set approach with the APPLY implementation of the ”for all symmetries” loop, and the calculation of the full symmetry group. The table summarises these differences.

	On demand	Ground set + APPLY	Full set
Max. number of permanently stored symmetries	0	polynomial	exponential
Preprocessing	no	Compute-GroundSet	Compute-GroundSet + APPLY
Integration into host algorithm	slow	fast	fastest

The table shows a trade–off between the slowing down of the host algorithm and the required memory for the symmetries to be stored permanently. Another trade–off concerns the slowing down of the host algorithm and the time required for pre–processing. Therefore there is a reason for the existence of all three approaches. A decision for one of the methods must be left to the users expectation about the size of the symmetry group and the size of the host problem. The smaller the symmetry group is, the more an approach with pre–processing is recommendable. The more we are afraid that the available memory could be exhausted, the more the ”on demand” approach qualifies for use (in the area of reachability analysis, space is often more restricted than running time).

Once having calculated a ground set, the decision between the second and third approach of the table can be based on the real size of the symmetry group that can be calculated according to Lemma 9.

The proposed algorithms have an exponential worst case running time, due to the existence of dead branches in the calculation tree. However, in average, the algorithm behaves much more friendly, since dead branches seem to appear rather seldom (though there is no profound average case analysis).

Up to now, not all of the presented algorithms are implemented. In the tools INA [RS97] and PROD [VHHP95], an algorithm for the calculation of the full symmetry group is implemented where ComputeGroundSet is augmented by an immediate composition of the calculated symmetries to a full group. The symmetry on demand approach exists in a test version that has not been released yet.

Acknowledgements. The author would like to thank his colleages Prof. Peter Starke, Stefan Haar, and Stephan Roch for discussions on the topic. The author is also grateful to two anonymous referees. Their suggestions improved the quality of the paper significantly.

References

[CDFH90] G. Chiola, C. Dutheillet, G. Franceschinis, 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, pp 387–410, 1990

[CEFJ96] E.M. Clarke, R. Enders, T. Filkorn, S. Jha: Exploiting symmetry in temporal logic model checking. Formal Methods in System Design, **9**, 77–104 (1996)

[CF90] G. Chiola, G. Franceschinis: Colored gspn models and automatic symmetry detection. In Proceedings of the 3rd Int. Workshop on Petri Nets and Performance Models 1989, Kyoto, Japan — Los Alamitos, CA, USA, pp 50–60. IEEE Computer Society Press, 1990

[CFG94] G. Chiola, G. Franceschinis, R. Gaeta: Modeling symmetric computer architectures by swns. Proceedings of the 15th Int. Conf. on Application and Theory of Petri nets, Zaragoza, Spain. LNCS **815**, 139–158 (1994)

[Cha82] E.J.H. Chang: Echo algorithms: Depth parallel operations on general graphs. IEEE Trans Software Eng, SE **8**(4), 391–401 (1982)

[DH91] C. Dutheillet, S. Haddad: Regular stochastic petri nets. Advances in Petri Nets 1990. LNCS **483**, 186–209 (1991)

[ES96] E.A. Emerson, A. P. Sistla: Symmetry and model checking. Formal Methods in System Design **9**, 105–131 (1996)

[GC95] R. Gaeta, G. Chiola: Efficient simulation of swn models. In Proc. of 6th Int. Workshop on Petri Nets and Performance Models, Durham, pp 137–146. IEEE Computer Soc. Press, 1995

[HIMZ95] S. Haddad, J. M. Ilié, M.Taghelit, B. Zouari: Symbolic reachability graph and partial symmetries. 16th International Conference on Application and Theory of Petri nets, pp 238–257, 1995

[HJJJ84] P. Huber, A. Jensen, L.O. Jepsen, K. Jensen: Towards reachability trees for high–level petri nets. In Advances in Petri Nets 1984, Lecture Notes on Computer Science 188, pp 215–233, 1984

[Jen92] K. Jensen: Coloured Petri Nets, volume 1 of EATCS Monographs on Theoretical Computer Science. Berlin, Heidelberg, New York, Springer, 1992

[Jen96] K. Jensen: Condensed state spaces for symmetrical coloured petri nets. Formal Methods in System Design **9**, 7–40 (1996)

[KRVW97] E. Kindler, W. Reisig, H. Völzer, R. Walter: Petri net based verification of distributed algorithms: an example. Formal Aspects of Computing 9, pp 409–424, 1997

[Pet90] L. Petrucci: Combining finkels and jensens reduction technique to build covering trees for coloured nets. Petri Net Newsletter **36**, 32–36 (1990)

[Rei85] W. Reisig: Petri Nets. EATCS Monographs on Theoretical Computer Science, 1985

[Rei98] W. Reisig: Elements of Distributed Algorithms. Berlin, Heidelberg, New York, Springer, 1998

[RS97] S. Roch, P. Starke: INA – Integrierter Netz–Analysator Version 1.7. Handbuch. Humboldt–University Berlin, Institute of Computer Science, 1997

[Sch93] K. Schmidt: Symmetries of petri nets. Petri Net Newsletter **43**, 9–25 (1993)

[SS91] K. Schmidt, P.H. Starke: An algorithm to compute the symmetries of petri nets. Petri Net Newsletter **40**, 25–30 (1991)

[Sta90] P. Starke: Analyse von Petri–Netz–Modellen. B.G. Teubner Stuttgart, 1990

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

[Tar79] R.E. Tarjan: A class of algorithms which require nonlinear time to maintain disjoint sets. J Comput Syst Sci **18**(2), 110–127 (1979)

[Tar83] R.E. Tarjan: Data Structures and Network Algorithms. Society for Industrial and Applied Mathematics, 1983

[Tiu94] M. Tiusanen: Symbolic, symmetry and stubborn set searches. In: R. Valette (ed) Proceedings of the 15th International Conference on Application and Theory of Petri Nets in Zaragoza, pp 511–530. Berlin, Heidelberg, New York, Springer, 1994

[Val78] R. Valk: Self–modifying nets, a natural extension of petri nets. Automata, Languages and Programming, LNCS, **62**, 464–476 (1978)

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

[VHHP95] K. Varpaaniemi, J. Halme, K. Hiekkanen, T. Pyssysalo: Prod reference manual. Technical Report B13, Digital Systems Laboratory, Helsinki University of Technology, August 1995

[WWV^{+}97] M. Weber, R. Walter, T. Vesper, W. Reisig, E. Kindler, J. Freiheit, J. Desel: DAWN – Petrinetzmodelle zur Verifikation Verteilter Algorithmen. Technical Report 88, Humboldt–University Berlin, 1997