Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 57
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 57
Espoo 1999 HUT-TCS-A57

DETECTING AND EXPLOITING DATA TYPE SYMMETRIES OF ALGEBRAIC SYSTEM NETS DURING REACHABILITY ANALYSIS

Tommi Junttila

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 57
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 57
Espoo 1999 HUT-TCS-A57

DETECTING AND EXPLOITING DATA TYPE SYMMETRIES OF ALGEBRAIC SYSTEM NETS DURING REACHABILITY ANALYSIS

Tommi Junttila

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietojenkäsittelyteorian laboratorio

Distribution:
Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O.Box 5400
FIN-02015 HUT
Tel. +358-0-451 1
Fax. +358-0-451 3369
E-mail: lab@tcs.hut.fi

ISBN 951-22-4885-9
ISSN 0783-5396

Picaset Oy
Helsinki 1999

ABSTRACT: The symmetry reduction method is a technique designed to alleviate the combinatorial state space explosion problem by exploiting the symmetries of state spaces. This work describes a way how state space symmetries of a high-level Petri net formalism, algebraic system nets, can be detected and exploited during the reachability analysis. The main idea is that permuting the domains of data types used in nets produces corresponding permutations to the state space level.

As the main result a sufficient condition is defined for data type domain permutations ensuring that the produced state space permutations are actually automorphisms i.e. symmetries. Since verification of the condition is shown to be a **co-NP**-complete problem, an approximation rule for the condition is also given. Use of the developed theory is illustrated by defining the class of extended well-formed nets and by examining data type symmetries of such nets. It is shown that checking the symmetricity of two markings of a net in this class is equivalent to checking whether two graphs are isomorphic.

KEYWORDS: Symmetry, reachability analysis, Petri nets, algebraic system nets.

Contents

1 INTRODUCTION

Verification of distributed and concurrent systems such as telecommunication protocols is an important task since such systems may be extremely complex and used in places where reliable error-free operation is highly desirable. One verification method is the state space based one in which the behaviors of (the model of) the system in question are mechanically enumerated, i.e. the reachability graph of the system is generated. Desired properties such as deadlock freedom are then checked automatically from the reachability graph. However, the approach suffers from the state space explosion problem which in essence means that the size of the reachability graph can be exponential with respect to the size of (the model of) the system. Several techniques have been proposed to alleviate this problem, see [Valmari 1998] for an overview.

The symmetry reduction method is one such technique exploiting the automorphisms (read: symmetries) of system's state space. State space symmetries are a result of regularities appearing in a system: for example, multiple instances of a same process type usually produce automorphisms to the state space. The main idea of the method is that only one of the symmetric states has to be explored during the reachability analysis. To see the benefits of this, consider a system consisting of n identical processes. Since processes are identical, global system states obtained by permuting the processes can be considered symmetric and therefore a global state can have up to $n!$ symmetric global states. Considering only one of these states during the reachability analysis can thus lead to exponential savings in the space required. It is very common that this kind of savings actually realize in practice, too. In the symmetry reduction method we thus generate a (often much smaller) reduced reachability graph, called a quotient reachability graph (QRG), instead of the original reachability graph. As a QRG is bisimilar to the original unreduced reachability graph, many interesting properties can be checked directly by using QRGs. By adding some additional information in the arcs of a QRG, the resulting annotated QRG will contain all the information of the original reachability graph. Therefore, arbitrary properties can be checked by using annotated QRGs although this may require (partial) unwinding of the annotated QRG.

However, since the goal is to avoid the generation of the full reachability graph (and enumeration of the state space), the information concerning its automorphisms has to be obtained some way other than direct calculation. But as the system description and the semantics of the description formalisms describe system's state space, they also contain information about its automorphisms. Thus the automorphism information has to be extracted from the system description. What this information is depends of course on the system description formalism. For instance, in the case of Place/Transition nets, the automorphisms of the net itself produce automorphisms to the state space, too, and is the desired information.

In this work we consider the problem of finding symmetry information from systems described in a high-level Petri net formalism called algebraic system nets [Reisig 1991; Kindler and Reisig 1996; Kindler and Völzer 1998].

Algebraic system nets are of interest because (i) use of algebras (with signatures) allows us to define both syntax and semantics of net inscriptions and (ii) definition of a class of algebras leads to a definition of a net class. However, algebraic specification techniques (in which algebras are defined by specifying constraint equations) are not considered since this would add a computationally hard extra layer to the approach; we assume that the algebras are described explicitly. Naturally we could also have used a similar net class called many-sorted nets [Billington 1989; International Organization for Standardization 1997].

Since algebraic system nets are high-level nets, our assumption is that nets are well-folded, i.e. similar parts of a net are folded together by use of data. Therefore the approach applicable for low-level Place/Transition nets taking advantage of the structural symmetries of a net [Starke 1991; Schmidt and Starke 1991; Schmidt 1997; 1999] is probably not very efficient for algebraic system nets. (We could unfold an algebraic system net into a low-level net but usually this unfolding is far too large to be manageable.) Since the similar parts are folded together by use of data, the symmetries are instead hidden in the way the data is used in algebraic system nets. We therefore try to find state space symmetries of algebraic system nets by permuting the domains of data types used in nets. These domain permutations are gradually extended to the state space level, producing state space permutations but not necessarily state space automorphisms. To ensure that the produced state space permutations are in fact symmetries, the data type domain permutations and the algebraic terms manipulating data in transitions have to be mutually compatible. In this work we define a general compatibility condition between domain permutations and terms of an algebra and analyze its complexity. Since it is shown to be a **co-NP**-complete problem to decide whether a domain permutation and a term are compatible, we will also present an approximation rule for compatibility that is based on lower-level compatibility between domain permutations and individual operations comprising a term.

We will illustrate the use of the developed theory by defining a net class called extended well-formed nets (EWFNs). (This name was chosen because the definition was inspired by that of well-formed nets [Chiola et al. 1991]. Another source of inspiration was the similar approach of Ip and Dill [1996].) In extended well-formed nets the basic data types, both predefined and user-definable, such as integers, Booleans, process identifiers etc. are classified according to the domain permutations and operations allowed on them. More complex data types are then build on the basic types by standard constructions such as lists, sets and arrays. We will show the compatibility of the standard operations on the data types w.r.t. to the domain permutations extended in a natural way from those allowed for basic types. A simple input language is sketched and we will also show in a constructive way that an important problem encountered in the generation of quotient reachability graphs, namely that of deciding the symmetricity of two markings of an EWF-net, is graph isomorphism-complete under polynomial-time many-one reductions. We also modify the very expressive temporal logic CTL* to EWF-nets.

1.1 RELATED WORK

The history of the symmetry reduction method dates back at least to work by Huber et al. [1985], where symmetries for colored Petri nets (CPNs) were introduced. The approach has since been further developed and applied at least in [Huber et al. 1991; Jensen 1995; 1996; Jørgensen and Kristensen 1999]. The fundamental difference between the approach in this work and that for CPNs is that in CPNs the exact syntax of the net inscriptions is not defined. This means that the net formalism itself does not provide much help for designing tools for automatic detection of symmetry information from a CPN. Thus it is the user that has to provide the symmetry information to the tool and also to check the correctness of the information. Furthermore, it is usually the user's task also to write the functions needed in the generation of QRGs; at least those that test whether two markings of the CPN are equivalent under the given symmetry group. All of these tasks can be a very hard and error prone.

Starke [1991] introduced symmetries for Place/Transition-nets, showing that symmetries of a P/T-net produce corresponding symmetries into the state space of the net. Basic properties between the reduced reachability graph and the original were shown and an algorithm for computing symmetries of a net was devised. This approach has been further studied in a series of articles, [Schmidt and Starke 1991; Schmidt 1997; 1999], where the symmetry calculation algorithm and generation of reduced reachability graph have been further improved. Advantages of this approach are that the symmetry detection and exploitation can be fully automatized. The main drawback is that the method is applicable primarily only to low-level nets.

Well-Formed Nets (WFNs) [Chiola et al. 1991] are a dialect of CPNs in which the data types and operations applicable on them are defined in a fixed way. This means that to some data types only certain restricted operations are applicable. Thus certain domain permutations on these data types are automatically compatible with the applicable operations and produce state space symmetries. Therefore some of the symmetries are immediately clear from the type definitions of a net description. Automatic tools can thus be written that check whether the syntax conforms to WFN-definitions. Since operations and data types are well-defined, tools can automatically calculate the functions needed in the generation of quotient reachability graphs. The price is that the syntax is fixed in the definition level and is thus very hard to change. In this work we describe an intermediate approach, in which the syntax is not fixed. Nevertheless, by using signatures and algebras, the syntax and semantics of data types and operations have a formal background. We are therefore able to give generalized sufficient conditions for compatibility between data type domain permutations and operations and to show that domain permutations compatible with used operations produce state space symmetries. When a new net description syntax (a new class of nets) is defined, it is sufficient to evaluate the compatibility of operations w.r.t. to domain permutations only once in the design phase.

Ip and Dill [Ip and Dill 1993; Ip 1996; Ip and Dill 1996] deal with the problem of finding symmetries in a fashion similar to WFNs. They introduce special data types called "scalar sets" for which only restricted operations can be applied.

Pandey [1997] and Pandey and Bryant [1997; 1999] discuss exploitation of symmetries in the context of verifying transistor-level circuits by symbolic trajectory evaluation.

The properties that one is able to check from reduced reachability graphs range from simple deadlock absence, safety and and Petri net specific properties [Chiola et al. 1991; 1997; Huber et al. 1991; Jensen 1995; Ip and Dill 1993; Ip 1996; Ip and Dill 1996] via model checking of a restricted form of temporal logic [Pandey and Bryant 1997; 1999] to full-fledged model checking. The foundations for model checking of temporal logic properties under symmetries were laid in 1993 by Clarke et al. [1993] and Emerson and Sistla [1993]. The topic has been further studied in [Clarke and Jha 1995; Emerson and Sistla 1995; Clarke et al. 1996; Emerson and Sistla 1996; 1997; Gyuris and Sistla 1997; Ajami et al. 1998; Clarke et al. 1998]. Furthermore, combining partial orders (another method for alleviating the state space explosion problem) and symmetries is studied by Emerson et al. [1997] and partial symmetries are introduced in [Haddad et al. 1995; Emerson and Trefler 1999].

1.2 OUTLINE OF THE THESIS

We begin by a brief introduction to the symmetry reduction method. In subsequent chapters we introduce the concepts of the chosen Petri net formalism, algebraic system nets. Chapter 3 first introduces the basic constructs of algebras and then defines how the domains of an algebra may be permuted and what does it mean for a term to be compatible with a domain permutation. Computational complexity issues concerning term compatibility are also discussed and an approximation rule for deciding term compatibility is given. Chapter 4 combines algebras with Petri nets and gives formal definition for the dialect of algebraic system nets we will use. Domain permutations defined in Chapter 3 are then extended to operate on the state space level of algebraic system nets in Chapter 5. To ensure that the produced state space permutations are in fact automorphisms, a sufficient condition stating that terms appearing as transitions guards and arc annotations should be compatible with used domain permutations is also given and proven in Chapter 5. To sum up and demonstrate the concepts of Chapters 3–5, Chapter 6 gives an illustrative net class called Extended Well-Formed Nets in which data types are divided into different classes according to the applicable operations. The chosen data types include not only some basic types but we also consider structured data types, such as lists and sets, found in many common high-level programming languages. We also describe what kind of data type domain permutations can be applied and what operations are compatible with them. Chapter 7 finally concludes the work.

Some parts of this work have been published before in [Junttila 1998; 1999].

1.3 CONTRIBUTIONS

The main contribution of this work is the definition of how data type domain permutations of an algebra produce symmetries to the state space of an algebraic system net using the algebra. A sufficient condition called term compatibility ensuring this is defined and proven. Computational complexity of term compatibility is also classified: it is shown to a **co-NP**-complete problem to decide whether a term and a data type domain permutation are compatible (under reasonable assumptions). Because of this result, an approximation rule is also given and proven.

Applicability of the developed theory is demonstrated by developing a net class similar to well-formed nets but having more complex data types. It is shown in a constructive way that comparing symmetricity of two markings of a net belonging to this net class is a graph isomorphism complete problem. The very expressive temporal logic CTL* is also modified for the net class.

A minor novelty is how error situations are handled in algebraic system nets by using a special form of error algebras.

2 THE SYMMETRY REDUCTION METHOD

The symmetry reduction method exploits the property present in some state spaces, namely that some states may be considered as equivalent (or symmetric) in the sense that they give rise to symmetric behavior. 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 machines in a client/server system). The basic idea of the method is that only one state in each set of symmetric states has to be examined during the reachability analysis and thus a (usually much) smaller reduced reachability graph can be generated and checked instead of the original reachability graph. Note that since the whole point of the symmetry reduction method is to avoid the enumeration of the state space/generation of the reachability graph, the state space symmetry information must be found in the system description that produces the state space prior to the reachability analysis. In general, the symmetry reduction method consists of three phases:

1. Finding some sort of system description level information that generates state space symmetries.
2. Exploiting the symmetry information during the reachability analysis by generating a reduced quotient reachability graph.
3. Checking the desired property from the quotient reachability graph.

In this chapter we first define the concept of state space symmetries by using Labeled Transition Systems (LTSs) as the formalism for state spaces. We formalize the relationship between system description level symmetry information and state space symmetries in Section 2.1.2. The generation of reduced reachability graphs and the connection to strong bisimulations is discussed in Section 2.2. Checking some simple properties on quotient reachability graphs is the topic of Section 2.2.3. Section 2.3 illustrates the symmetry reduction method by discussing symmetries of Place/Transition nets. The presentation in this chapter is by no means complete as, for instance, advanced model checking algorithms are not discussed, and the reader should consult the papers mentioned in Section 1.1 for more information.

2.1 STATE SPACES

Assume that we have a system description given, for instance, as a P/T-net or as a high-level Petri net. The semantics of the description formalism define the *state space* of the system, which in the case of Petri nets is the set of all markings and transition occurrences between them. We express states spaces by means of labelled transition systems. Formally, a *labelled transition system* (an LTS) is a tuple $\mathcal{L} = \langle Q, L, \Delta, q_{init} \rangle$, where Q is a non-empty set of *states*, L is a non-empty set of *transition labels* such that $Q \cap L = \emptyset$, $\Delta \subseteq Q \times L \times Q$

is the *transition relation* and $q_{init} \in Q$ is the *initial state*. For each $l \in L$ we define the relation $\xrightarrow[\mathcal{L}]{l}$ by $q \xrightarrow[\mathcal{L}]{l} q'$ iff $\langle q, l, q' \rangle \in \Delta$. The relation $\underset{\mathcal{L}}{\Longrightarrow}$ is the reflexive and transitive closure of $\bigcup_{l \in L} \xrightarrow[\mathcal{L}]{l}$. A (finite or infinite) sequence $\sigma = q_0 \xrightarrow[\mathcal{L}]{l_0} q_1 \xrightarrow[\mathcal{L}]{l_1} \ldots$, where $q_i \in Q$ and $l_i \in L$, is called a *path* starting from q_0 in $\mathcal{L}$. The *length* of σ is the number of states involved in σ (∞ in case of an infinite sequence).

The *reachability graph* of $\mathcal{L}$ is the LTS $\mathrm{RG}(\mathcal{L}) = \langle \vec{Q}, L, \vec{\Delta}, q_{init} \rangle$ such that $\vec{Q} = \{q \in Q \mid q_{init} \underset{\mathcal{L}}{\Longrightarrow} q\}$ and $\vec{\Delta} = \Delta \cap (\vec{Q} \times L \times \vec{Q})$. Therefore, in this work, a state space corresponds to all the possible states of a system, reachable or not, whereas the reachability graph is exactly the set of reachable states. An alternative, equivalent, way to define reachability graphs is to define them inductively — this makes some proofs easier. Formally:

Definition 2.1.1 (Reachability Graphs) *The* reachability graph *of $\mathcal{L}$ is the LTS $\mathrm{RG}(\mathcal{L}) = \langle \vec{Q}, L, \vec{\Delta}, q_{init} \rangle$, where $\vec{Q} \subseteq Q$ and $\Delta \subseteq Q \times L \times Q$ are defined inductively as follows: (i) $q_{init} \in \vec{Q}$, (ii) if $q \in \vec{Q}$ and $q \xrightarrow[\mathcal{L}]{l} q_1$, then $q_1 \in \vec{Q}$ and $\langle q, l, q_1 \rangle \in \vec{\Delta}$ and (iii) nothing else is in $\vec{Q}$ or $\vec{\Delta}$.*

2.1.1 State Space Symmetries

The equivalence classes of states that the symmetry reduction method utilizes are produced by automorphism groups operating on the state space. We first define a *permutation* of an LTS $\mathcal{L} = \langle Q, L, \Delta, q_{init} \rangle$ to be 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* $\mathcal{L}$ is denoted by $\mathrm{Perm}(\mathcal{L})$ and is a group under the function composition operator $\circ$. A sub-group of $\mathrm{Perm}(\mathcal{L})$ is called a *permutation group* of $\mathcal{L}$. Automorphisms are permutations that respect the transition relation:

Definition 2.1.2 (LTS Automorphisms) *A permutation ϕ of $\mathcal{L}$ is an* automorphism *of $\mathcal{L}$ if $\langle q, l, q_1 \rangle \in \Delta \Leftrightarrow \langle \phi(q), \phi(l), \phi(q_1) \rangle \in \Delta$ holds for all $q, q_1 \in Q$ and all $l \in L$. The* set of all automorphisms of $\mathcal{L}$ *is denoted by* $\mathrm{Aut}(\mathcal{L})$.

Note that $\mathrm{Aut}(\mathcal{L})$ is a sub-group of $\mathrm{Perm}(\mathcal{L})$ (there always exists one automorphism, namely the identity mapping). Given two elements $x, x' \in Q \cup L$ and a sub-group Φ of $\mathrm{Perm}(\mathcal{L})$, we say that x is *equivalent to x' under* Φ, denoted by $x \equiv_\Phi x'$, 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*, denoted by $[x]_\Phi$, is the set of all elements equivalent to it: $[x]_\Phi = \{x' \in Q \cup L \mid x \equiv_\Phi x'\}$. In case Φ is a sub-group of $\mathrm{Aut}(\mathcal{L})$, we may also say that x is *symmetric* to x' if $x \equiv_\Phi x'$. An element $x \in Q \cup L$ is *(self-)symmetric* under Φ if $[x]_\Phi = \{x\}$. For a path $\sigma = q_0 \xrightarrow[\mathcal{L}]{l_0} q_1 \xrightarrow[\mathcal{L}]{l_1} \ldots$ of length n in $\mathcal{L}$ and an automorphism $\phi \in \mathrm{Aut}(\mathcal{L})$, the sequence $\phi(\sigma) = \phi(q_0) \xrightarrow[\mathcal{L}]{\phi(l_0)} \phi(q_1) \xrightarrow[\mathcal{L}]{\phi(l_1)} \ldots$ is obviously also a path of length n in $\mathcal{L}$.

Example 2.1.3 Consider the very simple program for the mutual exclusion problem and its state space shown in Figure 2.1. The program consists of two shared state variables s_1 and s_2 with the domain $\{N, T, C\}$, where N denotes the non-critical section, T the trying section and C the critical section. The variables are manipulated by two asynchronous processes whose transitions are given by imposing pre- and postconditions on variables. For instance, transition t_1 of process 1 assigns the variable s_1 to value T in the next state if s_1 was set to N in the current one. Transitions are atomic and $\bar{\imath}$ means $3-i$. The right part of the figure shows the state space LTS $\mathcal{L}$ of the program. The set of states is $Q = \{N, T, C\} \times \{N, T, C\}$, where the first element describes the value of s_1 and the latter the value of s_2. Transition labels are the transitions names $L = \{t_1, e_1, l_1, t_2, e_2, l_2\}$ and the transition relation is depicted in the figure with arcs between states. $\langle N, N\rangle$ is the initial state pointed out by an arrow originating nowhere. The reachability graph of $\mathcal{L}$ is the $\mathcal{L}$ itself minus the state $\langle C, C\rangle$ and the arcs from it. The set of automorphisms of $\mathcal{L}$, $\mathrm{Aut}(\mathcal{L})$, consists of two permutations on $Q \cup L$: the identity mapping and the mapping

$$\begin{pmatrix} \langle N,N\rangle & \langle T,N\rangle & \langle N,T\rangle & \langle C,N\rangle & \langle T,T\rangle & \langle N,C\rangle & \langle C,T\rangle & \langle T,C\rangle & \langle C,C\rangle & t_1 & e_1 & l_1 & t_2 & e_2 & l_2 \\ \langle N,N\rangle & \langle N,T\rangle & \langle T,N\rangle & \langle N,C\rangle & \langle T,T\rangle & \langle C,N\rangle & \langle T,C\rangle & \langle C,T\rangle & \langle C,C\rangle & t_2 & e_2 & l_2 & t_1 & e_1 & l_1 \end{pmatrix}.$$

Note that, intuitively, the latter mapping corresponds to the swapping of process identities. The states $\langle T, N\rangle$ and $\langle N, T\rangle$ are symmetric under $\mathrm{Aut}(\mathcal{L})$ and the orbit of $\langle T, N\rangle$ is $[\langle T, N\rangle]_{\mathrm{Aut}(\mathcal{L})} = \{\langle T, N\rangle, \langle N, T\rangle\}$. ♣

$i \in \{1, 2\}$

Variables:
s_i with domains $\{N, T, C\}$

Transitions:
t_i: $s_i = N$ -> $s'_i = T$;
e_i: $s_i = T \wedge s_{\bar{\imath}} \neq C$ -> $s'_i = C$;
l_i: $s_i = C$ -> $s'_i = N$;

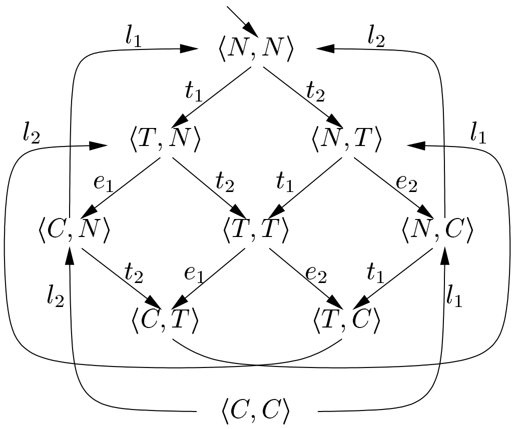

Figure 2.1: A (very simple) mutual exclusion program and its state space.

2.1.2 State Space Automorphisms from System Descriptions

Recall our assumption that a system $\mathfrak{S}$ is given in some system description formalism, such as concurrent programming language or, in this work, some sort of Petri net. The semantics of the description formalism describe the state space, an LTS denoted by $\mathrm{SS}(\mathfrak{S})$, of the system. Reachability analysis of the system $\mathfrak{S}$ is a procedure that builds the LTS $\mathrm{RG}(\mathrm{SS}(\mathfrak{S}))$ without explicitly constructing the state space LTS $\mathrm{SS}(\mathfrak{S})$. This is possible by using the inductive definition of reachability graphs: the system description contains the initial state and the formalism describes how successor states are generated. Because of the state space explosion, we want to exploit the automorphisms of the state space and generate a reduced quotient reachability graph

(QRG) (as described later) without ever generating the reachability graph or enumerating the state space. This means that we cannot compute the symmetries of the state space by using the state space itself but have to find the information that generates state space symmetries in the system description $\mathfrak{S}$. We formalize this as follows.

Definition 2.1.4 *Let Ψ be a permutation group (under an arbitrary binary operator $*$) operating on some elements of the system description $\mathfrak{S}$.* Ψ produces symmetries to the state space *if there is a group homomorphism* $f : \Psi \to \mathrm{Aut}(\mathrm{SS}(\mathfrak{S}))$.

We then know from the standard group theory that for any sub-group Ψ' of Ψ, $f(\Psi')$ is a sub-group of $\mathrm{Aut}(\mathrm{SS}(\mathfrak{S}))$. Furthermore, it is desirable, and in practice usually the case that given a $\psi \in \Psi'$ and a state or transition label x in $\mathrm{SS}(\mathfrak{S})$, the value $(f(\psi))(x)$ can be calculated in deterministic polynomial time.[1] Thus we can compute the $f(\Psi')$-orbit of a state q and compare whether two states, q and q', are symmetric under $f(\Psi')$ — although these computations are in most cases not executable in deterministic polynomial time (under plausible complexity theoretical assumptions) since there are possibly exponentially many symmetric states w.r.t. the length of a sufficiently succinct coding of Ψ'. Since the domain of a permutation $\psi \in \Psi$ is usually much smaller than the domain of the corresponding state space permutation $f(\psi)$, we actually store and handle the permutations in Ψ when building symmetry reduction algorithms rather than those in $f(\Psi)$.

Naturally the exact definitions of Ψ, $*$ and f depend on the system description formalism at hand. For Place/Transition-nets Ψ is a structure preserving permutation group on the places and transitions of a net and $*$ is the function composition operator $\circ$. For algebraic system nets considered in this work, Ψ will be a family of permutations on the data type domains used in a net while $*$ will be a data type-wise extension of $\circ$.

The group homomorphism connection between system description level permutations and state space automorphisms described here has, as far as the author is aware, not been explicitly stated before [Junttila 1999]. However, it has been implicitly used in almost all symmetry related works when describing how the regularities of a system description produce symmetries to the state space.

Example 2.1.5 Let us revisit Example 2.1.3. Now the program in Figure 2.1 is the system $\mathfrak{S}$ and the LTS on the right hand side of the figure is its state space $\mathcal{L} = \mathrm{SS}(\mathfrak{S})$. Take $\Psi = \mathrm{Perm}(\{1,2\}) = \{\psi_1 = \left(\begin{smallmatrix}1&2\\1&2\end{smallmatrix}\right), \psi_2 = \left(\begin{smallmatrix}1&2\\2&1\end{smallmatrix}\right)\}$ and define $*$ to be the function composition operator $\circ$. Define $f : \Psi \to \mathrm{Perm}(\mathrm{SS}(\mathfrak{S}))$ such that $f(\psi) = \phi$ where $\phi(y_i) = y_{\psi(i)}$ for $y \in \{t, e, l\}$ and $\phi(\langle v_1, v_2\rangle) = \langle v_{\psi(1)}, v_{\psi(2)}\rangle$ for $\langle v_1, v_2\rangle \in \{N, T, C\} \times \{N, T, C\}$. Now $(f(\psi_2))(t_1) = t_2$ and $(f(\psi_2))(\langle C, N\rangle) = \langle N, C\rangle$. Clearly f is a group homomorphism from Ψ to $\mathrm{Aut}(\mathrm{SS}(\mathfrak{S}))$ and $f(\Psi) = \mathrm{Aut}(\mathrm{SS}(\mathfrak{S}))$ as described in Example 2.1.3. ♣

[1] Care must be taken in claims that functions are polynomial-time computable. For instance, if mappings are represented as lists of “value, mapped value” pairs, then f itself is in most cases *not* a polynomial-time function since the domain and range of the function $f(\psi)$ can easily be exponentially large (even infinite) w.r.t. the domain of ψ.

2.1.3 Three Approaches to Symmetry Detection

In practice, given a system description in a certain formalism, how do we find out the system description level information that generates state space symmetries? Resembling the discussion by Schmidt [1997], we distinguish three different ways according to their degree of automatization.

Automatic Detection. In this approach there is an algorithm that, given a system description, computes a permutation group on the system description that under a fixed homomorphism type produces state space symmetries. An example of this approach is the symmetry reduction method for Place/Transition-nets [Starke 1991; Schmidt and Starke 1991; Schmidt 1997; 1999], see also Section 2.3. A resembling approach for a hardware description language is described in [Manku 1997; Manku et al. 1998]. The main advantage is obvious: no user interaction is required. So far this approach has been applied to low-level description formalisms only. We will consider some computational complexity issues concerning the automatic detection of symmetries of algebraic system nets (which is a high-level formalism) in Section 3.3.1.

Semi-Automatic (User-Hinted) Detection. This approach, used in [Chiola et al. 1991; Ip and Dill 1996], is based on use of special data types with restricted set of applicable operations. The operations are carefully chosen so that certain permutation types on the data type domains will produce state space symmetries when extended to them. This is a semi-automatic technique since the user is required to use these special data types or no state space symmetries will be exploited. Later in this work (Chapter 6) we describe a high-level net class called extended well-formed nets for which this symmetry detection technique can be used in a way similar to [Chiola et al. 1991; Ip and Dill 1996].

User Detects the Symmetries. [Jensen 1995; Pandey 1997; Pandey and Bryant 1997; 1999] In this approach the user is required to come up with symmetry information and describe it to the analysis tool. In some variants the correctness of the user-provided symmetry generating information is checked automatically [Pandey 1997; Pandey and Bryant 1997; 1999], while in others it is up to user to perform this task, too [Jensen 1995]. Needless to say, this approach requires heavy user intervention.

2.2 QUOTIENT REACHABILITY GRAPHS

We are now ready to define quotient reachability graphs. Basically, a quotient reachability graph is similar to the reachability graph of an LTS except that in the inclusion of successor states we may choose any symmetric state as the successor.

Definition 2.2.1 *Let* Φ *be a sub-group of* $\mathrm{Aut}(\mathcal{L})$. A quotient reachability graph of $\mathcal{L}$ under Φ *is an LTS* $\widetilde{\mathcal{L}} = \langle \widetilde{Q}, L, \widetilde{\Delta}, \widetilde{q}_{init} \rangle$ *such that* $q_{init} \equiv_\Phi \widetilde{q}_{init}$

and $\widetilde{Q} \subseteq Q$, $\widetilde{\Delta} \subseteq \widetilde{Q} \times L \times \widetilde{Q}$ *are defined inductively as follows:*

1. $\widetilde{q}_{init} \in \widetilde{Q}$;
2. *if* $q \in \widetilde{Q}$ *and* $q \xrightarrow[\mathcal{L}]{l} q_1$, *then* $\langle q, l, \widetilde{q}_1 \rangle \in \widetilde{\Delta}$ *and* $\widetilde{q}_1 \in \widetilde{Q}$ *for at least one* $\widetilde{q}_1$ *such that* $q_1 \equiv_\Phi \widetilde{q}_1$*; and*
3. *nothing else is in* $\widetilde{Q}$ or $\widetilde{\Delta}$.

