Fundamenta Informaticae 37 (1999) 269–289
IOS Press

Finding Symmetries of Algebraic System Nets *

Tommi A. Junttila†
Laboratory for Theoretical Computer Science
Department of Computer Science and Engineering
Helsinki University of Technology
P.O.Box 5400, FIN-02015 HUT, Finland
Tommi.Junttila@hut.fi

Abstract. The problem of finding symmetry information from algebraic system nets prior to the reachability graph generation is studied. The approach presented is based on well-formedness of transition descriptions, meaning that some data types in a net may be used in a symmetric way. Permutations on the domains of such data types produce symmetries on the state space level of the net, which in turn can be exploited during the reachability analysis. To ensure that the transitions behave symmetrically with respect to the chosen data domain permutations, a sufficient compatibility condition between data domain permutations and the algebraic terms used as transition guards and arc annotations is proposed. The solution is a general and flexible one as it does not fix the set of applicable operations, enabling the design of customized net classes. To help the process of deciding whether a term is compatible with a data domain permutation, an approximation rule for the compatibility condition is given.

Keywords: Symmetries, algebraic system nets, Petri nets, reachability analysis

1. Introduction

The symmetry reduction method is one of the techniques conceived to alleviate the state space explosion problem occurring in the reachability analysis of concurrent systems. Its history dates

*This work was supported by the Academy of Finland (project number 8309) and Helsinki Graduate School in Computer Science and Engineering (HeCSE).
†Address for correspondence: Laboratory for Theoretical Computer Science, Department of Computer Science and Engineering, Helsinki University of Technology, P.O.Box 5400, FIN-02015 HUT, Finland

back (at least) to the paper by Huber et al. [14]. The key idea of the method is that some states of a system can be considered as equivalent in the sense that they give rise to symmetric behaviour. Given an equivalence class of such states, it is necessary to inspect only one representative of the class during the reachability analysis. Thus a reduced reachability graph can be generated instead of the bigger, original reachability graph. Several properties of the system can then be checked by using the reduced graph, ranging from deadlock checking to full model checking of temporal logic formulae.

Because we want to avoid the generation of the full reachability graph (or the state space), the symmetry information must be found on the system description level prior to the reachability analysis. In this paper, we study the problem of how the symmetry information can be extracted from algebraic system nets. We are interested in algebraic system nets because (i) they permit the use of many kinds of data types (or sorts as they are called) in the nets, (ii) the use of algebraic techniques allows us to describe both the syntax and the semantics of the terms appearing in the nets, and (iii) by specifying different kinds of algebras we are able to define classes of nets for different customized purposes, such as for the modelling of telecommunication systems. Naturally, other similar formalisms, e.g. many-sorted high-level nets [2, 16], could be used as well.

The approach taken in this paper is based on "well-formedness" of system descriptions (cf. [3, 17]). Well-formedness means that the transitions of a system are formed in a way that they use data types symmetrically. This ensures that the system behaves in a symmetric way for different data values. Thus the symmetry information we are looking for is: How can the domains of data types used in an algebraic system net be permuted to produce symmetries on the state space level. We proceed by first defining how the domains can be permuted and how this produces permutations on the state space of the net. To make sure that produced permutations are actually symmetries (i.e. automorphisms of the state space) we must rule out the use of transitions that may behave asymmetrically. We do this by requiring that the terms used as transition guards and arc annotations in the net are compatible with the used domain permutations. To help the design and implementation of reachability analyzers exploiting symmetries, we also give an approximation rule with which it is easy to decide whether a term is compatible.

Our approach resembles (and is inspired by) those of Chiola et al. [3] and Ip and Dill [17]. However, our approach has an advantage of being more general. Instead of fixing all the available operations, we only impose sufficient conditions on them. This enables us to define different kinds of net classes, tailored for different uses.

The paper is organized as follows. The symmetry reduction method in general is introduced in the next section. The formalism of the paper, algebraic system nets, is presented in Sec. 3. Section 4 describes the main topics of the paper: How the state space permutations of algebraic system nets are produced by data domain permutations? What are the restrictions that have to be imposed on the terms used as transition guards and arc annotations to make them compatible with the permutations? An approximation rule for deciding the compatibility of terms is also given and its use is illustrated. Section 5 ends the paper with conclusions and discussion.

2. The Symmetry Reduction Method

The symmetry reduction method exploits the property present some in state spaces, namely that some states may be considered as equivalent (or symmetric) in the sense that they give rise to symmetric behaviour. Of course, not all systems exhibit symmetries but in many cases there is some source of symmetries. Typical sources include: (i) symmetries in the structure of a system, (ii) symmetric use of data (e.g. the famous bit in the alternating bit protocol), (iii) several instances of the same process type, and (iv) replicated components (e.g. n identical client processes).

Assume that we have a system description given, for example, as a Place/Transition net or as a high-level Petri net. The semantics for the description formalism define its *state space*, which in the case of Petri nets is the set of all markings and the transition occurrences between them. We express states spaces by means of *labeled transition systems* (LTSs). An LTS G is a tuple $G = \langle Q, L, \Delta, q_{init}\rangle$, where Q is a non-empty set of *states* with a distinguished *initial state* $q_{init} \in Q$, L is a non-empty set of *transition labels* such that $Q \cap L = \emptyset$ and $\Delta \subseteq Q \times L \times Q$ is a *transition relation*.

The equivalence classes of states that the method utilizes are produced by automorphism groups operating on the state space. Formally, a *permutation* of the LTS G is a bijective function $\phi : (Q \cup L) \to (Q \cup L)$ such that $\phi(Q) = Q$ and $\phi(L) = L$. The *set of all permutations of* G is denoted by $\mathrm{Perm}(G)$ and is a group under the composition function $\circ$. A sub-group of $\mathrm{Perm}(G)$ is called a *permutation group* of G. Symmetries are permutations that respect the transition relation: A permutation ϕ of G is a *symmetry* of G iff $\langle q, l, q_1\rangle \in \Delta \Leftrightarrow \langle \phi(q), \phi(l), \phi(q_1)\rangle \in \Delta$ holds. In other words, a symmetry of G is an automorphism of G. A permutation group Φ of G is a *symmetry group* iff all the permutations $\phi \in \Phi$ are symmetries of G. The *set of all symmetries of* G is denoted by $\mathrm{Sym}(G)$ and is a sub-group of $\mathrm{Perm}(G)$ (note that there always exists one symmetry, namely the identity mapping). Given two elements $x, x' \in Q \cup L$, we say that x is *symmetric to* x', denoted by $x \equiv_\Phi x'$, if and only if $(\exists \phi \in \Phi)(\phi(x) = x')$. Because of the group properties of Φ, $\equiv_\Phi$ is an equivalence relation on $Q \cup L$. For an element $x \in Q \cup L$, the Φ-*orbit of* x is $[x]_\Phi = \{x' \in Q \cup L \mid x \equiv_\Phi x'\}$.

Given a symmetry sub-group $\Phi \subseteq \mathrm{Sym}(G)$, it is sufficient to inspect only one representative of each equivalence class (Φ-orbit) of states during the reachability analysis. Thus a reduced reachability graph can be generated and substantial savings achieved both in the time and memory requirements (see e.g. [5, 17, 18, 25]). The properties that one is able to check from reduced reachability graphs range from simple deadlock absence, safety and Petri net specific properties [3, 4, 15, 17, 18] to full-fledged model checking of temporal logic formulae [1, 6, 7, 8, 12].

As the generation of the full reachability graph is to be avoided, we must find the *information generating the symmetries* on the system description level. This information typically is a permutation group Ψ (under an arbitrary binary operator $*$) operating on some elements of the system description. For a Place/Transition net, Ψ could be a permutation group permuting the places and transitions [27, 28, 29] whereas in this paper, Ψ will be a group permuting the domains of the data types used in an algebraic system net. Ψ will then produce symmetries to the state space LTS G, or formally, there is a group homomorphism $f : \Psi \to \mathrm{Sym}(G)$.[1]

2.1. An Example

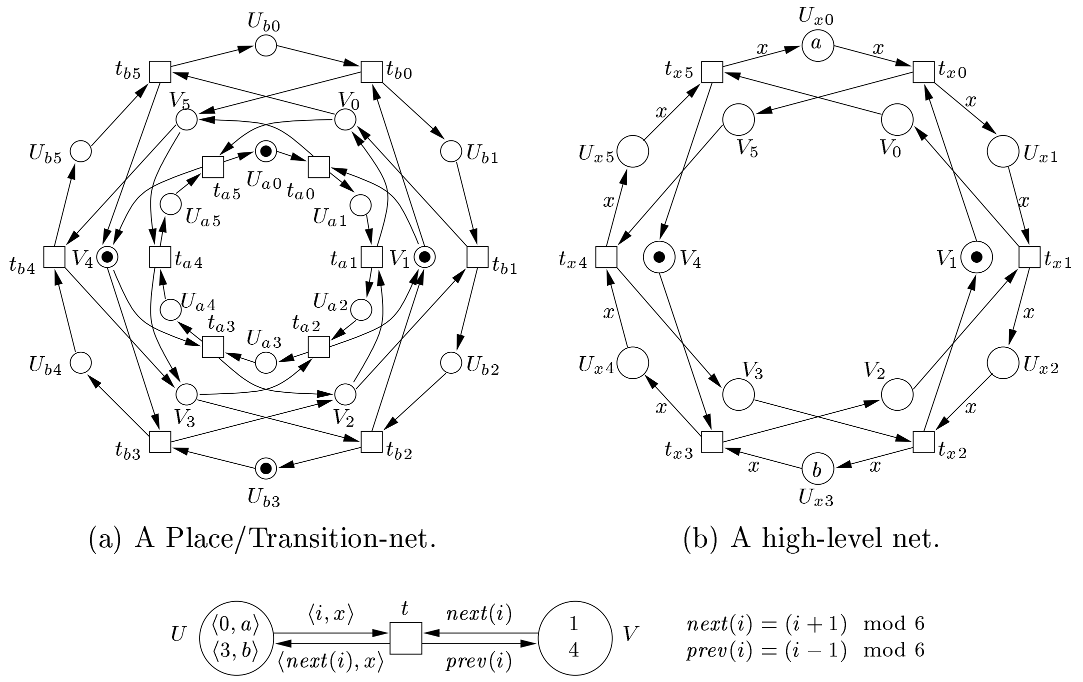
To make things more concrete, consider the nets in Fig. 1. They all model the same railway system and are 6-placed variants of the nets in [9].

