Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

SEN

Software Engineering

Software ENgineering

Symmetry in labeled transition systems

I.A. van Langevelde

REPORT SEN-R0303 JUNE 30, 2003

CWI is the National Research Institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organization for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-3711

Symmetry in Labeled Transition Systems

Izak van Langevelde
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT

Symmetry is defined for labeled transition systems, and it is shown how symmetrical systems can be symmetrically decomposed into components. The central question is under what conditions one such component may represent the whole system, in the sense that one symmetrical system is bisimilar to a second if and only if a component of the first is equivalent to a component of the second. The theory developed is illustrated by three case studies, i.e. the alternating bit protocol, Peterson's algorithm and the Dining Philosophers.

2000 Mathematics Subject Classification: 20B25 Finite automorphism groups of algebraic, geometric, or combinatorial structures; 68Q60 Specification and verification
1998 ACM Computing Classification System: D.2.4 Software/Program Verification; F.3.1 Specifying, verifying and reasoning about programs
Keywords and Phrases: labeled transition systems, symmetry, bisimulation equivalence, process algebra
Note: This work was carried out under project SEN2 "Specification and Analysis of Embedded Systems"

1. Introduction

Symmetry has been fruitfully exploited in the field of verification of communication protocols and distributed systems (e.g. [ID96, ES96, RS98]). For instance, communication protocols are typically insensitive to the actual data being sent and received, in which case the verification of the system can be reduced to the verification for one arbitrary datum. Classical approaches to symmetry, however, are formulated for a state-based view on systems, focusing on the assignment to state variables, as opposed to an label-based view, focusing on the sequences and combinations of observable actions as executed by the system.

Symmetry for state-based systems is defined in terms of a permutation of the state vector, i.e. the values of the state variables. For each system transition from one state to another there exists a transition from the permutation of the first to the permutation of the second. Once transitions are labeled by actions this symmetry is destroyed, since usually the action that labels the transition between the original states differs from the actions that label the transition between the permuted states. Instead of identical, the symmetrical behaviour is analogous.

This report introduces a generalisation of the classical notion of symmetry, in the sense that it covers the label-based view of systems. Instead of permutations on the state vector, it defines permutations of the actions that label transitions.

The work reported here serves as a theoretical basis. By itself, the notion of symmetry for labeled transition systems does not serve any direct practical application, since a labeled transition system is a primitive and voluminous formalisation, which is typically generated from specifications at a higher level of abstraction. Ultimately, work on symmetry should be lifted to this higher level.

The organisation of this report is as follows. Section 2 gives an overview of related work. Section 3 overviews the definitions of labeled transition systems and equivalence relations between these, and the theory of permutations. Section 4 introduces the notion of symmetry for labeled transition systems and Section 5 introduces symmetrical relations for these. Section 6 defines symmetrical reductions of labeled transition systems, addressing the question of whether a reduction can represent the original system. Section 7 describes how to benefit from symmetry in a specification. Section 8 illustrates

these notions on the hand of the alternating bit protocol, Peterson's mutual exclusion algorithm and the Dining Philosophers. Section 9 presents this report's conclusions and an overview of future work. The appendices contain the μCRL code for the case studies.

2. Related work

The use of the natural notion of symmetry in system analysis goes back to the 1980's. Initially, it was used in an ad hoc fashion in [AKS83], where it was applied in the verification of the *alternating bit protocol* by reducing a *selection/resolution model* to dimensions independent from the number of distinct messages handled. Later, in [Lub84] symmetry was used in a more general setting, to verify properties like livelocks and deadlocks for n parallel instances of a program with *Fetch&Add instructions*, by constructing a *reachability set description* independent from n. In the same year [HJJJ84] described how symmetry can be applied in the verification of properties like boundedness, coverability, reachability and liveness, for high-level Petri-Nets, and later in [Sta91] similar techniques were developed for deadlock and livelock checking in *P/T networks*. Finally, in [ID96] symmetry was exploited through a symmetrical data type of *scalar set* in reachability analysis. What these studies have in common, is that they target a limited range of properties to be verified.