We use $\mathrm{QRGs}(\mathcal{L}, \Phi)$ *to denote the* set of all quotient reachability graphs of $\mathcal{L}$ under Φ.

We say that a QRG $\widetilde{\mathcal{L}} = \langle \widetilde{Q}, L, \widetilde{\Delta}, \widetilde{q}_{init} \rangle \in \mathrm{QRGs}(\mathcal{L}, \Phi)$ is *minimal* (under Φ) if $\widetilde{q} \equiv_\Phi \widetilde{q}'$ implies $\widetilde{q} = \widetilde{q}'$ for all $\widetilde{q}, \widetilde{q}' \in \widetilde{Q}$.

Note that the reachability graph $\mathrm{RG}(\mathcal{L})$ is in $\mathrm{QRGs}(\mathcal{L}, \Phi)$ for any subgroup Φ of $\mathrm{Aut}(\mathcal{L})$. Algorithm 2.2.1 obviously calculates a quotient reachability graph of an LTS.

Algorithm 2.2.1 Calculation of quotient reachability graphs.

Require: An LTS $\mathcal{L} = \langle Q, L, \Delta, q_{init} \rangle$
Require: A sub-group Φ of $\mathrm{Aut}(\mathcal{L})$
Ensure: A quotient reachability graph $\widetilde{\mathcal{L}} \in \mathrm{QRGs}(\mathcal{L}, \Phi)$ of $\mathcal{L}$

Set $\widetilde{Q} \Leftarrow \emptyset$, $\widetilde{\Delta} \Leftarrow \emptyset$, and $W \Leftarrow \emptyset$
Choose $\widetilde{q}_{init} \in Q$ such that $\widetilde{q}_{init} \equiv_\Phi q_{init}$ {Take *any* symmetric state}
$W \Leftarrow W \cup \{\widetilde{q}_{init}\}$
while $W \neq \emptyset$ **do**
  Choose any $q \in W$ and set $W \Leftarrow W \setminus \{q\}$
  if $q \notin \widetilde{Q}$ **then**
    $\widetilde{Q} \Leftarrow \widetilde{Q} \cup \{q\}$ {Add state to $\widetilde{\mathcal{L}}$}
    for all q_1 such that $q \xrightarrow[\mathcal{L}]{l} q_1$ **do**
      Choose $\widetilde{q}_1$ such that $\widetilde{q}_1 \equiv_\Phi q_1$ {Take *any* symmetric state}
      $W \Leftarrow W \cup \{\widetilde{q}_1\}$
      $\widetilde{\Delta} \Leftarrow \widetilde{\Delta} \cup \langle q, l, \widetilde{q}_1 \rangle$ {Add transition to $\widetilde{\mathcal{L}}$}
    end for
  end if
end while
Return $\langle \widetilde{Q}, L, \widetilde{\Delta}, \widetilde{q}_{init} \rangle$

Example 2.2.2 Figure 2.2 shows two quotient reachability graphs of the LTS shown in Figure 2.1 under the automorphism group described in Examples 2.1.3 and 2.1.5. The one on the left hand side is minimal whereas the one on the right hand side is not. ♣

The approach for Well-Formed Nets presented in [Chiola et al. 1991] differs slightly from the one presented here: instead of choosing a symmetric state as successor, a symbolic state is build that represents all the states in an orbit. Symbolic states themselves are not states in the strict sense i.e. they do not belong to the original state space.

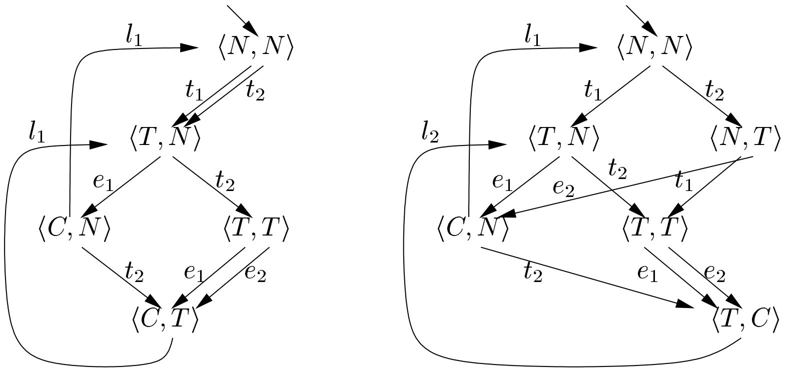

Figure 2.2: Quotient reachability graphs of the LTS shown in Figure 2.1.

2.2.1 The Strong Bisimulation Connection

We now show that all quotient reachability graphs under an automorphism group are strongly bisimilar.

Definition 2.2.3 *Let $\mathcal{L} = \langle Q, L, \Delta, q_{init}\rangle$ and $\mathcal{L}' = \langle Q', L', \Delta', q'_{init}\rangle$ be two LTSs. A relation $\mathcal{R} \subseteq ((Q \times Q') \cup (L \times L'))$ is a* strong bisimulation *between $\mathcal{L}$ and $\mathcal{L}'$ if for all $q \in Q$ and $q' \in Q'$, $q \ \mathcal{R} \ q'$ implies*

- *for all $l \in L$ and all $q_1 \in Q$ it holds that if $q \xrightarrow[\mathcal{L}]{l} q_1$, then there exists $l' \in L'$, $q'_1 \in Q'$ such that $q' \xrightarrow[\mathcal{L}']{l'} q'_1$, $l \ \mathcal{R} \ l'$ and $q_1 \ \mathcal{R} \ q'_1$; and*
- *for all $l' \in L'$ and all $q'_1 \in Q'$ it holds that if $q' \xrightarrow[\mathcal{L}']{l'} q'_1$, then there exists $l \in L$, $q_1 \in Q$ such that $q \xrightarrow[\mathcal{L}]{l} q_1$, $l \ \mathcal{R} \ l'$ and $q_1 \ \mathcal{R} \ q'_1$.*

Theorem 2.2.4 *Let Φ be a sub-group of $\mathrm{Aut}(\mathcal{L})$ and let $\widetilde{\mathcal{L}} = \langle \widetilde{Q}, L, \widetilde{\Delta}, \widetilde{q}_{init}\rangle$ and $\widetilde{\mathcal{L}}' = \langle \widetilde{Q}', L, \widetilde{\Delta}', \widetilde{q}'_{init}\rangle$ be two quotient reachability graphs of $\mathcal{L}$ under Φ. Then $\mathcal{R} = \equiv_\Phi \cap ((\widetilde{Q} \times \widetilde{Q}') \cup (L \times L))$ is a strong bisimulation relation between $\widetilde{\mathcal{L}}$ and $\widetilde{\mathcal{L}}'$. In addition, $\widetilde{q}_{init} \ \mathcal{R} \ \widetilde{q}'_{init}$.*

Proof. Clearly $\mathcal{R} \subseteq (\widetilde{Q} \times \widetilde{Q}') \cup (L \times L)$. Assume that $\widetilde{q} \ \mathcal{R} \ \widetilde{q}'$. Then $\widetilde{q} \in \widetilde{Q}$, $\widetilde{q}' \in \widetilde{Q}'$ and $\widetilde{q} \equiv_\Phi \widetilde{q}'$ by the definition of $\mathcal{R}$. If $\langle \widetilde{q}, l, \widetilde{q}_1\rangle \in \widetilde{\Delta}$, then $\widetilde{q}_1 \in \widetilde{Q}$ and there exists q_1 such that $\langle \widetilde{q}, l, q_1\rangle \in \Delta$ and $q_1 \equiv_\Phi \widetilde{q}_1$ by items 2 and 3 in the definition of quotient reachability graphs. Because $\widetilde{q} \equiv_\Phi \widetilde{q}'$, there exists l' and q'_1 such that $\langle \widetilde{q}', l', q'_1\rangle \in \Delta$, $l \equiv_\Phi l'$ and $q_1 \equiv_\Phi q'_1$ by the definition of $\mathrm{Aut}(\mathcal{L})$. Now because $\widetilde{q}' \in \widetilde{Q}'$, there exists $\langle \widetilde{q}', l', \widetilde{q}'_1\rangle \in \widetilde{\Delta}'$ such that $q'_1 \equiv_\Phi \widetilde{q}'_1$ by the definition of quotient reachability graphs. Since $\equiv_\Phi$ is an equivalence relation, $q'_1 \equiv_\Phi \widetilde{q}'_1$, $q_1 \equiv_\Phi q'_1$ and $q_1 \equiv_\Phi \widetilde{q}_1$ imply $\widetilde{q}_1 \equiv_\Phi \widetilde{q}'_1$. In addition, $\widetilde{q}'_1 \in \widetilde{Q}'$ and thus $\widetilde{q}_1 \ \mathcal{R} \ \widetilde{q}'_1$. Therefore, $\langle \widetilde{q}, l, \widetilde{q}_1\rangle \in \widetilde{\Delta}$ implies $(\exists l' \in L)(\exists \widetilde{q}'_1 \in \widetilde{Q}')\left(\langle \widetilde{q}', l', \widetilde{q}'_1\rangle \in \widetilde{\Delta}' \wedge l \ \mathcal{R} \ l' \wedge \widetilde{q}_1 \ \mathcal{R} \ \widetilde{q}'_1\right)$. $\langle \widetilde{q}', l', \widetilde{q}'_1\rangle \in \widetilde{\Delta}'$ implies $(\exists l \in L)(\exists \widetilde{q}_1 \in \widetilde{Q})\left(\langle \widetilde{q}, l, \widetilde{q}_1\rangle \in \widetilde{\Delta} \wedge l \ \mathcal{R} \ l' \wedge \widetilde{q}_1 \ \mathcal{R} \ \widetilde{q}'_1\right)$ with symmetric argumentation. From the item 1 in the definition of quotient reachability graphs it is obvious that $\widetilde{q}_{init} \ \mathcal{R} \ \widetilde{q}'_{init}$. □

Property 2.2.5 *Since* $\mathrm{RG}(\mathcal{L}) \in \mathrm{QRGs}(\mathcal{L}, \Phi)$, *there exists a strong bisimulation relation between the reachability graph and each quotient reachability graph of* $\mathcal{L}$.

2.2.2 Orbit Calculation

Of course, it does not make much sense to randomly pick a symmetric successor state (definition of QRGs and line 9 of Algorithm 2.2.1). A much better idea is to choose a symmetric state $\widetilde{q}_1$ that is already in $\widetilde{Q}$.

One approach for doing this is to pair-wisely compare the newly generated q_1 with each state in $\widetilde{Q}$ until a symmetric state is found and select that state as the successor (or if one is not found, use q_1 itself). For most interesting system description formalisms it is a graph isomorphism[2] hard problem to compare whether two states are symmetric (see e.g. [Clarke et al. 1996; Ip and Dill 1996]).

Another approach is to calculate and use a *representative state* for q_1. Formally:

Definition 2.2.6 *For a sub-group* Φ *of* $\mathrm{Aut}(\mathcal{L})$, *a function* $\mathrm{repr} : (Q \cup L) \to \wp(Q \cup L)$ *such that* $(\mathrm{repr}(x) = Y) \Rightarrow (Y \subseteq [x]_\Phi)$ *is a* representative function. *If* $|\mathrm{repr}(x)| = 1$ *for all* $x \in Q \cup L$, *then* repr *is* deterministic *and* $\mathrm{repr}(x) = \{y\}$ *may be written as* $\mathrm{repr}(x) = y$. *If* repr *is deterministic and* $\mathrm{repr}(x) = y$ *for all* $x \in [y]_\Phi$, *then* repr *is* canonical.

Needless to say, canonical representatives are highly desirable yet computing them can be relatively hard (usually at least as hard as graph isomorphism, sometimes provably **NP**-hard, see e.g., [Clarke et al. 1998]), depending on the system level permutation group producing the automorphism group Φ and on how canonicity is defined (i.e. which states are defined to be the canonical ones).

2.2.3 Some Simple Properties of Quotient Reachability Graphs

Let $\widetilde{\mathcal{L}} = \langle \widetilde{Q}, L, \widetilde{\Delta}, \widetilde{q}_{init} \rangle$ and $\widetilde{\mathcal{L}}' = \langle \widetilde{Q}', L, \widetilde{\Delta}', \widetilde{q}'_{init} \rangle$ be two quotient reachability graphs of an LTS $\mathcal{L} = \langle Q, L, \Delta, q_{init} \rangle$ under a sub-group Φ of $\mathrm{Aut}(\mathcal{L})$. Let $\vec{\mathcal{L}} = \mathrm{RG}(\mathcal{L}) = \langle \vec{Q}, L, \vec{\Delta}, q_{init} \rangle$.

We now show some simple properties of quotient reachability graphs, based on the observation that all QRGs are strongly bisimilar. Most of them are obvious (and presented in papers mentioned in section 1.1) and thus no proofs are presented. Recall that the reachability graph of $\mathcal{L}$ is a QRG of $\mathcal{L}$ and thus the following results hold for it, too.

First, states in different QRGs have symmetric counter-states:

Property 2.2.7 Correspondence of reachable states. *A state is in* $\widetilde{Q}$ *iff there is a symmetric state in* $\widetilde{Q}'$: $\widetilde{q} \in \widetilde{Q} \Leftrightarrow (\exists \widetilde{q}' \in \widetilde{Q}')\, (\widetilde{q} \equiv_\Phi \widetilde{q}')$

Direct consequences are the following:

[2] See e.g. [Köbler et al. 1993] for discussion on the graph isomorphism problem.

Property 2.2.8 Preservation of deadlocks. *A state q is a deadlock state if there are no $l \in L$ and $q' \in Q$ such that $q \xrightarrow[\mathcal{L}]{l} q'$. There is a deadlock state in $\widetilde{\mathcal{L}}$ iff there is a deadlock state in $\widetilde{\mathcal{L}}'$.*

Property 2.2.9 *If $\widetilde{\mathcal{L}}$ is minimal under Φ, then $|\widetilde{Q}| \leq |\widetilde{Q}'|$.*

Property 2.2.10 *Assume that $[q]_\Phi$ is finite for all $q \in \vec{Q}$. Then $\widetilde{Q}$ is finite iff $\vec{Q}$ is finite.*

A similar correspondence holds for paths:

Property 2.2.11 Correspondence of paths. *There is path $\widetilde{\sigma} = \widetilde{q}_0 \xrightarrow[\widetilde{\mathcal{L}}]{\widetilde{l}_0} \widetilde{q}_1 \cdots$ of length n starting from a state $\widetilde{q}_0 \in \widetilde{Q}$ iff from each state $\widetilde{q}'_0 \in \widetilde{Q}'$ such that $\widetilde{q}_0 \equiv_\Phi \widetilde{q}'_0$ starts a path $\widetilde{\sigma}' = \widetilde{q}'_0 \xrightarrow[\widetilde{\mathcal{L}}']{\widetilde{l}'_0} \widetilde{q}'_1 \cdots$ of length n such that $\widetilde{q}_i \equiv_\Phi \widetilde{q}'_i$ and $\widetilde{l}_i \equiv_\Phi \widetilde{l}'_i$ for all $0 \leq i \leq n$.*

Symmetricity of the initial state implies that for a reachable state, all the symmetric states are also reachable:

Property 2.2.12 *If q_{init} is symmetric under Φ, then $q \in \vec{Q} \Rightarrow \phi(q) \in \vec{Q}$ holds for all $q \in Q$ and all $\phi \in \Phi$. Furthermore, $\widetilde{q} \in \widetilde{Q} \Leftrightarrow (\forall q \in [\widetilde{q}]_\Phi) \left(q \in \vec{Q}\right)$. This also implies that $|\widetilde{Q}| \leq |\vec{Q}|$.*

Let AP be a set of atomic propositions. A *truth assignment* for $\mathcal{L}$ is a function $\tau : Q \to \wp(AP)$. We say that an atomic proposition $p \in AP$ is *invariant under* Φ if $(\forall q \in Q)(\forall \phi \in \Phi)\,(p \in \tau(q) \Leftrightarrow p \in \tau(\phi(q)))$. We then know by the work of Browne et al. [1988] that, in the (very expressive) temporal logic CTL*, a state satisfies a formula involving only invariant atomic propositions iff a state strongly bisimilar to it satisfies the formula. We therefore can check symmetry invariant CTL* properties on a QRG instead of the RG. For further reading about model checking under symmetries, see the papers mentioned in Section 1.1.

2.3 AN EXAMPLE: SYMMETRIES FOR PLACE/TRANSITION-NETS

To further illustrate the symmetry reduction method, we briefly review the approach for Place/Transition-nets [Starke 1991; Schmidt and Starke 1991; Schmidt 1997; 1999]. This formalism was selected because of the simplicity of definitions involved.

Let $N = \langle P, T, F, V, M_{init} \rangle$ be a *Place/Transition-net* (a P/T-net), where P is a finite non-empty set of *places*, T is a finite non-empty set of *transitions* such that $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (the flow relation), $V : F \to \mathbb{N}$ maps each arc in F with a *multiplicity* and $M_{init} \in [P \to \mathbb{N}]$ is the *initial marking* of N. The set $X = P \cup T$ denotes the *set of nodes* of N. A *marking* of N is a function $M : P \to \mathbb{N}$ and the *set of all markings* of N is denoted by $\mathbb{M}$. Note that an equivalent definition of markings is that a marking is a multi-set over P and we may thus write a marking as a formal sum over P.

For a transition t of N, we define the marking t^- as: for all $p \in P$, $t^-(p) = V(p,t)$ if $\langle p,t \rangle \in F$ and $t^-(p) = 0$ otherwise. Similarly, t^+ is defined by $t^+(p) = V(t,p)$ if $\langle t,p \rangle \in F$ and $t^+(p) = 0$ when $\langle t,p \rangle \notin F$. Transition t is *enabled* in a marking M if $t^- \leq M$.[3] If t is enabled in M, then it may *fire*, changing M into a new marking M' defined as: for all $p \in P$, $M'(p) = M(p) - t^-(p) + t^+(p)$. This is denoted by $M\ [t\rangle\ M'$. The relation $[\rangle \subseteq \mathbb{M} \times T \times \mathbb{M}$ for N is defined by $[\rangle = \{\langle M,t,M' \rangle \mid M\ [t\rangle\ M'\}$. The *state space* of N is the LTS $\mathrm{SS}(N) = \langle \mathbb{M}, T, [\rangle, M_{init} \rangle$.

In the Place/Transition net approach, symmetries on the state space level are produced by automorphisms on the net level:

Definition 2.3.1 (Net Automorphisms) *A* net automorphism *of N is a bijection $\psi : X \to X$ which*

1. *respects node type, i.e. $\psi(P) = P$ and $\psi(T) = T$;*
2. *respects the arc relation: for all $x, y \in X$, $\langle x,y \rangle \in F$ iff $\langle \psi(x), \psi(y) \rangle \in F$; and*
3. *respects arc multiplicities: $V(x,y) = V(\psi(x), \psi(y))$ for all $\langle x,y \rangle \in F$.*

The set of all automorphisms *of N is denoted by* $\mathrm{Aut}(N)$ *and is a group under $\circ$.*

In [Schmidt and Starke 1991; Schmidt 1997], Schmidt and Starke have developed an algorithm which computes the generators of the net automorphisms group $\mathrm{Aut}(N)$. Although finding the automorphisms of a graph (a P/T-net can be seen as a directed graph) is computationally a relatively hard task (presumably not achievable in deterministic polynomial time), it seems to be quite tractable in practice.

Example 2.3.2 Consider the P/T-net for a railroad system in Figure 2.3 (a 6-section variant of that in [Genrich 1991]). The structure of the net 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. Thus the net automorphism group, call it Ψ, of the net is the internal direct product (see Definition A.2.5) 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. For instance, $\psi_1^{\mathrm{rot3}} = \left(\begin{smallmatrix} U_{a0} & U_{a1} & U_{a2} & U_{a3} & U_{a4} & U_{a5} & U_{b0} & \cdots & U_{b5} & V_0 & \cdots & V_5 & t_{a0} & \cdots & t_{a5} & t_{b0} & \cdots & t_{b5} \\ U_{a3} & U_{a4} & U_{a5} & U_{a0} & U_{a1} & U_{a2} & U_{b3} & \cdots & U_{b2} & V_3 & \cdots & V_2 & t_{a3} & \cdots & t_{a2} & t_{b3} & \cdots & t_{b2} \end{smallmatrix}\right) \in \Psi_1$. Ψ_2 is the group consisting of the identity mapping and the permutation swapping the train identities: $\psi_2^{\mathrm{swap}} = \left(\begin{smallmatrix} U_{a0} & \cdots & U_{a5} & U_{b0} & \cdots & U_{b5} & V_0 & \cdots & V_5 & t_{a0} & \cdots & t_{a5} & t_{b0} & \cdots & t_{b5} \\ U_{b0} & \cdots & U_{b5} & U_{a0} & \cdots & U_{a5} & V_0 & \cdots & V_5 & t_{b0} & \cdots & t_{b5} & t_{a0} & \cdots & t_{a5} \end{smallmatrix}\right)$.

[3] For two markings, M and M', we define that $M \leq M'$ iff $M(p) \leq M'(p)$ for all $p \in P$.

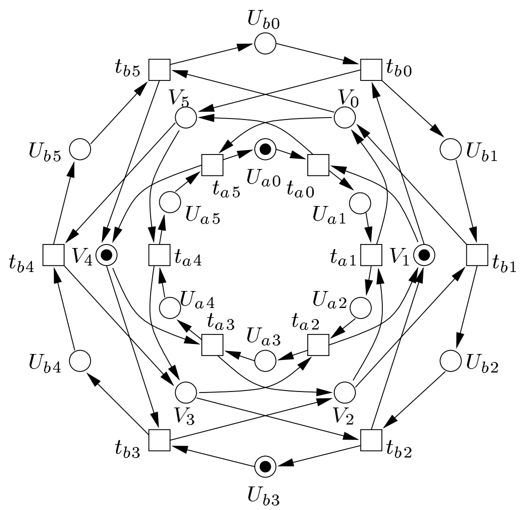

Figure 2.3: A P/T-net for a railroad system.

Combining these two groups, we have $\Psi = \Psi_1 \circ \Psi_2 = \{\psi_1 \circ \psi_2 \mid \psi_1 \in \Psi_1 \wedge \psi_2 \in \Psi_2\}$ and that, for instance,

$$\psi^{\text{rot3,swap}} = \psi_1^{\text{rot3}} \circ \psi_2^{\text{swap}} = \begin{pmatrix} U_{a0} & \cdots & U_{a5} & U_{b0} & \cdots & U_{b5} & V_0 & \cdots & V_5 & t_{a0} & \cdots & t_{a5} & t_{b0} & \cdots & t_{b5} \\ U_{b3} & \cdots & U_{b2} & U_{a3} & \cdots & U_{a2} & V_3 & \cdots & V_2 & t_{b3} & \cdots & t_{b2} & t_{a3} & \cdots & t_{a5} \end{pmatrix}$$

is also in Ψ. ♣

A net automorphism ψ of N is extended to operate on markings by defining the corresponding permutation $\psi_{\mathbb{M}} \in [\mathbb{M} \to \mathbb{M}]$ by $\big(\psi_{\mathbb{M}}(M)\big)(p) = M(\psi^{-1}(p))$, or equivalently, by $\big(\psi_{\mathbb{M}}(M)\big)(\psi(p)) = M(p)$. It is easy to see that $\psi_{\mathbb{M}}$ is indeed a bijection on $\mathbb{M}$. The following was shown by Starke [1991]:

Lemma 2.3.3 *Let ψ be a net automorphism, t a transition and M, M' markings of N. Then the following hold:*

- $\psi_{\mathbb{M}}(t^-) = (\psi(t))^-$ *and* $\psi_{\mathbb{M}}(t^+) = (\psi(t))^+$
- $M\ [t\rangle\ M' \Leftrightarrow \psi_{\mathbb{M}}(M)\ [\psi(t)\rangle\ \psi_{\mathbb{M}}(M')$.

It is quite straightforward to see that the net automorphisms of N produce corresponding automorphisms to the state space of N.

Theorem 2.3.4 *For each $\psi \in \mathrm{Aut}(N)$, $h(\psi) = \psi_{\mathrm{SS}(N)} = \{\psi_{\mathbb{M}} \cup \psi|_T\}$ is an automorphism of $\mathrm{SS}(N)$.*[4]

Proof. A direct consequence of Lemma 2.3.3 and the fact that $\psi_{\mathbb{M}}$ and $\psi|_T$ are disjoint bijections on $\mathbb{M}$ and T, respectively. □

Thus h is a group homomorphism from $\mathrm{Aut}(N)$ to $\mathrm{Aut}(\mathrm{SS}(N))$.

[4]For a function $f : A \to B$ and a set $A_1 \subseteq A$, $f|_{A_1} : A_1 \to B$ such that $f|_{A_1}(a) = f(a)$ for all $a \in A_1$ is the restriction of f to A_1

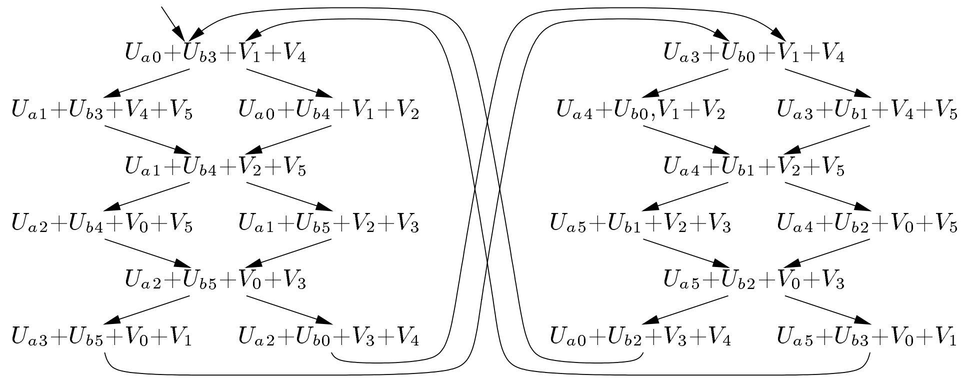

Figure 2.4: Reachability graph of the net in Figure 2.3.

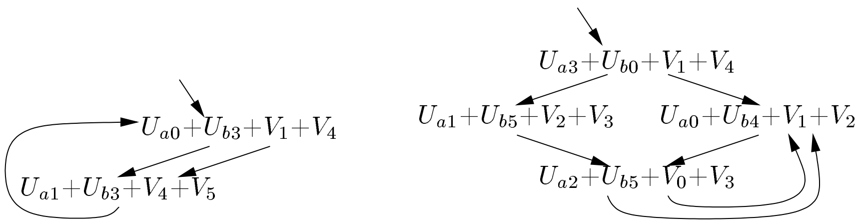

Figure 2.5: Two quotient reachability graphs of the net in Figure 2.3.

Example 2.3.5 Consider again the net in Figure 2.3 and its symmetries explained in Example 2.3.2. The reachability graph of the net is shown in Figure 2.4. We can see that the automorphisms of the net actually produce automorphisms to the state space (only the reachable part is shown in Figure 2.4). The reachability graph exhibits the train swapping symmetry produced by the net automorphism ψ_2^{swap}. In this case, $M = U_{a1} + U_{b3} + V_4 + V_5$ becomes $M' = \psi_2^{\text{swap}}{}_{\mathbb{M}}(M) = \psi_2^{\text{swap}}(U_{a1}) + \psi_2^{\text{swap}}(U_{b3}) + \psi_2^{\text{swap}}(V_4) + \psi_2^{\text{swap}}(V_5) = U_{a3} + U_{b1} + V_4 + V_5$. The symmetry produced by rotating the net is also present, e.g. M' is mapped to $M'' = \psi_1^{\text{rot3}}{}_{\mathbb{M}}(M') = (\psi_1^{\text{rot3}} \circ \psi_1^{\text{swap}})_{\mathbb{M}}(M) = \psi_1^{\text{rot3}}(U_{a3}) + \psi_1^{\text{rot3}}(U_{b1}) + \psi_1^{\text{rot3}}(V_4) + \psi_1^{\text{rot3}}(V_5) = U_{a0} + U_{b4} + V_1 + V_2$. Thus $U_{a1}+U_{b3}+V_4+V_5$ is symmetric to $U_{b4}+U_{a0}+V_1+V_2$. In fact, the reachability graph is composed of markings belonging to two equivalence classes under the action of Ψ: (i) The first class is contains markings in which the distance between the trains is three sections, e.g. $U_{a0}+U_{b3}+V_1+V_4$, $U_{a1}+U_{b4}+V_2+V_5$, $U_{a2} + U_{b5} + V_0 + V_3$ and $U_{a3} + U_{b0} + V_1 + V_4$. (ii) Markings in the second class are such that the (shorter) distance between the trains is two sections, e.g. $U_{a1} + U_{b3} + V_4 + V_5$, $U_{a0} + U_{b4} + V_1 + V_2$ and $U_{a5} + U_{b1} + V_2 + V_3$. Figure 2.5 shows two quotient reachability graphs of the net. ♣

Schmidt [1999] discusses different ways how to utilize the symmetry information during the generation of QRGs. Two approaches check whether a marking symmetric to a newly generated one M is already in the QRG: the first iterates trough all net symmetries ϕ in $\mathrm{Aut}(N)$ checking whether $(h(\phi))\,(M)$ is already in the QRG while the second iterates through all markings already in the QRG and for each marking checks whether it is symmetric to M. The third approach presented calculates a deterministic but not necessarily canonical representative marking for M by using the group $\mathrm{Aut}(N)$.