Let us first examine the symmetries present on the system description level, i.e. in the Petri nets. The structure of the Place/Transition net in Fig. 1(a) is symmetric in two respects: (i) The trains a and b (the U-places) access the track sections (the V places) in the same way, independent of which section is concerned. Thus the net exhibits circular symmetry obtained by rotating the net. (ii) The trains behave symmetrically and therefore the inner and outer rings can be swapped (i.e. the identities of trains can be swapped). For instance, the situation in which the train a is in the section 1 and b in 5 is symmetric to the opposite situation where the train a occupies the section 5 and the train b section 1. The high-level net in Fig. 1(b) shows only the circular symmetry in the structural level. The symmetry obtained by swapping the trains is not anymore visible in the structure but is produced by the symmetric use of data in the transitions. Unfolding the net into the P/T-net would again reveal the symmetry on the structural level. The high-level net in Fig. 1(c) has no structural symmetries but both symmetries are encoded in the way the transition uses data types. A similar kind of distinction between data and structural symmetries was made by Pandey and Bryant in the context of transistor-level circuit verification with symmetries [25].

All the nets in Fig. 1 produce reachability graphs which are isomorphic to that of Fig. 1(c), shown in Fig. 2 (the transition instance names are not shown on the arcs). The format for the markings (the nodes of the graph) is that the tokens in the place U are shown first and then the tokens in the place V. The initial marking is denoted by an arrow originating nowhere. We can see that the symmetries of the nets actually produce automorphisms to the state space (only the reachable part is shown in Fig. 2). The reachability graph exhibits the rotation symmetry, so that we can rotate the markings, for instance, 3 railway sections forwards. In this case, '$\langle 0,a\rangle+\langle 4,b\rangle$,1+2' becomes '$\langle 3,a\rangle+\langle 1,b\rangle$,4+5' etc. The symmetry produced by swapping the trains is also present, e.g. '$\langle 3,a\rangle+\langle 1,b\rangle$,4+5' is mapped to '$\langle 1,a\rangle+\langle 3,b\rangle$,4+5'. Combining these two automorphisms, we see that '$\langle 0,a\rangle+\langle 4,b\rangle$,1+2' is symmetric to '$\langle 1,a\rangle+\langle 3,b\rangle$,4+5'. Moreover, the reachability graph is actually composed of markings belonging to two symmetry equivalence classes: (i) The first class is contains markings in which the distance between the trains is three sections, e.g. '$\langle 0,a\rangle+\langle 3,b\rangle$,1+4', '$\langle 1,a\rangle+\langle 4,b\rangle$,2+5', '$\langle 2,a\rangle+\langle 5,b\rangle$,0+3' and '$\langle 3,a\rangle+\langle 0,b\rangle$,1+4'. (ii) Markings in the second class are such that the distance between the trains is two sections, e.g. '$\langle 1,a\rangle+\langle 3,b\rangle$,4+5', '$\langle 0,a\rangle+\langle 4,b\rangle$,1+2' and '$\langle 5,a\rangle+\langle 1,b\rangle$,2+3'.

Based on the symmetry information discussed above, a reduced reachability graph can be generated. Figure 3 shows three possible outcomes. The one on the left side is the result of a normal depth first search (DFS) algorithm which uses the concept of canonical representative states. Whenever a new state is generated, it is transformed into the corresponding canonical

[1] $f : \Psi \to \mathrm{Sym}(G)$ is a group homomorphism iff for all $\psi, \psi' \in \Psi$, $f(\psi * \psi') = f(\psi) \circ f(\psi')$.



(a) A Place/Transition-net.

(b) A high-level net.

(c) Another high-level net.

Figure 1. Nets for a railway system.

representative state (which in this case is the lexicographically smallest state in the orbit of the state). All the states in the same orbit have the same canonical representative. After the transformation, the normal search is continued (it is checked whether the (representative) state is already visited etc.). The reduced reachability graph in the middle shows the outcome of the same algorithm but now the calculation of the representative marking is only approximated i.e. the representative state is not anymore unique. And the one on the right side is the result of a DFS algorithm that compares whether a state symmetrical to the newly generated is already visited and discards the state if it is the case. We can check simple properties of the railway system from all three reduced reachability graphs. For example, we see that the system is deadlock free and that there is no state in which the trains are in the same section (these properties can be verified from the reduced reachability graphs; see the papers mentioned in the beginning of this section). For detailed reachability graph generation algorithm descriptions see e.g. [6, 7, 17, 18, 27]. For complexity results concerning the orbit problem (are two states in the same orbit, calculation of the canonical representative state) refer, e.g., to [5, 6, 17].

During the example we have actually found and utilized the symmetry generating information of the nets. For the P/T-net the generating group Ψ is the internal direct product of the permutation groups Ψ_1 and Ψ_2, where Ψ_1 is the set of all rotation permutations of the net respecting node types (i.e. mapping places to places and transitions to transitions). For instance, $\psi_1^{(3)} = \begin{pmatrix} U_{a0} & U_{a1} & U_{a2} & U_{a3} & U_{a4} & U_{a5} & U_{b0} & \cdots & U_{b5} & V_0 & \cdots & V_5 & t_{a0} & \cdots & t_{a5} & t_{b0} & \cdots & t_{b5} \\ U_{a3} & U_{a4} & U_{a5} & U_{a0} & U_{a1} & U_{a2} & U_{b3} & \cdots & U_{b2} & V_3 & \cdots & V_2 & t_{a3} & \cdots & t_{a2} & t_{b3} & \cdots & t_{b2} \end{pmatrix} \in \Psi_1$. Ψ_2 is the

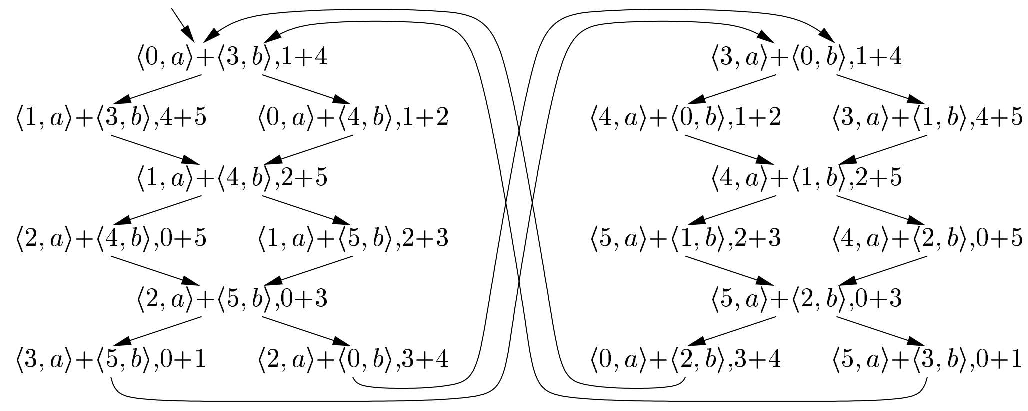


Figure 2. The reachability graph of the railway system in Fig. 1(c).

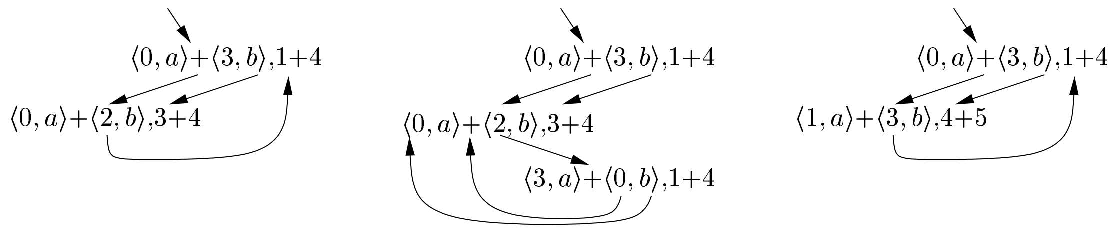


Figure 3. Reduced reachability graphs of the railway system in Fig. 1(c).

group consisting of the identity mapping and the permutation swapping the trains: $\psi_2^{(1)} = \left(\begin{smallmatrix} U_{a0} & \cdots & U_{a5} & U_{b0} & \cdots & U_{b5} & V_0 & \cdots & V_5 & t_{a0} & \cdots & t_{a5} & t_{b0} & \cdots & t_{b5} \\ U_{b0} & \cdots & U_{b5} & U_{a0} & \cdots & U_{a5} & V_0 & \cdots & V_5 & t_{b0} & \cdots & t_{b5} & t_{a0} & \cdots & t_{a5} \end{smallmatrix}\right)$. This is the information that the algorithms of Schmidt and Starke deduce automatically from nets [27, 28]. On the other hand, the symmetry generating information of the high-level net in Fig. 1(c) operates on the domains of the data types of the net. The elements of the group Ψ are now families of permutations operating on the domain $D_{\mathsf{Trains}} = \{a, b\}$ of the train identities, the domain $D_{\mathsf{Sections}} = \{0, 1, 2, 3, 4, 5, 6\}$ of the railway sections and the composite domain $D_{\mathsf{SecsXTrains}} = D_{\mathsf{Sections}} \times D_{\mathsf{Trains}}$ of the place U. For instance, a permutation family ψ consisting of the following data domain permutations is in Ψ: $\psi_{\mathsf{Trains}} = \left(\begin{smallmatrix} a & b \\ b & a \end{smallmatrix}\right)$, $\psi_{\mathsf{Sections}} = \left(\begin{smallmatrix} 0 & 1 & \cdots & 5 \\ 3 & 4 & \ldots & 2 \end{smallmatrix}\right)$, and $\psi_{\mathsf{SecsXTrains}} = \left(\begin{smallmatrix} \langle 0,a\rangle & \langle 1,a\rangle & \cdots & \langle 5,a\rangle & \langle 0,b\rangle & \cdots & \langle 5,b\rangle \\ \langle 3,b\rangle & \langle 4,b\rangle & \cdots & \langle 2,b\rangle & \langle 3,a\rangle & \cdots & \langle 2,a\rangle \end{smallmatrix}\right)$. This is the kind of symmetry information we try to find in this paper. The data domain permutations in Ψ are then extended to operate on markings in a way that, for example, '$\langle 0, a\rangle + \langle 4, b\rangle$,1+2' is mapped to '$\langle 1, a\rangle + \langle 3, b\rangle$,4+5'.

2.2. Three Approaches for Finding the Symmetries

In general, the problem of finding symmetries from the system description can be approached in three ways (cf. [27]). In the first, it is the user that deduces the symmetries and gives them, in some form or another, to the reachability analyzer [15, 18, 19]. The drawbacks of this approach are obvious; the user must really know what s/he is doing, must know the system thoroughly and must be familiar with the theory of symmetries. In addition, it may be the user's responsibility to also write algorithms that check whether two states are symmetric [18]. These tasks can be very hard and error-prone.