It was not until the 1990's that symmetry was applied in a more general setting to verify arbitrary properties. The use of symmetry in the reduction of Kripke structures to facilitate the model checking of formulas in the temporal logic CTL^* was independently addressed in [CFJ93] and [ES96], with the difference that the former was centered around *binary decision diagrams* while the latter also addressed properties in the μ-calculus. In each of these, symmetry is defined for unlabeled transition systems.

The application of symmetry for labeled transition systems, where state transition are labeled by actions, has received relatively few attention. The only known approach is [RS98] where symmetry is used in test generation in a trace semantics context. As a consequence of symmetry being defined for an equivalence which is coarser than bisimulation, the notion of symmetry is weaker, which makes the set of symmetrical systems larger and which allows more freedom in the definition of symmetrical reduction, or *kernel* in the terminology of [RS98].

Symmetry reductions have outgrown the stage of elegant theoretical research, and made it into a number of state-of-the-art verification tools. Especially the scalar set notion from [ID96] seems to make good practical implementations, as it was included in the $Mur\varphi$ verification system [DDHY92] and the SymmSpin [BDH00] extension to the well-known model checker Spin [Hol91].

A related issue is *data independence* [LN00, Wol86]. For a wide range of systems neither the system behaviour nor the properties to be verified depend on the data manipulated by the system. The major results in this area show that under these conditions it suffices to show that the system satisfies the properties for only a finite number of finite data types, which guarantees that the system is correct for all data types. It appears that data independent systems are symmetrical, and it is conceivable how work on data independence could benefit from symmetry research.

A more loosely related topic is the theory of *abstract interpretations* [CC77], which aims at reducing a system to a system at a higher level of abstraction, possibly reducing infinite-state systems to finite systems. For some classes of systems, like communication protocols, where symmetry can be interpreted as 'the data handled is arbitrary and can thus be left unspecified', symmetry reduction closely resembles abstract interpretation.

A final relative is the voluminous work on *partial order reduction* [God90], which is motivated from the observation that in parallel systems any interleaving of parallel actions is equivalent to any other one, which suggests that the system can be reduced to one interleaving. For systems consisting of a number of similar parallel processes, this procedures bears some resemblance to symmetry reduction.

The current report is meant as a generalisation of existing research in that it studies symmetry for labeled transition systems with a bisimulation semantics. As a first step, the report fixes a number of central notions and properties; subsequent research aims at practical applications of symmetry in the tool set centered around the process-algebraic language μCRL [BFG$^+$01].

3. Preliminaries

This report formalises the notion of symmetry of labeled transition systems using well-known group-theoretic notions like permutations. This section overviews labeled transition systems and permutations, for more information on the former the reader is referred to the literature of process algebra (see [Fok00] for an introduction) and any introduction to algebra.

3.1 Labeled transition systems

The label-based view on computing centers around a system model which defines the system in terms of states and transitions, labeled by actions, between these. In the pure label-based view the nature of these states is left abstract.

Definition 3.1 (labeled transition system)
A *labeled transition system* $\mathcal{S} = \langle S, \mathcal{L}, \rightarrow, s_0 \rangle$ consists of a set of states S, a set of action labels $\mathcal{L}$, a transition relation $\rightarrow$: $S \times \mathcal{L} \times S$ and an initial state $s_0 \in S$. The set of all labeled transition systems is denoted by Sys and the set of all possible states is denoted by Σ.

Example 1
The example that will be used throughout this report is the *Positive Acknowledgement with Retransmission (PAR)* protocol, part of the TCP protocol suite [Def81]. The main principle of this protocol is that data is accompanied by a sequence number, which is to be acknowledged by the receiver before the sender is allowed to send another datum with the next sequence number; it can be easily verified that it suffices to calculate sequence numbers modulo 2, effectively reducing them to alternating bits which is why PAR is nicked *alternating bit protocol* in the more popular literature. For the sake of presentation, the protocol is simplified in that its channels may not drop acknowledgements, although they may drop data packets.