3 ALGEBRAS AND DOMAIN PERMUTATIONS

Later in this work we will be dealing with algebraic system nets (ASNs). As their name implies, these nets use algebras as means to represent and manipulate data. This chapter first reviews basics of algebras. For reasons that become apparent later in Chapter 6, we use a home-made form of error algebras (cf. [Wirsing 1990]). This is in contrast to other works on algebraic (system) nets, e.g. [Reisig 1991; Kindler and Reisig 1996; Kindler and Völzer 1998]. We next introduce the concept of domain permutations for algebras which is later used when we study the symmetries of algebraic system nets. Another idea crucial to ASN symmetries is the concept of compatibility between terms and domain permutations for an algebra, introduced in Section 3.3. We also study the complexity of deciding whether a term is compatible with a domain permutation, which turns out to be a **co-NP**-complete problem. To circumvent this problem, we present an approximation scheme for term compatibility in Section 3.4.

3.1 SIGNATURES AND ALGEBRAS

Signatures define the used sort (read "data type") names and the names of operations that can be applied to the sorts.

Definition 3.1.1 (Signatures) *A* signature $SIG = \langle \mathcal{S}, \mathscr{F} \rangle$ *consists of*

- *a non-empty set* $\mathcal{S}$ *of* sort names *(aka* sorts*); and*
- *a pairwise disjoint family* $\mathscr{F} = (\mathscr{F}_{\sigma,s})_{\sigma \in \mathcal{S}^*, s \in \mathcal{S}}$ *of* operation symbols.

An operation symbol $f \in \mathscr{F}_{s_1 \dots s_n, s}$ stands for an operation from $s_1, \dots, s_n$ to s, where $s_1, \dots, s_n$ are the *domain* (or argument) sorts and s is the *range* sort of f. The set $\mathscr{F}_{\epsilon,s}$ is called the set of *SIG-constant symbols of sort s* (ϵ denotes the empty string). The disjointness of $\mathscr{F}$ is a mere technicality because we can always rename the operations, we only have to take care that terms (defined later) have unambiguous interpretation. We may thus, for instance, use the same operation symbol "+" to denote addition in the contexts of both natural numbers and multi-sets as long as it is understood which operation is meant. We follow the notation that sorts are written with capitalized sans serif, e.g., Bool is used to denote the sort for Booleans.

Definition 3.1.2 (Variables) *A pairwise disjoint family* $\mathcal{X} = (\mathcal{X}_s)_{s \in \mathcal{S}}$ *such that* $\mathcal{X} \cap \mathscr{F} = \emptyset$ *is called a* family of *SIG*-variables.

We require that the families of variables and operations are disjoint to avoid confusions between constants (which are operations) and variables.

Using variables, constant symbols and operation symbols, we can build *SIG*-terms as follows.

Definition 3.1.3 (Terms) *The set* $\mathbf{T}_s^{SIG}(\mathcal{X})$ *of SIG*-terms of sort s over $\mathcal{X}$ *is the minimal set (in the set inclusion sense) defined inductively by the following rules.*

1. $\mathcal{X}_s \subseteq \mathbf{T}_s^{SIG}(\mathcal{X})$.

2. For $n \geq 0$, if $f \in \mathscr{F}_{s_1 \ldots s_n, s}$ and $T_i \in \mathbf{T}_{s_i}^{SIG}(\mathcal{X})$ for all $1 \leq i \leq n$, then $f(T_1, \ldots, T_n) \in \mathbf{T}_s^{SIG}(\mathcal{X})$.[1]

The set $\mathbf{T}_s^{SIG} = \mathbf{T}_s^{SIG}(\emptyset)$ denotes the set of *SIG-ground terms of sort* s, $\mathbf{T}^{SIG}(\mathcal{X}) = \bigcup_{s \in \mathcal{S}} \mathbf{T}_s^{SIG}(\mathcal{X})$ is the set of *SIG-terms over* $\mathcal{X}$ and $\mathbf{T}^{SIG} = \bigcup_{s \in \mathcal{S}} \mathbf{T}_s^{SIG}$ is the set of *SIG-ground terms*. Note that we may also write terms in mix-fix notation whenever no confusion can arise. As an example, a term $+(a, b)$ for $+ \in \mathscr{F}_{\mathsf{Nat.Nat,Nat}}$ may be written as $a + b$.

An algebra corresponding to a signature *SIG* gives an interpretation for the sorts and operations of the signature. It assigns each sort a domain set and each operation symbol a function.

Definition 3.1.4 (Algebras) *A SIG-*error algebra *(or simply a SIG-*algebra*)* $\mathscr{A} = \langle \mathscr{D}^{\mathscr{A}}, \mathscr{F}^{\mathscr{A}} \rangle$ *consists of:*

1. *A family* $\mathscr{D}^{\mathscr{A}} = \left(\mathscr{D}_s^{\mathscr{A}}\right)_{s \in \mathcal{S}}$ *of non-empty* domain sets *for sorts. It is assumed that no* $\mathscr{D}_s^{\mathscr{A}}$ *contains the element* $\mathfrak{err}$ *while* $\tilde{\mathscr{D}}_s^{\mathscr{A}}$ *denotes the set* $\mathscr{D}_s^{\mathscr{A}} \cup \{\mathfrak{err}\}$.

2. *A family* $\mathscr{F}^{\mathscr{A}} = \left(f^{\mathscr{A}}\right)_{f \in \mathscr{F}}$ *of* operations. *For each* $f \in \mathscr{F}_{s_1 \ldots s_n, s}$, $n \geq 0$, *the operation* $f^{\mathscr{A}}$ *is a mapping* $f^{\mathscr{A}} : \tilde{\mathscr{D}}_{s_1}^{\mathscr{A}} \times \cdots \times \tilde{\mathscr{D}}_{s_n}^{\mathscr{A}} \to \tilde{\mathscr{D}}_s^{\mathscr{A}}$.

Unless it is otherwise stated, we implicitly assume that each operation returns $\mathfrak{err}$ if any of its arguments is $\mathfrak{err}$ (actually, we will consider only two exceptions in Section 6.3). We say that an operation $f^{\mathscr{A}}$ is *safe* if $f^{\mathscr{A}}(v_1, \ldots, v_n) = \mathfrak{err}$ implies that at least one of $v_1, \ldots, v_n$ is $\mathfrak{err}$.

We can assign values to *SIG*-variables under an *SIG*-algebra:

Definition 3.1.5 (Assignments) *An* assignment β *to the variables in a family* $\mathcal{X}$ *of SIG-variables (under an algebra* $\mathscr{A}$*) is a mapping* $\beta : \mathcal{X} \to \mathscr{D}^{\mathscr{A}}$ *such that* $\beta(x) \in \mathscr{D}_s^{\mathscr{A}}$ *for each variable* $x \in \mathcal{X}_s$.

The *set of all assignments* to $\mathcal{X}$ under $\mathscr{A}$ is denoted by $\mathrm{Ass}(\mathscr{A}, \mathcal{X})$. Note that variables cannot be assigned to the error value. Given an assignment to variables, we evaluate *SIG*-terms as follows.

Definition 3.1.6 (Evaluations of Terms) *An assignment* $\beta \in \mathrm{Ass}(\mathscr{A}, \mathcal{X})$ *is extended to the corresponding* evaluation of terms, $eval_\beta^{\mathscr{A}} : \mathbf{T}^{SIG}(\mathcal{X}) \to \tilde{\mathscr{D}}^{\mathscr{A}}$, *by the following inductive definition for each* $T \in \mathbf{T}^{SIG}(\mathcal{X})$.

1. *If* $T = x \in \mathcal{X}_s$, *then* $eval_\beta^{\mathscr{A}}(T) = \beta(T)$.

2. *If* $T = f(T_1, \ldots, T_n)$, $n \geq 0$, $f \in \mathscr{F}_{s_1 \ldots s_n, s}$ *and* $T_i \in \mathbf{T}_{s_i}^{SIG}(\mathcal{X})$ *for* $1 \leq i \leq n$, *then* $eval_\beta^{\mathscr{A}}(T) = f^{\mathscr{A}}(eval_\beta^{\mathscr{A}}(T_1), \ldots, eval_\beta^{\mathscr{A}}(T_n))$.

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 $eval^{\mathscr{A}}(T)$ instead of $eval_\beta^{\mathscr{A}}(T)$ (for any assignment β). Furthermore, for terms composed only of safe operations the evaluation value is never $\mathfrak{err}$.

[1] For a constant term $\mathsf{c}()$, we may simply write c instead.

Example 3.1.7 Consider a simple "Place/Transition-net signature" $PTSig = \langle \mathcal{S}, \mathscr{F} \rangle$ with three sorts, $\mathcal{S} = \{\mathsf{Bool}, \mathsf{Nat}, \mathsf{Token}\}$, where Token is a data type for the "black token" of P/T-nets. For Booleans we have the constants $\mathscr{F}_{\epsilon,\mathsf{Bool}} = \{\mathsf{true}, \mathsf{false}\}$, an unary operation $\mathscr{F}_{\mathsf{Bool},\mathsf{Bool}} = \{\mathsf{neg}\}$, and binary operations $\mathscr{F}_{\mathsf{Bool}.\mathsf{Bool},\mathsf{Bool}} = \{\mathsf{and}, \mathsf{or}, \mathsf{implies}, \mathsf{equiv}\}$. Natural numbers can be accessed and manipulated with constants $\{0, 1, 2, \dots\} = \mathscr{F}_{\epsilon,\mathsf{Nat}}$, comparison operations $\{\mathsf{equal}, \mathsf{inequal}, \mathsf{less}, \mathsf{greater}, \dots\} = \mathscr{F}_{\mathsf{Nat}.\mathsf{Nat},\mathsf{Bool}}$ and normal arithmetic operations $\{\mathsf{plus}, \mathsf{monus}, \mathsf{times}, \mathsf{div}\} = \mathscr{F}_{\mathsf{Nat}.\mathsf{Nat},\mathsf{Nat}}$. The constant $\bullet \in \mathscr{F}_{\epsilon,\mathsf{Token}}$ is the only operation for the token data type. For a family $\mathcal{X} = (\mathcal{X}_{\mathsf{Bool}} = \{x, y\}, \mathcal{X}_{\mathsf{Nat}} = \{i, j\}, \mathcal{X}_{\mathsf{Token}} = \emptyset)$ of *PTSig*-variables, the following are *PTSig*-terms over $\mathcal{X}$: true, x, $\mathsf{and}(x, y)$, $\mathsf{or}(x, \mathsf{implies}(x, \mathsf{false}))$, $\mathsf{equal}(i, j)$, $\mathsf{and}(\mathsf{less}(i, \mathsf{plus}(j, 6)), x)$, $\bullet$ and so on.

One possible algebra $PTAlg = \langle \mathscr{D}^{PTAlg}, \mathscr{F}^{PTAlg} \rangle$ for the above signature is the following obvious one. The domains are $\mathscr{D}^{PTAlg}_{\mathsf{Bool}} = \mathbb{B}$, $\mathscr{D}^{PTAlg}_{\mathsf{Nat}} = \mathbb{N}$ and $\mathscr{D}^{PTAlg}_{\mathsf{Token}} = \{\bullet\}$. The operations (including constants) are associated with functions, e.g., $\mathsf{true}^{PTAlg}() = true$, $\mathsf{or}^{PTAlg}(v_1, v_2) = true$ iff $v_1 = true$ or $v_2 = true$, $5^{PTAlg} = 5$, $\mathsf{plus}^{PTAlg}(m, n) = m + n$, $\mathsf{monus}^{PTAlg}(m, n) = m \dot{-} n$ ($m - n$ if $m \geq n$ and 0 otherwise), $\bullet^{PTAlg}() = \bullet$ and so on. Given an assignment on the family $\mathcal{X}$ of *PTSig*-variables above, for instance $\beta = \{x \mapsto false, y \mapsto true, i \mapsto 3, j \mapsto 2\}$, we can evaluate the above *PTSig*-terms:

$$
\begin{aligned}
eval^{PTAlg}_{\beta}(\mathsf{true}) &= \mathsf{true}^{PTAlg}() = true \\
eval^{PTAlg}_{\beta}(x) &= \beta(x) = false \\
eval^{PTAlg}_{\beta}(\mathsf{and}(x, y)) &= \mathsf{and}^{PTAlg}(eval^{PTAlg}_{\beta}(x), eval^{PTAlg}_{\beta}(y)) \\
&= \mathsf{and}^{PTAlg}(false, true) = false \\
eval^{PTAlg}_{\beta}(\mathsf{or}(x, \mathsf{implies}(x, \mathsf{false}))) &= true \\
eval^{PTAlg}_{\beta}(\mathsf{equal}(i, j)) &= false \\
eval^{PTAlg}_{\beta}(\mathsf{and}(\mathsf{less}(i, \mathsf{plus}(j, 6)), x)) &= false \\
eval^{PTAlg}_{\beta}(\bullet) &= \bullet
\end{aligned}
$$

♣

Since Booleans play a special role in algebraic system nets, we define the following:

Definition 3.1.8 *SIG is a* Boolean signature *if it includes the sort* Bool. *In that case we require that* $\mathscr{D}^{\mathscr{A}}_{\mathsf{Bool}} = \mathbb{B}$ *for any SIG-algebra* $\mathscr{A}$.

3.2 DOMAIN PERMUTATIONS

In this section we will define a way how the domains of sorts are permuted.

Formally, a *permutation* on a set A is a bijective function from A to itself. The *set of all permutations* on A is denoted by $\mathrm{Perm}(A)$ and is a group under the function composition operation $\circ$.

Assume a signature $SIG = \langle \mathcal{S}, \mathscr{F} \rangle$ and a corresponding SIG-algebra $\mathscr{A} = \langle \mathscr{D}^{\mathscr{A}}, \mathscr{F}^{\mathscr{A}} \rangle$.

Definition 3.2.1 (Domain Permutations) A domain permutation ψ_s *for a sort* $s \in \mathcal{S}$ *is a member of* $\mathrm{Perm}(\mathscr{D}_s^{\mathscr{A}})$ *while a* domain permutation group *for* s *is a sub-group of* $\mathrm{Perm}(\mathscr{D}_s^{\mathscr{A}})$. ψ_s *is implicitly extended to* $\tilde{\mathscr{D}}_s^{\mathscr{A}}$ *so that* $\psi_s(\mathfrak{err}) = \mathfrak{err}$.

A domain permutation *for a set of sorts,* $S' \subseteq \mathcal{S}$*, is a family* $\psi_{S'} = (\psi_s)_{s \in S'}$ *where for each* $s \in S'$*,* ψ_s *is a domain permutation for* s. A domain permutation group *for* S' *is a non-empty set* $\Psi_{S'}$ *of domain permutations for* S' *such that* $\langle \Psi_{S'}, * \rangle$ *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* $s \in S'$.

A domain permutation group for $\mathcal{S}$ is also called a domain permutation group for $\mathscr{A}$.

Imagine that we have specified a group Ψ_s of domain permutations for each sort $s \in S'$, $S' \subseteq \mathcal{S}$, individually. We can combine them into a domain permutation group for S' by taking any appropriate sub-group of the "exterior direct product"

$$\bigotimes_{s \in S'} \Psi_s = \{(\psi_s)_{s \in S'} \mid \psi_s \in \Psi_s \text{ for each } s \in S'\}.$$

It is quite straightforward to see that $\bigotimes_{s \in S'} \Psi_s$ is a group under the binary operator $*$ defined in Definition 3.2.1. Furthermore, $\bigotimes_{s \in S'} \Psi_s$ is a sub-group of $\bigotimes_{s \in S'} \mathrm{Perm}(\mathscr{D}_s^{\mathscr{A}})$. The definition of $\bigotimes$ resembles the standard exterior direct product construction of group theory (see Definition A.2.4 or, e.g., [Grimaldi 1985]). The only difference is that it is expressed by using families instead of Cartesian products, which saves us from defining artificial total orders on the set of sorts.

Domain permutations are extended to variable assignments by simply permuting the assigned values.

Definition 3.2.2 *Let* $\psi_{\mathcal{S}} = (\psi_s)_{s \in \mathcal{S}}$ *be a domain permutation for* $\mathscr{A}$*. Let* $\beta \in \mathrm{Ass}(\mathscr{A}, \mathcal{X})$ *be an assignment to a family* $\mathcal{X} = (\mathcal{X}_s)_{s \in \mathcal{S}}$ *of MSIG-variables. Then the* corresponding permuted variable assignment $\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)$ *is defined by* $\big(\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)\big)(x) = \psi_s(\beta(x))$ *for all sorts* $s \in \mathcal{S}$ *and all variables* $x \in \mathcal{X}_s$.

Clearly $\psi_{\mathrm{Ass}(\mathcal{X})}(\beta) \in \mathrm{Ass}(\mathscr{A}, \mathcal{X})$, $\Psi_{\mathrm{Ass}(\mathcal{X})} = \{\psi_{\mathrm{Ass}(\mathcal{X})} \mid \psi_{\mathcal{S}} \in \Psi_{\mathcal{S}}\}$ is a sub-group of $\mathrm{Perm}(\mathrm{Ass}(\mathscr{A}, \mathcal{X}))$ for a domain permutation group $\Psi_{\mathcal{S}}$ for $\mathscr{A}$ and the mapping from $\Psi_{\mathcal{S}}$ to $\Psi_{\mathrm{Ass}(\mathcal{X})}$ is a group homomorphism.

Example 3.2.3 Consider the P/T-net algebra described in Example 3.1.7. The set $\Psi_{\mathcal{S}} = \{\psi_{\mathcal{S}}, \psi'_{\mathcal{S}}\}$ where

$$\psi_{\mathcal{S}} = (\psi_{\mathsf{Bool}} = \mathrm{Id}(\mathbb{B}), \psi_{\mathsf{Nat}} = \mathrm{Id}(\mathbb{N}), \psi_{\mathsf{Token}} = \mathrm{Id}(\{\bullet\}))$$

and

$$\psi'_{\mathcal{S}} = \left(\psi_{\mathsf{Bool}} = \left(\begin{smallmatrix} \mathit{false} & \mathit{true} \\ \mathit{true} & \mathit{false} \end{smallmatrix}\right), \psi_{\mathsf{Nat}} = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & \cdots \\ 0 & 2 & 1 & 3 & 4 & \cdots \end{smallmatrix}\right), \psi_{\mathsf{Token}} = \mathrm{Id}(\{\bullet\})\right)$$

is a domain permutation group for *PTAlg* ($\mathrm{Id}(A)$ denotes the identity mapping on the set A). $\psi'_{\mathcal{S}}$ maps the assignment $\beta = \{t \mapsto \mathit{true}, x \mapsto 2, y \mapsto 10\}$ on variables in $\mathcal{X} = (\mathcal{X}_{\mathsf{Bool}} = \{t\}, \mathcal{X}_{\mathsf{Nat}} = \{x, y\}, \mathcal{X}_{\mathsf{Token}} = \emptyset)$ to the assignment $\psi'_{\mathrm{Ass}(\mathcal{X})}(\beta) = \{t \mapsto \mathit{false}, x \mapsto 1, y \mapsto 10\}$. ♣

3.3 TERM COMPATIBILITY

We say that a term and a domain permutation are compatible if it holds for each variable assignment on the variables appearing in the term that for the permuted variable assignment the evaluation result of the term is similarly permuted or both evaluations return $\mathfrak{err}$. Formally:

Definition 3.3.1 (Compatible Terms) *A term $T \in \mathbf{T}_s^{SIG}(\mathcal{X})$ and a domain permutation $\psi_{\mathcal{S}}$ are* compatible *if*

$$eval_{\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)}(T) = \psi_s(eval_\beta(T))$$

holds for all assignments $\beta \in \mathrm{Ass}(\mathscr{A}, \mathcal{X})$. The term T is compatible with a domain permutation group $\Psi_{\mathcal{S}}$ if it is compatible with all $\psi_{\mathcal{S}} \in \Psi_{\mathcal{S}}$.

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

Example 3.3.2 Continuing Example 3.2.3, term $\mathsf{neg}(x)$ is compatible with $\psi'_{\mathcal{S}}$ whereas $\mathsf{or}(t, \mathsf{and}(u, \mathsf{true}))$ is not since

$$eval_{\{t \mapsto true, u \mapsto false\}}(\mathsf{or}(t, \mathsf{and}(u, \mathsf{true}))) = true$$

but

$$eval_{\{t \mapsto false, u \mapsto true\}}(\mathsf{or}(t, \mathsf{and}(u, \mathsf{true}))) = true,$$

not *false* as required. ♣

3.3.1 Complexity of Deciding Term Compatibility

We next study the computational complexity of issues concerning term compatibility. For basics on computational complexity, see e.g. [Garey and Johnson 1979; Papadimitriou 1995].

We use polynomial-time many-one reductions and make the following assumptions. An arbitrary algebra $\mathscr{A}$ with finite domains is assumed. Furthermore, we assume that each operation in $\mathscr{A}$ is computable in deterministic polynomial time. That is, for each $f \in \mathscr{F}_{s_1 \ldots s_n, s} \in \mathscr{F}$, when given $v_1 \in \tilde{\mathscr{D}}_{s_1}^{\mathscr{A}}, \ldots, v_n \in \tilde{\mathscr{D}}_{s_n}^{\mathscr{A}}$, the value $f^{\mathscr{A}}(v_1, \ldots, v_n)$ is computable in deterministic polynomial time (i.e. $f^{\mathscr{A}}$ belongs to the class **FP**). This means that given an assignment for variables in a family $\mathcal{X}$ of variables and a term $T \in \mathbf{T}^{SIG}(\mathcal{X})$, we can clearly evaluate T in deterministic polynomial time. Domain permutations are given in a form such that it is possible to apply them to elements of domains and variable assignments in deterministic polynomial time. That is, for instance, given a domain permutation $\psi_{\mathcal{S}}$ and an element $v \in \tilde{\mathscr{D}}_s^{\mathscr{A}}$, $s \in \mathcal{S}$, as input, the value $\psi_s(v)$ can be computed in deterministic polynomial time.

We first study how hard it is to tell whether a given domain permutation and a term are compatible. This is an important question since it would be highly desirable to verify that user-given domain permutations are compatible with the terms in her/his ASN description.

Problem 3.3.3 TERM COMPATIBILITY WITH RESPECT TO A DOMAIN PERMUTATION (TC1). *Given a term T and a domain permutation $\psi_{\mathcal{S}}$, is T compatible with $\psi_{\mathcal{S}}$?*

Theorem 3.3.4 TC1 *is* **co-NP**-*complete.*

Proof. We can guess a disqualifying assignment β on the variables occurring in T in non-deterministic polynomial time and then calculate and compare $eval_{\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)}(T)$ and $\psi_s(eval_\beta(T))$ in (deterministic) polynomial time. We then return "yes" if they were not equal and "no" otherwise. This non-deterministic polynomial time computation clearly accepts the complement of TC1. Thus TC1 is in **co-NP**.

We show **co-NP**-hardness of TC1 by reducing VALIDITY, a well-known **co-NP**-complete problem, to it. Assume that we are dealing with a very simple algebra, namely the Boolean algebra with one sort, Bool with the domain $\mathbb{B}$, and a truth-functionally complete set of operations (such as $\{\wedge, \vee, \neg\}$ with the usual interpretations). Take any Boolean formula f over a set $\mathcal{X}$ of Boolean variables. Define $f' = x \vee \neg f$ where x is a new Boolean variable not in $\mathcal{X}$ and let $\mathcal{X}'$ be $\mathcal{X}$ augmented with x. Clearly f' can be expressed as a term over $\mathcal{X}'$ with our Boolean algebra. Now consider the only non-identity domain permutation $\psi_{\mathcal{S}}$ in our algebra: the one flipping the truth values *true* and *false*. Now f' is evaluated to true for all assignments on $\mathcal{X}'$ such that $x \mapsto true$. But, in order f' to be compatible with $\psi_{\mathcal{S}}$, this means that for all $\psi_{\mathcal{S}}$-permuted assignments, i.e. for all assignments where $x \mapsto false$, f' must evaluate to *false*. Therefore $\neg f$ must be false for all assignments on $\mathcal{X}$ meaning that f has to evaluate to *true* for all assignments on $\mathcal{X}$. Thus f is valid if and only if f' is compatible with ψ. □

As is seen in the proof, the major complexity source in the term compatibility problem is not the domain permutation but rather the global nature of the definition of term compatibility. That is, term compatibility is quantified over all assignments on the variables appearing in a term.

Interestingly, the same problem for domain permutation *groups* remains in **co-NP**. However, from the practical point of view, it should be noted that the input size for the problem grows and thus, in practice, the problem for groups is a bit harder.

Problem 3.3.5 TERM COMPATIBILITY WITH RESPECT TO A DOMAIN PERMUTATION GROUP *aka* TC2. *Given a term T and a domain permutation group $\Psi_{\mathcal{S}}$, is T compatible with $\Psi_{\mathcal{S}}$?*

Theorem 3.3.6 TC2 *is* **co-NP**-*complete.*

Proof. The **co-NP**-hardness follows directly from the proof of Theorem 3.3.4 (setting Ψ to consist of the the identity mapping and the truth value flipping permutation ψ, f' is compatible with Ψ iff f is valid).

The complexity theoretical upper bound might depend on the succinctness of the representation of $\Psi_{\mathcal{S}}$. In the following we study two cases, for both of which the upper bound is **co-NP**.

If $\Psi_{\mathcal{S}}$ is given as an enumeration of all the domain permutations in it, we can solve the problem in **co-NP** by non-deterministically choosing first a disqualifying domain permutation $\psi_{\mathcal{S}}$ and then proceeding just like in the inclusion part in the proof of Theorem 3.3.4.

However, enumeration of all permutations is not an economical way of representing permutation groups. A much better way is to only store a set of generators of the group (see, e.g., [Butler 1991; Kreher and Stinson 1999]). For instance, the group of all permutations on an n element set can be represented by two generator permutations although the group itself has $n!$ permutations. Moreover, there are algorithms with which one can, given a set of generator permutations, build a compact representation of the permutation group in deterministic polynomial time w.r.t. the size of the permuted set and generator permutations. After this, one can query in deterministic polynomial time whether a permutation belongs to the group [Furst et al. 1980; Butler 1991; Kreher and Stinson 1999]. Now if $\Psi_{\mathcal{S}}$ is given by its generators, we can solve the problem TC2 by non-deterministically guessing a disqualifying permutation $\psi_{\mathcal{S}}$, checking (in deterministic polynomial time, possibly generating a compact representation for $\Psi_{\mathcal{S}}$) whether $\psi_{\mathcal{S}} \in \Psi_{\mathcal{S}}$, and if this was the case, non-deterministically choosing a disqualifying assignment and proceeding in the same way as in the inclusion part of Theorem 3.3.4. Thus TC2 is in **co-NP** even for groups given by generator permutations. □

The existence of compatible domain permutations for a term is another interesting question.

Problem 3.3.7 EXISTENCE OF COMPATIBLE DOMAIN PERMUTATIONS *or* ECDP. *Given a term T, is there a non-identity domain permutation $\psi_{\mathcal{S}}$ such that T is compatible with $\psi_{\mathcal{S}}$?*

Theorem 3.3.8 ECDP *is in* $\mathbf{NP}^{\mathbf{NP}[1]}$ *(the class of decision problems solvable by an* **NP***-machine with one access to an* **NP***-oracle).*

Proof. We can first non-deterministically guess a (non-identity) domain permutation $\psi_{\mathcal{S}}$ in polynomial time (the length of $\psi_{\mathcal{S}}$ is a constant) and then call a **co-NP**-oracle to find out whether it is compatible with T. Thus ECDP is in $\mathbf{NP}^{\mathbf{co\text{-}NP}[1]}$ (which equals to $\mathbf{NP}^{\mathbf{NP}[1]}$). □

The author does not know the exact hardness of ECDP: certainly it is **co-NP**-hard (with an ECDP machine we can answer whether the truth value flipping domain permutation in the proof of Theorem 3.3.4 is compatible since it is the only non-identity one). Author's current (somewhat weak) conjecture is that ECDP is not in $\mathbf{P}^{\mathbf{NP}}$.

Let us compare ECDP to the GRAPH AUTOMORPHISM problem asking whether a graph has non-trivial automorphisms. Although these problems have quite a similar flavor, the main difference is that for graphs it is easy (read: in **P**) to check whether a given permutation of nodes is an automorphism whereas checking whether a domain permutation is compatible with a term is (under plausible complexity theoretical assumptions) a harder problem. Therefore, if a yes answer is given to an ECDP instance together with a witness domain permutation $\psi_{\mathcal{S}}$, it is relatively hard to verify the correctness of the answer.

3.4 APPROXIMATING TERM COMPATIBILITY

In the light of results presented at the end of the previous section, an approximation scheme for term compatibility is highly desirable. We achieve one such scheme by considering a lower level compatibility, namely that between domain permutations and individual operations. We then show that if all operations appearing in a term are compatible, then the whole term is compatible with a domain permutation. The connection of operation compatibility to the notion of algebra homomorphisms is also shown.

3.4.1 Operation Compatibility

Definition 3.4.1 (Compatible Operations) *An operation $f^{\mathscr{A}}$, where $f \in \mathscr{F}_{s_1 \ldots s_n, s}$, and a domain permutation $\psi_{\mathcal{S}}$ are* compatible *if it holds for all $v_1 \in \tilde{\mathscr{D}}^{\mathscr{A}}_{s_1}, \ldots, v_n \in \tilde{\mathscr{D}}^{\mathscr{A}}_{s_n}$ that*

$$f^{\mathscr{A}}(\psi_{s_1}(v_1), \ldots, \psi_{s_n}(v_n)) = \psi_s(f^{\mathscr{A}}(v_1, \ldots, v_n)).$$

An operation f is compatible with a domain permutation group $\Psi_{\mathcal{S}}$ if it is compatible with all $\psi_{\mathcal{S}} \in \Psi_{\mathcal{S}}$.