The second approach, the one studied in this work, is based on the "well-formedness" of system descriptions of certain kind [3, 17, 21]. Usually, the used system description language is a high-level one, involving various, possibly complex, data types. The well-formedness means that only certain kinds of restricted operations can be applied to some data types. This implies that the state spaces of such systems will contain symmetries which are produced by permuting the domains of the restricted data types. Thus the symmetries can be deduced directly from the type information of the system description; no expensive computations are needed. This is one of the greatest advantages of this approach. The symmetry information can also be presented in a very compact form inside the analyzer; there is no need to store the generators of the symmetry group explicitly. The knowledge of what kind of permutations can be applied to the domains of data types is sufficient. On the other hand, only certain predefined classes of symmetry groups, such as full permutation groups, rotation permutation groups and their compositions, can be handled. This is also a slightly semi-automatic approach in the sense that the user must use right kind of data types to describe the system and in this way hint the symmetries to the analyzer. It might also be the case that the approach does not detect some of the state space symmetries.

The third way to solve the problem of finding symmetries is to develop algorithms that find the symmetries from the system description automatically [5, 24, 27, 28]. This approach is mainly applied to low-level system descriptions such as low-level Petri nets or transistor-level circuits. Given a such description, the analyzer tool typically calculates an automorphism group of the description which produces the state space symmetries. Although the computation of automorphisms is a computationally hard task in theory, it seems to be quite tractable in practice. Besides that it is totally automatic, this approach can also handle any kind of symmetries; it is not restricted to certain types like the second approach. The greatest drawback of this approach is that the system description has to be a low-level one. In the context of Petri nets this means that a high-level net must be unfolded into the equivalent low-level net. When a high-level net contains lots of data or is otherwise too large, the unfolding might become impossible although the reachability analysis could still be within the limits of what the tools can handle. This phenomenon is caused by the fact that often not all places and transitions of the unfolded net may actually be used.

To justify the selection of our approach, notice that we are dealing with high-level nets in this paper. Our assumption is that the modeller of a net (or the computer program generating the net) has folded similar parts of the net into high-level constructs. Therefore, we expect that the structure of the net has very few or no symmetries. Moreover, we assume that the nets are so large that they cannot be unfolded. If they were unfoldable, we could also use the third approach mentioned above.

3. Algebraic System Nets

In this section we introduce a high-level Petri net formalism called algebraic system nets. Basically, an algebraic system net is a Petri net augmented with algebraic definitions. The presentation is based on [22, 23, 26].

Some basic notations must be defined first. In the rest of the paper, we use the set $\mathbb{B} = \{true, false\}$ for Boolean values and $\mathbb{N} = \{0, 1, 2, \dots\}$ for natural numbers. For two sets, A and B, $[A \to B]$ means the set of all functions from A to B. For a set A, $\mathcal{MS}(A)$ denotes the set of all multi-sets over A, i.e., the set $[A \to \mathbb{N}]$. We assume the usual multi-set operations, e.g., if $a, b \in \mathcal{MS}(A)$ are two multi-sets, then (i) $a = b$ iff $a(x) = b(x)$ for all $x \in A$, (ii) $a \geq b$ iff $a(x) \geq b(x)$ for all $x \in A$, (iii) $(a + b)(x) = a(x) + b(x)$ for all $x \in A$ and (iv) if $a \geq b$, then $(a - b)(x) = a(x) - b(x)$ for all $x \in A$. A multi-set $b \in \mathcal{MS}(A)$ can also be written as a formal sum $\Sigma_{a \in A} b(a) \times a$.

3.1. Signatures and Algebras

Signatures. A *signature* $SIG = \langle S, \Omega \rangle$ consists of a non-empty set S of *sorts* and a pairwise disjoint family $\Omega = (\Omega_{\sigma,s})_{\sigma \in S^*, s \in S}$ of *operation symbols*. The set $\Omega_{\varepsilon,s}$ is called the set of *SIG-constant symbols of sort s* (ε is the empty string). The disjointness of Ω is a mere technicality because we can always rename the operations. *SIG* is a *boolean signature* if $\mathsf{Bool} \in S$ and a *natural signature* if $\mathsf{Nat} \in S$. An operation symbol $\omega \in \Omega_{s_1 \dots s_n, s}$ stands for an operation from $s_1, \dots, s_n$ to s. A pairwise disjoint family $X = (X_s)_{s \in S}$ such that $X \cap \Omega = \emptyset$ is called a *set of SIG-variables.* (For a family $A = (A_i)_{i \in I}$, we sometimes write A to denote $\bigcup_{i \in I} A_i$.)

Using variables, constant symbols and operation symbols, we can build *SIG*-terms as follows. The set $\mathbf{T}_s^{SIG}(X)$ of *SIG-terms of sort s over X* is the minimal set (in the set inclusion sense) defined inductively by the following rules: (i) $X_s \subseteq \mathbf{T}_s^{SIG}(X)$, (ii) $\Omega_{\varepsilon,s} \subseteq \mathbf{T}_s^{SIG}(X)$, and (iii) for $n \geq 1$, if $\omega \in \Omega_{s_1 \dots s_n, s}$ and $T_i \in \mathbf{T}_{s_i}^{SIG}(X)$ for all $1 \leq i \leq n$, then $\omega(T_1, \dots, T_n) \in \mathbf{T}_s^{SIG}(X)$. The set $\mathbf{T}_s^{SIG} = \mathbf{T}_s^{SIG}(\emptyset)$ is the set of *SIG-ground terms of sort s*, the set $\mathbf{T}^{SIG}(X) = \bigcup_{s \in S} \mathbf{T}_s^{SIG}(X)$ is the set of *SIG-terms over X* and the set $\mathbf{T}^{SIG} = \bigcup_{s \in S} \mathbf{T}_s^{SIG}$ is the set of *SIG-ground terms.*

Algebras. An algebra corresponding to a signature *SIG* gives an interpretation for sorts and operations. It assigns each sort a domain and each operation symbol a function. Formally, a *SIG-algebra* $\mathcal{A} = \langle D, \Omega_{\mathcal{A}} \rangle$ consists of a family $D = (D_s)_{s \in S}$ of non-empty *domain sets* for sorts and a family $\Omega_{\mathcal{A}} = (\omega_{\mathcal{A}})_{\omega \in \Omega}$ of *operations.* For each $\omega \in \Omega_{s_1 \dots s_n, s}$, $n \geq 1$, the operation $\omega_{\mathcal{A}}$ is a mapping $\omega_{\mathcal{A}} : D_{s_1} \times \cdots \times D_{s_n} \to D_s$. For a constant $c \in \Omega_{\varepsilon,s}$, we define $c_{\mathcal{A}} : \emptyset \to D_s$. If *SIG* is a boolean signature, then $D_{\mathsf{Bool}} = \mathbb{B}$. Similarly, if *SIG* is a natural signature, then $D_{\mathsf{Nat}} = \mathbb{N}$. Given an *SIG*-algebra, we can assign values to *SIG*-variables: If X is a family of *SIG*-variables, then an *assignment* β *to variables in* X is a mapping $\beta : X \to D$ such that $\beta(x) \in D_s$ for each variable $x \in X_s$. The *set of all assignments to* X is denoted by $\mathrm{Ass}(X)$.

An assignment β to variables is extended to the corresponding *evaluation of terms*, $eval_\beta : \mathbf{T}^{SIG}(X) \to D$, by the following inductive definition: (i) if $T = x \in X_s$, then $eval_\beta(T) = \beta(T)$,

(ii) if $T = c \in \Omega_{\varepsilon,s}$, then $eval_\beta(T) = c_{\mathcal{A}}()$, and (iii) if $T = \omega(T_1, \ldots, T_n)$, $\omega \in \Omega_{s_1 \ldots s_n, s}$ and $T_i \in \mathbf{T}^{SIG}_{s_i}(X)$, then $eval_\beta(T) = \omega_{\mathcal{A}}(eval_\beta(T_1), \ldots, eval_\beta(T_n))$. Note that for ground terms all evaluations yield the same value because variables are not involved in ground terms. For a ground term T we may thus write $eval(T)$ instead of $eval_\beta(T)$ (for any assignment β).

Multi-Set Signatures and Algebras. We now introduce multi-set signatures and their corresponding algebras, which are special cases of signatures and algebras, respectively. We distinguish some ground sorts and assign each of them a corresponding multi-set sort. The domain of a multi-set sort is the set of all multi-sets over the domain of the corresponding ground sort. The definition is based on [22, 23]. However, we use a stricter definition in the sense that we require that each sort is either a ground sort or a multi-set sort. This is a mere technicality because we can always augment the set of sorts by the corresponding (multi-set) sorts even though we actually never use them.

For a signature $SIG = \langle S, \Omega \rangle$, let $GS, MS \subseteq S$ be two non-empty disjoint sets such that $GS \cup MS = S$ and $GS \cap MS = \emptyset$. GS is called the set of *ground sorts* and MS is the set of *multi-set sorts*. $MSIG = \langle S, \Omega, ms \rangle$ is a *multi-set-signature* iff $ms : GS \to MS$ is a bijective mapping. A SIG-algebra is a *$MSIG$-algebra* iff $D_{ms(s)} = \mathcal{MS}(D_s)$ for all $s \in GS$.

As each multi-set-signature $MSIG = \langle S, \Omega, ms \rangle$ is a specialization of the corresponding signature $SIG = \langle S, \Omega \rangle$, variables, terms, assignments and evaluations are well-defined for multi-set signatures, too. Furthermore, each $MSIG$-algebra is a SIG-algebra.

3.2. Algebraic System Nets

We will now combine the algebraic techniques described above with Petri nets. The result is a class of high-level Petri nets, called the class of algebraic system nets.