The labeled transition system in Figure 1 shows the protocol for two data elements d_1 and d_2. The actions of receiving a datum from and sending a datum to the sending and receiving user of the protocol are $r1(datum)$ and $s4(datum)$ respectively. The actions for sending and the receiving of a datum with a sequence number are $c2(datum, number)$ and $c3(datum, number)$, and the actions for sending and receiving an acknowledgement are $c5(number)$ and $c6(number)$. The action that corresponds to the dropping of a datum is $c3(e)$.

From Figure 1 the notion of symmetry is clear: each of the white nodes corresponds with one of the black states, in the sense that each transition with an action on a datum $d1$ between black nodes corresponds to a transition with the same action with $d2$ instead of $d1$, between white nodes. A special role is played by the gray states, which correspond to themselves. This notion of symmetry will be formalised in Section 4.

3.2 Equivalences for labeled transition systems

Equivalence for labeled transition systems comes in a vast range of flavours. This report generalises beyond specific equivalences by focusing on equivalences which satisfy certain properties. To support the intuition that these are not overly restrictive, it will be proven that a number of well known equivalences, i.e. isomorphism, strong bisimulation and weak bisimulation, indeed satisfy the requirements.

The notion of relation between labeled transition systems is usually defined as a relation between states, leaving the context of systems implicit. In the shape of things to come, the notion will be made more precise.

Definition 3.2 (system relation)
A *system relation* is a relation $R : Sys \times Sys \times (\Sigma \times \Sigma)$ satisfying $\langle \langle S_1, \rightarrow_1, s_1 0 \rangle, \langle S_2, \rightarrow_2, s_2 0 \rangle, R' \rangle \in R \Leftrightarrow R' \subseteq S_1 \times S_2$. When no confusion arises, the notation $\langle \mathcal{S}_1, \mathcal{S}_2, R' \rangle \in R$ will be simplified to $\mathcal{S}_1 R \mathcal{S}_2$ and if the context is clear, sRt will be written for $sR't$.

Equivalences typically come in sets, in the sense that equivalent systems are generally not just related by one single equivalence relation. For this, it is useful to define sets of relations.