Hence, an operation is compatible with a domain permutation if for a permuted input arguments it will give a similarly permuted output. An immediate consequence of the definition is that a domain permutation compatible with a constant c (recall that constants are operations) cannot permute its value to other values since $\mathsf{c}^{\mathscr{A}}() = \psi_s(\mathsf{c}^{\mathscr{A}}())$ must hold. Also note that in the usual case when an operation returns $\mathfrak{err}$ whenever and only when any of the arguments is $\mathfrak{err}$, it suffices to consider the compatibility under the non-error arguments since $\mathfrak{err}$ is never permuted.

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

Lemma 3.4.2 *Let $\psi_{\mathcal{S}} = (\psi_s)_{s \in \mathcal{S}}$ be a domain permutation and let $T \in \mathbf{T}^{SIG}_s(\mathcal{X})$ be a term such that the operations appearing in it are compatible with $\psi_{\mathcal{S}}$. Then $eval_{\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)}(T) = \psi_s(eval_\beta(T))$ holds for all assignments β on $\mathcal{X}$, i.e. T is compatible with $\psi_{\mathcal{S}}$.*

Proof. By induction on the structure of the term. Induction base. Case 1 (constants). If $T = f \in \mathscr{F}_{\epsilon,s}$, the $eval_\beta(T) = f^{\mathscr{A}}() = eval_{\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)}(T)$. But by the compatibility of $f^{\mathscr{A}}$ and $\psi_{\mathcal{S}}$ we have $f^{\mathscr{A}}() = \psi_s(f^{\mathscr{A}}())$. Case 2 (variables). If $T = x \in \mathcal{X}_s$, then $eval_{\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)}(T) = \psi_{\mathrm{Ass}(\mathcal{X})}(\beta)(T) = \psi_s(\beta(T)) = \psi_s(eval_\beta(T))$.

Induction hypothesis. Let $T_i \in \mathbf{T}^{SIG}_{s_i}(\mathcal{X})$, $1 \leq i \leq n$, be terms compatible with $\psi_{\mathcal{S}}$ (so that $eval_{\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)}(T_i) = \psi_{s_i}(eval_\beta(T_i))$).

Induction step. Let $T = f(T_1, \ldots, T_n) \in \mathbf{T}^{SIG}_s(\mathcal{X})$ where $f \in \mathscr{F}_{s_1,\ldots,s_n,s}$. Now $eval_{\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)}(T) = f^{\mathscr{A}}(eval_{\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)}(T_1), \ldots, eval_{\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)}(T_n))$ is by the induction hypothesis equal to $f^{\mathscr{A}}(\psi_{s_1}(eval_\beta(T_1)), \ldots, \psi_{s_n}(eval_\beta(T_n)))$. Since $f^{\mathscr{A}}$ is compatible with $\psi_{\mathcal{S}}$, $f^{\mathscr{A}}(\psi_{s_1}(eval_\beta(T_1)), \ldots, \psi_{s_n}(eval_\beta(T_n)))$ equals to $\psi_s(f^{\mathscr{A}}(eval_\beta(T_1), \ldots, eval_\beta(T_n))) = \psi_s(eval_\beta(T))$. $\square$

Example 3.4.3 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 equality checking term $\mathsf{c} = \mathsf{d}$, where c and d are constants of a sort s and $= \in \mathscr{F}_{s.s,\mathsf{Bool}}$ such that $=^{\mathscr{A}}(x, y) = \mathit{true}$ iff x and y are the same element. The constants c and d are not compatible with any domain permutation permuting the elements $\mathsf{c}^{\mathscr{A}}()$ and $\mathsf{d}^{\mathscr{A}}()$ in $\mathscr{D}_s^{\mathscr{A}}$. But the term itself is compatible with all domain permutations that leave the domain of Bool intact. This is because $\mathsf{c}^{\mathscr{A}}()$ is the same element as $\mathsf{d}^{\mathscr{A}}()$ iff $\psi_s(\mathsf{c}^{\mathscr{A}}())$ and $\psi_s(\mathsf{d}^{\mathscr{A}}())$ are the same (by the bijective nature of permutations). ♣

Approximating term compatibility through operation compatibility is extremely handy when it comes to implementation since it allows fast checking of term compatibility. That is, following the approaches taken by Ip and Dill [1996] and Chiola et al. [1991], the type system of an implementation language is such that allows definition of special type classes to which only certain restricted operations can be applied. Operations are analyzed by hand *once* at the language design phase to find out what kind of domain permutations are compatible with them. Thus checking the compatibility of a term corresponds to checking that only allowed restricted operations appear in it.

3.4.2 The Homomorphism Connection

There is an interesting (and somewhat obvious) connection between our definition of operation compatibility and the traditional notion of algebra homomorphisms [Wirsing 1990].[2] Let $\mathit{SIG} = \langle \mathcal{S}, \mathscr{F} \rangle$ be a signature and let $\mathscr{A} = \langle \mathscr{D}^{\mathscr{A}}, \mathscr{F}^{\mathscr{A}} \rangle$ and $\mathscr{A}' = \langle \mathscr{D}^{\mathscr{A}'}, \mathscr{F}^{\mathscr{A}'} \rangle$ be two SIG-algebras. A *SIG-homomorphism h* from $\mathscr{A}$ to $\mathscr{A}'$ is a family of mappings $\left(h_s : \tilde{\mathscr{D}}_s^{\mathscr{A}} \to \tilde{\mathscr{D}}_s^{\mathscr{A}'}\right)_{s \in \mathcal{S}}$ such that for each $f \in \mathscr{F}_{s_1 \ldots s_n, s}$ and all $v_1 \in \tilde{\mathscr{D}}_{s_1}^{\mathscr{A}}, \ldots, v_n \in \tilde{\mathscr{D}}_{s_n}^{\mathscr{A}}$,

$$h_s(f^{\mathscr{A}}(v_1, \ldots, v_n)) = f^{\mathscr{A}'}(h_{s_1}(v_1), \ldots, h_{s_n}(v_n)).$$

When each h_s is a bijection, h is a *SIG-isomorphism*. Furthermore, if h is a SIG-isomorphism from a SIG-algebra $\mathscr{A}$ to itself, we say that h is a *SIG-automorphism* of $\mathscr{A}$. It should be clear that the set of all SIG-automorphisms of $\mathscr{A}$ forms a group under the binary operation $*$ of Definition 3.2.1. This group is denoted by $\mathrm{Aut}(\mathscr{A})$.

The definition of SIG-isomorphisms is similar to our definition for compatibility between operations and domain permutations (Definition 3.4.1). Thus, if all the operations are compatible with a domain permutation, then the domain permutation (extended to errors) is a SIG-automorphism of $\mathscr{A}$ and each (similarly extended) domain permutation group is a sub-group of $\mathrm{Aut}(\mathscr{A})$.

[2] This connection was realized by the author after the definition operation compatibility; therefore it does not appear in the previous works ([Junttila 1998; 1999]).

4 ALGEBRAIC SYSTEM NETS

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

4.1 MULTI-SET SIGNATURES AND ALGEBRAS

Since multi-sets (see Appendix A.1) play an important role in algebraic system nets (as well as in Petri nets in general), we now introduce the concept of multi-set signatures and their corresponding algebras, which are special cases of signatures and algebras, respectively. The idea is to 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 [Kindler and Reisig 1996; Kindler and Völzer 1998]. 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. Formally we define the following.

Definition 4.1.1 (Multi-Set Signatures) *For a signature $SIG = \langle \mathcal{S}, \mathscr{F} \rangle$, let $GS, MS \subseteq \mathcal{S}$ be two non-empty disjoint sets such that $GS \cup MS = \mathcal{S}$ and $|GS| = |MS|$. GS is called the set of* ground sorts *and MS is the set of* multi-set sorts. *$MSIG = \langle \mathcal{S}, \mathscr{F}, ms \rangle$ is a* multi-set signature *if $ms : GS \to MS$ is a bijective mapping. A SIG-algebra $\mathscr{A} = \langle \left(\mathscr{D}_s^{\mathscr{A}}\right)_{s \in \mathcal{S}}, \mathscr{F}^{\mathscr{A}} \rangle$ is a $MSIG$-*algebra *if $\mathscr{D}^{\mathscr{A}}_{ms(s)} = \mathcal{MS}(\mathscr{D}_s^{\mathscr{A}})$ for all $s \in GS$.*

As each multi-set-signature $MSIG = \langle \mathcal{S}, \mathscr{F}, ms \rangle$ is a specialization of the corresponding signature $SIG = \langle \mathcal{S}, \mathscr{F} \rangle$, variables and terms are well-defined for multi-set signatures, too. Furthermore, each $MSIG$-algebra is a SIG-algebra and error values, assignments and evaluations are defined as earlier.

Below we list some standard multi-set operations. Let $s \in GS$ be a ground sort.

1. OPERATION: $\mathsf{plus} \in \mathscr{F}_{ms(s).ms(s),ms(s)}$ or $\cdot + \cdot$ in mix-fix notation
   DEFINITION: $\mathsf{plus}^{\mathscr{A}}(b_1, b_2) = b_1 + b_2$
2. OPERATION: $\mathsf{times} \in \mathscr{F}_{\mathsf{Nat}.ms(s),ms(s)}$ aka $\cdot \cdot$
   DEFINITION: $\mathsf{times}^{\mathscr{A}}(n, b) = n \cdot b$
3. OPERATIONS: $\mathsf{equals}, \mathsf{lessOrEqual} \in \mathscr{F}_{ms(s).ms(s),\mathsf{Bool}}$ aka $\cdot = \cdot$ and $\cdot \leq \cdot$, respectively
   DEFINITIONS: $\mathsf{equals}^{\mathscr{A}}(b_1, b_2) = \mathit{true}$ iff $b_1 = b_2$
   $\mathsf{lessOrEqual}^{\mathscr{A}}(b_1, b_2) = \mathit{true}$ iff $b_1 \leq b_2$
4. OPERATION: $\mathsf{unitms} \in \mathscr{F}_{s,ms(s)}$ aka $\cdot \in \mathscr{F}_{s,ms(s)}$
   DEFINITION: $\mathsf{unitms}^{\mathscr{A}}(v) = 1 \cdot v$

We may thus abuse the notation a bit and use each term of the ground sort s as a term of multi-set sort $ms(s)$. We only have to avoid ambiguities, e.g., it must be clear from the context whether the term $x + y$ of a multi-set sort means $\mathsf{unitms}(x) + \mathsf{unitms}(y)$ or $\mathsf{unitms}(x + y)$, where x and y are terms of the corresponding ground sort. That is, in case of overloading operators (such as the operator $+$ above), the semantics must be somehow made clear.

5. OPERATION: $\mathsf{all} \in \mathscr{F}_{\epsilon,ms(s)}$
   DEFINITION: $\mathsf{all}^{\mathscr{A}}() = \Sigma_{v \in \mathscr{D}_s^{\mathscr{A}}} 1 \cdot v$

   This operation returns the multi-set consisting of one copy of each element in the domain of s. Note that the error value is not included in it.

6. OPERATION: $\Psi\text{-}\mathsf{orbit} \in \mathscr{F}_{s,ms(s)}$ such that $\Psi_{\mathcal{S}}$ is a domain permutation group for the used algebra $\mathscr{A}$
   DEFINITION: $\Psi\text{-}\mathsf{orbit}^{\mathscr{A}}(v) = \Sigma_{v' \in [v]_{\Psi_{\mathcal{S}}}} 1 \cdot v'$

   The $\Psi_{\mathcal{S}}$-*orbit of* v is the set of elements in $\mathscr{D}_s^{\mathscr{A}}$ to which v can be permuted by domain permutations in $\Psi_{\mathcal{S}}$. It is denoted and defined by $[v]_{\Psi_{\mathcal{S}}} = \{v' \in \mathscr{D}_s^{\mathscr{A}} \mid (\exists \psi_{\mathcal{S}} \in \Psi_{\mathcal{S}})(v' = \psi_s(v))\}$. This operation returns the $\Psi_{\mathcal{S}}$-orbit of an element v and therefore domains of ground sorts can be divided into "static sub-classes" (w.r.t. a domain permutation group) as is done by Chiola et al. [1991].

7. OPERATION: $\mathsf{multiplicity} \in \mathscr{F}_{ms(s).s,\mathsf{Nat}}$ aka $\cdot(\cdot)$
   DEFINITION: $\mathsf{multiplicity}^{\mathscr{A}}(b, v) = b(v)$

   Returns the multiplicity of the element v in the multi-set b.

8. OPERATION: $\mathsf{size} \in \mathscr{F}_{ms(s),\mathsf{Nat}}$ aka $|\cdot|$
   DEFINITION: $\mathsf{size}^{\mathscr{A}}(b) = \sum_{v \in \mathscr{D}_s^{\mathscr{A}}} b(v)$ if the sum is finite and $\mathfrak{err}$ if it is not.

   Returns the number of all elements in a multi-set i.e. its size. In the case of an infinite multi-set, the error return value was chosen because introducing ∞ into the domain of Nat could cause some inconveniences. In practise, all our multi-sets are finite.

Example 4.1.2 We can extend the signature $PTSig = \langle \mathcal{S}, \mathscr{F} \rangle$ of Example 3.1.7 to a multi-set signature $PTMSig = \langle \mathcal{S}', \mathscr{F}', ms \rangle$ as follows. First, let $\mathcal{S}' = GS \cup MS$, where $GS = \mathcal{S}$ and $MS = \{\mathsf{MSBool}, \mathsf{MSNat}, \mathsf{MSToken}\}$. Let $ms(\mathsf{Bool}) = \mathsf{MSBool}$, $ms(\mathsf{Nat}) = \mathsf{MSNat}$ and $ms(\mathsf{Token}) = \mathsf{MSToken}$. In the extended algebra *PTMAlg* we have $\mathscr{D}_{\mathsf{MSBool}}^{PTMAlg} = \mathcal{MS}(\mathbb{B})$, $\mathscr{D}_{\mathsf{MSNat}}^{PTMAlg} = \mathcal{MS}(\mathbb{N})$ and $\mathscr{D}_{\mathsf{MSToken}}^{PTMAlg} = \mathcal{MS}(\{\bullet\})$. Note that $\mathcal{MS}(\{\bullet\}) = [\{\bullet\} \to \mathbb{N}]$ and thus $\mathscr{D}_{\mathsf{MSToken}}^{PTMAlg}$ essentially equals to $\mathbb{N}$ (a multi-set of black tokens is fully characterized by the number of tokens in it). As a basic set of multi-set operations we use the ones listed above. For instance, $\mathsf{3} \cdot \bullet$ (actually $\mathsf{times}(\mathsf{3}, \mathsf{unitms}(\bullet))$) always evaluates to multi-set of three black tokens whereas $x \cdot y + \mathsf{1} \cdot \mathsf{true}$ evaluates to the multi-set $3 \cdot \mathit{false} + 1 \cdot \mathit{true}$ in $\mathscr{D}_{\mathsf{MSBool}}^{PTMAlg}$ under an assignment mapping x to 3 and y to *false*. ♣

4.2 ALGEBRAIC SYSTEM NETS

An algebraic system net is basically a Petri net augmented with algebraic insciptions on the transitions and the arcs.

Definition 4.2.1 (Algebraic System Nets) *An* algebraic system net *is a tuple*

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

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);*
- $MSIG = \langle \mathcal{S}, \mathscr{F}, ms \rangle$ *is a Boolean multi-set-signature with a set* GS *of ground sorts;*
- $\mathscr{A} = \langle \mathscr{D}^{\mathscr{A}}, \mathscr{F}^{\mathscr{A}} \rangle$ *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* $v(t)$ *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*
- $m_{init} : P \to \mathbf{T}^{MSIG}$ *such that* $m_{init}(p) \in \mathbf{T}^{MSIG}_{ms(d(p))}$ *is the* symbolic initial marking *assigning each place an* $MSIG$*-ground term of place's multi-set sort.*

A *marking* M of $\mathcal{N}$ is a function that assigns each place $p \in P$ a multi-set over its domain: $M(p) \in \mathscr{D}^{\mathscr{A}}_{ms(d(p))}$ for all $p \in P$. The *set of all markings* of $\mathcal{N}$ is denoted by $\mathbb{M}$. The *initial marking* M_{init} is the evaluation of the symbolic initial marking: $M_{init}(p) = eval^{\mathscr{A}}(m_{init}(p))$ for all $p \in P$ (we assume that $eval^{\mathscr{A}}(m_{init}(p)) \neq \mathfrak{err}$).

Let $t \in T$ be a transition of $\mathcal{N}$. 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: for all $p \in P$, $t^-_\beta(p) = eval^{\mathscr{A}}_\beta(a(p, t))$ if $\langle p, t \rangle \in F$ and $t^-_\beta(p) = \emptyset$ otherwise. Similarly, the *output effect function* t^+_β is defined by: $t^+_\beta(p) = eval^{\mathscr{A}}_\beta(a(t, p))$ 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 $M\ [t_\beta\rangle$, if

1. $eval^{\mathscr{A}}_\beta(g(t)) = true$,
2. $t^-_\beta(p) \neq \mathfrak{err}$ and $M(p) \geq t^-_\beta(p)$ for each $p \in P$, and

3. $t^+_\beta(p) \neq \mathfrak{err}$ for each $p \in P$.

If t_β is enabled in M, it 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 $\mathrm{SS}(\mathcal{N}) = \langle \mathbb{M}, \mathbb{T}, [\rangle, M_{init}\rangle$, 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 4.2.2 Recall the definition of Place/Transition nets (Section 2.3) and the multi-set signature *PTMSig* and the algebra *PTMAlg* built in Examples 3.1.7 and 4.1.2. A Place/Transition-net $N = \langle P, T, F, V, M_{init}\rangle$ can now be interpreted as an algebraic system net

$$\mathcal{N} = \langle\langle P, T, F\rangle, \mathit{PTMSig}, \mathit{PTMAlg}, d, v, a, g, m_{init}\rangle,$$

where $d(p) = \mathsf{Token}$ for all $p \in P$, $v(t) = \emptyset$ for all $t \in T$, $a(f) = \mathsf{unitms}(\mathsf{n})$ where $\mathsf{n} \in \mathscr{F}_{\epsilon,\mathsf{Nat}}$ is the constant evaluating to $n = V(f)$, $g(t) = \mathsf{true}$ for all $t \in T$, and $m_{init}(p) = \mathsf{unitms}(\mathsf{n})$ where the constant $\mathsf{n} \in \mathscr{F}_{\epsilon,\mathsf{Nat}}$ evaluates to $M_{init}(p)$ for all $p \in P$. It is clear that both nets have the same (isomorphic) behavior. ♣

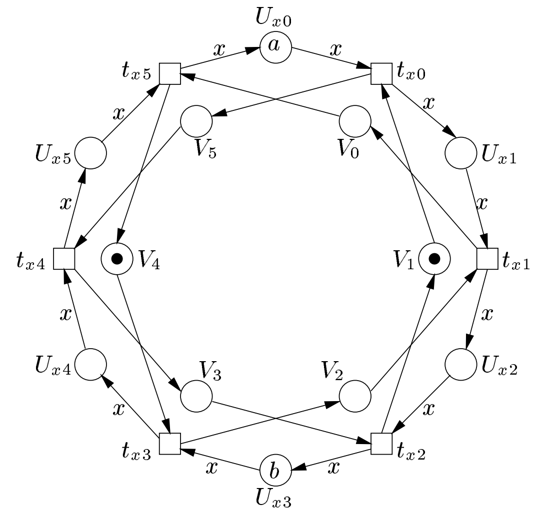

Figure 4.1: A partially folded algebraic system net of the net in Figure 2.3

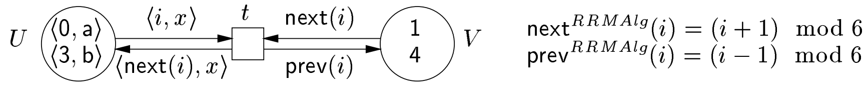

Figure 4.2: Yet another folded version of the net in Figure 2.3

Example 4.2.3 Consider the nets for the railroad system in Figures 4.1 and 4.2. We now define the multi-set signature *RRMSig* and the corresponding multi-set algebra *RRMAlg* for them by extending *PTMSig* and *PTMAlg* from Examples 3.1.7 and 4.1.2.

The set of sorts is augmented with (i) the ground sort Trains for the trains, Secs for the railroad sections, and the composite sort $\mathsf{SecsXTrains}$ and (ii) the corresponding multi-set sorts, e.g. MSSecs. For the trains, the domain is $\mathscr{D}^{RRMAlg}_{\mathsf{Trains}} = \{a, b\}$ and the operations consist of the constants a and b such that $\mathsf{a}^{RRMAlg}() = a$ and $\mathsf{b}^{RRMAlg}() = b$. For the railroad sections we have the domain $\mathscr{D}^{RRMAlg}_{\mathsf{Secs}} = \{0, 1, \ldots, 5\}$ and the corresponding constants $\mathsf{0}, \mathsf{1}, \ldots, \mathsf{5}$. The operations $\mathsf{next}, \mathsf{prev} \in \mathscr{F}_{\mathsf{Secs},\mathsf{Secs}}$ are defined as in the legend of Figure 4.2. The domain of the sort $\mathsf{SecsXTrains}$ is $\mathscr{D}^{RRMAlg}_{\mathsf{SecsXTrains}} = \mathscr{D}^{RRMAlg}_{\mathsf{Secs}} \times \mathscr{D}^{RRMAlg}_{\mathsf{Trains}}$ and the operation $\langle\cdot,\cdot\rangle \in \mathscr{F}_{\mathsf{Secs}.\mathsf{Trains},\mathsf{SecsXTrains}}$ is defined in the obvious way. For the corresponding multi-set sorts, e.g. MSSecs, we use the usual multi-set operations.

The domain of each place V_i, $0 \leq i \leq 5$ in Figure 4.1 is Token and the domain of U_{xi}, $0 \leq i \leq 5$, is Trains. Each transition has $\mathcal{X} = (\mathcal{X}_{\mathsf{Trains}} = \{x\})$ as its variables and the guards of all transitions are the constant true. The arc annotations of arc with no annotations drawn are $\bullet$.

Similarly, the domain of the place V in Figure 4.2 is Secs and that of U is $\mathsf{SecsXTrains}$. The variables of t are $\mathcal{X} = (\mathcal{X}_{\mathsf{Secs}} = \{i\}, \mathcal{X}_{\mathsf{Trains}} = \{x\})$ and its guard is 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 $\mathsf{msadd} \in \mathscr{F}_{\mathsf{MSSecsXTrains}.\mathsf{MSSecsXTrains},\mathsf{MSSecsXTrains}}$ written in the more familiar infix notation. In the initial marking $\{U \mapsto \langle 0, a\rangle + \langle 3, b\rangle, V \mapsto 1 + 4\}$, the transition t is enabled for mode $\{i \mapsto 0, x \mapsto a\}$ and firing it leads to the marking $\{U \mapsto \langle 1, a\rangle + \langle 3, b\rangle, V \mapsto 4 + 5\}$. The reachable part of the state space (the reachability graph) of the net is shown in Figure 4.3. The initial state is marked with an arrow originating from nowhere and the labels on the transitions are not shown. Note that the nets in Figures 2.3, 4.1 and 4.2 have isomorphic state spaces (compare Figures 2.4 and 4.3).

Note that the structural symmetries appearing in the P/T-net (recall Example 2.3.2) are only partially or not at all present in the nets in Figures 4.1 and 4.2. In Figure 4.1 the rotation symmetry is still present on the structural level whereas the train identity swapping symmetry is hidden in the way that the sort Trains is used. The net in Figure 4.2 does not exhibit neither rotation nor swapping symmetry on the structural level. Still both symmetries are present on the state space level! Detecting these symmetries is the topic of the next chapter. ♣

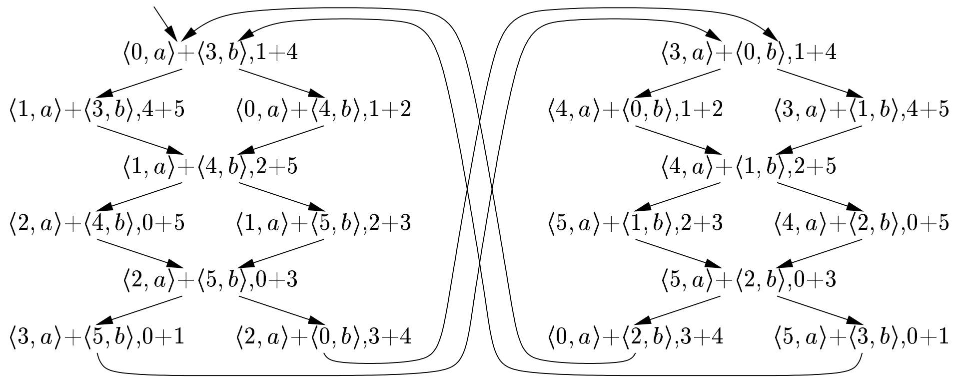

Figure 4.3: The reachable part of the state space of the net in Figure 4.2.

5 SYMMETRIES FOR ALGEBRAIC SYSTEM NETS

In this chapter we extend domain permutations for algebras to multi-set algebras. We also consider the compatibility of some standard multi-set operations. We then proceed by describing how the domain permutations for multi-set algebras are extended to the state space level of algebraic system nets. We show that requiring that the terms attached to transitions are compatible with the domain permutations is a sufficient condition to ensure that produced state space permutations are automorphisms.

5.1 WELL-FORMED DOMAIN PERMUTATIONS

As the Booleans have a special role in the definition of transition guards (the firing rule), it is desirable not to 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 state space automorphisms. We therefore define the following.

Definition 5.1.1 (Well-Formed Domain Permutations) *A domain permutation $\psi_{\mathcal{S}} = (\psi_s)_{s\in\mathcal{S}}$ is* well-formed *if* $\mathsf{Bool} \in \mathcal{S}$ *implies* $\psi_{\mathsf{Bool}}(v) = v$ *for all* $v \in \mathbb{B}$. *A domain permutation group is* well-formed *if 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. A positive note is that the (in)equality check operations $=, \neq \in \mathscr{F}_{s.s,\mathsf{Bool}}$ with standard interpretation of comparing whether two elements are (not) the same, are compatible with all well-formed domain permutations for all sorts $s \in \mathcal{S}$. This is a direct consequence of the bijective nature of domain permutations and of the definition of well-formed domain permutations.

5.2 DOMAIN PERMUTATIONS OF MULTI-SET ALGEBRAS

Assume now that the considered signature is a multi-set signature $MSIG = \langle \mathcal{S}, \mathscr{F}, ms\rangle$ with the set GS of ground sorts and that $\mathscr{A}$ is actually an $MSIG$-algebra (fulfilling the requirements imposed in Definition 4.1.1). A domain permutation (group) for GS is in this case called a *ground domain permutation (group)* for $\mathscr{A}$. Ground domain permutations are extended to multi-set sorts by the following definition, which states that the corresponding multi-set permutations are produced by permuting the elements of multi-sets but leaving the multiplicities intact.

Definition 5.2.1 *Let $\psi_{GS} = (\psi_s)_{s\in GS}$ be a ground domain permutation for $\mathscr{A}$. The corresponding* domain permutation *$\psi_{\mathcal{S}} = (\psi'_s)_{s\in\mathcal{S}}$ is defined by*

1. *for all $s \in GS$, $\psi'_s = \psi_s$ and*

2. *for all* $s \in GS$, $\psi'_{ms(s)} : \mathcal{MS}(\mathscr{D}_s^{\mathscr{A}}) \to \mathcal{MS}(\mathscr{D}_s^{\mathscr{A}})$ *such that for each multi-set* $b = \sum_{v \in \mathscr{D}_s^{\mathscr{A}}} b(v) \cdot v$ *in* $\mathcal{MS}(\mathscr{D}_s^{\mathscr{A}})$, $\psi'_{ms(s)}(b) = \sum_{v \in \mathscr{D}_s^{\mathscr{A}}} b(v) \cdot \psi'_s(v)$.

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

It follows directly from the definition that $\Psi_{\mathcal{S}}$ is a group under $*$. Furthermore, $\Psi_{\mathcal{S}}$ is a sub-group of $\bigotimes_{s \in \mathcal{S}} \mathrm{Perm}(\mathscr{D}_s^{\mathscr{A}})$ and the extension of Ψ_{GS} to $\Psi_{\mathcal{S}}$ is actually a group isomorphism. From now on we assume that domain permutations for multi-set algebras fulfill the requirements of the above definition.

5.2.1 Compatibility of Standard Multi-Set Operations

As the domain permutations for multi-set sorts are dictated by the domain permutations for the corresponding ground sorts, the following properties hold.