Definition 3.1. (Algebraic System Nets) An *algebraic system net* $\mathcal{N}$ is a tuple

$$\mathcal{N} = \langle N, MSIG, \mathcal{A}, d, v, a, g, m_0 \rangle,$$

where

- $N = \langle P, T, F \rangle$ is an *underlying net* consisting of a finite non-empty set P of *places*, a finite set T of *transitions* such that $P \cap T = \emptyset$, and a set $F \subseteq (P \times T) \cup (T \times P)$ of *arcs* (the flow relation);
- $MSIG = \langle S, \Omega, ms \rangle$ is a Boolean multi-set-signature with a set GS of ground sorts;
- $\mathcal{A} = \langle D, \Omega_{\mathcal{A}} \rangle$ is a $MSIG$-algebra;
- $d : P \to GS$ is a *sort assignment function* assigning a ground sort to each place;
- v assigns each transition $t \in T$ a finite family X_t of $MSIG$-variables;
- a is an *arc annotation function* assigning each arc $\langle t, p \rangle, \langle p, t \rangle \in F$, where $p \in P$ and $t \in T$, an $MSIG$-term of place's multi-set sort over transition's variables. That is, $a(t,p), a(p,t) \in \mathbf{T}^{MSIG}_{ms(d(p))}(v(t))$;

- g assigns each transition $t \in T$ a *transition guard*, which is a Boolean term over transition's variables, i.e., $g(t) \in \mathbf{T}^{MSIG}_{\mathsf{Bool}}(v(t))$; and
- $m_0 : P \to \mathbf{T}^{MSIG}$ such that $m_0(p) \in \mathbf{T}^{MSIG}_{ms(d(p))}$ for each $p \in P$ is the *symbolic initial marking* assigning each place an *MSIG*-ground term of place's multi-set sort.

In the rest of this section we assume a fixed algebraic system net $\mathcal{N}$. A *marking* M of $\mathcal{N}$ is a function that assigns each place $p \in P$ a multi-set over its domain: $M(p) \in D_{ms(d(p))}$ for all $p \in P$. The *set of all markings* of $\mathcal{N}$ is denoted by $\mathbb{M}$. The *initial marking* M_0 is the evaluation of the symbolic initial marking: $M_0(p) = eval(m_0(p))$ for all $p \in P$.

Let t be a transition. An assignment β to its variables $v(t)$ is called a *mode for* t and the pair $t_\beta = \langle t, \beta \rangle$ is a *transition instance*. The *set of all transition instances* of $\mathcal{N}$ is denoted by $\mathbb{T}$. Note that $\mathbb{M}$ and $\mathbb{T}$ are disjoint because P and T are disjoint. For a transition instance t_β, the *input effect function* t^-_β is defined by: $t^-_\beta(p) = eval_\beta(a(p,t))$ if $\langle p, t \rangle \in F$ and $t^-_\beta(p) = \emptyset$ otherwise for all $p \in P$. Similarly, the *output effect function* t^+_β is defined by: $t^+_\beta(p) = eval_\beta(a(t,p))$ if $\langle t, p \rangle \in F$ and $t^+_\beta(p) = \emptyset$ when $\langle t, p \rangle \notin F$. A transition instance t_β is *enabled in a marking* M, denoted by $M\,[t_\beta\rangle$, iff (i) $eval_\beta(g(t)) = true$ and (ii) $M(p) \geq t^-_\beta(p)$ for all $p \in P$. An enabled transition instance may *fire* and transform M into the new marking M_1 calculated as: $M_1(p) = M(p) - t^-_\beta(p) + t^+_\beta(p)$ for all $p \in P$. This is denoted by $M\,[t_\beta\rangle\,M_1$. The *state space of* $\mathcal{N}$ is the LTS $\mathcal{S}(\mathcal{N}) = \langle \mathbb{M}, \mathbb{T}, [\rangle, M_0 \rangle$, where $[\rangle = \{\langle M_1, t_\beta, M_2 \rangle \in \mathbb{M} \times \mathbb{T} \times \mathbb{M} \mid M_1\,[t_\beta\rangle\,M_2\}$.

Example 3.1. Consider the railway system net in Fig. 1(c). We now define the multi-set signature $MSIG^{\mathrm{rw}} = \langle S^{\mathrm{rw}}, \Omega^{\mathrm{rw}}, ms^{\mathrm{rw}} \rangle$ and the corresponding multi-set algebra $\mathcal{A}^{\mathrm{rw}} = \langle D^{\mathrm{rw}}, \Omega^{\mathrm{rw}}_{\mathcal{A}^{\mathrm{rw}}} \rangle$ for it. The set S^{rw} of sorts consists of the ground sorts Bool for the Booleans, Trains for the trains, $\mathsf{Sections}$ for the railway sections, the composite sort $\mathsf{SecsXTrains}$ for the place U and of the corresponding multi-set sorts, e.g. $\mathsf{MSSections}$. The function ms^{rw} maps each ground sort into the corresponding multi-set sort. The domain of the Booleans is $\mathbb{B}$ and the operations for it are the usual Boolean operations, e.g. $\wedge \in \Omega^{\mathrm{rw}}_{\mathsf{Bool.Bool,Bool}}$, and the constants true and false, all defined in the obvious way. For the trains, the domain is $D_{\mathsf{Trains}} = \{a, b\}$ and the operations consists of the constants a and b such that $\mathsf{a}_{\mathcal{A}^{\mathrm{rw}}}() = a$ and $\mathsf{b}_{\mathcal{A}^{\mathrm{rw}}}() = b$. For the railway sections we have the domain $D_{\mathsf{Sections}} = \{0, 1, \ldots, 5\}$ and corresponding constants $\mathsf{0}, \ldots, \mathsf{5}$. The operations $next, prev \in \Omega_{\mathsf{Sections,Sections}}$ are defined as in the legend of the figure. $\mathsf{Sections}$ is the domain of the place V. The composite sort $\mathsf{SecsXTrains}$ has the domain $D_{\mathsf{SecsXTrains}} = D_{\mathsf{Sections}} \times D_{\mathsf{Trains}}$ and an operation $\langle \cdot, \cdot \rangle \in \Omega_{\mathsf{Sections.Trains,SecsXTrains}}$, defined in the obvious way. For the multi-set sorts, e.g. $\mathsf{MSSections}$, we define the usual multi-set operations such as the multi-set addition operation. Note that we also write, for instance, $\langle i, x \rangle$ to denote the multi-set $1 \times \langle i, x \rangle$ of sort $\mathsf{MSSecsXTrains}$ (that is, we implicitly use the operation $\cdot$ that maps each element of a ground sort s to the corresponding singleton multi-set of the corresponding multi-set sort). The guard of the transition t is the constant true. The symbolic initial marking of place U is the ground term $\langle \mathsf{0}, \mathsf{a} \rangle + \langle \mathsf{3}, \mathsf{b} \rangle$, where $+$ is the multi-set addition operation $+ \in \Omega_{\mathsf{MSSecsXTrains.MSSecsXTrains,MSSecsXTrains}}$ written in infix notation. In the initial marking $\{U \to \langle 0, a \rangle + \langle 3, b \rangle, V \to 1 + 4\}$, the transition t is enabled for mode $\{i \to 0, x \to a\}$ and firing it leads to the marking $\{U \to \langle 1, a \rangle + \langle 3, b \rangle, V \to 4 + 5\}$.

4. Symmetries for Algebraic System Nets

In this section we present an approach to find state space symmetries of algebraic system nets. Recall our assumption that the algebraic system nets we consider are well folded. Thus the symmetries based on the structure of a net are of little or no use (without unfolding the net, of course). Instead, the symmetries are produced by symmetric use of data. We therefore try to obtain symmetries by permuting the data domains of the net, meaning the domains of sorts. The domain permutations are then extended into markings and transition instances. In other words, data domain permutations produce state space permutations. This is the topic of Sec. 4.1.

Although domain permutations produce state space permutations, not all of them necessarily produce state space symmetries. This is because transitions may behave asymmetrically with respect to the domain permutations. In other words, the condition distinguishing state space symmetries from permutations is violated. To enforce symmetric behaviour, we must impose some restrictions on the terms we use as transition guards and arc annotations. We do this in Sec. 4.2 by introducing a relation called compatibility between domain permutations and terms. We show that when all the terms used in a net as transition guards and arc annotations are compatible with a group of domain permutations, then the permutation group produced into the state space is actually a symmetry group.

In the end of this section, we give an approximation rule for deciding term compatibility. It is based on a lower-level compatibility between domain permutations and the individual operations comprising terms.

4.1. Domain Permutations

We will first define how the domains of the ground sorts of a multi-set algebra are permuted. The permutations are then gradually extended to operate on the state space level.

Assume a multi-set signature $MSIG = \langle S, \Omega, ms \rangle$ with the set GS of ground sorts and a corresponding $MSIG$-algebra $\mathcal{A} = \langle D, \Omega_{\mathcal{A}} \rangle$. The ground domain permutations only operate within the domains so the elements of a domain are not permuted to the elements of other domains:

Definition 4.1. (Domain Permutations) A *ground domain permutation* for $\mathcal{A}$ is a family $\psi_{GS} = (\psi_s)_{s \in GS}$, where for each $s \in GS$, ψ_s is a bijective function on D_s. A *ground domain permutation group* for $\mathcal{A}$ is a non-empty set Ψ_{GS} of ground domain permutations for $\mathcal{A}$ such that $\langle \Psi_{GS}, * \rangle$ is a group, where $*$ is the sort-wise function composition operator on families of domain permutations, i.e., $(\psi_s)_{s \in S'} * (\psi'_s)_{s \in S'} = (\psi''_s)_{s \in S'}$ such that $\psi''_s = \psi_s \circ \psi'_s$ for all $s \in S'$, where $S' \subseteq S$ is any non-empty set of sorts.

If we have specified a group Ψ_s of permutations on D_s for each ground sort $s \in GS$ individually (the binary group operator being the function composition operator $\circ$), we can combine them into a ground domain permutation group for $\mathcal{A}$ by taking any appropriate sub-group of $\bigotimes_{s \in GS} \Psi_s = \{(\psi_s)_{s \in GS} \mid (\forall s \in GS)(\psi_s \in \Psi_s)\}$. The definition of $\bigotimes$ resembles the exterior direct product construction of the standard group theory (see e.g. [10]). The only difference is that it is expressed by using families instead of Cartesian products, which saves us from defining artificial total orders for the set of sorts. It is quite straightforward to see that $\bigotimes_{s \in GS} \Psi_s$ is a group under the binary operator $*$ defined in Def. 4.1.

We extend ground domain permutations for multi-set sorts by the following definition, which states that the corresponding permutations are produced by permuting the elements of multi-sets but leaving the multiplicities intact.

Definition 4.2. Let $\psi_{GS} = (\psi_s)_{s \in GS}$ be a ground domain permutation for $\mathcal{A}$. The function f maps ψ_{GS} to the corresponding *domain permutation* $\psi_S = f(\psi_{GS}) = (\psi'_s)_{s \in S}$ defined by

1. for all $s \in GS$, $\psi'_s = \psi_s$ and
2. for all $s \in GS$, $\psi'_{ms(s)} : \mathcal{MS}(D_s) \to \mathcal{MS}(D_s)$ is such that for each multi-set $b = \sum_{v \in D_s} b(v) \times v$ in $\mathcal{MS}(D_s)$, $\psi'_{ms(s)}(b) = \sum_{v \in D_s} b(v) \times \psi'_s(v)$.