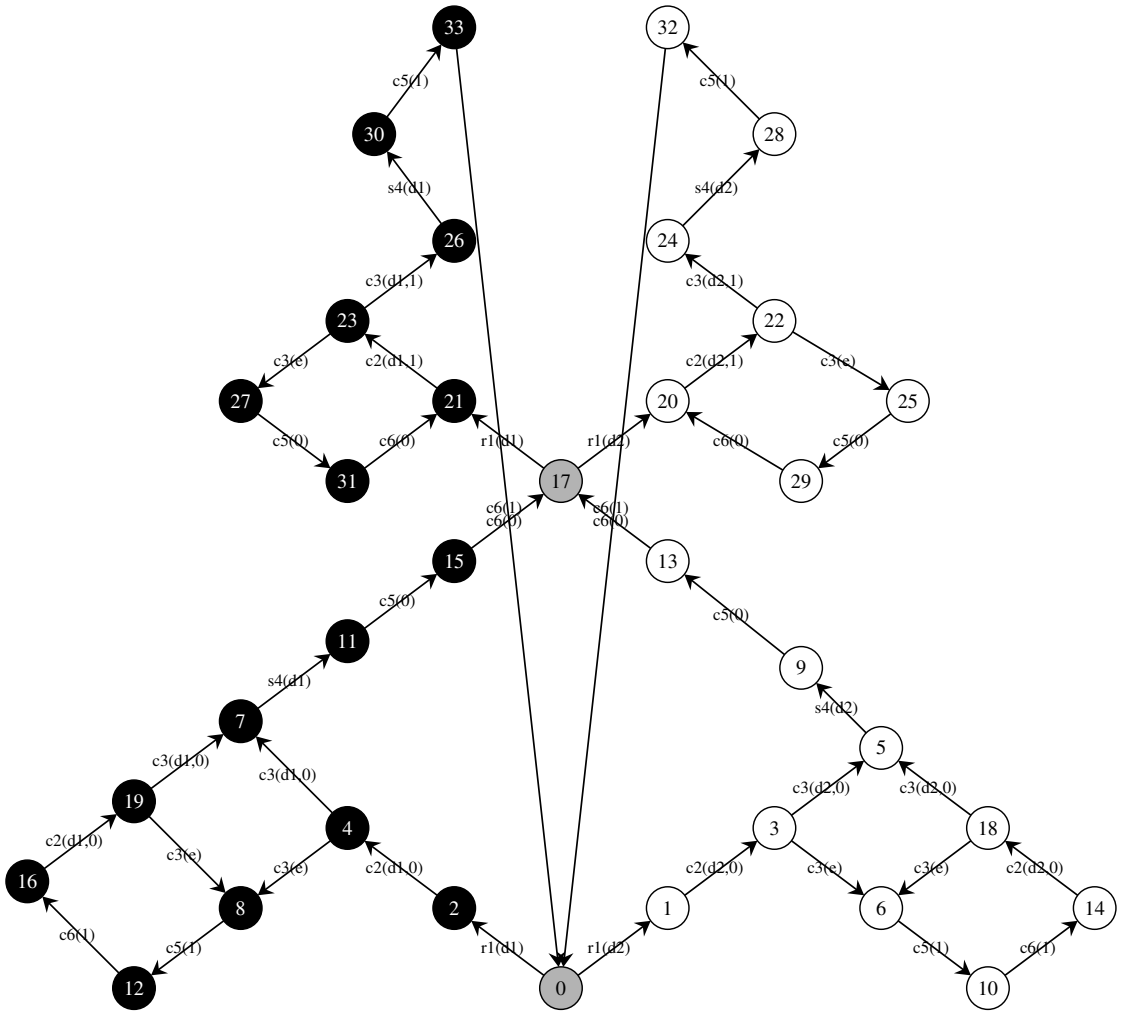


Figure 1: Labeled transition system for the alternating bit protocol

Definition 3.3 (set of relations)
Suppose $\mathcal{R}$ is a set of system relations, then two systems $\mathcal{S}_1, \mathcal{S}_2$ are related by $\mathcal{R}$, denoted by $\mathcal{S}_1 \mathcal{R} \mathcal{S}_2$, iff $\mathcal{S}_1 R \mathcal{S}_2$ for some $R \in \mathcal{R}$.

The strictest notion of equivalence after identity is that of isomorphism, which only allows system to differ in the designation of their states, leaving all labeled transitions between these intact; its origin is in graph theory (e.g. [Hof82]).

Definition 3.4 (isomorphism)
Suppose $\mathcal{S}_1 = \langle S_1, \mathcal{L}_1, \rightarrow_2, s_{01} \rangle$ and $\mathcal{S}_2 = \langle S_2, \mathcal{L}_2, \rightarrow_2, s_{02} \rangle$ are labeled transition systems, then an isomorphism from $\mathcal{S}_1$ to $\mathcal{S}_2$ is a bijection $f : S_1 \rightarrow S_2$ satisfying:
i if $s_1 \stackrel{a}{\rightarrow}_1 s_2$ then $f(s_1) \stackrel{a}{\rightarrow}_2 f(s_2)$;
ii $f(s_{01}) = s_{02}$.
If $\mathcal{S}_1 = \mathcal{S}_2$ then f is an *automorphism*. If there exists an isomorphism from $\mathcal{S}_1$ to $\mathcal{S}_2$ then $\mathcal{S}_1$ and $\mathcal{S}_2$ are *isomorph*, which is denoted by $\mathcal{S}_1 \equiv \mathcal{S}_2$.