Lemma 5.2.2 *Each* $\psi_{ms(s)} : \mathcal{MS}(\mathscr{D}_s^{\mathscr{A}}) \to \mathcal{MS}(\mathscr{D}_s^{\mathscr{A}})$ *as defined in Definition 5.2.1 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. $\psi_{ms(s)}(n \cdot b) = n \cdot \psi_{ms(s)}(b)$, *for each* $n \in \mathbb{N}$
4. *For any* $b_1, b_2 \in \mathcal{MS}(\mathscr{D}_s^{\mathscr{A}})$, $b_1 \leq b_2$ *iff* $\psi_{ms(s)}(b_1) \leq \psi_{ms(s)}(b_2)$

Proof. First, $\psi_{ms(s)}(\emptyset) = \sum_{v \in \mathscr{D}_s^{\mathscr{A}}} \emptyset(v) \cdot \psi_s(v) = \emptyset$. The item 2 holds since

$$\begin{aligned}
\psi_{ms(s)}(b_1 + b_2) &= \sum_{v \in \mathscr{D}_s^{\mathscr{A}}} (b_1 + b_2)(v) \cdot \psi_s(v) \\
&= \sum_{v \in \mathscr{D}_s^{\mathscr{A}}} (b_1(v) + b_2(v)) \cdot \psi_s(v) \\
&= \sum_{v \in \mathscr{D}_s^{\mathscr{A}}} b_1(v) \cdot \psi_s(v) + \sum_{v \in \mathscr{D}_s^{\mathscr{A}}} b_2(v) \cdot \psi_s(v) \\
&= \psi_{ms(s)}(b_1) + \psi_{ms(s)}(b_2).
\end{aligned}$$

Similarly, $\psi_{ms(s)}(n \cdot b) = \sum_{v \in \mathscr{D}_s^{\mathscr{A}}} (n \cdot b)(v) \cdot \psi_s(v) = \sum_{v \in \mathscr{D}_s^{\mathscr{A}}} nb(v) \cdot \psi_s(v) = n \cdot \sum_{v \in \mathscr{D}_s^{\mathscr{A}}} b(v) \cdot \psi_s(v) = n \cdot \psi_{ms(s)}(b)$. 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 $v \in \mathscr{D}_s^{\mathscr{A}}$. By bijectivity of ψ_s this is equal 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 $v \in \mathscr{D}_s^{\mathscr{A}}$. This in turn equals to that $b_1(v) \leq b_2(v)$ for all $v \in \mathscr{D}_s^{\mathscr{A}}$ and thus to the requirement that $b_1 \leq b_2$. □

We may now investigate the compatibility of the standard operations on multi-set sorts described in Section 4.1 with respect to an arbitrary domain permutation $\psi_{\mathcal{S}}$ for a multi-set algebra $\mathscr{A}$. Let $s \in \mathcal{S}$ be a ground sort and $ms(s)$ its corresponding multi-set sort.

1. The multi-set addition operation $\mathsf{plus} \in \mathscr{F}_{ms(s).ms(s),ms(s)}$ is compatible with all domain permutations (fulfilling the conditions of Definition 5.2.1) because $\mathsf{plus}^{\mathscr{A}}(\psi_{ms(s)}(v_1), \psi_{ms(s)}(v_2)) = \psi_{ms(s)}(v_1) + \psi_{ms(s)}(v_2)$ equals to $\psi_{ms(s)}(v_1+v_2) = \psi_{ms(s)}(\mathsf{plus}^{\mathscr{A}}(v_1, v_2))$ by Lemma 5.2.2.

2. The scalar multiplication operation, $\mathsf{times} \in \mathscr{F}_{\mathsf{Nat}.ms(s),ms(s)}$, is compatible with all domain permutations in which the domain of Nat is not permuted by applying Lemma 5.2.2.

3. The two multi-set comparison operations equals and $\mathsf{lessOrEqual}$ in $\mathscr{F}_{ms(s).ms(s),\mathsf{Bool}}$ are, by Lemma 5.2.2, compatible with all domain permutations as long as the domain of the Booleans is not permuted (i.e. with all well-formed domain permutations).

4. The unit-multi-set operation $\mathsf{unitms} \in \mathscr{F}_{s,ms(s)}$ is compatible with all domain permutations since $\mathsf{unitms}^{\mathscr{A}}(\psi_s(v)) = 1 \cdot \psi_s(v)$ equals by Definition 5.2.1 to $\psi_{ms(s)}(1 \cdot v) = \psi_{ms(s)}(\mathsf{unitms}^{\mathscr{A}}(v))$.

5. The $\mathsf{all} \in \mathscr{F}_{\epsilon,ms(s)}$ constant operation returning the whole domain of s is compatible with all domain permutations. This is because each domain permutation ψ_s appearing in any $\psi_{\mathcal{S}} \in \Psi_{\mathcal{S}}$ is a bijection on $\mathscr{D}_s^{\mathscr{A}}$ and thus $\psi_{ms(s)}(\Sigma_{v \in \mathscr{D}_s^{\mathscr{A}}} 1 \cdot v) = \Sigma_{v \in \mathscr{D}_s^{\mathscr{A}}} 1 \cdot v$.

6. The operation $\Psi\text{-}\mathsf{orbit} \in \mathscr{F}_{s,ms(s)}$ returning $\Psi_{\mathcal{S}}$-orbit of an element v, $\Psi\text{-}\mathsf{orbit}^{\mathscr{A}}(v) = \Sigma_{v' \in [v]_{\Psi_{\mathcal{S}}}} 1 \cdot v'$, is compatible because of the way that the permutations for multi-set sorts are defined in Definition 5.2.1 and because (definition of $[v]_{\Psi_{\mathcal{S}}}$ and group properties of $\Psi_{\mathcal{S}}$) the orbit elements cannot be permuted out of the orbit by any permutation in $\Psi_{\mathcal{S}}$.

7. The operation $\mathsf{multiplicity} \in \mathscr{F}_{ms(s).s,\mathsf{Nat}}$ returning the multiplicity of an element in a multi-set is compatible with all domain permutations that do not permute the domain of the natural numbers. This is because $\left(\psi_{ms(s)}(b)\right)(\psi_s(v)) = b(v)$ directly by Definition 5.2.1.

8. The operation $\mathsf{size} \in \mathscr{F}_{ms(s),\mathsf{Nat}}$ returning the size of a multi-set is compatible with all domain permutations that leave the domain of the natural numbers intact. This is because the sum of the multiplicities is preserved when ground domain permutations are extended to multi-sets in Definition 5.2.1.

5.2.2 Quantified Operations

Let $T \in \mathbf{T}^{MSIG}_{\mathsf{Bool}}(\mathcal{X})$ be a Boolean term, where $\{x\} \subseteq \mathcal{X}$ for an arbitrary variable x of a sort s. Let us say that x is *the quantified variable* of T whereas $\mathcal{X} \setminus \{x\}$ is the (possibly empty) set of *free variables* appearing in T. Let the free variables $\mathcal{X} \setminus \{x\} = \{x_1, \ldots, x_n\}$ be ordered in an arbitrary order and let s_i be the sort of the free variable x_i, $1 \leq i \leq n$. We can now form a new operation of sort $ms(s)$, namely "$\mathsf{all}\ x : s\ \mathsf{s.t.}\ T(x_1, \ldots, x_n, x)$" $\in$

$\mathscr{F}_{s_1 \ldots s_n, ms(s)}$, whose interpretation is defined by the characteristic set

$$\text{``all } x : s \text{ s.t. } T(x_1, \ldots, x_n, x)\text{''}^{\mathscr{A}}(v_1, \ldots, v_n) = \{v \in \mathscr{D}_s^{\mathscr{A}} \mid eval_{\{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n, x \mapsto v\}}(T) = true\}.$$

That is, it returns the multi-set which contains one copy of each element in the domain of sort s that fulfills the predicate T. The free variables of T are bound to the arguments of the operation.

Luckily, the compatibility of such new quantified operations can be established by the compatibility of the predicate term. This means that we do not have to analyze the compatibility of each quantified operation separately, but can rely on the analysis of other operations and terms.

Theorem 5.2.3 *Let $T \in \mathbf{T}^{MSIG}_{\mathsf{Bool}}(\mathcal{X})$ be a Boolean term that is compatible with a well-formed domain permutation $\psi_{\mathcal{S}}$. Then, for any variable $x \in \mathcal{X}$ of sort s, the operation "all $x : s$ s.t. $T(x_1, \ldots, x_n, x)$" is compatible with $\psi_{\mathcal{S}}$.*

Proof. First, $eval_{\{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n, x \mapsto v\}}(T)$ equals by well-formedness of $\psi_{\mathcal{S}}$ to $\psi_{\mathsf{Bool}}(eval_{\{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n, x \mapsto v\}}(T))$, which in turn is by Definition 3.3.1 of compatible terms equal to $eval_{\{x_1 \mapsto \psi_{s_1}(v_1), \ldots, x_n \mapsto \psi_{s_n}(v_n), x \mapsto \psi_s(v)\}}(T)$. Thus the (multi-)set "all $x : s$ s.t. $T(x_1, \ldots, x_n, x)$"$^{\mathscr{A}}(v_1, \ldots, v_n)$ includes an element $v \in \mathscr{D}_s^{\mathscr{A}}$ iff "all $x : s$ s.t. $T(x_1, \ldots, x_n, x)$"$^{\mathscr{A}}(\psi_{s_1}(v_1), \ldots, \psi_{s_n}(v_n))$ contains $\psi_s(v)$. But this is exactly how ground domain permutations are extended to multi-sets (Definition 5.2.1) and therefore we may conclude that "all $x : s$ s.t. $T(x_1, \ldots, x_n, x)$" is compatible with $\psi_{\mathcal{S}}$. □

Example 5.2.4 Let PId be a sort for 5 process identifiers with the domain $\{pid_1, \ldots, pid_5\}$. Let $\psi_{\mathcal{S}}$ be a domain permutation in which we have $\psi_{\mathsf{PId}} = \begin{pmatrix} pid_1 & pid_2 & pid_3 & pid_4 & pid_5 \\ pid_2 & pid_3 & pid_4 & pid_1 & pid_5 \end{pmatrix}$. Now the Boolean equality testing term $x \neq y$, where x and y are variables of sort PId, is compatible with $\psi_{\mathcal{S}}$. Then also the operation "all x : PId s.t. $x \neq y$" is compatible with $\psi_{\mathcal{S}}$ (y is bound to the operation's argument). For instance, "all x : PId s.t. $x \neq y$"$^{\mathscr{A}}(pid_1) = 1 \cdot pid_2 + \cdots + 1 \cdot pid_5$ which is permuted to $\psi_{\mathcal{S}}(1 \cdot pid_2 + \cdots + 1 \cdot pid_5) = 1 \cdot pid_1 + 1 \cdot pid_3 + 1 \cdot pid_4 + 1 \cdot pid_5$. This equals to "all x : PId s.t. $x \neq y$"$^{\mathscr{A}}(pid_2) =$ "all x : PId s.t. $x \neq y$"$^{\mathscr{A}}(\psi_{\mathsf{PId}}(pid_1))$. ♣

5.2.3 From Domain Permutations To State Space Permutations

Assume an algebraic system net $\mathcal{N} = \langle\langle P, T, F\rangle, MSIG, \mathscr{A}, d, v, a, g, m_{init}\rangle$ using $MSIG$ as its signature and $\mathscr{A}$ as its $MSIG$-algebra. We say that a domain permutation group for $\mathscr{A}$ is a *domain permutation group for $\mathcal{N}$*. The next step is to extend domain permutations to the state space level of $\mathcal{N}$. Recall that $\mathbb{M}$ and $\mathbb{T}$ are the mutually disjoint sets of all markings and all transition instances of $\mathcal{N}$, respectively.

For markings, we permute the multi-sets of each individual place but do not permute the places.

Definition 5.2.5 (Permuted Markings) *Let $\psi_{\mathcal{S}} = (\psi_s)_{s \in \mathcal{S}}$ be a domain permutation for $\mathcal{N}$ and let M be a marking of $\mathcal{N}$. The corresponding permuted marking $\psi_{\mathbb{M}}(M)$ is defined by $(\psi_{\mathbb{M}}(M))(p) = \psi_{ms(d(p))}(M(p))$ for all $p \in P$.*

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

Definition 5.2.6 (Permuted Transition Instances) *Let $\psi_{\mathcal{S}} = (\psi_s)_{s\in\mathcal{S}}$ be a domain permutation for $\mathcal{N}$ and let $t_\beta = \langle t, \beta\rangle$ be a transition instance of $\mathcal{N}$. The corresponding permuted 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 $\Psi_{\mathcal{S}}$ for $\mathcal{N}$ is extended to the state space level by the following definition:

$$\Psi_{\mathrm{SS}(\mathcal{N})} = \{\psi_{\mathbb{M}} \cup \psi_{\mathbb{T}} \mid \psi_{\mathcal{S}} \in \Psi_{\mathcal{S}}\}.$$

But is $\Psi_{\mathrm{SS}(\mathcal{N})}$ a permutation group of the state space $\mathrm{SS}(\mathcal{N}) = \langle \mathbb{M}, \mathbb{T}, [\rangle, M_{init}\rangle$ of $\mathcal{N}$? Clearly, the answer is "yes".

Lemma 5.2.7 *$\langle \Psi_{\mathrm{SS}(\mathcal{N})}, \circ\rangle$ is a permutation group for* $\mathrm{SS}(\mathcal{N})$.

Proof. Because each ψ_s in $\psi_{\mathcal{S}}$ for a ground sort $s \in GS$ is a permutation on $\mathscr{D}_s^{\mathscr{A}}$, the corresponding function $\psi_{ms(s)}$ for the multi-set sort is a permutation on $\mathcal{MS}(\mathscr{D}_s^{\mathscr{A}})$, directly by Definition 5.2.1. 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 $\psi_{\mathrm{Ass}(\mathcal{X})}$ on variable assignments to $\mathcal{X}$ is a permutation on $\mathrm{Ass}(\mathscr{A}, \mathcal{X})$. 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 $\psi_{\mathrm{SS}(\mathcal{N})} = \psi_{\mathbb{M}} \cup \psi_{\mathbb{T}}$ is a permutation on $\mathbb{M} \cup \mathbb{T}$ such that $\psi_{\mathrm{SS}(\mathcal{N})}(\mathbb{M}) = \mathbb{M}$ and $\psi_{\mathrm{SS}(\mathcal{N})}(\mathbb{T}) = \mathbb{T}$.

We know from group theory that if $f : G \to H$ is homomorphism from a group $\langle G, \circ\rangle$ to a group $\langle H, *\rangle$, then $f(G')$ is a sub-group of H for any sub-group G' of G (see e.g. [Grimaldi 1985; Humphreys 1996]). We thus have to show that our extension from $\Psi_{\mathcal{S}}$ to $\Psi_{\mathrm{SS}(\mathcal{N})}$ fulfills the homomorphism condition (see Definition A.2.2):

$$(\psi_{\mathcal{S}} * \psi'_{\mathcal{S}})_{\mathrm{SS}(\mathcal{N})} = \psi_{\mathrm{SS}(\mathcal{N})} \circ \psi'_{\mathrm{SS}(\mathcal{N})}$$

for all $\psi_{\mathcal{S}}, \psi'_{\mathcal{S}} \in \Psi_{\mathcal{S}}$. Let $\psi''_{\mathcal{S}} = (\psi''_s)_{s\in\mathcal{S}}$ be the domain permutation $\psi''_{\mathcal{S}} = \psi_{\mathcal{S}} * \psi'_{\mathcal{S}}$ for any $\psi_{\mathcal{S}}, \psi'_{\mathcal{S}} \in \Psi_{\mathcal{S}}$. First, note that $\psi''_{ms(s)} = \psi_{ms(s)} \circ \psi'_{ms(s)}$ for each ground sort $s \in GS$: $(\psi_{ms(s)} \circ \psi'_{ms(s)})(b) = \psi_{ms(s)}(\psi'_{ms(s)}(b)) = \psi_{ms(s)}(\sum_{v\in\mathscr{D}_s^{\mathscr{A}}} b(v) \cdot \psi'_s(v)) = \sum_{v\in\mathscr{D}_s^{\mathscr{A}}} b(v) \cdot \psi_s(\psi'_s(v)) = \sum_{v\in\mathscr{D}_s^{\mathscr{A}}} b(v) \cdot ((\psi_s \circ \psi'_s)(v)) = \sum_{v\in\mathscr{D}_s^{\mathscr{A}}} b(v) \cdot \psi''_s(v) = \psi''_{ms(s)}(b)$ for all $b \in \mathcal{MS}(\mathscr{D}_s^{\mathscr{A}})$. Now for each marking $M \in \mathbb{M}$ and each place $p \in P$, it holds that

$$\begin{aligned}
\big(\psi''_{\mathbb{M}}(M)\big)(p) &= \psi''_{ms(d(p))}(M(p)) = (\psi_{ms(d(p))} \circ \psi'_{ms(d(p))})(M(p)) \\
&= \psi_{ms(d(p))}(\psi'_{ms(d(p))}(M(p))) = \big(\psi_{\mathbb{M}}(\psi'_{\mathbb{M}}(M))\big)(p) \\
&= \big(\psi_{\mathbb{M}}(M) \circ \psi'_{\mathbb{M}}(M)\big)(p).
\end{aligned}$$

Similarly, for each variable assignment β to variables in an $\mathcal{X}$, we can prove that $\psi''_{\mathrm{Ass}(\mathcal{X})}(\beta) = (\psi_{\mathrm{Ass}(\mathcal{X})} \circ \psi'_{\mathrm{Ass}(\mathcal{X})})(\beta)$. This in turn implies that for each transition instance $t_\beta \in \mathbb{T}$, we have $\psi''_{\mathbb{T}}(t_\beta) = \langle t, \psi''_{\mathrm{Ass}(v(t))}(\beta)\rangle = \langle t, (\psi_{\mathrm{Ass}(\mathcal{X})}(\psi'_{\mathrm{Ass}(\mathcal{X})}(\beta)))\rangle = (\psi_{\mathbb{T}} \circ \psi'_{\mathbb{T}})(t_\beta)$. Thus the homomorphism condition holds. □

In the proof above we actually show that the extension from $\Psi_{\mathcal{S}}$ to $\Psi_{\mathrm{SS}(\mathcal{N})}$ is a group homomorphism from $\Psi_{\mathcal{S}}$ to $\mathrm{Perm}(\mathrm{SS}(\mathcal{N}))$.

Example 5.2.8 Let us revisit the railroad Example 4.2.3 and consider domain permutations for the net in Figure 4.1. For each sort s, $\mathrm{Perm}(\mathscr{D}_s^{\mathscr{A}})$ denotes the set of all permutations of the domain of s. Now choose the ground domain permutation group Ψ_{GS} for *RRMAlg* to be the sub-group of $\bigotimes_{s\in GS}\mathrm{Perm}(\mathscr{D}_s^{\mathscr{A}})$ such that each $\psi_{GS} = (\psi_{\mathsf{Bool}}, \psi_{\mathsf{Secs}}, \psi_{\mathsf{Trains}}, \psi_{\mathsf{SecsXTrains}})$ belonging to Ψ_{GS} satisfies the following properties: (i) ψ_{Bool} is the identity mapping, (ii) ψ_{Secs} is a rotation of $\mathscr{D}^{\mathscr{A}}_{\mathsf{Secs}}$, 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) $\psi_{\mathsf{SecsXTrains}}(\langle s,t\rangle) = \langle\psi_{\mathsf{Secs}}(s), \psi_{\mathsf{Trains}}(t)\rangle$ for all $\langle s,t\rangle \in \mathscr{D}^{\mathscr{A}}_{\mathsf{Secs}} \times \mathscr{D}^{\mathscr{A}}_{\mathsf{Trains}}$. For instance, the domain permutation $\psi_{GS} = \Big(\psi_{\mathsf{Bool}} = \left(\begin{smallmatrix}\mathit{true}&\mathit{false}\\\mathit{true}&\mathit{false}\end{smallmatrix}\right), \psi_{\mathsf{Secs}} = \left(\begin{smallmatrix}0&1&2&3&4&5\\3&4&5&0&1&2\end{smallmatrix}\right), \psi_{\mathsf{Trains}} = \left(\begin{smallmatrix}a&b\\b&a\end{smallmatrix}\right), \psi_{\mathsf{SecsXTrains}} = \left(\begin{smallmatrix}\langle 0,a\rangle&\langle 1,a\rangle&\langle 2,a\rangle&\langle 3,a\rangle&\langle 4,a\rangle&\langle 5,a\rangle&\langle 0,b\rangle&\langle 1,b\rangle&\langle 2,b\rangle&\langle 3,b\rangle&\langle 4,b\rangle&\langle 5,b\rangle\\\langle 3,b\rangle&\langle 4,b\rangle&\langle 5,b\rangle&\langle 0,b\rangle&\langle 1,b\rangle&\langle 2,b\rangle&\langle 3,a\rangle&\langle 4,a\rangle&\langle 5,a\rangle&\langle 0,a\rangle&\langle 1,a\rangle&\langle 2,a\rangle\end{smallmatrix}\right)\Big)$ is in Ψ_{GS}.

When ψ_{GS} is extended to the multi-set sort $\mathsf{MSSecsXTrains}$ and is applied to the multi-set $2\cdot\langle 2,a\rangle + \langle 4,b\rangle$, the result is the multi-set $\langle 1,a\rangle + 2\cdot\langle 5,b\rangle$. If the variables i and x are of sorts Secs 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 $\{U\to\langle 0,a\rangle+\langle 4,b\rangle, V\to 1+2\}$. ♣

However, as the following example illustrates, $\Psi_{\mathrm{SS}(\mathcal{N})}$ is not necessarily a sub-group of $\mathrm{Aut}(\mathrm{SS}(\mathcal{N}))$.

Example 5.2.9 Like in the previous example, consider again the domain permutations for the net in Figure 4.1. This time we choose the ground domain permutation group to consist only of identity mapping and the mapping ψ_{GS} swapping the sections 0 and 1: $\psi_{GS} = (\psi_{\mathsf{Bool}}, \psi_{\mathsf{Secs}}, \psi_{\mathsf{Trains}}, \psi_{\mathsf{SecsXTrains}})$ such that $\psi_{\mathsf{Bool}} = \left(\begin{smallmatrix}\mathit{true}&\mathit{false}\\\mathit{true}&\mathit{false}\end{smallmatrix}\right)$, $\psi_{\mathsf{Secs}} = \left(\begin{smallmatrix}0&1&2&3&4&5\\1&0&2&3&4&5\end{smallmatrix}\right)$, $\psi_{\mathsf{Trains}} = \left(\begin{smallmatrix}a&b\\a&b\end{smallmatrix}\right)$ and $\psi_{\mathsf{SecsXTrains}} = \left(\begin{smallmatrix}\langle 0,a\rangle&\langle 1,a\rangle&\langle 2,a\rangle&\langle 3,a\rangle&\langle 4,a\rangle&\langle 5,a\rangle&\langle 0,b\rangle&\langle 1,b\rangle&\langle 2,b\rangle&\langle 3,b\rangle&\langle 4,b\rangle&\langle 5,b\rangle\\\langle 1,a\rangle&\langle 0,a\rangle&\langle 2,a\rangle&\langle 3,a\rangle&\langle 4,a\rangle&\langle 5,a\rangle&\langle 1,b\rangle&\langle 0,b\rangle&\langle 2,b\rangle&\langle 3,b\rangle&\langle 4,b\rangle&\langle 5,b\rangle\end{smallmatrix}\right)$. Now it holds that $\{U\to\langle 0,a\rangle+\langle 3,b\rangle, V\to 1+4\}\ [t_{\{i\mapsto 0,x\mapsto a\}}\rangle\ \{U\to\langle 1,a\rangle+\langle 3,b\rangle, V\to 4+5\}$ but under the action of ψ_{GS}, the transition instance $\psi_{\mathbb{T}}(t_{\{i\mapsto 0,x\mapsto a\}}) = t_{\{i\mapsto 1,x\mapsto a\}}$ is not enabled in $\psi_{\mathbb{M}}(\{U\to\langle 0,a\rangle+\langle 3,b\rangle, V\to 1+4\}) = \{U\to\langle 1,a\rangle+\langle 3,b\rangle, V\to 0+4\}$. Thus the state space automorphism condition of Definition 2.1.2 is violated. We may either say that the transition t breaks the state space automorphism induced by the domain permutation group or that the domain permutation group was not compatible with the transition. ♣

5.3 COMPATIBLE ASNS

To solve the problem of automorphism-breaking transitions described at the end of the last section, we exploit the concept of term compatibility (Section 3.3).

Definition 5.3.1 (Compatible ASNs) *We say that an algebraic system net $\mathcal{N}$ is* compatible *with a well-formed domain permutation group $\Psi_{\mathcal{S}}$ if*

1. *all transition guards $g(t)$ are compatible with $\Psi_{\mathcal{S}}$ and*
2. *all arc annotations $a(p,t), a(t,p)$ are compatible with $\Psi_{\mathcal{S}}$.*

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 the 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 work, namely that if we use a algebraic net with a compatible well-formed domain permutation group, the group will induce an automorphism group for the state space of the net.

Theorem 5.3.2 *Let $\mathcal{N}$ be an algebraic system net compatible with a well-formed domain permutation group $\Psi_{\mathcal{S}}$. Then $\Psi_{\mathrm{SS}(\mathcal{N})}$ is an automorphism group for $\mathrm{SS}(\mathcal{N})$.*

Proof. By Theorem 5.2.7, $\Psi_{\mathrm{SS}(\mathcal{N})}$ is a permutation group for $\mathrm{SS}(\mathcal{N})$. 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 $\psi_{\mathrm{SS}(\mathcal{N})} \in \Psi_{\mathrm{SS}(\mathcal{N})}$ it holds that

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

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

Recall that $\psi_{\mathrm{SS}(\mathcal{N})}(t_\beta) = \psi_{\mathbb{T}}(t_\beta) = \langle t, \psi_{\mathrm{Ass}(\mathcal{X})}(\beta)\rangle$. The guard value of $\psi_{\mathrm{SS}(\mathcal{N})}(t_\beta)$, $eval_{\psi_{\mathrm{Ass}(\mathcal{X})}(\beta)}(g(t))$, is equal to $\psi_{\mathsf{Bool}}(eval_\beta(g(t)))$ by Lemma 3.4.2. This 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 $\psi_{\mathrm{SS}(\mathcal{N})}(t_\beta)$.

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 5.2.2). 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 $t^-_{\psi(\beta)}(p) = eval_{\psi_{\mathrm{Ass}(v(t))}(\beta)}(a(p,t))$ equals by the definition of compatible terms (Definition 3.3.1) to $\psi_{ms(d(p))}(eval_\beta(a(p,t)))$. On the other hand, $t^-_\beta(p) = eval_\beta(a(p,t))$ and therefore we have shown that $t^-_{\psi(\beta)}(p) = \psi_{ms(d(p))}(t^-_\beta(p))$. In similar vein we can show that if $\langle t,p\rangle \in F$, then $t^+_{\psi(\beta)}(p) = \psi_{ms(d(p))}(t^+_\beta(p))$.

As a consequence of $t^-_{\psi(\beta)}(p) = \psi_{ms(d(p))}(t^-_\beta(p))$ and the definition of domain permutations, $t^-_{\psi(\beta)}(p) = \mathfrak{err}$ iff $\psi_{ms(d(p))}(t^-_\beta(p)) = \mathfrak{err}$. Similarly for $t^+_{\psi(\beta)}$. For markings we have $\big(\psi_{\mathrm{SS}(\mathcal{N})}(M)\big)(p) = \psi_{ms(d(p))}(M(p))$ by Definition 5.2.5. Now $M(p) \geq t^-_\beta(p)$ iff $\psi_{ms(d(p))}(M(p)) \geq \psi_{ms(d(p))}(t^-_\beta(p))$ iff

$\psi_{\mathrm{SS}(\mathcal{N})}(M)(p) \geq t^-_{\psi(\beta)}(p)$ by Lemma 5.2.2 and the result above. Combined with the result of guards, we see that t_β is enabled in M_1 iff $\psi_{\mathrm{SS}(\mathcal{N})}(t_\beta)$ is enabled in $\psi_{\mathrm{SS}(\mathcal{N})}(M_1)$.

Finally, assume that t_β is enabled in M_1 and $M_1\ [t_\beta\rangle\ M_2$. Then (and only then) $\psi_{\mathrm{SS}(\mathcal{N})}(t_\beta)$ is enabled in $\psi_{\mathrm{SS}(\mathcal{N})}(M_1)$. Using the results above for $t^-_{\psi(\beta)}(p)$ and $t^+_{\psi(\beta)}(p)$ we have that $\big(\psi_{\mathrm{SS}(\mathcal{N})}(M_1)\big)(p) - t^-_{\psi(\beta)}(p) + t^+_{\psi(\beta)}(p)$ equals to $\psi_{ms(d(p))}(M_1(p)) - \psi_{ms(d(p))}(t^-_\beta(p)) + \psi_{ms(d(p))}(t^+_\beta(p))$, which by linearity of $\psi_{ms(d(p))}$ (Lemma 5.2.2) and by the definition of M_2 is equal to $\psi_{ms(d(p))}(M_1(p) - t^-_\beta(p) + t^+_\beta(p)) = \psi_{ms(d(p))}(M_2(p)) = \psi_{\mathrm{SS}(\mathcal{N})}(M_2)(p)$. $\square$

6 EXTENDED WELL-FORMED NETS

To demonstrate the use of the concepts and theory developed in this work, we give an illustrative net class in this chapter. We do this by imposing constraints on the multi-set signatures and algebras used in the nets. The resulting class of signatures and algebras is called the class of *Extended Well-Formed Net* signatures and algebras, respectively, and the net class using these signatures and algebras is called *Extended Well-Formed Nets* (EWF-nets or EWFNs).

Multi-set sorts of EWFN signatures and algebras are as usual while the ground sorts consists of two sets. First, there are some *basic* sorts such as Booleans, natural numbers, the black token sort, bytes, rotation and permutations sorts (described in a moment) and so on. Building on these basic sorts, a superset of sorts called *structured* sorts is defined by using traditional programming language constructs such as lists, arrays and sets.

The set of basic sorts is partitioned into three sets according to the domain permutations allowed to the sorts. The first set is called the set of *ordered* sorts for which only identity permutations are allowed. Consequently, all possible operations concerning these sorts are compatible with the allowed domain permutation. The second set is the set of *rotation* sorts. The domains of these sorts are assumed to finite and ordered in some total order. The allowed domain permutations consists of rotations of the elements meaning that only (in)equality test operations and the successor functions are compatible. The third set of sorts is that of *unordered* sorts for which all the domain permutations are allowed implying that only (in)equality test operations are compatible. Needless to say, this approach is largely inspired by the works of Ip and Dill [1996], and Chiola et al. [1991].