A ground domain permutation group Ψ_{GS} is extended to the corresponding *domain permutation group* for $\mathcal{A}$ by $\Psi_S = f(\Psi_{GS}) = \{f(\psi_{GS}) \mid \psi_{GS} \in \Psi_{GS}\}$.

It follows directly from the definition that Ψ_S is a group under $*$ and that the function f is a group isomorphism from Ψ_{GS} to Ψ_S. As the domain permutations for multi-set sorts are dictated by the domain permutations for the corresponding ground sorts, the following properties hold.

Lemma 4.1. *Each $\psi_{ms(s)} : \mathcal{MS}(D_s) \to \mathcal{MS}(D_s)$ as defined in Def. 4.1 satisfies the following:*

1. *$\psi_{ms(s)}$ maps empty multi-set to empty multi-set*
2. *$\psi_{ms(s)}$ is linear, that is, $\psi_{ms(s)}(b_1 + b_2) = \psi_{ms(s)}(b_1) + \psi_{ms(s)}(b_2)$*
3. *$\psi_{ms(s)}(n \times b) = n \times \psi_{ms(s)}(b)$, for each $n \in \mathbb{N}$*
4. *For any $b_1, b_2 \in \mathcal{MS}(D_s)$, $b_1 \leq b_2$ iff $\psi_{ms(s)}(b_1) \leq \psi_{ms(s)}(b_2)$*

Proof:
First, $\psi_{ms(s)}(\emptyset) = \sum_{v \in D_s} \emptyset(v) \times \psi_s(v) = \emptyset$. The item 2 is true because $\psi_{ms(s)}(b_1 + b_2) = \sum_{v \in D_s} (b_1 + b_2)(v) \times \psi_s(v) = \sum_{v \in D_s} (b_1(v) + b_2(v)) \times \psi_s(v) = \sum_{v \in D_s} b_1(v) \times \psi_s(v) + \sum_{v \in D_s} b_2(v) \times \psi_s(v) = \psi_{ms(s)}(b_1) + \psi_{ms(s)}(b_2)$. Similarly, $\psi_{ms(s)}(n \times b) = \sum_{v \in D_s} (n \times b)(v) \times \psi_s(v) = \sum_{v \in D_s} n \times b(v) \times \psi_s(v) = n \times \sum_{v \in D_s} b(v) \times \psi_s(v) = n \times \psi_{ms(s)}(b)$. Finally, $\psi_{ms(s)}(b_1) \leq \psi_{ms(s)}(b_2)$ iff $\big(\psi_{ms(s)}(b_1)\big)(v) \leq \big(\psi_{ms(s)}(b_2)\big)(v)$ for all $v \in D_s$. By the bijective nature of ψ_s, this is equivalent to that $\big(\psi_{ms(s)}(b_1)\big)(\psi_s(v)) \leq \big(\psi_{ms(s)}(b_2)\big)(\psi_s(v))$ for all $v \in D_s$. This in turn equals to that $b_1(v) \leq b_2(v)$ for all $v \in D_s$ and thus to the requirement that $b_1 \leq b_2$. □

We extend domain permutations for variable assignments by simply permuting the assigned values.

Definition 4.3. Let $\psi_S = (\psi_s)_{s \in S}$ be a domain permutation for $\mathcal{A}$. Let β be an assignment to a family $X = (X_s)_{s \in S}$ of *MSIG*-variables. Then the corresponding permuted variable assignment $\psi_{\mathrm{Ass}(X)}(\beta)$ is defined by $\big(\psi_{\mathrm{Ass}(X)}(\beta)\big)(x) = \psi_s(\beta(x))$ for all sorts $s \in S$ and all variables $x \in X_s$.

Now, assume that we have an algebraic system net $\mathcal{N}$ using the *MSIG* as its signature and $\mathcal{A}$ as its *MSIG*-algebra. We say that a domain permutation group for $\mathcal{A}$ is a *domain permutation group for $\mathcal{N}$*. The next step is to extend domain permutations to the state space level of $\mathcal{N}$. For markings, we permute the multi-sets of each individual place but do not permute the places.

Definition 4.4. (Permuted Markings) Let $\psi_S = (\psi_s)_{s \in S}$ be a domain permutation for $\mathcal{N}$ and let $M \in \mathbb{M}$ be a marking of $\mathcal{N}$. The corresponding permuted marking $\psi_{\mathbb{M}}(M)$ is defined by $\big(\psi_{\mathbb{M}}(M)\big)(p) = \psi_{ms(d(p))}(M(p))$ for all $p \in P$.

In a similar way, a domain permutation is extended to transition instances. We simply permute the variable binding of a transition instance but leave the transition name intact.

Definition 4.5. (Permuted Transition Instances) Let $\psi_S = (\psi_s)_{s \in S}$ be a domain permutation for $\mathcal{N}$ and let $t_\beta = \langle t, \beta \rangle$ be a transition instance of $\mathcal{N}$. The corresponding transition instance $\psi_{\mathbb{T}}(t_\beta)$ is defined by $\psi_{\mathbb{T}}(t_\beta) = \langle t, \psi_{\mathrm{Ass}(v(t))}(\beta) \rangle$. We may also denote $\psi_{\mathbb{T}}(t_\beta)$ by $t_{\psi(\beta)}$.

We specify that a domain permutation group Ψ_S for $\mathcal{N}$ is extended to the state space level by the following definition: $\Psi_{\mathcal{S}(\mathcal{N})} = \{\psi_{\mathbb{M}} \cup \psi_{\mathbb{T}} \mid \psi_S \in \Psi_S\}$. But is $\Psi_{\mathcal{S}(\mathcal{N})}$ a permutation group of the state space $\mathcal{S}(\mathcal{N})$ of $\mathcal{N}$? Clearly, the answer is yes.

Lemma 4.2. *$\langle \Psi_{\mathcal{S}(\mathcal{N})}, \circ \rangle$ is a permutation group for $\mathcal{S}(\mathcal{N})$.*

Proof:
Because each ψ_s in ψ_S for a ground sort $s \in GS$ is a permutation on D_s, the corresponding function $\psi_{ms(s)}$ for the multi-set sort is a permutation on $\mathcal{MS}(D_s)$, directly by Def. 4.2. Based on this, it is easy to see that the corresponding function $\psi_{\mathbb{M}}$ on the set $\mathbb{M}$ of all markings of $\mathcal{N}$ is also a permutation. Similarly, each corresponding function $\psi_{\mathrm{Ass}(X)}$ on variable assignments to X is a permutation on $\mathrm{Ass}(X)$. Therefore, the function $\psi_{\mathbb{T}}$ is a permutation on the set $\mathbb{T}$ of all transition instances of $\mathcal{N}$. As $\mathbb{M}$ and $\mathbb{T}$ are disjoint, the union $\psi_{\mathcal{S}(\mathcal{N})} = \psi_{\mathbb{M}} \cup \psi_{\mathbb{T}}$ is a permutation on $\mathbb{M} \cup \mathbb{T}$ such that $\psi_{\mathcal{S}(\mathcal{N})}(\mathbb{M}) = \mathbb{M}$ and $\psi_{\mathcal{S}(\mathcal{N})}(\mathbb{T}) = \mathbb{T}$.

The group properties of $\Psi_{\mathcal{S}(\mathcal{N})}$ follow directly from the group properties of Ψ_S. For instance, consider the closure property. We have to show that if $\psi_{\mathcal{S}(\mathcal{N})}$ and $\psi'_{\mathcal{S}(\mathcal{N})}$ are in $\Psi_{\mathcal{S}(\mathcal{N})}$, then $\psi_{\mathcal{S}(\mathcal{N})} \circ \psi'_{\mathcal{S}(\mathcal{N})}$ is also in $\Psi_{\mathcal{S}(\mathcal{N})}$. Let $\psi_S = (\psi_s)_{s \in S}$ and $\psi'_S = (\psi'_s)_{s \in S}$ be the two domain permutations in Ψ_S producing $\psi_{\mathcal{S}(\mathcal{N})}$ and $\psi'_{\mathcal{S}(\mathcal{N})}$, respectively. Let $\psi''_S = (\psi''_s)_{s \in S}$ be the domain permutation $\psi''_S = \psi_S * \psi'_S$. Because Ψ_S is a group under $*$, ψ''_S is also in Ψ_S. First, we show that $\psi''_{ms(s)} = \psi_{ms(s)} \circ \psi'_{ms(s)}$ for each ground sort $s \in GS$: $(\psi_{ms(s)} \circ \psi'_{ms(s)})(b) = \psi_{ms(s)}(\psi'_{ms(s)}(b)) = \psi_{ms(s)}(\sum_{v \in D_s} b(v) \times \psi'_s(v)) = \sum_{v \in D_s} b(v) \times \psi_s(\psi'_s(v)) = \sum_{v \in D_s} b(v) \times ((\psi_s \circ \psi'_s)(v)) = \sum_{v \in D_s} b(v) \times \psi''_s(v) = \psi''_{ms(s)}(b)$ for all $b \in \mathcal{MS}(D_s)$. Now for each marking $M \in \mathbb{M}$ and each place $p \in P$, $\big(\psi''_{\mathbb{M}}(M)\big)(p) = \psi''_{ms(d(p))}(M(p)) = (\psi_{ms(d(p))} \circ \psi'_{ms(d(p))})(M(p)) = \psi_{ms(d(p))}(\psi'_{ms(d(p))}(M(p))) = \big(\psi_{\mathbb{M}}(\psi'_{\mathbb{M}}(M))\big)(p) = \big(\psi_{\mathbb{M}}(M) \circ \psi'_{\mathbb{M}}(M)\big)(p)$. Thus $\psi_{\mathbb{M}}$ is closed under $\circ$. Similarly, for each variable assignment β to variables in an X, we can prove that $\psi''_{\mathrm{Ass}(X)}(\beta) = (\psi_{\mathrm{Ass}(X)} \circ \psi'_{\mathrm{Ass}(X)})(\beta)$. This in turn implies that for each transition instance $t_\beta \in \mathbb{T}$, we have $\psi''_{\mathbb{T}}(t_\beta) = \langle t, \psi''_{\mathrm{Ass}(v(t))}(\beta) \rangle = \langle t, (\psi_{\mathrm{Ass}(X)}(\psi'_{\mathrm{Ass}(X)}(\beta))) \rangle = (\psi_{\mathbb{T}} \circ \psi'_{\mathbb{T}})(t_\beta)$. Thus $\psi_{\mathbb{T}}$ is closed under $\circ$ and $\psi_{\mathcal{S}(\mathcal{N})}$ is closed under $\circ$.