Definition 3.5 (strong bisimulation)
Suppose $\mathcal{S}_1 = \langle S_1, \mathcal{L}, \rightarrow_1, s_1 \rangle$ and $\mathcal{S}_2 = \langle S_2, \mathcal{L}, \rightarrow_2, s_2 \rangle$ are labeled transition systems. A *strong bisimulation* between $\mathcal{S}_1$ and $\mathcal{S}_2$ is a relation $R : S_1 \times S_2$ which satisfies the following:
i $s_1 R s_2$
ii if $t_1 R t_2$ and $t_1 \stackrel{a}{\rightarrow}_1 t_1'$ then $t_1' R t_2'$ and $t_2 \stackrel{a}{\rightarrow}_2 t_2'$ for a state $t_2' \in S_2$
iii if $t_1 R t_2$ and $t_2 \stackrel{a}{\rightarrow}_2 t_2'$ then $t_1' R t_2'$ and $t_1 \stackrel{a}{\rightarrow}_1 t_1'$ for a state $t_1' \in S_1$
If a strong bisimulation between $\mathcal{S}_1$ and $\mathcal{S}_2$ exists then these systems are called *strong bisimilar*; this is denoted by $\mathcal{S}_1 \sim_s \mathcal{S}_2$.

In the following definition, the notation $\rightarrow^*$ is used for the reflexive and transitive closure of $\rightarrow$. The action τ plays a special role here, in that it represents the hidden, or internal, actions of the system.

Definition 3.6 (weak bisimulation)
Suppose $\mathcal{S}_1 = \langle S_1, \mathcal{L}, \rightarrow_1, s_1 \rangle$ and $\mathcal{S}_2 = \langle S_2, \mathcal{L}, \rightarrow_2, s_2 \rangle$ are labeled transition systems, then a *weak bisimulation* between $\mathcal{S}_1$ and $\mathcal{S}_2$ is a relation $R : S_1 \times S_2$ which satisfies the following:
- $s_1 R s_2$
- if $t_1 R t_2$ and $t_1 \stackrel{a}{\rightarrow}_1 t_1'$ then $t_1' R t_2'$ and $t_2 \stackrel{\tau}{\rightarrow}_2^* \cdot \stackrel{a}{\rightarrow}_2 \cdot \stackrel{\tau}{\rightarrow}_2^* t_2'$ for a state $t_2' \in S_2$ or $a = \tau$ and $t_1' R t_2$.
- if $t_1 R t_2$ and $t_2 \stackrel{a}{\rightarrow}_2 t_2'$ then $t_1' R t_2'$ and $t_1 \stackrel{\tau}{\rightarrow}_1^* \cdot \stackrel{a}{\rightarrow}_1 \cdot \stackrel{\tau}{\rightarrow}_1^* t_1'$ for a state $t_1' \in S_1$ or $a = \tau$ and $t_1 R t_2'$.
If a weak bisimulation between $\mathcal{S}_1$ and $\mathcal{S}_2$ exists then these systems are called *weakly bisimilar*; this is denoted by $\mathcal{S}_1 \sim_w \mathcal{S}_2$.

The choice of equivalences presented here is relatively arbitrary, and certainly meant as complete. For a wide spectrum of equivalences the reader is refered to [Gla90].

3.3 Permutations
A permutation of a set corresponds to a reordering of the elements of the set. The permutations of a given set form a *group*, i.e. it is closed under composition, composition is associative, it contains the identity function and each permutation has an inverse. Therefore, the theory of permutations may draw from the vast body of group theory. This report just relies on the basic notion of permutation.

Definition 3.7 (permutation)
Suppose S is a set.
i a *permutation* of S is a bijection $\pi : S \rightarrow S$.
ii a *cycle of length* k is a permutation π for which distinct $s_1, \cdots, s_k \in S$ exist satisfying $\pi(s_i) = s_{(i \bmod k)+1}$ $(1 \leqslant i \leqslant k)$, and $\pi(x) = x$ for all $x \in S / \{s_1, \cdots, s_k\}$.
iii the order of a permutation π is the smallest n for which π^n is the identity function.

Lemma 3.8
The set of permutations of a set S forms a group under composition.