We proceed as follows. First we study the basic sorts and their sets of allowed domain permutations. Structured sorts are then introduced in Section 6.2 where the allowed domain permutation groups for basic sorts are also extended in an unambiguous way to corresponding allowed domain permutation groups for structured sorts. The standard operations concerning the chosen structured sorts are also studied in that section. This study shows that the standard operations are very well compatible with the allowed domain permutations, justifying our choices. The allowed domain permutations for structured sorts are then in turn extended to multi-set sorts in the usual way (Definition 5.2.1). We give an illustrative example of these concepts and a simple input language syntax for EWF-nets in Section 6.4. Section 6.5 discusses the important topic of deciding whether two markings of an EWF-net are symmetric. It is proven that this problem is polynomial-time many-one equivalent to the graph isomorphism problem. In Section 6.6 we will the final gap needed for implementing a full-fledged analysis tool for EWF-nets: we describe and briefly discuss a variant of the very expressive temporal logic CTL* for EWF-nets.

Assume a multi-set signature $MSIG = \langle \mathcal{S}, \mathscr{F}, ms \rangle$ and a corresponding multi-set algebra $\mathcal{E} = \langle \mathscr{D}^{\mathcal{E}}, \mathscr{F}^{\mathcal{E}} \rangle$. In the following sections we define what constraints they have to fulfill in order to be called as EWFN-signatures and -algebras. In the course we also build the *set* $\Theta_{\mathcal{S}}$ *of allowed domain permu-*

tations for $\mathcal{E}$ which is a well-formed domain permutation group.

6.1 BASIC SORTS

The set of basic sorts, $Basic\mathcal{S}$, is partitioned into three disjoint subsets: *ordered* sorts $Ord\mathcal{S}$, *rotation* sorts $Rot\mathcal{S}$ and *unordered* sorts $Unord\mathcal{S}$.

The domains of ordered sorts are arbitrary non-empty sets and the allowed domain permutations for them leave the domains intact. That is, if $s \in Ord\mathcal{S}$, then $\Theta_s = \{\mathrm{Id}(\mathscr{D}_s^{\mathcal{E}})\}$ (the identity mapping). The sorts Bool and Nat are assumed to be included in $Ord\mathcal{S}$. All operations that involve only ordered sorts are allowed. And since $\mathsf{Bool} \in Ord\mathcal{S}$, $\Theta_{\mathcal{S}}$ will be well-formed.

For each rotation sort $s \in Rot\mathcal{S}$, it is assumed that the domain $\mathscr{D}_s^{\mathcal{E}}$ is non-empty and finite. Moreover, there is an attached successor function succ_s such that $\bigcup_{k=0}^{|\mathscr{D}_s^{\mathcal{E}}|-1} \mathrm{succ}_s^k(v) = \mathscr{D}_s^{\mathcal{E}}$ holds for each $v \in \mathscr{D}_s^{\mathcal{E}}$.[1] Now the set of allowed permutations Θ_s is $\{\mathrm{succ}_s^k \mid 0 \leq k \leq |\mathscr{D}_s^{\mathcal{E}}| - 1\}$. This is a group under $\circ$ since each succ_s^k is a permutation on $\mathscr{D}_s^{\mathcal{E}}$, succ_s^0 is the identity mapping, $\mathrm{succ}_s^k \circ \mathrm{succ}_s^{k'} = \mathrm{succ}_s^{k+k' \mod |\mathscr{D}_s^{\mathcal{E}}|}$ and the associativity holds by the associativity of $\circ$. The compatible operations include (i) $=, \neq \in \mathscr{F}_{s.s,\mathsf{Bool}}$, the usual equality and inequality tests and (ii) k-$\mathsf{succ} \in \mathscr{F}_{s,s}$, the *$k$-successor* function defined by $\mathsf{k\text{-}succ}^{\mathcal{E}}(v) = \mathrm{succ}_s^k(v)$ for each $1 \leq k \leq |\mathscr{D}_s^{\mathcal{E}}| - 1$. We may write succ or next for 1-succ and prev for $(|\mathscr{D}_s^{\mathcal{E}}| - 1)$-$\mathsf{succ}$.

The domain of each unordered sort $s \in Unord\mathcal{S}$ is assumed to be non-empty. The group of allowed permutations is the set of all permutations on the domain i.e. $\Theta_s = \mathrm{Perm}(\mathscr{D}_s^{\mathcal{E}})$. The compatible operations include the usual equality and inequality tests $=, \neq \in \mathscr{F}_{s.s,\mathsf{Bool}}$.

Note that rotation and unordered sorts may also have other operations, such as constants, that are not compatible with allowed domain permutations. However, use of these operations should be limited into description of symbolic initial markings (whose symmetricity is not usually required).

The allowed domain permutation group for the set of basic sorts is the "exterior direct product" $\Theta_{Basic\mathcal{S}} = \bigotimes_{s \in Basic\mathcal{S}} \Theta_s$.

6.2 STRUCTURED SORTS

We now show a way how to build structured sorts over basic sorts. We use the familiar constructs such as lists, arrays and sets, found in many programming languages. This new set of sorts is called the set of *structured sorts* and it is a proper infinite superset of the set of basic sorts. We then extend the allowed domain permutations of basic sorts to structured sorts in a natural and unambiguous way. Finally, we introduce a large set of standard operations concerning structured sorts and study their compatibility with allowed domain permutations. Luckily it turns out that the standard operations are usually compatible with allowed domain permutations.

[1] For a function $f : A \to A$, f^k means $\underbrace{f \circ \ldots \circ f}_{k \text{ times}}$ and f^0 is the identity mapping.

6.2.1 Sorts and Domains

For a set of basic sorts, $Basic\mathcal{S}$, the corresponding set of *structured sorts over* $Basic\mathcal{S}$, $Struct\mathcal{S}(Basic\mathcal{S})$, is the smallest set fulfilling the following inductive rules.

1. $s \in Struct\mathcal{S}(Basic\mathcal{S})$ for each $s \in Basic\mathcal{S}$.
2. $\mathsf{List}(s) \in Struct\mathcal{S}(Basic\mathcal{S})$ if $s \in Struct\mathcal{S}(Basic\mathcal{S})$.
3. $\mathsf{Struct}(\text{field1} : s_1, \ldots, \text{fieldn} : s_n) \in Struct\mathcal{S}(Basic\mathcal{S})$ if $s_1, \ldots, s_n \in Struct\mathcal{S}(Basic\mathcal{S})$ for $n \geq 1$ (field1, ..., fieldn are arbitrary, mutually distinct *field names*).
4. $\mathsf{Set}(s) \in Struct\mathcal{S}(Basic\mathcal{S})$ if $s \in Struct\mathcal{S}(Basic\mathcal{S})$.
5. $\mathsf{Array}(n, s) \in Struct\mathcal{S}(Basic\mathcal{S})$ if $n \in \mathbb{N}_+$ and $s \in Struct\mathcal{S}(Basic\mathcal{S})$.
6. $\mathsf{AssocArray}(s_1, s_2) \in Struct\mathcal{S}(Basic\mathcal{S})$ if $s_1, s_2 \in Struct\mathcal{S}(Basic\mathcal{S})$.
7. $\mathsf{Union}(s_1, \ldots, s_n) \in Struct\mathcal{S}(Basic\mathcal{S})$ if $n \in \mathbb{N}_+$ and $s_1, \ldots, s_n \in Struct\mathcal{S}(Basic\mathcal{S})$.

$Struct\mathcal{S}(Basic\mathcal{S})$ is used as the set of ground sorts in the EWFN-signatures and -algebras.

The domains of sorts are defined in the way that corresponds to the usual idea of the defined data types.

1. The domain $\mathscr{D}^{\mathcal{E}}_s$ of each basic sort $s \in Basic\mathcal{S}$ is as defined earlier.
2. If $s = \mathsf{List}(s_1)$, then $\mathscr{D}^{\mathcal{E}}_s = (\mathscr{D}^{\mathcal{E}}_{s_1})^*$.
3. If $s = \mathsf{Struct}(\text{f1} : s_1, \ldots, \text{fn} : s_n)$, then $\mathscr{D}^{\mathcal{E}}_s = \mathscr{D}^{\mathcal{E}}_{s_1} \times \cdots \times \mathscr{D}^{\mathcal{E}}_{s_n}$.
4. If $s = \mathsf{Set}(s_1)$, then $\mathscr{D}^{\mathcal{E}}_s = \wp(\mathscr{D}^{\mathcal{E}}_{s_1})$, where $\wp(\mathscr{D}^{\mathcal{E}}_{s_1})$ is the set of all subsets of $\mathscr{D}^{\mathcal{E}}_{s_1}$ (the powerset of $\mathscr{D}^{\mathcal{E}}_{s_1}$).
5. If $s = \mathsf{Array}(n, s_1)$, then $\mathscr{D}^{\mathcal{E}}_s = \underbrace{\mathscr{D}^{\mathcal{E}}_s \times \cdots \times \mathscr{D}^{\mathcal{E}}_s}_{n \text{ times}}$.
6. If $s = \mathsf{AssocArray}(s_1, s_2)$, then $\mathscr{D}^{\mathcal{E}}_s$ is the set of all partial functions from $\mathscr{D}^{\mathcal{E}}_{s_1}$ to $\mathscr{D}^{\mathcal{E}}_{s_2}$.[2]
7. If $s = \mathsf{Union}(s_1, \ldots, s_n)$, then we define that $\mathscr{D}^{\mathcal{E}}_s = \{\{s_1\} \times \mathscr{D}^{\mathcal{E}}_{s_1}\} \cup \cdots \cup \{\{s_n\} \times \mathscr{D}^{\mathcal{E}}_{s_n}\}$.

Note that we tag the elements in the union domains by the sort names to be able to distinguish the elements of overlapping constituent sorts. Thus the sort name is retrievable later for some purposes, e.g. we can use the sort name to denote a message type (see Section 6.4).

[2] A partial function from a set A to a set B is a subset f of $A \times B$ such that each $a \in A$ appears at most once as the first component of pairs in f. If $\langle a, b\rangle \in f$ we write $f(a) = b$ and say that $f(a)$ is defined; otherwise we write $f(a) = \nearrow$ and say that $f(a)$ is undefined. $f[a \mapsto b]$ denotes the partial function from A to B such that $f[a \mapsto b](a) = b$ and $f[a \mapsto b](a') = f(a')$ for all $a' \in A$ such that $a' \neq a$.

6.2.2 Domain Permutations

The allowed domain permutations of basic sorts are now extended in an unambiguous way to allowed domain permutations of structured sorts.

Each $\theta_{Basic\mathcal{S}} = (\theta_s)_{s \in Basic\mathcal{S}} \in \Theta_{Basic\mathcal{S}}$ is extended to the corresponding domain permutation $\theta_{Struct\mathcal{S}(Basic\mathcal{S})} = (\theta_s)_{s \in Struct\mathcal{S}(Basic\mathcal{S})}$ for structured sorts by the following inductive definition.

1. If $s \in Basic\mathcal{S}$, then θ_s is already defined.
2. If $s = \mathsf{List}(s_1)$, then for all $v = \langle v_1, \ldots, v_n \rangle \in (\mathscr{D}^{\mathcal{E}}_{s_1})^*$ we define that $\theta_s(v) = \langle \theta_{s_1}(v_1), \ldots, \theta_{s_1}(v_n) \rangle$.
3. If $s = \mathsf{Struct}(\mathrm{f1} : s_1, \ldots, \mathrm{fn} : s_n)$, then for all $v = \langle v_1, \ldots, v_n \rangle \in \mathscr{D}^{\mathcal{E}}_{s_1} \times \cdots \times \mathscr{D}^{\mathcal{E}}_{s_n}$ we define that $\theta_s(v) = \langle \theta_{s_1}(v_1), \ldots, \theta_{s_n}(v_n) \rangle$.
4. If $s = \mathsf{Set}(s_1)$ and $v \in \wp(\mathscr{D}^{\mathcal{E}}_{s_1})$, then $\theta_s(v) = \{\theta_{s_1}(a) \mid a \in v\}$.
5. If $s = \mathsf{Array}(n, s_1)$ and $v = \langle v_1, \ldots, v_n \rangle \in (\mathscr{D}^{\mathcal{E}}_{s_1})^n$, then we define that $\theta_s(v) = \langle \theta_{s_1}(v_1), \ldots, \theta_{s_1}(v_n) \rangle$.
6. If $s = \mathsf{AssocArray}(s_1, s_2)$ and $a \in \mathscr{D}^{\mathcal{E}}_s$ is a partial function from $\mathscr{D}^{\mathcal{E}}_{s_1}$ to $\mathscr{D}^{\mathcal{E}}_{s_2}$, then $\theta_s(a)$ is the partial function such that $\langle i, v \rangle \in a$ iff $\langle \theta_{s_1}(i), \theta_{s_2}(v) \rangle \in \theta_s(a)$.
7. If $s = \mathsf{Union}(s_1, \ldots, s_n)$ and $v = \langle s', v' \rangle$ for some $s' \in \{s_1, \ldots, s_n\}$ and $v' \in \mathscr{D}^{\mathcal{E}}_{s'}$, then $\theta_s(v) = \langle s', \theta_{s'}(v') \rangle$.

$\Theta_{Struct\mathcal{S}(Basic\mathcal{S})} = \{\theta_{Struct\mathcal{S}(Basic\mathcal{S})} \mid \theta_{Basic\mathcal{S}} \in \Theta_{Basic\mathcal{S}}\}$ is the group of allowed domain permutations for structured sorts corresponding to the group of allowed domain permutations $\Theta_{Basic\mathcal{S}}$ for basic sorts. It can be easily seen that $\Theta_{Struct\mathcal{S}(Basic\mathcal{S})}$ is a sub-group of $\bigotimes_{s \in Struct\mathcal{S}(Basic\mathcal{S})} \mathrm{Perm}(\mathscr{D}^{\mathcal{E}}_s)$ and that there is actually a group isomorphism from $\Theta_{Basic\mathcal{S}}$ to $\Theta_{Struct\mathcal{S}(Basic\mathcal{S})}$.

6.2.3 Operations for Structured Sorts

In the following we present the operations for each structured sort type all of which are compatible with the set $\Theta_{\mathcal{S}}$ of allowed permutations. Some of these operations have erroneous invocations in which case we return the special $\mathfrak{err}$ element. In fact, these operations are the reason why error algebras had to be introduced in the first place. For a net class simpler than EWFNs normal algebras would have sufficed.

Naturally, the (in)equality test operations $=, \neq \in \mathscr{F}_{s.s,\mathsf{Bool}}$ for each structured sort s are included. Thus we can, for example, compare two lists $l, l' \in \mathscr{D}^{\mathcal{E}}_{\mathsf{List}(s)}$ for equality by using the operation $= \in \mathscr{F}_{\mathsf{List}(s).\mathsf{List}(s),\mathsf{Bool}}$.

Lists. Assume a list sort $\mathsf{List}(s)$. We consider the following operations.

- OPERATION: $\mathsf{emptyList} \in \mathscr{F}_{\epsilon,\mathsf{List}(s)}$
  DEFINITION: $\mathsf{emptyList}^{\mathcal{E}}() = \langle\rangle$

This is the empty list constant. Since an empty list is always permuted to itself (Sec. 6.2.2), the operation is compatible with all allowed domain permutations.

- OPERATION: $\mathsf{isEmpty} \in \mathscr{F}_{\mathsf{List}(s),\mathsf{Bool}}$

  DEFINITION: $\mathsf{isEmpty}^{\mathcal{E}}(l) = \mathit{true}$ iff $l = \langle\rangle$

  The operation tells whether a list is empty. The operation is clearly compatible with all allowed domain permutations because a list is permuted to an empty list if and only if it is empty (and because all allowed domain permutations are well-formed, not permuting the Booleans).

- OPERATION: $\mathsf{length} \in \mathscr{F}_{\mathsf{List}(s),\mathsf{Nat}}$

  DEFINITION: $\mathsf{length}^{\mathcal{E}}(\langle v_1, v_2, \ldots, v_n\rangle) = n$

  Returns the number of items in the list. The operation is clearly compatible with all allowed domain permutations because the length of a list is not affected by allowed permutations and because the domain of Nat is not permuted.

- OPERATIONS: $\mathsf{first}, \mathsf{last} \in \mathscr{F}_{\mathsf{List}(s),s}$

  DEFINITIONS: $\mathsf{first}^{\mathcal{E}}(\langle v_1, v_2, \ldots, v_n\rangle) = v_1$ if $n \geq 1$ and $\mathfrak{err}$ if $n = 0$
  $\mathsf{last}^{\mathcal{E}}(\langle v_1, v_2, \ldots, v_n\rangle) = v_n$ if $n \geq 1$ and $\mathfrak{err}$ if $n = 0$

  The operations return the first (last) element of the list. First consider the case of non-empty list $l = \langle v_1, \ldots, v_n\rangle$, where $n \geq 1$. Now $\mathsf{first}^{\mathcal{E}}(\langle v_1, \ldots, v_n\rangle) = v_1$ and $\mathsf{first}^{\mathcal{E}}(\theta_{\mathsf{List}(s)}(\langle v_1, \ldots, v_n\rangle))$ is equivalent to $\mathsf{first}^{\mathcal{E}}(\langle \theta_s(v_1), \ldots, \theta_s(v_n)\rangle) = \theta_s(v_1)$. The case is similar for last and thus the operations would be compatible with all allowed 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. It is most convenient to choose this value to be the special error value.

- OPERATIONS: $\mathsf{removeFirst}, \mathsf{removeLast} \in \mathscr{F}_{\mathsf{List}(s),\mathsf{List}(s)}$

  DEFINITIONS:
  $$\mathsf{removeFirst}^{\mathcal{E}}(\langle v_1, v_2, \ldots, v_n\rangle) = \begin{cases} \langle v_2, \ldots, v_n\rangle & \text{if } n \geq 1 \\ \mathfrak{err} & \text{if } n = 0 \end{cases}$$
  $$\mathsf{removeLast}^{\mathcal{E}}(\langle v_1, \ldots, v_{n-1}, v_n\rangle) = \begin{cases} \langle v_1, \ldots, v_{n-1}\rangle & \text{if } n \geq 1 \\ \mathfrak{err} & \text{if } n = 0 \end{cases}$$

  The delete operations $\mathsf{removeFirst}, \mathsf{removeLast} \in \mathscr{F}_{\mathsf{List}(s),\mathsf{List}(s)}$ are compatible with all allowed domain permutations since the empty list case is defined to return the special, non-permuted error value.

- OPERATIONS: $\mathsf{addFirst}, \mathsf{addLast} \in \mathscr{F}_{\mathsf{List}(s).s,\mathsf{List}(s)}$

  DEFINITIONS: $\mathsf{addFirst}^{\mathcal{E}}(\langle v_1, \ldots, v_n\rangle, v) = \langle v, v_1, \ldots, v_n\rangle$
  $\mathsf{addLast}^{\mathcal{E}}(\langle v_1, \ldots, v_n\rangle, v) = \langle v_1, \ldots, v_n, v\rangle$

  These operations append the list in the beginning/end and are clearly compatible with all allowed domain permutations

- OPERATION: $\mathsf{conc} \in \mathscr{F}_{\mathsf{List}(s).\mathsf{List}(s),\mathsf{List}(s)}$

DEFINITION: $\mathsf{conc}^{\mathcal{E}}(\langle v_1, v_2, \dots, v_n\rangle, \langle u_1, u_2, \dots, u_m\rangle) = \langle v_1, v_2, \dots, v_n, u_1, u_2, \dots, u_m\rangle$

The list concatenation operation is also compatible with all allowed domain permutations for obvious reasons.

- OPERATIONS: $\mathsf{element} \in \mathscr{F}_{\mathsf{List}(s).\mathsf{Nat},s}$ and $\mathsf{setElement} \in \mathscr{F}_{\mathsf{List}(s).\mathsf{Nat}.s,s}$

  DEFINITIONS: $\mathsf{element}^{\mathcal{E}}(\langle v_1, ..., v_n\rangle, i) = \begin{cases} v_i & \text{if } 1 \le i \le n \\ \mathfrak{err} & \text{otherwise} \end{cases}$

  $\mathsf{setElement}^{\mathcal{E}}(\langle v_1, \dots, v_{i-1}, v_i, v_{i+1}, \dots, v_n\rangle, i, v) = \begin{cases} \langle v_1, \dots, v_{i-1}, v, v_{i+1}, \dots, v_n\rangle & \text{if } 1 \le i \le n \\ \mathfrak{err} & \text{otherwise} \end{cases}$

  Random access operations for lists. Compatibility clearly requires that the index value i is not permuted i.e. the domain of Nat is not permuted, which is the case with allowed domain permutations.

Actually, because lists are a generalized case of two important data types, namely stacks and first-in first-out buffers (FIFOs), we have covered them, too. To have a FIFO sort, just rename the sort to FIFO and use the operations so that $\mathsf{addLast}$ becomes send and $\mathsf{receive}$ is simulated by first and $\mathsf{removeFirst}$. Similarly, for the sort Stack, substitute push for $\mathsf{addFirst}$ and simulate pop with first and $\mathsf{removeFirst}$.

Structures. For a structure sort $\mathsf{Struct}(\mathrm{f1} : s_1, \dots, \mathrm{fn} : s_n)$ we do not need many operations.

- OPERATION: $\mathsf{getf}i \in \mathscr{F}_{\mathsf{Struct}(\mathrm{f1}:s_1,\dots,\mathrm{fn}:s_n),s_i}$, or $\mathsf{f}i$, for each $1 \le i \le n$

  DEFINITION: $\mathsf{getf}i^{\mathcal{E}}(\langle v_1, \dots, v_n\rangle) = v_i$

  The operation $\mathsf{getf}i$ simply returns the value of the field "fi".

- OPERATION: $\mathsf{setf}i \in \mathscr{F}_{\mathsf{Struct}(\mathrm{f1}:s_1,\dots,\mathrm{fn}:s_n).s_i,\mathsf{Struct}(\mathrm{f1}:s_1,\dots,\mathrm{fn}:s_n)}$, $1 \le i \le n$

  DEFINITION: $\mathsf{setf}i^{\mathcal{E}}(\langle v_1, \dots, v_{i-1}, v_i, v_{i+1} \dots, v_n\rangle, v_i') = \langle v_1, \dots, v_{i-1}, v_i', v_{i+1} \dots, v_n\rangle$

  Similarly, the operations $\mathsf{setf}i$ set the value of the field "fi".

- OPERATION: $\mathsf{makeStruct} \in \mathscr{F}_{s_1.\dots.s_n,\mathsf{Struct}(\mathrm{f1}:s_1,\dots,\mathrm{fn}:s_n)}$ or $\langle\underbrace{\cdot, \dots, \cdot}_{n \text{ times}}\rangle$

  DEFINITION: $\mathsf{makeStruct}^{\mathcal{E}}(v_1, \dots, v_n) = \langle v_1, \dots, v_n\rangle$

  The operation $\mathsf{makeStruct}$ returning the structure composed of the argument values is useful in some situations. For instance, it is more convenient to have an expression $\mathsf{makeStruct}(x, \mathsf{5})$ (or, equivalently, $\langle x, \mathsf{5}\rangle$) on an arc of a net than have a transition variable y of sort $\mathsf{Struct}(\mathrm{f1} : \mathsf{Nat}, \mathrm{f2} : \mathsf{Nat})$ on the arc and the comparison $\mathsf{f1}(y) = x \wedge \mathsf{f2}(y) = \mathsf{5}$ in the transition guard.

All these operations are compatible with all allowed domain permutations.

Sets. Let $V_1, V_2 \in \wp(\mathscr{D}_s^{\mathcal{E}})$ be two sets belonging to the domain of a set sort $\mathsf{Set}(s)$.

- OPERATION: $\mathsf{card} \in \mathscr{F}_{\mathsf{Set}(s),\mathsf{Nat}}$ or $|\cdot|$ in mix-fix notation
  DEFINITION: $\mathsf{card}^{\mathcal{E}}(V_1) = |V_1|$

  The operation card returning the cardinality of a set is compatible with all allowed domain permutations since they do not permute the domain of Nat.

- OPERATION: $\mathsf{in} \in \mathscr{F}_{s.\mathsf{Set}(s),\mathsf{Bool}}$ aka $\cdot \in \cdot$
  DEFINITION: $\mathsf{in}^{\mathcal{E}}(v, V_1) = \mathit{true} \Leftrightarrow v \in V_1$

  The inclusion test operation is compatible with all allowed domain permutations since $\mathsf{in}^{\mathcal{E}}(\theta_s(v), \theta_{\mathsf{Set}(s)}(V_1)) = \mathit{true}$ iff $\theta_s(v) \in \theta_{\mathsf{Set}(s)}(V_1)$ iff $\theta_s(v) \in \{\theta_s(v') \mid v' \in V_1\}$ iff $v \in V_1$ iff $\mathsf{in}^{\mathcal{E}}(v, V_1) = \mathit{true} = \theta_{\mathsf{Bool}}(\mathit{true})$.

- OPERATION: $\mathsf{complement} \in \mathscr{F}_{\mathsf{Set}(s),\mathsf{Set}(s)}$ aka $\bar{\cdot}$
  DEFINITION: $\mathsf{complement}^{\mathcal{E}}(V_1) = \{v \in \mathscr{D}_s^{\mathcal{E}} \mid v \notin V_1\}$

  The set complement operation is compatible with all allowed domain permutations since

$$\begin{aligned}
\mathsf{complement}^{\mathcal{E}}(\theta_{\mathsf{Set}(s)}(V_1)) &= \{v \in \mathscr{D}_s^{\mathcal{E}} \mid v \notin \theta_{\mathsf{Set}(s)}(V_1)\} \\
&= \{v \in \mathscr{D}_s^{\mathcal{E}} \mid v \notin \{\theta_s(v') \mid v' \in V_1\}\} \\
&= \{v \in \mathscr{D}_s^{\mathcal{E}} \mid v \in \{\theta_s(v') \mid v' \notin V_1\}\} \\
&= \{\theta_s(v') \mid v' \in \overline{V_1}\} \\
&= \theta_{\mathsf{Set}(s)}(\overline{V_1}) \\
&= \theta_{\mathsf{Set}(s)}(\mathsf{complement}^{\mathcal{E}}(V_1))
\end{aligned}$$

  by using bijectivity of θ_s.

- OPERATION: $\mathsf{union} \in \mathscr{F}_{\mathsf{Set}(s).\mathsf{Set}(s),\mathsf{Set}(s)}$ aka $\cdot \cup \cdot$
  DEFINITION: $\mathsf{union}^{\mathcal{E}}(V_1, V_2) = \{v \in \mathscr{D}_s^{\mathcal{E}} \mid v \in V_1 \vee v \in V_2\}$

  The set union operation is compatible with all allowed domain permutations because

$$\begin{aligned}
&\mathsf{union}^{\mathcal{E}}(\theta_{\mathsf{Set}(s)}(V_1), \theta_{\mathsf{Set}(s)}(V_2)) \\
&= \{v \in \mathscr{D}_s^{\mathcal{E}} \mid v \in \theta_{\mathsf{Set}(s)}(V_1) \vee v \in \theta_{\mathsf{Set}(s)}(V_2)\} \\
&= \{v \in \mathscr{D}_s^{\mathcal{E}} \mid v \in \{\theta_s(v') \mid v' \in V_1\} \vee v \in \{\theta_s(v') \mid v' \in V_2\}\} \\
&= \{v \in \mathscr{D}_s^{\mathcal{E}} \mid v \in \{\theta_s(v') \mid v' \in V_1 \vee v' \in V_2\}\} \\
&= \{\theta_s(v') \mid v' \in V_1 \vee v' \in V_2\} \\
&= \theta_{\mathsf{Set}(s)}(\mathsf{union}^{\mathcal{E}}(V_1, V_2)).
\end{aligned}$$

- OPERATION: $\mathsf{intersection} \in \mathscr{F}_{\mathsf{Set}(s).\mathsf{Set}(s),\mathsf{Set}(s)}$ aka $\cdot \cap \cdot$
  DEFINITION: $\mathsf{intersection}^{\mathcal{E}}(V_1, V_2) = \{v \in \mathscr{D}_s^{\mathcal{E}} \mid v \in V_1 \wedge v \in V_2\}$

  By reasoning similar to the case of set union operation, we can show that the set intersection operation is compatible with all allowed domain permutations.

- OPERATION: $\mathsf{setMinus} \in \mathscr{F}_{\mathsf{Set}(s).\mathsf{Set}(s),\mathsf{Set}(s)}$ aka $\cdot \setminus \cdot$

DEFINITION: $\mathsf{setMinus}^{\mathcal{E}}(V_1, V_2) = \{v \in \mathscr{D}_s^{\mathcal{E}} \mid v \in V_1 \land v \notin V_2\}$

Compatible with all allowed domain permutations, refer to the set union operation.

- OPERATION: "$\mathsf{All}\ x : s\ \mathsf{s.t.}\ T(x_1, \dots, x_n, x)$" $\in \mathscr{F}_{s_1 \dots s_n, \mathsf{Set}(s)}$ where $T \in \mathbf{T}_{\mathsf{Bool}}(\mathcal{X})$ and $\mathcal{X}$ consists of the variable x of sort s and $x_1, \dots, x_n$ of sorts $s_1, \dots, s_n$, respectively

  DEFINITION: "$\mathsf{All}\ x : s\ \mathsf{s.t.}\ T(x_1, \dots, x_n, x)$"$^{\mathcal{E}}(v_1, \dots, v_n) =$
  $$\{v \in \mathscr{D}_s^{\mathcal{E}} \mid eval_{\{x_1 \mapsto v_1, \dots, x_n \mapsto v_n, x \mapsto v\}}(T) = true\}$$

  See Section 5.2.2.

Arrays. The difference between the operations for structures and arrays is that array operations have the index as an argument. Let $\mathsf{Array}(n, s)$ be an array sort.

- OPERATION: $\mathsf{newArray} \in \mathscr{F}_{s, \mathsf{Array}(n,s)}$

  DEFINITION: $\mathsf{newArray}^{\mathcal{E}}(v) = \langle \underbrace{v, \dots, v}_{n \text{ times}} \rangle$

  Builds an n-element array whose elements are initialized to the argument value v. Clearly compatible with all allowed domain permutations.