The other group properties follow in a similar way. The proof actually shows that there is a group homomorphism from Ψ_S to $\Psi_{\mathcal{S}(\mathcal{N})}$. $\square$

Example 4.1. Let us revisit our train example (Ex. 3.1). For each sort s, let $\mathrm{Perm}(D_s)$ denote the set of all permutations of the domain of s. We choose the ground domain permutation group Ψ_{GS} for $\mathcal{A}^{\mathrm{trains}}$ to be the sub-group of $\bigotimes_{s \in GS} \mathrm{Perm}(D_s)$ such that each $\psi_{GS} =$

$(\psi_{\mathsf{Bool}}, \psi_{\mathsf{Sections}}, \psi_{\mathsf{Trains}}, \psi_{\mathsf{SecsXTrains}})$ belonging to Ψ_{GS} satisfies the following properties: (i) ψ_{Bool} is the identity mapping, (ii) ψ_{Sections} is a rotation of D_{Sections}, e.g. $\left(\begin{smallmatrix}0&1&2&3&4&5\\3&4&5&0&1&2\end{smallmatrix}\right)$, (iii) ψ_{Trains} is either $\left(\begin{smallmatrix}a&b\\a&b\end{smallmatrix}\right)$ or $\left(\begin{smallmatrix}a&b\\b&a\end{smallmatrix}\right)$, and (iv) $\psi_{\mathsf{SecsXTrains}}(\langle s,t\rangle) = \langle \psi_{\mathsf{Sections}}(s), \psi_{\mathsf{Trains}}(t)\rangle$ for all $\langle s,t\rangle \in D_{\mathsf{Sections}} \times D_{\mathsf{Trains}}$. For instance, $\psi_{GS} = \left(\psi_{\mathsf{Bool}} = \left(\begin{smallmatrix}\mathit{true}&\mathit{false}\\\mathit{true}&\mathit{false}\end{smallmatrix}\right), \psi_{\mathsf{Sections}} = \left(\begin{smallmatrix}0&1&2&3&4&5\\3&4&5&0&1&2\end{smallmatrix}\right), \psi_{\mathsf{Trains}} = \left(\begin{smallmatrix}a&b\\b&a\end{smallmatrix}\right), \psi_{\mathsf{SecsXTrains}} = \left(\begin{smallmatrix}\langle 0,a\rangle&\langle 1,a\rangle&\langle 2,a\rangle&\langle 3,a\rangle&\langle 4,a\rangle&\langle 5,a\rangle&\langle 0,b\rangle&\langle 1,b\rangle&\langle 2,b\rangle&\langle 3,b\rangle&\langle 4,b\rangle&\langle 5,b\rangle\\\langle 3,b\rangle&\langle 4,b\rangle&\langle 5,b\rangle&\langle 0,b\rangle&\langle 1,b\rangle&\langle 2,b\rangle&\langle 3,a\rangle&\langle 4,a\rangle&\langle 5,a\rangle&\langle 0,a\rangle&\langle 1,a\rangle&\langle 2,a\rangle\end{smallmatrix}\right)\right)$ is in Ψ_{GS}.

When ψ_{GS} is extended to the multi-set sort $\mathsf{MSSecsXTrains}$ and is applied to the multi-set $2 \times \langle 2,a\rangle + \langle 4,b\rangle$, the result is the multi-set $\langle 1,a\rangle + 2 \times \langle 5,b\rangle$. If the variables i and x are of sorts $\mathsf{Sections}$ and Trains, respectively, then ψ_{GS}, extended to variable assignments, maps the assignment $\{i \to 4, x \to a\}$ to $\{i \to 1, x \to b\}$. Similarly, ψ_{GS}, when extended to markings, transforms $\{U \to \langle 1,a\rangle + \langle 3,b\rangle, V \to 4+5\}$ into $\{U \to \langle 0,a\rangle + \langle 4,b\rangle, V \to 1+2\}$.

Example 4.2. To see that domain permutations do not necessarily produce state space symmetries, assume that we have modelled a system with three client processes communicating with a server. A part of the net is shown in Fig. 4. The client process identifiers are modelled with a sort CId with the domain $D_{\mathsf{CId}} = \{cId_1, cId_2, cId_3\}$ and the corresponding constants $\mathsf{cId}_1, \mathsf{cId}_2$ and cId_3. the sort CId is the domain of the places c_CS and $c\_crashed$. Consider a domain permutation ψ_S in which $\psi_{\mathsf{CId}}(cId_3) = cId_1$. The transition *crash* with the guard $x = \mathsf{cId}_3$, modelling that the client number 3 can crash in its critical section, can fire in a marking where $M(c\_CS) = cId_3$ but not in the permuted marking in which $(\psi_{\mathbb{M}}(M))(c\_CS) = cId_1$. Thus the transition *crash* breaks the symmetries because its guard is not compatible with the applied permutation.

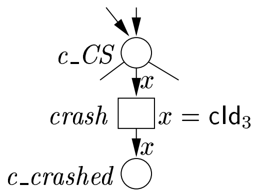


Figure 4. A symmetry-breaking transition.

4.2. Term Compatibility

To solve the problem of symmetry-breaking transitions described at the end of the last subsection, we define compatibility between domain permutations and terms. Basically, compatibility means that for a permuted variable assignment, the evaluation result of a term should be similarly permuted. As it turns out, when this condition is applied to the terms appearing in an algebraic system net (with one additional restriction), the domain permutations indeed produce state space symmetries. Formally:

Definition 4.6. (Compatible Terms) A term $T \in \mathbf{T}^{SIG}_s(X)$ and a domain permutation ψ_S are *compatible* iff $eval_{\psi_{\mathrm{Ass}(X)}(\beta)}(T) = \psi_s(eval_\beta(T))$ holds for all assignments β on X. The term T is compatible with a domain permutation group Ψ_S iff it is compatible with all $\psi_S \in \Psi_S$.

Consequently, a term consisting only of a variable is by the definition compatible with all possible domain permutations because $eval_{\psi_{\mathrm{Ass}(X)}(\beta)}(x) = \big(\psi_{\mathrm{Ass}(X)}(\beta)\big)(x) = \psi_s(\beta(x)) = \psi_s(eval_\beta(x))$ for any variable x of sort s.

As we use the Booleans in the definition of transition guards and the firing rule, we may not permute the domain of the Booleans. This is because swapping the truth value of a transition guard could make the transition disabled for some permutations and enabled for some, which would certainly break the symmetries. Formally:

Definition 4.7. (Well-Formed Domain Permutations) We say that a domain permutation $\psi_S = (\psi_s)_{s \in S}$ is *well-formed* iff $\mathsf{Bool} \in S$ implies $\psi_{\mathsf{Bool}}(v) = v$ for all $v \in \mathbb{B}$. A domain permutation group is *well-formed* iff all the domain permutations in it are well-formed.

Note that the use of well-formed domain permutations does not prevent us from using other sorts that are isomorphic to the sort Bool, to which we can apply the domain permutation swapping true to false and vice versa.

Definition 4.8. (Well-Formed Algebraic System Nets) We say that an algebraic system net $\mathcal{N}$ is *well-formed* with respect to a well-formed domain permutation group Ψ_S iff

1. all transition guards $g(t)$ are compatible with Ψ_S and
2. all arc annotations $a(p,t), a(t,p)$ are compatible with Ψ_S.

The requirement that the transition guards are compatible is a necessary one. If the guard is true for a mode of a transition, then (and only then) it should also be true for all permuted modes. Otherwise a transition instance could be enabled in a marking but a permuted instance could be disabled in a similarly permuted marking. The second requirement concerning the arc annotations is only a sufficient condition since it could be the case that an arc annotation is not compatible but the transition guard eliminates the incompatible parts of the arc annotation (in the extreme case, the guard could be invariably false). Note that we do not require the compatibility of the symbolic initial marking.

We can now prove the main result of this paper, namely that if we use a well-formed algebraic net with a compatible well-formed domain permutation group, the group will induce a symmetry group for the state space of the net.

Theorem 4.1. *If an algebraic system net $\mathcal{N}$ is well-formed with respect to a well-formed domain permutation group Ψ_S, then $\Psi_{\mathcal{S}(\mathcal{N})}$ is a symmetry group for $\mathcal{S}(\mathcal{N})$.*

Proof:
By Theorem 4.2, $\Psi_{\mathcal{S}(\mathcal{N})}$ is a permutation group for $\mathcal{S}(\mathcal{N})$. We now have to show that for all $M_1, M_2 \in \mathbb{M}$, for all $t_\beta \in \mathbb{T}$ and for all $\psi_{\mathcal{S}(\mathcal{N})} \in \Psi_{\mathcal{S}(\mathcal{N})}$ it holds that

$$M_1\ [t_\beta\rangle\ M_2 \Leftrightarrow \psi_{\mathcal{S}(\mathcal{N})}(M_1)\ [\psi_{\mathcal{S}(\mathcal{N})}(t_\beta)\rangle\ \psi_{\mathcal{S}(\mathcal{N})}(M_2).$$

Take any $M_1, M_2 \in \mathbb{M}$, any $t_\beta \in \mathbb{T}$ and any $\psi_{\mathcal{S}(\mathcal{N})} \in \Psi_{\mathcal{S}(\mathcal{N})}$. Suppose that $\psi_{\mathcal{S}(\mathcal{N})}$ is induced by a domain permutation $\psi_S \in \Psi_S$.

First, recall that $\psi_{\mathcal{S}(\mathcal{N})}(t_\beta) = \psi_{\mathbb{T}}(t_\beta) = \langle t, \psi_{\mathrm{Ass}(X)}(\beta)\rangle$. Now $eval_{\psi_{\mathrm{Ass}(X)}(\beta)}(g(t))$ equals by compatibility (Def. 4.6) to $\psi_{\mathsf{Bool}}(eval_\beta(g(t)))$, which in turn equals to $eval_\beta(g(t))$ by the fact that $(\forall v \in \mathbb{B})(\psi_{\mathsf{Bool}}(v) = v)$ for well-formed domain permutations. Thus the guard is true for t_β iff it is true for $\psi_{\mathcal{S}(\mathcal{N})}(t_\beta)$.