A useful property of permutations used in this report is that a permutation can be written as a set of disjoint cycles. Thus, a permutation partitions its domain into a number of disjoint *orbits*.

Lemma 3.9
A permutation is the product of disjoint cycles of length greater than 1.

A cycle of length k can be written in the form $\langle a, \pi(a), \pi^2(a), \cdots, \pi^{k-1}(a)\rangle$. A permutation will be written as the composition of its disjoint cycles. Finally, the elements permuted in one cycle form an orbit.

Definition 3.10 (orbit)
Suppose π is a permutation on set S. The π-*orbit* of $x \in S$, denoted by $orb_\pi(x)$, is defined as $orb_\pi(x) = \{\pi^n(x) \mid n \geqslant 0\}$. The *period of* x, denoted by $per(x)$, is defined as $per(x) = \mid orb_\pi(x) \mid$.

Lemma 3.11
Suppose π is a permutation on a set S, then the relation $\sim_\pi$ defined by $x \sim_\pi y \Leftrightarrow orb_\pi(x) = orb_\pi(y)$ is an equivalence relation.

The next section extends the theory of permutations to labeled transition systems.

4. Symmetrical labeled transition systems

Symmetry in labeled transition system is induced by a permutation of action labels. If permuting the labels in a labeled transition system yields an isomorphic system, then the system is symmetric with respect to this label permutation.

Definition 4.1 (label permutation)
Suppose $\mathcal{S} = \langle S, \mathcal{L}, \rightarrow, s_0\rangle$ is a labeled transition system.

i An *label permutation* for $\mathcal{S}$ is a permutation $\pi : \mathcal{L} \rightarrow \mathcal{L}$.

ii The definition of π is extended to apply to transition relations by $t \xrightarrow{a} t' \Leftrightarrow t \overset{\pi(a)}{\pi(\rightarrow)} t'$

iii The definition of π is extended to apply to labeled transition systems by $\pi\mathcal{S} = \langle S, \mathcal{L}, \pi(\rightarrow), s_0\rangle$

Definition 4.2 (state permutation)
Suppose $\mathcal{S} = \langle S, \mathcal{L}, \rightarrow, s_0\rangle$ is a labeled transition system.

i A *state permutation* for $\mathcal{S}$ is a permutation $\sigma : S \rightarrow S$.

ii The definition of σ is extended to apply to transition relations by $t \xrightarrow{a} t' \Leftrightarrow \sigma t \overset{a}{\sigma(\rightarrow)} \sigma t'$

iii The definition of σ is extended to apply to labeled transition systems by $\sigma\mathcal{S} = \langle S, \mathcal{L}, \sigma(\rightarrow), \sigma s_0\rangle$

The notions of state permutation and isomorphism are closely related.

Lemma 4.3
Suppose $\mathcal{S}$ is a labeled transition system and σ is a state permutation, then σ is an isomorphism from $\mathcal{S}$ to $\sigma\mathcal{S}$.

Proof
Trivial. □

Since state permutations act on states and label permutations act on labels, the two are independent.

Lemma 4.4
Suppose $\mathcal{S} = \langle S, \mathcal{L}, \rightarrow, s_0\rangle$ is a labeled transition system, π is a label permutation for $\mathcal{S}$ and σ is an isomorphism for $\mathcal{S}$, then

i $\pi\sigma \rightarrow = \sigma\pi \rightarrow$
ii $\pi\sigma\mathcal{S} = \sigma\pi\mathcal{S}$

Proof
i For arbitrary $\langle s, a, s' \rangle \in \rightarrow$ it holds that $\pi\sigma\langle s, a, s' \rangle = \pi\langle \sigma s, a, \sigma s' \rangle = \langle \sigma s, \pi a, \sigma s' \rangle = \sigma\langle s, \pi a, s' \rangle = \sigma\pi\langle s, a, s' \rangle$. Hence, $\pi\sigma \rightarrow = \sigma\pi \rightarrow$
ii By Definition 4.1 and (i), $\pi\sigma\mathcal{S} = \pi\langle \sigma S, \mathcal{L}, \sigma \rightarrow, \sigma s_0 \rangle = \langle S, \mathcal{L}, \pi\sigma \rightarrow, \sigma s_0 \rangle = \langle \sigma S, \mathcal{L}, \sigma\pi \rightarrow, \sigma s_0 \rangle = \sigma\pi\mathcal{S}$

□

Now, label permutations and state permutations can be combined into one system permutation.