- OPERATIONS: $\mathsf{element} \in \mathscr{F}_{\mathsf{Array}(n,s_1).\mathsf{Nat}, s_1}$ aka $\cdot[\cdot]$ and $\mathsf{setElement} \in \mathscr{F}_{\mathsf{Array}(n,s_1).\mathsf{Nat}.s_1, \mathsf{Array}(n,s_1)}$ aka $\cdot[\cdot] := \cdot$

  DEFINITIONS: $\mathsf{element}^{\mathcal{E}}(\langle v_1, ..., v_n \rangle, i) = \begin{cases} v_i & \text{if } 1 \leq i \leq n \\ \mathfrak{err} & \text{otherwise} \end{cases}$

  $$\mathsf{setElement}^{\mathcal{E}}(\langle v_1, \dots, v_{i-1}, v_i, v_{i+1}, \dots, v_n \rangle, i, v) = \begin{cases} \langle v_1, \dots, v_{i-1}, v, v_{i+1}, \dots, v_n \rangle & \text{if } 1 \leq i \leq n \\ \mathfrak{err} & \text{otherwise} \end{cases}$$

  The operation $\mathsf{element} \in \mathscr{F}_{\mathsf{Array}(n,s_1).\mathsf{Nat}, s_1}$ returns the value of the nth element of an array while $\mathsf{setElement} \in \mathscr{F}_{\mathsf{Array}(n,s_1).\mathsf{Nat}.s_1, \mathsf{Array}(n,s_1)}$ sets its value. An access out of bounds of the array must return a non-permuted error value, similarly to the case of an empty list in the list operation first. And since $\mathsf{element}^{\mathcal{E}}(\langle v_1, \dots, v_n \rangle, i) = v_i$ and $\mathsf{element}^{\mathcal{E}}(\theta_{\mathsf{Array}(n,s_1)}(\langle v_1, \dots, v_n \rangle), \theta_{\mathsf{Nat}}(i)) = \theta_{s_1}(v_{\theta_{\mathsf{Nat}}(i)})$, the domain of naturals must not be permuted. But this is the case with allowed domain permutations and therefore the operations are compatible with them.

Association Arrays. Let $s = \mathsf{AssocArray}(s_1, s_2)$ be an association array sort.

- OPERATION: $\mathsf{newAssocArray} \in \mathscr{F}_{\epsilon, \mathsf{AssocArray}(s_1,s_2)}$

  DEFINITION: $\mathsf{newAssocArray}^{\mathcal{E}}() = \emptyset$

  Creates an empty association array. It is obviously compatible with all allowed domain permutations since an empty association array is always permuted to itself.

- OPERATION: $\mathsf{isDefined} \in \mathscr{F}_{\mathsf{AssocArray}(s_1,s_2).s_1, \mathsf{Bool}}$

DEFINITION: $\mathsf{isDefined}^{\mathcal{E}}(a,i) = \begin{cases} \mathit{true} & \text{if } a(i) \text{ is defined} \\ \mathit{false} & \text{otherwise} \end{cases}$

Tests whether an association is defined for an index element i. Compatible with all allowed domain permutations because (only) a defined association is mapped to a defined association.

- OPERATIONS: $\mathsf{get} \in \mathscr{F}_{\mathsf{AssocArray}(s_1,s2).s_1,s_2}$ and
  $\mathsf{set} \in \mathscr{F}_{\mathsf{AssocArray}(s_1,s2).s_1.s_2,\mathsf{AssocArray}(s_1,s2)}$

  DEFINITIONS: $\mathsf{get}^{\mathcal{E}}(a,i) = \begin{cases} a(i) & \text{if } a(i) \text{ is defined} \\ \mathfrak{err} & \text{otherwise} \end{cases}$
  $\mathsf{set}^{\mathcal{E}}(a,i,v) = a[i \mapsto v]$

  Operations for getting and setting associations. If $\mathsf{get}^{\mathcal{E}}(a,i) = v$, then $\mathsf{get}^{\mathcal{E}}(\theta_s(a), \theta_{s_1}(i)) = \theta_{s_2}(v)$ directly by the definition of how allowed domain permutations are defined for association arrays. Therefore get is compatible with all allowed domain permutations. Similar argumentation holds set.

Unions. For unions we have operations that assign and fetch elements from the union, taking care that the element stored in a union is of correct sort.

- OPERATION: $\mathsf{make}_s \in \mathscr{F}_{s,\mathsf{Union}(s_1,\ldots,s_n)}$ for each $s \in \{s_1, \ldots, s_n\}$
  DEFINITION: $\mathsf{make}_s{}^{\mathcal{E}}(v) = \langle s, v \rangle$

  Creates an union element out of a constituent sort element. Obviously compatible with all allowed domain permutations.

- OPERATION: $\mathsf{isOfSort}_s \in \mathscr{F}_{\mathsf{Union}(s_1,\ldots,s_n),\mathsf{Bool}}$ for $s \in \{s_1, \ldots, s_n\}$
  DEFINITION: $\mathsf{isOfSort}_s{}^{\mathcal{E}}(\langle s', v \rangle) = \mathit{true}$ if $s' = s$ and *false* otherwise

  Checks the sort of the union element.

- OPERATION: $\mathsf{cast}_s \in \mathscr{F}_{\mathsf{Union}(s_1,\ldots,s_n),s}$ aka $(s)\cdot$ for $s \in \{s_1, \ldots, s_n\}$
  DEFINITION: $\mathsf{cast}_s{}^{\mathcal{E}}(\langle s', v \rangle) = v$ if $s' = s$ and $\mathfrak{err}$ otherwise

  "Casts" an union element back to its base sort (error if sorts mismatch).

6.3 ERROR HANDLING

Consider a situation in which we would like have a transition guard that evaluates to *true* if the first element of a list l (from an input place) equals to x and *false* otherwise. We do not want the guard evaluate to $\mathfrak{err}$ since it is usually convenient, when implementing reachability analyzers, to abort the generation of the reachability graph when an error is evaluated. Instead, since we have considered the empty list case, we would like the analysis to continue. The first attempt, the term $\mathsf{neg}(\mathsf{isEmpty}(l)) \wedge \mathsf{first}(l) = x$, fails because in case l is empty the term evaluates to $\mathfrak{err}$, not *false*. We therefore add the following error detection and correction operations for each sort $s \in \mathcal{S}$. These are the only exceptional operations in this work that have a value different to $\mathfrak{err}$ in case an argument value is $\mathfrak{err}$.

The error detection operation $\mathsf{isOK} \in \mathscr{F}_{s,\mathsf{Bool}}$ is such that $\mathsf{isOK}^{\mathcal{E}}(v) = \mathit{true}$ if $v \in \mathscr{D}_s^{\mathcal{E}}$ and $\mathsf{isOK}^{\mathcal{E}}(\mathfrak{err}) = \mathit{false}$.

The error correction operation "if-then-else" $\mathsf{ite}_s \in \mathscr{F}_{\mathsf{Bool}.s.s,s}$ is defined so that $\mathsf{ite}_s{}^{\mathcal{E}}(\mathit{true}, v_1, v_2) = v_1$, $\mathsf{ite}_s{}^{\mathcal{E}}(\mathit{false}, v_1, v_2) = v_2$ and $\mathsf{ite}_s{}^{\mathcal{E}}(\mathfrak{err}, v_1, v_2) = \mathfrak{err}$ for all $v_1, v_2 \in \tilde{\mathscr{D}}_s^{\mathcal{E}}$. The ite_s operation may also be written in mix-fix notation as $\mathsf{if} \cdot \mathsf{then} \cdot \mathsf{else} \cdot$.

Clearly both of these operations are compatible with all well-formed domain permutations (recall Def. 5.1.1) since $\mathfrak{err}$ and Boolean values are not permuted.

Now the problem presented in the beginning of this section can be solved with the term $\mathsf{if}\ \mathsf{neg}(\mathsf{isEmpty}(l))\ \mathsf{then}\ \mathsf{first}(l) = x\ \mathsf{else}\ \mathit{false}$.

6.4 EXAMPLES

We next show some examples and also sketch a simple input language for EWF-nets.

Example 6.4.1 Let us interpret the folded version of the railroad system (Figure 4.2) as an EWF-net. In addition to some predefined sorts such as the Booleans and natural numbers, we have the unordered sort Trains with the domain $\{a, b\}$ for trains and the rotation sort Secs with the domain $\{0, 1, \ldots, 5\}$ for railroad sections. The successor function $\mathrm{succ}_{\mathsf{Secs}}$ for Secs is defined in the obvious way: as addition modulo 6. The operation next is a synonym for the 1-successor function and prev denotes the 5-successor function $\mathrm{succ}^5_{\mathsf{Secs}}$. The domain of the place V of Figure 4.2 is Secs as previously whereas the domain of the place U is Struct(section : Secs, train : Trains). The variables x and i of the transition t are as in Example 4.2.3 and the annotation of the arc from the transition t to the place U is the term $\mathsf{unitms}(\mathsf{makeStruct}(\mathsf{succ}(i), x))$. Note that the set of allowed domain permutations of this EWFN-version corresponds to the domain permutations described in Example 5.2.8. ♣

6.4.1 A Textual Input Language Example

To illustrate a possible textual syntax for EWFNs, we consider a (partial) net description for a hypothetical client/server model.

Let us begin with sort definitions:

```
newsort PId UnorderedType(5);
newsort SeqNum RotationType(10);
newsort AckMSG Struct(sender: PId, seqnum: SeqNum);
newsort RequestMSG Struct(sender: PId, seqnum: SeqNum);
newsort Msg Union(AckMSG, DataMSG);
```

and so on. Above we have defined an unordered sort PId with the domain $\{\mathit{pid1}, \ldots, \mathit{pid5}\}$ which is used to identify different processes and a rotation sort SeqNum with the domain $\{\mathit{seqnum1}, \ldots, \mathit{seqnum10}\}$ for sequence numbers of messages. An acknowledgement message is modeled with a structured sort AckMSG where the sender field gives the identifier of the process which sent the message (needed in order to reply the sender) and the seqnum field tells what is the sender's opinion about the current sequence number. The sort Msg is the union of AckMSG and RequestMSG.

The places are defined next by attaching each place a name and a sort.

```
...
place server_wait_ack: PId;
place server_current_seq_num: Struct(pid: PId, seqnum: SeqNum) =
  all x : Struct(pid:PId, seqnum:SeqNum) s.t. seqnum(x)=seqnum1;
place server_acknowled: PId;
place server_queues: Struct(pid: PId, FIFO(Msg));
...
```

Note that the initial marking $1 \cdot \langle pid1, seqnum1 \rangle + \cdots + 1 \cdot \langle pid5, seqnum1 \rangle$ for place *server_current_seq_num* is not symmetric w.r.t. to the allowed domain permutations since by rotating the sequence numbers we get a different marking. Since the symmetricity of initial markings is not required, we can use operations such as the constant seqnum1 which are not compatible with the allowed domain permutations in the declaration of initial markings.

Finally, the transitions are defined.

```
...
transition server_rec_ack {
  vars { pid: Pid, snum: SeqNum, q: FIFO }
  in { server_wait_ack: pid,
       server_current_seqnum: <pid, snum>,
       server_queues: <pid, q> }
  out { server_acknowledged: pid,
        server_current_seqnum: <pid, succ(snum)>,
        server_queues: <pid, deleteFirst(q)> }
  guard { if !isEmpty(q)
          then isOfSortAckMsg(first(q)) &
               seqnum((AckMsg)first(q)) = snum
          else false }
}
...
```

The transition above models the situation in which a server i is in the state waiting for the acknowledgement from a client. This is denoted by presence of token *pidi* in the place *server_wait_ack*. When the corresponding input queue in the place *server_queues* is not empty, the first message in the queue is an acknowledgement message and the expected sequence number in the place *server_current_seqnum* matches with the one in the message, then the message is removed from the queue and the server moves to the acknowledged state. Furthermore, the sequence number expected next is the successor of the previous. Note that because the elements in a union sort are tagged with the sort name, information about the message type can be retrieved by using the isOfSort operation.

We have used the mix-fix notation in some places, e.g. `<pid, snum>` actually means `MakeStruct(pid,snum)` and `pid` in the input and output arcs of the transition means `unitms(pid)`. Similarly, the term `(AckMsg)first(q)` corresponds to `castAckMsg(first(q))`. Also observe that we use the if-then-else operation in the transition guard to prevent it evaluation to $\mathfrak{err}$ in case the list q is empty.

Recall that deciding whether two states (markings in the case of Petri nets) are symmetric is a sub-task used in the calculation of quotient reachability graphs (Section 2.2.2). We now show that deciding whether two markings of an EWF-net are symmetric under the allowed domain permutation group is a polynomial-time many-one equivalent to the graph isomorphism problem (for more information about the complexity of the graph isomorphism problem see [Köbler et al. 1993]). The inclusion part (reduction from the marking symmetry problem to graph isomorphism) is done in a constructive way enabling the use of existing graph isomorphism checking algorithms for checking symmetricity of two markings.

We begin with the problem definition:

Problem 6.5.1 MARKING SYMMETRY. *Given an EWF-net $\mathcal{N}$ and two of its markings, decide whether they are symmetric under net's set of allowed domain permutations.*

The graph isomorphism problem is defined as follows.

Problem 6.5.2 GRAPH ISOMORPHISM. *An unlabeled, undirected graph is a pair $\langle V, E\rangle$ where V is a non-empty set of* vertices *and $E \subseteq V \times V$ is the anti-reflexive and symmetric set of* edges. *Two undirected and unlabeled graphs, $G_1 = \langle V_1, E_1\rangle$ and $G_2 = \langle V_2, E_2\rangle$, are* isomorphic *if there is a bijective mapping $\pi : V_1 \to V_2$ such that $\langle v, v'\rangle \in E_1 \Leftrightarrow \langle \pi(v), \pi(v')\rangle \in E_2$ for all $v, v' \in V_1$. The problem* GRAPH ISOMORPHISM *is to decide whether two graphs are isomorphic.*

We first show GRAPH ISOMORPHISM-hardness of MARKING SYMMETRY:

Lemma 6.5.3 GRAPH ISOMORPHISM *polynomial-time many-one reduces to* MARKING SYMMETRY.

Proof. (Inspired by a proof by Ip [1996, Theorem 3.4].) Assume that we are given two undirected graphs, $G_1 = \langle V_1, E_1\rangle$ and $G_2 = \langle V_2, E_2\rangle$. Construct the EWF-net $\mathcal{N}$ consisting of places *edges* and *vertices*, and including the unordered sort Vertices with the domain $V_1 \cup V_2$ in the algebra. Let the domain of the place *vertices* be Vertices and let the domain of *edges* be Struct(e1 : Vertices, e2 : Vertices). For the graph G_i, $1 = 1, 2$, consider the marking M_i such that $M_i(\mathit{edges}) = \sum_{v \in V_i} 1 \cdot v$ and $M_i(\mathit{vertices}) = \sum_{\langle v,v'\rangle \in E_i} 1 \cdot \langle v, v'\rangle$. Clearly M_1 and M_2 are symmetric under $\mathcal{N}$'s set of allowed domain permutations iff G_1 and G_2 are isomorphic. We have thus polynomial-time many-one reduced GRAPH ISOMORPHISM to MARKING SYMMETRY. □

We next show that MARKING SYMMETRY can be polynomial-time many-one reduced to GRAPH ISOMORPHISM. For this we use an extended definition of graphs:

Definition 6.5.4 *A* labeled directed graph *is a triple* $G = \langle V, E, L \rangle$ *where* V *is a finite non-empty set of* vertices, $E \subseteq V \times V$ *is the set of* edges *and* L *assigns each vertex with an arbitrary* label. *Two labeled directed graphs,* $G_1 = \langle V_1, E_1, L_1 \rangle$ *and* $G_2 = \langle V_2, E_2, L_2 \rangle$ *, are* isomorphic *if there is a bijective mapping* $\pi : V_1 \to V_2$ *such that* $\langle v, v' \rangle \in E_1 \Leftrightarrow \langle \pi(v), \pi(v') \rangle \in E_2$ *and* $L_1(v) = L_2(\pi(v))$ *for all* $v, v' \in V_1$.

The *labeled directed graph isomorphism problem*, call it LDGI, is to decide whether two such graphs are isomorphic. Miller [1979] has shown that unlabeled directed graph isomorphism problem DGI is polynomial-time many-one reducible to undirected graph isomorphism problem GRAPH ISOMORPHISM. Using similar graph-theoretical gadget constructs it is easy to show that LDGI is polynomial-time many-one reducible to DGI and thus to GRAPH ISOMORPHISM. Furthermore, GRAPH ISOMORPHISM is a special case of LDGI in which all vertices have the same label and all edges are bidirectional. This sums up to that LDGI is polynomial-time many-one equivalent to GRAPH ISOMORPHISM and therefore we do not distinguish them in this work.

Now assume an EWF-net $\mathcal{N} = \langle N, MSIG, \mathcal{E}, d, v, a, g, m_{init} \rangle$. In the following we build for a marking M of $\mathcal{N}$ a corresponding directed labeled graph G_M such that two markings M_1 and M_2 are symmetric under the extension $\Theta_{\mathrm{SS}(\mathcal{N})}$ of net's set $\Theta_{\mathcal{S}}$ of allowed domain permutations to the state space level iff G_{M_1} and G_{M_2} are isomorphic.

Definition 6.5.5 *For a marking* M *of* $\mathcal{N}$*, the corresponding graph* $G_M = \langle V_M, E_M, L_M \rangle$ *is defined inductively by the following rules:*

- *For each place* p *and each* $v \in \mathscr{D}^{\mathcal{E}}_{d(p)}$ *such that* $(M(p))\,(v) = n \geq 1$, V_M *includes the vertex* $\langle p, n, v \rangle$ *with the label* $\langle p, n \rangle$. *Furthermore,* $\langle d(p), v \rangle \in V_M$ *and there is an edge from* $\langle p, n, v \rangle$ *to* $\langle d(p), v \rangle$ *in* E_M.
- *If* $\langle s, v \rangle \in V_M$ *for* $s = \mathsf{List}(s_1)$ *and* $v = \langle v_1, \ldots, v_n \rangle \in (\mathscr{D}^{\mathcal{E}}_{s_1})^*$*, then*
  1. $L_M(\langle s, v \rangle) = s$,
  2. *for each* $1 \leq i \leq n$, G_M *has the vertex* $\langle s, v, i \rangle$ *with label* $\langle s, i \rangle$ *and the vertex* $\langle s_1, v_i \rangle$ *(label defined later inductively)*
  3. $\langle \langle s, v \rangle, \langle s, v, i \rangle \rangle$ *and* $\langle \langle s, v, i \rangle, \langle s_1, v_i \rangle \rangle$ *are in* E_M *for* $1 \leq i \leq n$.
- *If* $\langle s, v \rangle \in V_M$ *for a structure sort* $s = \mathsf{Struct}(\mathrm{f1} : s_1, \ldots, \mathrm{fn} : s_n)$ *and value* $v = \langle v_1, \ldots, v_n \rangle \in \mathscr{D}^{\mathcal{E}}_{s_1} \times \cdots \times \mathscr{D}^{\mathcal{E}}_{s_n}$*, then*
  1. $L_M(\langle s, v \rangle) = s$,
  2. *for each* $1 \leq i \leq n$, $\langle s, v, i \rangle$ *with label* $\langle s, i \rangle$ *and* $\langle s_i, v_i \rangle$ *are in* V_M *and*
  3. *for each* $1 \leq i \leq n$, $\langle \langle s, v \rangle, \langle s, v, i \rangle \rangle, \langle \langle s, v, i \rangle, \langle s_i, v_i \rangle \rangle \in E_M$.
- *If* $\langle s, v \rangle \in V_M$ *for* $s = \mathsf{Array}(n, s_1)$ *and* $v = \langle v_1, \ldots, v_n \rangle \in (\mathscr{D}^{\mathcal{E}}_{s_1})^n$*, then*
  1. $L_M(\langle s, v \rangle) = s$,

2. *for each* $1 \leq i \leq n$, $\langle s, v, i\rangle$ *with label* $\langle s, i\rangle$ *and* $\langle s_1, v_i\rangle$ *are in* V_M *and*
3. *for each* $1 \leq i \leq n$, $\langle\langle s, v\rangle, \langle s, v, i\rangle\rangle, \langle\langle s, v, i\rangle, \langle s_1, v_i\rangle\rangle \in E_M$.

- *If* $\langle s, v\rangle \in V_M$ *where* $s = \mathsf{AssocArray}(s_1, s_2)$ *and* $v \subseteq \mathscr{D}^{\mathcal{E}}_{s_1} \times \mathscr{D}^{\mathcal{E}}_{s_2}$, *then*

  1. $L_M(\langle s, v\rangle) = s$,
  2. *for each* $\langle i, v'\rangle \in v$, $\langle s, i, v'\rangle$, $\langle s, i, v', \text{index}\rangle$ *and* $\langle s, i, v', \text{value}\rangle$ *each with label* s *are in* V_M, *and so are* $\langle s_1, i\rangle$ *and* $\langle s_2, v'\rangle$; *and*
  3. *for each* $\langle i, v'\rangle \in v$, G_M *has edges from* $\langle s, v\rangle$ *to* $\langle s, i, v'\rangle$, *from* $\langle s, i, v'\rangle$ *to* $\langle s, i, v', \text{index}\rangle$, *from* $\langle s, i, v'\rangle$ *to* $\langle s, i, v', \text{value}\rangle$, *from* $\langle s, i, v', \text{index}\rangle$ *to* $\langle s_1, i\rangle$ *and from* $\langle s, i, v', \text{value}\rangle$ *to* $\langle s_2, v'\rangle$.

- *If* $\langle s, v\rangle \in V_M$ *for* $s = \mathsf{Set}(s_1)$ *and* $v \subseteq \mathscr{D}^{\mathcal{E}}_{s_1}$, *then*

  1. $L_M(\langle s, v\rangle) = s$,
  2. *for each* $v' \in v$, $\langle s_1, v'\rangle$ *is in* V_M *and*
  3. $\langle\langle s, v\rangle, \langle s_1, v'\rangle\rangle$ *is in* E_M *for each* $v' \in v$.

- *If* $\langle s, v\rangle \in V_M$ *for* $s = \mathsf{Union}(s_1, \ldots, s_n)$ *and* $v = \langle s_i, v'\rangle$ *for some* $1 \leq i \leq n$ *such that* $v_i \in \mathscr{D}^{\mathcal{E}}_{s_i}$, *then*

  1. $L_M(\langle s, v\rangle) = s$,
  2. $\langle s_i, v'\rangle$ *is in* V_M *and*
  3. $\langle\langle s, v\rangle, \langle s_i, v'\rangle\rangle$ *is in* E_M.

- *If* $\langle s, v\rangle \in V_M$ *for* $s \in \mathit{Basic}\mathcal{S}$ *and* $v \in \mathscr{D}^{\mathcal{E}}_s$, *then*

  1. *If* $s \in \mathit{Ord}\mathcal{S}$, *then* $L_M(\langle s, v\rangle) = \langle s, v\rangle$.
  2. *If* $s \in \mathit{Rot}\mathcal{S}$, *then* $L_M(\langle s, v\rangle) = s$, $\langle s, v'\rangle \in V_M$ *for all* $v' \in \mathscr{D}^{\mathcal{E}}_s$ *and* $\langle\langle s, v'\rangle, \langle s, \mathrm{succ}_s(v')\rangle\rangle \in E_M$ *for all* $v' \in \mathscr{D}^{\mathcal{E}}_s$.
  3. *If* $s \in \mathit{Unord}\mathcal{S}$, *then* $L_M(\langle s, v\rangle) = s$.

- *Nothing else is in* V_M *or in* E_M.

The labels of the nodes are chosen in a way that preserves necessary information about the domains of values as well as the order of the children (in case of structures, lists etc.).

Example 6.5.6 Consider the marking $M = \{U \mapsto \langle 0, a\rangle + \langle 3, a\rangle, V \mapsto 1 + 4\}$ of the EWFN-version of our railroad Example 6.4.1 (cf. Example 4.2.3). The essential skeleton of the corresponding graph G_M is depicted in Figure 6.1. A lot of label and vertex name information is omitted from the graph for the sake of readability. ♣

Lemma 6.5.7 *For an EWF-net* $\mathcal{N}$, *two of its markings,* M_1 *and* M_2, *are symmetric under* $\Theta_{\mathrm{SS}(\mathcal{N})}$ *iff* G_{M_1} *and* G_{M_2} *are isomorphic.*

Proof. By induction on the structure of G_{M_1} and G_{M_2}. □

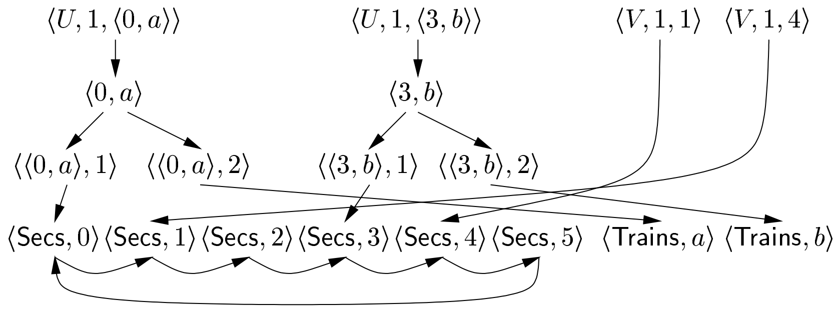

Figure 6.1: The (simplified) graph corresponding to the marking $M = \{U \mapsto \langle 0, a\rangle + \langle 3, a\rangle, V \mapsto 1 + 4\}$.

However, there is a small problem in the construction of G_M: its size is not necessarily polynomial with respect to the coding of the net $\mathcal{N}$ and the marking M. The problem is that if the marking includes an element of a rotation basic type, then all the elements in that rotation type are added into G_M (cf. Example 6.5.6 and Figure 6.1). And since numbers are usually coded in binary and the coding (description) of the net $\mathcal{N}$ does not necessarily enumerate all the elements of rotation basic sorts but may rather have a definition of form `newsort Secs RotationType(10000);`, enumerating all the elements in G_M may in such a case lead to a graph that is exponentially bigger w.r.t. joint size of $\mathcal{N}$ and M. To cure this anomaly, we do the following trick. Suppose that there is a sequence of vertices of form $\langle s, v\rangle$, $\langle s, \mathrm{succ}_s(v)\rangle, \ldots,$ $\langle s, \mathrm{succ}_s^{k-1}(v)\rangle$, $\langle s, \mathrm{succ}_s^{k}(v)\rangle$, where $\langle s, \mathrm{succ}_s(v)\rangle, \ldots, \langle s, \mathrm{succ}_s^{k-1}(v)\rangle$ have no edges coming from vertices other than those in the sequence. That is, they are vertices that we had to add into G_M even though they do not appear in the marking M. We now simply replace the sequence with vertices $\langle s, v\rangle$, $\langle s, k\rangle$, $\langle s, \mathrm{succ}_s^{k}(v)\rangle$, where the label of $\langle s, k\rangle$ is the integer k. Clearly this modified graph construction is of polynomial size w.r.t. $\mathcal{N}$ and M and can be calculated directly from them without constructing the original graph first. Furthermore, Lemma 6.5.7 stays intact since $\langle s, v\rangle$, $\langle s, k\rangle$, $\langle s, \mathrm{succ}_s^{k}(v)\rangle$ of the modified G_{M_1} can be mapped to their counter-parts in the modified G_{M_2} iff $\langle s, v\rangle$, $\langle s, \mathrm{succ}_s(v)\rangle, \ldots, \langle s, \mathrm{succ}_s^{k-1}(v)\rangle$, $\langle s, \mathrm{succ}_s^{k}(v)\rangle$ of the original G_{M_1} can mapped into the corresponding counter-parts of the original G_{M_2}.

As a corollary we have the following:

Lemma 6.5.8 MARKING SYMMETRY *polynomial-time many-one reduces to* GRAPH ISOMORPHISM. □

Combining this with Lemma 6.5.3 we have:

Theorem 6.5.9 MARKING SYMMETRY *is polynomial-time many-one equivalent to* GRAPH ISOMORPHISM. □

Consequently, computing a canonical representative for a marking under the set $\Theta_{\mathcal{S}}$ of allowed domain permutations is computationally equivalent to finding a canonical representative for a graph. And note that although the graph G_M may be a directed acyclic graph (DAG) (in case no rotation sorts are present), the complexity result we obtained is valid since the isomorphism problem for DAGs is polynomial-time many-one equivalent to GRAPH ISOMORPHISM.

The construction and the proof above actually enable us to use many existing algorithms and implementations for graph isomorphism to solve the marking symmetry and canonization problems. See e.g. [McKay 1990; Kreher and Stinson 1999] for such algorithms.

As a word of warning it should be noted that the construction from a marking to a graph was possible only because of the simplicity of the allowed domain permutation groups for basic types. This is because it is not known whether for each permutation group on a set of n elements there is a graph of polynomial size w.r.t. n whose automorphism group is isomorphic to the desired group [Hoffmann 1982, page 246]. Therefore it is not known if we could build similar graph gadgets for arbitrary domain permutation groups like we did for those of rotation sorts in polynomial time.

6.6 TEMPORAL LOGIC EWFN-CTL*

After building a quotient reachability graph of an EWF-net $\mathcal{N}$, we would like to verify various properties about it. Based on the properties listed in Section 2.2.3, some simple ones such as deadlock freedom can be checked directly on the QRG. In the following we modify the very expressive temporal logic CTL* [Clarke et al. 1986; Emerson and Halpern 1986] to EWF-nets by defining the atomic propositions and their semantics.