Recall that we may write $t_{\psi(\beta)}$ for $\psi_{\mathbb{T}}(t_\beta)$. We now show that $t^-_{\psi(\beta)}(p) = \psi_{ms(d(p))}(t^-_\beta(p))$ and $t^+_{\psi(\beta)}(p) = \psi_{ms(d(p))}(t^+_\beta(p))$ for all places $p \in P$. There are two different cases. Case 1. If $\langle p,t\rangle \notin F$, then $t^-_{\psi(\beta)}(p) = \emptyset = t^-_\beta(p) = \psi_{ms(d(p))}(t^-_\beta(p))$ because $\psi_{ms(d(p))}$ maps empty multi-sets to empty multi-sets (Lemma 4.1). Similarly, if $\langle t,p\rangle \notin F$, then $t^+_{\psi(\beta)}(p) = \psi_{ms(d(p))}(t^+_\beta(p))$. Case 2. If $\langle p,t\rangle \in F$, then $t^-_{\psi(\beta)}(p) = eval_{\psi_{\mathrm{Ass}(v(t))}(\beta)}(a(p,t)) = \psi_{ms(d(p))}(eval_\beta(a(p,t)))$ by the definition of compatible terms (Def. 4.6). On the other hand, $t^-_\beta(p) = eval_\beta(a(p,t))$ and thus $t^-_{\psi(\beta)}(p) = \psi_{ms(d(p))}(t^-_\beta(p))$. In the same way, we can show that if $\langle t,p\rangle \in F$, then $t^+_{\psi(\beta)}(p) = \psi_{ms(d(p))}(t^+_\beta(p))$.

For markings we have $\big(\psi_{\mathcal{S}(\mathcal{N})}(M)\big)(p) = \psi_{ms(d(p))}(M(p))$ by Def. 4.4. Therefore $M(p) \geq t^-_\beta(p)$ iff $\psi_{ms(d(p))}(M(p)) \geq \psi_{ms(d(p))}(t^-_\beta(p))$ iff $\psi_{\mathcal{S}(\mathcal{N})}(M)(p) \geq t^-_{\psi(\beta)}(p)$ by Lemma 4.1 and the result above. Combined with the result for guards, we see that t_β is enabled in M_1 iff $\psi_{\mathcal{S}(\mathcal{N})}(t_\beta)$ is enabled in $\psi_{\mathcal{S}(\mathcal{N})}(M_1)$.

Finally, assume that t_β is enabled in M_1 and $M_1\,[t_\beta\rangle\, M_2$. Then $\psi_{\mathcal{S}(\mathcal{N})}$ is enabled in $\psi_{\mathcal{S}(\mathcal{N})}(M_1)$. Using the results above for $t^-_{\psi(\beta)}(p)$ and $t^+_{\psi(\beta)}(p)$, we have: $\big(\psi_{\mathcal{S}(\mathcal{N})}(M_1)\big)(p) - t^-_{\psi(\beta)}(p) + t^+_{\psi(\beta)}(p) = \psi_{ms(d(p))}(M_1(p)) - \psi_{ms(d(p))}(t^-_\beta(p)) + \psi_{ms(d(p))}(t^+_\beta(p))$, which by linearity of $\psi_{ms(d(p))}$ (Lemma 4.1) and by the definition of M_2 equals to $\psi_{ms(d(p))}(M_1(p) - t^-_\beta(p) + t^+_\beta(p)) = \psi_{ms(d(p))}(M_2(p)) = \psi_{\mathcal{S}(\mathcal{N})}(M_2)(p)$. □

4.3. Approximating Term Compatibility

In practice, when building analyzers for algebraic system nets, deciding whether a possibly long and complex term is compatible with a domain permutation may be problematic. Luckily, we can address this problem by approximating the term compatibility. We define a lower-level compatibility, between operations and domain permutations, and then show that if all the operations appearing in a term are compatible, then the term is compatible, too.

Definition 4.9. (Compatible Operations) We say that an operation $\omega \in \Omega_{s_1\ldots s_n,s}$ and a domain permutation ψ_S are *compatible* if and only if for all $v_1 \in D_{s_1}, \ldots, v_n \in D_{s_n}$ it holds that $\omega_{\mathcal{A}}(\psi_{s_1}(v_1), \ldots, \psi_{s_n}(v_n)) = \psi_s(\omega_{\mathcal{A}}(v_1, \ldots, v_n))$. An operation ω is compatible with a domain permutation group Ψ_S iff it is compatible with all $\psi_S \in \Psi_S$.

Hence, an operation is compatible with a domain permutation if for a permuted input, it will give a similarly permuted output. An immediate consequence of the definition is that a domain permutation compatible with a constant ω (recall that constants are operations) cannot permute its value to other values because $\omega_{\mathcal{A}}() = \psi_s(\omega_{\mathcal{A}}())$ must hold. On the other hand, a positive note is that the operations $=, \neq \in \Omega_{s.s,\mathsf{Bool}}$ with their usual interpretations are compatible with all well-formed domain permutations for all sorts $s \in S$. So is the operation $\mathit{ifThenElse} \in \Omega_{\mathsf{Bool}.s.s,s}$ returning the value of the second (third) argument if the first evaluates to true (false).

We can now prove that terms composed only of variables and compatible operations are compatible.

Lemma 4.3. *Let $\psi_S = (\psi_s)_{s \in S}$ be a domain permutation. Let $T \in \mathbf{T}_s^{SIG}(X)$ be a term such that the operations appearing in it are compatible with ψ_S. Then $eval_{\psi_{\mathrm{Ass}(X)}(\beta)}(T) = \psi_s(eval_\beta(T))$ holds for all assignments β on X, i.e. T is compatible with ψ_S.*

Proof:
By induction on the structure of the term. Induction base: (i) If $T = \omega \in \Omega_{\varepsilon,s}$, then $eval_\beta(T) = \omega_{\mathcal{A}}()$ and $eval_{\psi_{\mathrm{Ass}(X)}(\beta)}(T) = \omega_{\mathcal{A}}()$. But by the compatibility of T and ψ_S we have $\omega_{\mathcal{A}}() = \psi_s(\omega_{\mathcal{A}}())$. (ii) If $T = x \in X_s$, then $eval_\beta(T) = \beta(T)$ and $eval_{\psi_{\mathrm{Ass}(X)}(\beta)}(T) = \psi_{\mathrm{Ass}(X)}(\beta)(T) = \psi_s(\beta(T))$. Induction hypothesis: Let $T_i \in \mathbf{T}_{s_i}^{SIG}(X)$, $1 \leq i \leq n$, be terms compatible with ψ_S such that $eval_{\psi_{\mathrm{Ass}(X)}(\beta)}(T_i) = \psi_{s_i}(eval_\beta(T_i))$. Induction step: Let $T = \omega(T_1, \ldots, T_n) \in \mathbf{T}_s^{SIG}(X)$ and $\omega \in \Omega_{s_1 \ldots s_n, s}$. Now $eval_\beta(T) = \omega_{\mathcal{A}}(eval_\beta(T_1), \ldots, eval_\beta(T_n))$. On the other hand, $eval_{\psi_{\mathrm{Ass}(X)}(\beta)}(T) = \omega_{\mathcal{A}}(eval_{\psi_{\mathrm{Ass}(X)}(\beta)}(T_1), \ldots, eval_{\psi_{\mathrm{Ass}(X)}(\beta)}(T_n))$ equals by the induction hypothesis to $\omega_{\mathcal{A}}(\psi_{s_1}(eval_\beta(T_1)), \ldots, \psi_{s_n}(eval_\beta(T_n)))$. Because ω is compatible with ψ_S, $\omega_{\mathcal{A}}(\psi_{s_1}(eval_\beta(T_1)), \ldots, \psi_{s_n}(eval_\beta(T_n))) = \psi_s(\omega_{\mathcal{A}}(eval_\beta(T_1), \ldots, eval_\beta(T_n)))$ and thus $eval_{\psi_{\mathrm{Ass}(X)}(\beta)}(T) = \psi_s(eval_\beta(T))$. □

Therefore, the requirement that the operations appearing in a term are compatible is a sufficient condition for the term to be compatible. To show that it is not a necessary one, consider the term $\mathsf{cld}_1 = \mathsf{cld}_3$ of the sort Cld described in Ex. 4.2. The constant operations, cld_1 and cld_3, appearing in the term are not compatible with any well-formed domain permutation permuting the elements cld_1 and cld_3 of D_{Cld}. But the term itself is compatible with all well-formed domain permutations. This is because it always evaluates to *false* regardless of the permutation applied to Cld.

Operations Concerning Multi-Set Sorts. Operations on multi-set sorts are of great importance for us because the arc annotation terms are of multi-set sorts. Recollect that the permutations of multi-set sorts are dictated in a systematic way by the permutations of the corresponding ground sorts (Def. 4.2). We may thus investigate the operations concerning multi-set sorts without any knowledge of the corresponding ground sorts. In the following we study some fundamental operations on multi-set sorts with respect to an arbitrary domain permutation ψ_S for the multi-set algebra $\mathcal{A}$. Let $s \in S$ be a ground sort and $ms(s)$ its corresponding multi-set sort.

1. The multi-set addition operation $+ \in \Omega_{ms(s).ms(s),ms(s)}$ such that $+_{\mathcal{A}}(b_1, b_2) = b_1 + b_2$ is compatible because $+_{\mathcal{A}}(\psi_{ms(s)}(b_1), \psi_{ms(s)}(b_2)) = \psi_{ms(s)}(b_1) + \psi_{ms(s)}(b_2) = \psi_{ms(s)}(b_1 + b_2)$ by Lemma 4.1
2. $\times \in \Omega_{\mathsf{Nat}.ms(s),ms(s)}$, defined as $\times_{\mathcal{A}}(n, b) = n \times b$, is compatible provided that n is a constant that is not permuted (by Lemma 4.1)
3. $=, \leq \in \Omega_{ms(s).ms(s),\mathsf{Bool}}$, the usual multi-set comparison operations, are by Lemma 4.1 compatible as long as the domain of the Booleans is not permuted. This means that they are compatible with all well-formed domain permutations

4. The operation $\cdot \in \Omega_{s,ms(s)}$ (the element itself) such that $v_{\mathcal{A}} = 1 \times v$ for all $v \in D_s$, is compatible because we have $(\psi_s(v))_{\mathcal{A}} = 1 \times \psi_s(v) = \psi_{ms(s)}(1 \times v)$ by Def. 4.2. We may therefore abuse the notation and use each term of the ground sort s as a term of multi-set sort $ms(s)$
5. The operation $all \in \Omega_{ms(s)}$ returning the whole domain of s, $all_{\mathcal{A}}() = \Sigma_{v \in D_s} 1 \times v$, is compatible. This is because each domain permutation ψ_s appearing in any $\psi_S \in \Psi_S$ is a bijection on D_s and thus $\psi_{ms(s)}(\Sigma_{v \in D_s} 1 \times v) = \Sigma_{v \in D_s} 1 \times v$

Moreover, consider an element v of D_s and a domain permutation group Ψ_s for s. The Ψ_s-*orbit of* v is a set of elements of D_s defined by $[v]_{\Psi_s} = \{v' \in D_s \mid (\exists \psi_s \in \Psi_s)(v' = \psi_s(v))\}$. Notice that the orbit elements cannot be permuted out of the orbit by any permutation in Ψ_s. Thus, the constant operation $[v] \in \Omega_{ms(s)}$ returning the multi-set consisting of the orbit of v, i.e. $[v]_{\mathcal{A}}() = \Sigma_{v' \in [v]_{\Psi_s}} 1 \times v'$, is compatible because of the way that the permutations for multi-set sorts are defined in Def. 4.2. Thus, if the domain of a ground sort is divided into "static sub-classes" as is done in [3], we can take and use each sub-class as a multi-set similarly their approach.

Example 4.3. Consider the data type of lists, not available in Well-Formed nets [3] nor in the approach of Ip and Dill [17]. Assume that we have a ground sort s and the corresponding list sort $\mathsf{List}(s)$ with the domain $D_{\mathsf{List}(s)} = D_s^*$. For all domain permutations, we define that $\psi_{\mathsf{List}(s)}(l) = \langle \psi_s(v), \ldots, \psi_s(v_n) \rangle$ for all $l = \langle v_1, \ldots, v_n \rangle \in D_s^*$, where ψ_s is the domain permutation for the sort s.

Let us consider the compatibility of some usual operations we would like to apply to lists. The operation $empty \in \Omega_{\mathsf{List}(s),\mathsf{Bool}}$ tells whether a list is empty. The operation is clearly compatible with all well-formed domain permutations because a list is permuted to an empty list if and only if it is empty. The operations $first, last \in \Omega_{\mathsf{List}(s),s}$ return the first (the last) element. First consider the case when $l = \langle v_1, v_2, \ldots, v_n \rangle$ is a list and $n \geq 1$. Now $first_{\mathcal{A}}(l) = v_1$ and $first_{\mathcal{A}}(\psi_{\mathsf{List}(s)}(l)) = first_{\mathcal{A}}(\langle \psi_s(v_1), \psi_s(v_2), \ldots, \psi_s(v_n) \rangle) = \psi_s(v_1)$ for all domain permutations. The case is similar for *last* and thus the operations would be compatible with all domain permutations. But since an empty list (the case when $n = 0$) is permuted to an empty list and there is nothing to return in either case, the operations must return a value that is not permuted, e.g. a special error value. To sum up, the operations are compatible with all domain permutations in which the value returned for an empty list is not permuted. The delete operations $removeFirst, removeLast \in \Omega_{\mathsf{List}(s),\mathsf{List}(s)}$ are compatible with all domain permutations if they return the empty list for the empty list, that is, $removeFirst_{\mathcal{A}}(\langle\rangle) = \langle\rangle$. On the other hand, sometimes it could be desirable to return a special, non-permuted error value as in the case of *first* and *last*. The operations $addFirst, addLast \in \Omega_{\mathsf{List}(s).s,\mathsf{List}(s)}$ appending the list in the beginning/end, are compatible with all domain permutations. So is the list concatenation operation $conc \in \Omega_{\mathsf{List}(s).\mathsf{List}(s),\mathsf{List}(s)}$. The operations $length \in \Omega_{\mathsf{List}(s),\mathsf{Nat}}$ and $isInList \in \Omega_{\mathsf{List}(s).s,\mathsf{Bool}}$ with the obvious interpretations are compatible with all well-formed domain permutations that do not permute the natural numbers.

Actually, because lists are a generalized case of two important data types, namely stacks and FIFO's, we have covered them, too. To have a FIFO sort, just rename the sort to FIFO and the operations so that *addLast* becomes *send* and *getFirst* is *receive*. Similarly, for the sort Stack, substitute *push* for *addFirst* and *pop* for *getFirst*.

As an example of the use of lists, consider the client/server example 4.2. We could have the place *s_wlockreq* with the domain List(CId) for storing the identifiers of clients that have requested a write lock from the server. The server then grants the lock in turns by reading the first element of the list. This is compatible, for instance, with a domain permutation ψ_S swapping the identifiers of clients number 1 and 3, in which case a list $l = \langle cId_1, cId_3 \rangle$ is permuted to $\psi_{\mathsf{List(CId)}}(l) = \langle cId_3, cId_1 \rangle$.

A same kind of analysis could be carried out for other similar data types such as structures, sets, arrays and unions.

5. Conclusions

In this article, we have defined how permutations applied to the domains of the data types used in an algebraic system net produce permutaations to the state space of the net. To ensure that the produced state space permutations are symmetries, i.e. automorphisms of the state space, we have proposed a sufficient compatibility condition between the data domain permutations and the terms appearing in the transitions of the net. We have also given an approximation rule for the compatibility condition to help the process of deciding term compatibility. We have also briefly illustrated the use of our approach and the approximation rule by considering the data type of lists, which had not been studied in the context of automatic symmetry deduction before. By performing a similar kind of analysis for other data types, we can classify the operations that can be applied to them as compatible or incompatible. After this kind of human-performed classification (which is performed only once for a data type), it is quite simple for a tool to check whether the operations used in a net are compatible with the used domain permutations.

There exists also an interesting, not yet researched scenario for combining high-level symmetry detection with the low-level approach. It arises in cases when the high-level formalism, like the one of this paper, supports symmetry detection and a high-level net is also unfoldable into the low-level net in practice. Then we could first find the symmetries from the high-level net and then unfold the net into the low-level net, and also "unfold" the already found symmetry information to the low-level net analyzer. The low-level net analyzer would not have to find these symmetries again, but could use them directly or as a basis for finding the rest of the symmetries that were possibly not found by the high-level analysis.

References

[1] K. Ajami, S. Haddad, and J-M. Ilié. Exploiting symmetry in linear time temporal logic model checking: One step beyond. In B. Steffen, editor, *Tools and Algorithms for the Construction and Analysis of Systems; 4th International Conference, TACAS'98, Lisbon,*

Portugal, March/April 1998, volume 1384 of *Lecture Notes in Computer Science*, pages 52–67. Springer-Verlag, Berlin, Germany, 1998.

[2] J. Billington. Many-sorted high-level nets. In *Proceedings of the Third International Workshop on Petri Nets and Performance Models (PNPM89)*, pages 166–179, Kyoto, Japan, 11–13 December 1989. IEEE Computer Society Press. Reprinted in [20], pages 123–136.

[3] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well-formed coloured nets and their symbolic reachability graph. In Jensen and Rozenberg [20], pages 373–396.

[4] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A symbolic reachability graph for coloured Petri nets. *Theoretical Computer Science*, 176:39–65, 1997.

[5] E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla. Symmetry reductions in model checking. In Hu and Vardi [13], pages 147–158.

[6] E. M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2):77–104, August 1996.

[7] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, August 1996.

[8] E. A. Emerson and A. P. Sistla. Utilizing symmetry when model checking under fairness assumptions: An automata-theoretic approach. *ACM Transactions on Programming Languages and Systems*, 19(4):617–638, July 1997.

[9] H. J. Genrich. Predicate/transition nets. In Jensen and Rozenberg [20], pages 3–43.

[10] R. P. Grimaldi. *Discrete and Combinatorial Mathematics.* Addison-Wesley Publishing Company, Reading, Massachusetts, USA, 1985.

[11] O. Grumberg, editor. *Computer Aided Verification: 9th International Conference, CAV'97, Haifa, Israel, June 1997*, volume 1254 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1997.

[12] V. Gyuris and A. P. Sistla. On-the-fly model checking under fairness that exploits symmetry. In Grumberg [11], pages 232–243.

[13] A. J. Hu and M. Y. Vardi, editors. *Computer Aided Verification: 10th International Conference, CAV'98, Vancouver, BC, Canada, June/July 1998*, volume 1427 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1998.

[14] P. Huber, A. M. Jensen, L. O. Jepsen, and K. Jensen. Towards reachability trees for high-level Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets 1984*, volume 188 of *Lecture Notes in Computer Science*, pages 215–233. Springer-Verlag, Berlin, Germany, 1985.

[15] P. Huber, A. M. Jensen, L. O. Jepsen, and K. Jensen. Reachability trees for high-level Petri nets. In Jensen and Rozenberg [20], pages 319–350.

[16] International Organization for Standardization. *Committee Draft ISO/IEC 15909: High-level Petri Nets – Concepts, Definitions and Graphical Notation*, October 2, 1997. Version 3.4.

[17] C. N. Ip and D. L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1/2):41–76, August 1996.

[18] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use: Volume 2, Analysis Methods.* Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1995.

[19] K. Jensen. Condensed state spaces for symmetrical coloured Petri nets. *Formal Methods in System Design*, 9(1/2):7–40, August 1996.

[20] K. Jensen and G. Rozenberg, editors. *High-level Petri Nets; Theory and Application.* Springer-Verlag, Berlin, Germany, 1991.

[21] T. A. Junttila. Towards well-formed algebraic system nets. In H.-D. Burkhard, L. Czaja, and P. Starke, editors, *Workshop Concurrency, Specification & Programming*, number 110 in Informatik-Bericht, pages 116–127. Humboldt-Universität zu Berlin, 1998.

[22] E. Kindler and W. Reisig. Algebraic system nets for modelling distributed algorithms. *Petri Net Newsletter*, 51:16–31, December 1996.

[23] E. Kindler and H. Völzer. Flexibility in algebraic nets. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets 1998; Proceedings of the 19th International Conference, ICATPN'98; Lisbon, Portugal, June 1998*, volume 1420 of *Lecture Notes in Computer Science*, pages 345–364. Springer-Verlag, Berlin, Germany, 1998.

[24] G. S. Manku, R. Hojati, and R. Brayton. Structural symmetry and model checking. In Hu and Vardi [13], pages 159–171.

[25] M. Pandey and R. E. Bryant. Exploiting symmetry when verifying transistor-level circuits by symbolic trajectory evaluation. In Grumberg [11], pages 244–255.

[26] W. Reisig. Petri nets and algebraic specifications. *Theoretical Computer Science*, 80(1):1–34, March 1991. Reprinted in [20], pages 137–170.

[27] K. Schmidt. How to calculate symmetries of Petri nets. Technical Report MATH-AL-8-1997, Technische Universität Dresden, Germany, September 1997. Revised version accepted for publication in Acta Informatica.

[28] K. Schmidt and P. H. Starke. An algorithm to compute the symmetries of Petri nets. *Petri Net Newsletter*, 40:25–30, December 1991.

[29] P. H. Starke. Reachability analysis of Petri nets using symmetries. *Journal of Mathematical Modelling and Simulation in System Analysis*, 8(4/5):293–303, 1991.