Assume an algebraic system net $\mathcal{N} = \langle N, \mathit{MSIG}, \mathcal{E}, d, v, a, g, m_{init} \rangle$. The definition of atomic propositions for $\mathcal{N}$ is most naturally done in an algebraic way by using the same algebra that the net involves. For this purpose we introduce a family $\mathcal{M}$ of MSIG-variables: $\mathcal{M}$ has for each place $p \in P$ a variable M_p of sort $ms(d(p))$. Atomic propositions are then simply Boolean terms over $\mathcal{M}$ i.e. the set $AP = \mathbf{T}^{\mathit{MSIG}}_{\mathsf{Bool}}(\mathcal{M})$ serves as the (infinite) set of atomic propositions. To give semantics for atomic propositions, we introduce a variable assignment for $\mathcal{M}$ corresponding to a marking of the net: for a marking $M \in \mathbb{M}$, β_M is the variable assignment for $\mathcal{M}$ such that $\beta_M(M_p) = M(p)$. That is, each variable M_p is assigned the marking of the place p. We now construct the truth assignment function $\tau : \mathbb{M} \to \wp(AP)$ for the set $\mathbb{M}$ of markings of $\mathcal{N}$: for each atomic proposition $T \in \mathbf{T}^{\mathit{MSIG}}_{\mathsf{Bool}}(\mathcal{M})$ and marking M,

$$T \in \tau(M) \text{ iff } eval^{\mathcal{E}}_{\beta_M}(T) = true.$$

It can be assumed that the evaluation never leads to $\mathfrak{err}$ (however, should this happen anyway on the implementation level, the analysis is stopped and the user is reported about an erroneous temporal logic specification).

We now define the rest of the logic EWFN-CTL* in the standard way.

Definition 6.6.1 (Syntax) EWFN-CTL* *has two types of formulas:* state formulas *and* path formulas. *A state formula is either:*

- T, *if* $T \in AP$.
- *If* f_1 *and* f_2 *are state formulas, then* $\neg f_1$ *and* $f_1 \vee f_2$ *are state formulas.*

- *If g is a path formula, then $\mathbf{E}(g)$ is a state formula.*

A path formula is either:

- *A state formula.*
- *If g_1 and g_2 are path formulas, then $\neg g_1$, $g_1 \vee g_2$, $\mathbf{X}g_1$ and $g_1\mathbf{U}g_2$ are path formulas.*

EWFN-CTL$^\star(AP)$ *is the set of all state formulas generated by the above rules for the set AP of atomic propositions.*

As usual, we can use the following abbreviations: *true* for $\varphi \vee \neg\varphi$, *false* for $\neg true$, $\varphi \wedge \psi$ for $\neg(\neg\varphi \vee \neg\psi)$, $\varphi \Rightarrow \psi$ for $\neg\varphi \vee \psi$, $\varphi\mathbf{V}\psi$ for $\neg(\neg\varphi\mathbf{U}\neg\psi)$, $\mathbf{F}\varphi$ for $true\mathbf{U}\varphi$, $\mathbf{G}\varphi$ for $\neg\mathbf{F}\neg\varphi$ and $\mathbf{A}\varphi$ for $\neg\mathbf{E}\neg\varphi$.

Let $\mathcal{L} = \langle \mathbb{M}', \mathbb{T}, [\rangle', M'_{init}\rangle$ be the state space of $\mathcal{N}$, the reachability graph of $\mathcal{N}$ or a quotient reachability graph of $\mathcal{N}$ under $\Theta_{\mathrm{SS}(\mathcal{N})}$. As usual, it is assumed that each deadlock marking in $\mathcal{L}$ is augmented with a dummy self-loop so that all paths in $\mathcal{L}$ can be extended to infinite ones. The semantics of EWFN-CTL* on $\mathcal{L}$ are defined in the following.

Definition 6.6.2 (Semantics) *$M \models f$ means that the state formula f holds in the marking M. $\sigma \models g$ means that the path formula g holds along the infinite path σ in $\mathcal{L}$. Assume that f_1 and f_2 are state formulas and that g_1 and g_2 are path formulas. The relation $\models$ is defined inductively as follows:*

1. *$M \models T$ iff $T \in \tau(M)$ for an atomic proposition $T \in AP$.*
2. *$M \models \neg f_1$ iff $M \not\models f_1$.*
3. *$M \models f_1 \vee f_2$ iff $M \models f_1$* or *$M \models f_2$.*
4. *$M \models \mathbf{E}(g_1)$ iff there exists an infinite path σ starting with M such that $\sigma \models g_1$.*
5. *$\sigma \models f_1$ iff M is the first state in σ and $M \models f_1$.*
6. *$\sigma \models \neg g_1$ iff $\sigma \not\models g_1$.*
7. *$\sigma \models g_1 \vee g_2$ iff $\sigma \models g_1$* or *$\sigma \models g_2$.*
8. *$\sigma \models \mathbf{X}g_1$ iff $\sigma^1 \models g_1$.*[3]
9. *$\sigma \models g_1\mathbf{U}g_2$ iff there exists a $k \geq 0$ such that $\sigma^k \models g_2$ and for all $0 \leq j < k$, $\sigma^j \models g_1$.*

Now it follows directly from the well-formedness of allowed domain permutations and from Lemma 3.4.2 that if an atomic proposition T consists only of operations compatible with the group $\Theta_{\mathcal{S}}$ of allowed domain permutations of $\mathcal{N}$, then T is invariant under $\Theta_{\mathrm{SS}(\mathcal{N})}$. And EWFN-CTL* formulas

[3]For an infinite path $\sigma = M_0 \xrightarrow[\mathcal{L}]{t_{\beta,0}} M_1 \xrightarrow[\mathcal{L}]{t_{\beta,1}} \cdots M_i \xrightarrow[\mathcal{L}]{t_{\beta,i}} \cdots$, σ^i is the infinite suffix $M_i \xrightarrow[\mathcal{L}]{t_{\beta,i}} \cdots$.

involving only invariant atomic propositions can be evaluated on any quotient reachability graph of $\mathcal{N}$, as noted in Section 2.2.3. Model checking of formulas containing non-invariant i.e. asymmetric atomic propositions has to be done by more advanced techniques found in the papers mentioned in Section 1.1.

Example 6.6.3 Consider the net in Figure 6.2 modeling a simple mutual exclusion system. We define that the sort Pid is an unordered sort with n elements $\{PId_1, \ldots, PId_n\}$ modeling the process identifiers. The symbolic initial marking for the place *idle* is "all x : PId s.t. *true*" meaning that the initial marking of *idle* is $PId_1 + \cdots + PId_n$. The place *free* contains one black token and the rest of the places are empty in the initial marking. The reachability graph of the net for $n = 2$ is shown in Figure 6.3 and two of its quotient reachability graphs are shown in Figure 6.4. The markings of the places are written in the order *idle*, *trying*, *critical*, *free*.

First consider the safety property formula

$$\mathbf{AG}(\mathsf{size}(M_{critical}) \leq 1)$$

asserting that there should be at most one process in the critical section at each time. This formula is true in the initial marking M_{init} as can be seen by inspecting the reachability graph of the net in Figure 6.3. And since the atomic proposition $\mathsf{size}(M_{critical}) \leq 1$ in the formula is invariant under the net's allowed domain permutations, the formula is true when evaluated in the representative initial states of the quotient reachability graphs in Figure 6.4.

On the other hand, the liveness formula

$$\mathbf{AG}\,(\mathsf{multiplicity}(M_{trying}, \mathsf{PId}_1) \geq 1 \Rightarrow \mathbf{F}\mathsf{multiplicity}(M_{critical}, \mathsf{PId}_1) \geq 1)$$

stating that process 1 should eventually get into the critical section if it is trying, contains asymmetric atomic propositions. Consequently, its value in the initial marking of the reachability graph is false but true when evaluated in the initial marking of the upper quotient reachability graph in Figure 6.4. Thus simply model checking quotient reachability graphs with formulas involving asymmetric propositions is not sound.

The *Indexed CTL** of Emerson and Sistla [1996; 1997] and the associated model checking algorithms can be modified to solve this problem. For instance, expressing the fairness property that each process always finally gets from the trying state to the critical state can be done with the following formula:

$$\bigwedge_{p:\mathsf{PId}} \mathbf{AG}\,(\mathsf{multiplicity}(M_{trying}, p) \geq 1 \Rightarrow \mathbf{F}\mathsf{multiplicity}(M_{critical}, p) \geq 1)\,.$$

♣

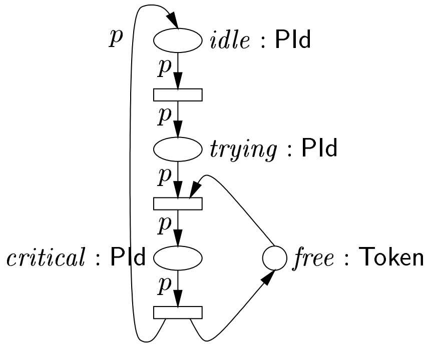

Figure 6.2: An EWF-net for a simple mutual exclusion problem.

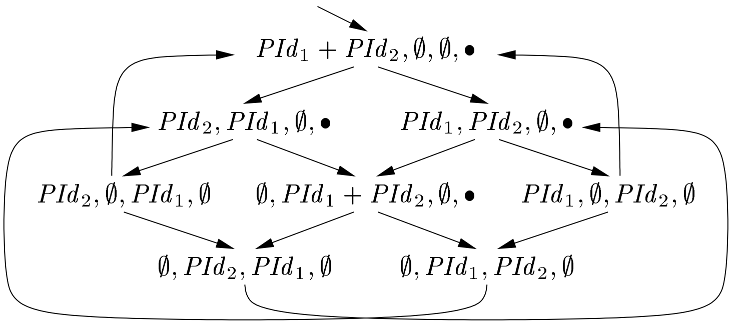

Figure 6.3: The reachability graph of the net in Figure 6.2 for two processes.

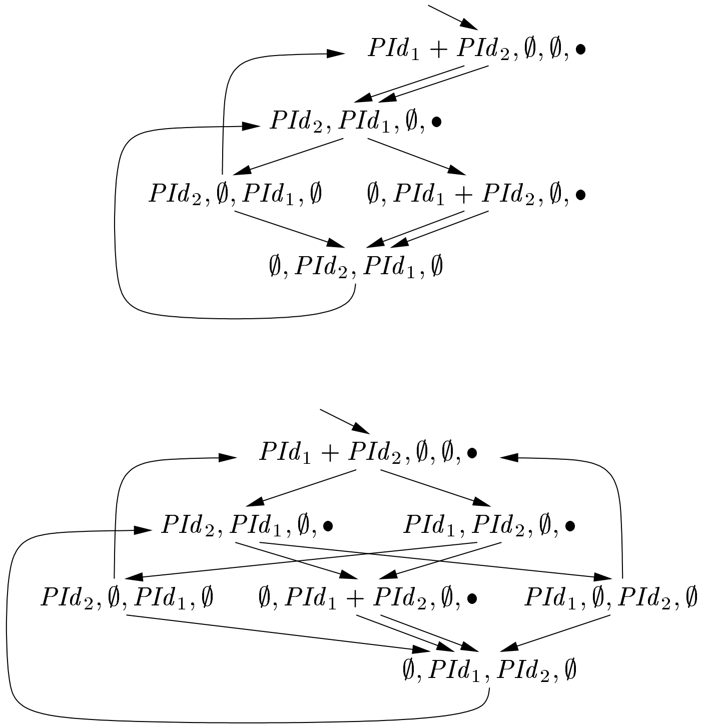

Figure 6.4: Two quotient reachability graphs of the net in Figure 6.2 for two processes.

7 CONCLUSIONS

We began this work by giving a brief introduction (a unified theory) for the symmetry reduction method in the context of reachability analysis. We then defined the variant of error algebras used in this work as means to represent and manipulate data. The algebraic concepts were then combined with Petri nets to a definition of the net class studied in this work: algebraic system nets. It was then defined how permutations on the data type domains produce state space permutations of algebraic system nets. In order to make sure that the state space permutations are in fact symmetries we defined and proved a sufficient condition that the domain permutations and the algebraic terms attached to the transitions and arcs should be mutually compatible. The concept of term compatibility w.r.t. a domain permutation was defined to essentially mean that evaluation of a term under permuted variable assignments gives a similarly permuted result. The problem of deciding term compatibility was shown to be **co-NP**-complete. Motivated by this fact, an approximation rule was devised stating that if the operations appearing in a term are compatible, then the term itself is also compatible with respect to a domain permutation.

Finally all the concepts were demonstrated together when we defined the class of extended well-formed nets. The net class was restricted in a way that only certain operations could be used for different classes of basic types. This meant that certain domain permutations called allowed domain permutations are compatible with the operations. Structured types and their operations were then introduced and it was shown that all the operations we could conceive were in fact compatible with allowed domain permutations extended to the structured sorts. It was shown that it is a graph isomorphism-complete problem to decide whether two markings of an EWFN are symmetric. The proof was constructive in a sense that based on it a practical implementation using existing graph isomorphism checking software could be build. An illustrative temporal logic was also described for EWFNs. Basing on this demonstration, it should be relatively easy to build similar net classes for which data type symmetries could be exploited.

As closing note recall that our method is built on the concept of domain permutations and therefore different data types must exists in order to efficiently detect symmetries. This holds even if we devised an algorithm to decide term compatibility and to find compatible domain permutations. Thus the method still requires that the user uses many different (special) types provided by the net class. A (non-trivial) improvement would be to automatically "retype" algebras and nets to isomorphic (in some sense) ones having more types to exploit.

A MATHEMATICAL PRELIMINARIES

A.1 SETS, MULTI-SETS AND FAMILIES

Sets etc. We use the set $\mathbb{B} = \{true, false\}$ for Boolean values and the set $\mathbb{N} = \{0, 1, 2, \dots\}$ 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, $\wp(A)$ denotes the set of all subsets of A i.e. the *power-set* of A.

Multi-Sets. For a non-empty set A, a *multi-set* b over A is a function $b : A \to \mathbb{N}$. The set $\mathcal{MS}(A)$ denotes the *set of all multi-sets* over A, i.e., the set $[A \to \mathbb{N}]$. A multi-set $b \in \mathcal{MS}(A)$ can also be written as a *formal sum* $\sum_{a \in A} b(a) \cdot a$; each $b(a)$ is called the *multiplicity* of the element a in b. Elements with multiplicity 0 as well as multiplicities of 1 can be omitted i.e. $0 \cdot a + 1 \cdot b + 3 \cdot c$ may be written as $b + 3 \cdot c$. The *empty multi-set* $\sum_{a \in A} 0 \cdot a$ is denoted by $\emptyset$. A set $A' \subseteq A$ can be interpreted as the multi-set $\sum_{a \in A'} 1 \cdot a$.

Let $b_1, b_2 \in \mathcal{MS}(A)$ be multi-sets over a set A. We define the following relations and operations:

1. $b_1 = b_2$ iff $b_1(a) = b_2(a)$ for all $a \in A$ (equality)
2. $b_1 \leq b_2$ iff $b_1(a) \leq b_2(a)$ for all $a \in A$ (less-or-equal)
3. $b_1 + b_2 = \sum_{a \in A}(b_1(a) + b_2(a))a$ (addition)
4. $n \cdot b_1 = \sum_{a \in A}(nb_1(a)) \cdot a$ for each $n \in \mathbb{N}$ (scalar multiplication)

Families. For a set I, $A = (A_i)_{i \in I}$ is a *family* of sets if A_i is a set for each $i \in I$. A is *pairwise disjoint* if $i \neq j \Rightarrow A_i \cap A_j = \emptyset$ for all $i, j \in I$. For a family $A = (A_i)_{i \in I}$, we often write A to denote the union $\bigcup_{i \in I} A_i$, e.g. $a \in A$ actually means $a \in \bigcup_{i \in I} A_i$. A is *finite* iff $\bigcup_{i \in I} A_i$ is finite. We may also denote a family by $A = (i_A)_{i \in I}$, where i_A is a set for each $i \in I$ while the other definitions remain similar.

A.2 GROUPS

The definitions for groups are based on [Grimaldi 1985; Humphreys 1996]. If G is a non-empty set and $\circ$ is a binary operation on G, then $\langle G, \circ \rangle$ is a *group* if

1. $a, b \in G$ imply $a \circ b \in G$ (closure of G under $\circ$);
2. for $a, b, c \in G$, $a \circ (b \circ c) = (a \circ b) \circ c$ (the associative property);
3. $(\exists e \in G)(\forall a \in G)(a \circ e = e \circ a = a)$ (the existence of identity); and
4. $(\forall a \in G)(\exists a^{-1} \in G)(a \circ a^{-1} = a^{-1} \circ a = e)$ (existence of inverses).

For a group $\langle G, \circ \rangle$, we may also say that G is group under $\circ$ and if the binary operator on G is clear from the context, we may simply say that G is a group. Similarly, we may write $g_1 \circ g_2$ as $g_1 g_2$.

For a set A we use $\mathrm{Perm}(A)$ to denote the group consisting of all permutations on A under the function composition operator $\circ$. The word permutation and $\mathrm{Perm}(A)$ may also have slightly different but intuitively similar meanings in some parts of the work (e.g. Section 2.1.1).

Definition A.2.1 (Subgroups) *Let G be a group and $\emptyset \neq H \subseteq G$. If H is a group under the binary operation of G, then we call H a* subgroup *of G, denoted by $H \leq G$. H is* normal *iff for all $g \in G$, $gHg^{-1} \in H$ (i.e. for all $h \in H$ and all $g \in G$, $ghg^{-1} \in H$).*

Definition A.2.2 (Group Morphisms) *Let $\langle G, \circ \rangle$ and $\langle H, * \rangle$ be groups. A function $f : G \to H$ is called a* group homomorphism *if for all $a, b \in G$, $f(a \circ b) = f(a) * f(b)$. We say that f is a* group isomorphism *if it is one-to-one and onto. In this case, G and H are said to be* isomorphic groups.

Theorem A.2.3 *If $f : G \to H$ is a homomorphism, then*

1. *$f(a^{-1}) = f(a)^{-1}$; and*
2. *$f(S)$ is a subgroup of H for any subgroup S of G.*

Definition A.2.4 (External Direct Products) *Let $G_1, \ldots, G_n$ be groups under binary operations $\circ_1, \ldots, \circ_n$, respectively. The* external direct product *of $G_1, \ldots, G_n$ is the group consisting of the set of tuples*

$$G_1 \times \cdots \times G_n = \{\langle g_1, \ldots, g_n \rangle \mid g_1 \in G_1, \ldots, g_n \in G_n\}$$

under the binary operation

$$\langle g_1, \ldots, g_n \rangle * \langle g'_1, \ldots, g'_n \rangle = \langle g_1 \circ_1 g'_1, \ldots, g'_n \circ_n g'_n \rangle.$$

Definition A.2.5 (Internal Direct Products) *Let $\{G_i \mid 1 \leq i \leq n\}$ be subgroups of G. G is the* internal direct product *of $\{G_i \mid 1 \leq i \leq n\}$ if*

1. *each G_i is a normal subgroup of G and*
2. *every element of G can be written uniquely in the form $g = g_1 g_2 \ldots g_n$ with g_i in G_i $(1 \leq i \leq n)$.*

Bibliography

AJAMI, K., HADDAD, S., AND ILIÉ, J.-M. 1998. Exploiting symmetry in linear time temporal logic model checking: One step beyond. In *Tools and Algorithms for the Construction and Analysis of Systems; 4th International Conference, TACAS'98, Lisbon, Portugal, March/April 1998*, B. Steffen, Ed. Lecture Notes in Computer Science, vol. 1384. Springer-Verlag, Berlin, Germany, 52–67.

BILLINGTON, J. 1989. Many-sorted high-level nets. In *Proceedings of the Third International Workshop on Petri Nets and Performance Models (PNPM89)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 166–179. Reprinted in [Jensen and Rozenberg 1991], pages 123–136.

BROWNE, M. C., CLARKE, E. M., AND GRÜMBERG, O. 1988. Characterizing finite Kripke structures in propositional temporal logic. *Theor. Comput. Sci. 59,* 1–2 (July), 115–131.

BUTLER, G. 1991. *Fundamental Algorithms for Permutation groups.* Lecture Notes in Computer Science, vol. 559. Springer-Verlag, Berlin, Germany.

CHIOLA, G., DUTHEILLET, C., FRANCESCHINIS, G., AND HADDAD, S. 1991. On well-formed coloured nets and their symbolic reachability graph. See Jensen and Rozenberg [1991], 373–396.

CHIOLA, G., DUTHEILLET, C., FRANCESCHINIS, G., AND HADDAD, S. 1997. A symbolic reachability graph for coloured Petri nets. *Theor. Comput. Sci. 176*, 39–65.

CLARKE, E. M., EMERSON, E. A., JHA, S., AND SISTLA, A. P. 1998. Symmetry reductions in model checking. See Hu and Vardi [1998], 147–158.

CLARKE, E. M., EMERSON, E. A., AND SISTLA, A. P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems 8,* 2 (Apr.), 244–263.

CLARKE, E. M., ENDERS, R., FILKORN, T., AND JHA, S. 1996. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design 9,* 1/2 (Aug.), 77–104.

CLARKE, E. M., FILKORN, T., AND JHA, S. 1993. Exploiting symmetry in temporal logic model checking. See Courcoubetis [1993], 450–462.

CLARKE, E. M. AND JHA, S. 1995. Symmetry and induction in model checking. In *Computer Science Today: Recent Trends and Developments,* J. van Leeuwen, Ed. Lecture Notes in Computer Science, vol. 1000. Springer-Verlag, Berlin, Germany, 455–470.

COURCOUBETIS, C., Ed. 1993. *Computer Aided Verification, 5th International Conference, CAV'93, Elounda, Greece, June 28 – July 1, 1993, Proceedings.* Lecture Notes in Computer Science, vol. 697. Springer-Verlag, Berlin, Germany.

Emerson, E. A. and Halpern, J. Y. 1986. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM 33,* 1 (Jan.), 151–178.

Emerson, E. A., Jha, S., and Peled, D. 1997. Combining partial order and symmetry reductions. In *Tools and Algorithms for the Construction and Analysis of Systems; Third International Workshop, TACAS'97, Enschede, The Netherlands, April 1997*, E. Brinksma, Ed. Lecture Notes in Computer Science, vol. 1217. Springer-Verlag, Berlin, Germany, 19–34.

Emerson, E. A. and Sistla, A. P. 1993. Symmetry and model checking. See Courcoubetis [1993], 463–478.

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

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

Emerson, E. A. and Sistla, P. 1995. Utilizing symmetry when model checking under fairness assumptions: An automata-theoretic approach. In *Computer Aided Verification: 7th International Conference, CAV'95, Liège, Belgium, July 1995*, P. Wolper, Ed. Lecture Notes in Computer Science, vol. 939. Springer-Verlag, Berlin, Germany, 309–324.

Emerson, E. A. and Trefler, R. J. 1999. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. Tech. Rep. TR-99-04, The University of Texas at Austin, Department of Computer Sciences. Mar.

Furst, M., Hopcroft, J., and Luks, E. 1980. Polynomial-time algorithms for permutation groups. In *21st Annual Symposium on Foundations of Computer Science*. IEEE, 36–41.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco.

Genrich, H. J. 1991. Predicate/transition nets. See Jensen and Rozenberg [1991], 3–43.

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

Grumberg, O., Ed. 1997. *Computer Aided Verification: 9th International Conference, CAV'97, Haifa, Israel, June 1997.* Lecture Notes in Computer Science, vol. 1254. Springer-Verlag, Berlin, Germany.

Gyuris, V. and Sistla, A. P. 1997. On-the-fly model checking under fairness that exploits symmetry. See Grumberg [1997], 232–243.

Haddad, S., Ilié, J.-M., Taghelit, M., and Zouari, B. 1995. Symbolic reachability graph and partial symmetries. In *Application and*

Theory of Petri Nets 1995; Proceedings of the 16th International Conference; Turin, Italy, June 1995, G. D. Michelis and M. Diaz, Eds. Lecture Notes in Computer Science, vol. 935. Springer-Verlag, Berlin, Germany, 238–257.

HOFFMANN, C. M. 1982. *Group-Theoretic Algorithms and Graph Isomorphism.* Lecture Notes in Computer Science, vol. 136. Springer-Verlag, Berlin, Germany.

HU, A. J. AND VARDI, M. Y., Eds. 1998. *Computer Aided Verification: 10th International Conference, CAV'98, Vancouver, BC, Canada, June/July 1998.* Lecture Notes in Computer Science, vol. 1427. Springer-Verlag, Berlin, Germany.

HUBER, P., JENSEN, A., JEPSEN, L., AND JENSEN, K. 1991. Reachability trees for high-level Petri nets. See Jensen and Rozenberg [1991], 319–350.

HUBER, P., JENSEN, A. M., JEPSEN, L. O., AND JENSEN, K. 1985. Towards reachability trees for high-level Petri nets. In *Advances in Petri Nets 1984*, G. Rozenberg, Ed. Lecture Notes in Computer Science, vol. 188. Springer-Verlag, Berlin, Germany, 215–233.

HUMPHREYS, J. F. 1996. *A Course in Group Theory.* Oxford University Press, Oxford.

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

IP, C. N. 1996. State reduction methods for automatic formal verification. Ph.D. thesis, Department of Computer Science, Stanford University.

IP, C. N. AND DILL, D. L. 1993. Efficient verification of symmetric concurrent systems. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'93).* IEEE, 230–234.

IP, C. N. AND DILL, D. L. 1996. Better verification through symmetry. *Formal Methods in System Design 9,* 1/2 (Aug.), 41–76.

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

JENSEN, K. 1996. Condensed state spaces for symmetrical coloured Petri nets. *Formal Methods in System Design 9,* 1/2 (Aug.), 7–40.

JENSEN, K. AND ROZENBERG, G., Eds. 1991. *High-level Petri Nets; Theory and Application.* Springer-Verlag, Berlin, Germany.

JØRGENSEN, J. B. AND KRISTENSEN, L. M. 1999. Computer aided verification of Lamport's fast mutual exclusion algorithm using colored Petri nets and occurrence graphs with symmetries. *IEEE Trans. Parall. Distrib. Syst. 10,* 7 (July), 714–732.

JUNTTILA, T. A. 1998. Towards well-formed algebraic system nets. In *Workshop Concurrency, Specification & Programming*, H.-D. Burkhard, L. Czaja, and P. Starke, Eds. Number 110 in Informatik-Bericht. Humboldt-Universität zu Berlin, 116–127.

JUNTTILA, T. A. 1999. Finding symmetries of algebraic system nets. *Fundamenta Informaticae 37*, 3 (Feb.), 269–289.

KINDLER, E. AND REISIG, W. 1996. Algebraic system nets for modelling distributed algorithms. *Petri Net Newsletter 51*, 16–31.

KINDLER, E. AND VÖLZER, H. 1998. Flexibility in algebraic nets. In *Application and Theory of Petri Nets 1998; Proceedings of the 19th International Conference, ICATPN'98; Lisbon, Portugal, June 1998*, J. Desel and M. Silva, Eds. Lecture Notes in Computer Science, vol. 1420. Springer-Verlag, Berlin, Germany, 345–364.

KÖBLER, J., SCHÖNING, U., AND TORÁN, J. 1993. *The Graph Isomorphism Problem: Its Structural Complexity*. Progress in Theoretical Computer Science. Birkhäuser, Boston, USA.

KREHER, D. L. AND STINSON, D. R. 1999. *Combinatorial Algorithms: Generation, Enumeration and Search*. CRC Press, Boca Raton, Florida, USA.

MANKU, G. S. 1997. Structural symmetries and model checking. Master's thesis UCB/ERL M97/92, Department of Computer Science and Engineering, University of California at Berkeley.

MANKU, G. S., HOJATI, R., AND BRAYTON, R. 1998. Structural symmetry and model checking. See Hu and Vardi [1998], 159–171.

MCKAY, B. D. 1990. Nauty user's guide (version 1.5). Tech. Rep. TR-CS-90-02, Computer Science Department, Australian National University.

MILLER, G. L. 1979. Graph isomorphism, general remarks. *J. Comput. Syst. Sci. 18*, 2 (Apr.), 128–142.

PANDEY, M. 1997. Formal verification of memory arrays. Tech. Rep. CMU-CS-97-162, School of Computer Science, Carnegie Mellon University. May. PhD thesis.

PANDEY, M. AND BRYANT, R. E. 1997. Exploiting symmetry when verifying transistor-level circuits by symbolic trajectory evaluation. See Grumberg [1997], 244–255.

PANDEY, M. AND BRYANT, R. E. 1999. Exploiting symmetry when verifying transistor-level circuits by symbolic trajectory evaluation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18*, 7 (July), 918–935.

PAPADIMITRIOU, C. H. 1995. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, USA.

REISIG, W. 1991. Petri nets and algebraic specifications. *Theor. Comput. Sci. 80*, 1 (Mar.), 1–34. Reprinted in [Jensen and Rozenberg 1991], pages 137–170.

SCHMIDT, K. 1997. How to calculate symmetries of Petri nets. Tech. Rep. MATH-AL-8-1997, Technische Universität Dresden, Germany. Sept. Revised version accepted for publication in Acta Informatica.

SCHMIDT, K. 1999. Integrating low level symmetries into reachability analysis. Informatik-Bericht 122, Humboldt-Universität zu Berlin, Institut für Informatik, Berlin, Germany.

SCHMIDT, K. AND STARKE, P. H. 1991. An algorithm to compute the symmetries of Petri nets. *Petri Net Newsletter 40*, 25–30.

STARKE, P. H. 1991. Reachability analysis of Petri nets using symmetries. *Systems Analysis Modelling Simulation 8*, 4/5, 293–303.

VALMARI, A. 1998. The state explosion problem. In *Lectures on Petri Nets I: Basic Models*, W. Reisig and G. Rozenberg, Eds. Lecture Notes in Computer Science, vol. 1491. Springer-Verlag, Berlin, Germany.

WIRSING, M. 1990. Algebraic specification. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B: Formal models and semantics. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, Chapter 13, 675–788.

HELSINKI UNIVERSITY OF TECHNOLOGY LABORATORY FOR THEORETICAL COMPUTER SCIENCE
RESEARCH REPORTS

ISBN 951-22-4885-9
ISSN 0783-5396