Definition 4.5 (system permutation)
Suppose $\mathcal{S} = \langle S, \mathcal{L}, \rightarrow, s_0 \rangle$ is a labeled transition system, then a *system permutation* for $\mathcal{S}$ is a composition $\pi\sigma$, where π is a label permutation for $\mathcal{S}$ and σ is a state permutation for $\mathcal{S}$.

Permutations respect isomorphisms, in the sense that permuted labeled transition systems are isomorphic and only if the originals are. This follows from the following lemma, which states the independence of permutation and isomorphism.

Lemma 4.6
Suppose $\mathcal{S}_1, \mathcal{S}_2$ are labeled transition systems and π is a label permutation, then $\mathcal{S}_1 \equiv \mathcal{S}_2 \Leftrightarrow \pi\mathcal{S}_1 \equiv \pi\mathcal{S}_2$

Proof
Suppose σ is an isomorphism from $\mathcal{S}_1$ to $\mathcal{S}_2$. Then it follows that this same σ is an isomorphism from $\pi\mathcal{S}_1$ to $\pi\mathcal{S}_2$, by $\sigma\pi\mathcal{S}_1 = \pi\sigma\mathcal{S}_1 = \pi\mathcal{S}_2$, using the previous Lemma. The reverse half of the proof is analogous. □

The analogue of Lemma 3.8 for labeled transition systems holds trivially.

Lemma 4.7
i The set of label permutations of a labeled transition system $\mathcal{S}$ is a group.
ii The set of state permutations of a labeled transition system $\mathcal{S}$ is a group.
iii The set of system permutations of a labeled transition system $\mathcal{S}$ is a group.

Proof
i Directly from Definition 4.1 and Lemma 3.8.
ii Directly from Definition 4.2 and Lemma 3.8.
iii Suppose $\pi_1\sigma_1$ and $\pi_2\sigma_2$ are system permutations for $\mathcal{S}$. First, using Lemma 4.4 it follows that $\pi_1\sigma_1\pi_2\sigma_2 = \pi_1\pi_2\sigma_1\sigma_2$. By (ii) the composition $\sigma_1\sigma_2$ is a state permutation for $\mathcal{S}$, and by (i) the composition $\pi_1\pi_2$ is a label permutation for $\mathcal{S}$. Hence, $\sigma_1\pi_1\sigma_2\pi_2$ is a system permutation isomorphism for $\mathcal{S}$. Second, by (i) and (ii) both π_1, π_2 and σ_1, σ_2 have an inverse, so $(\pi_1\sigma_1)^{-1} = \sigma_1^{-1}\pi_1^{-1}$, i.e. also $\pi_1\sigma_1$ has an inverse.

□

Definition 4.8 (permutation isomorphism)
Suppose $\mathcal{S}_1, \mathcal{S}_2$ are labeled transition systems and π is a label permutation for both $\mathcal{S}_1$ and $\mathcal{S}_2$, and σ is an isomorphism from $\mathcal{S}_1$ to $\mathcal{S}_2$. Then $\pi\sigma$ is a *permutation isomorphism* with respect to π. If $\mathcal{S}_1 = \mathcal{S}_2$ then $\pi\sigma$ is a *permutation automorphism.*

Lemma 4.9
Suppose $\mathcal{S}$ is a labeled transition system and $\pi\sigma$ is a system permutation, then $\pi\sigma$ is a permutation isomorphism from $\mathcal{S}$ to $\pi\sigma\mathcal{S}$.

Proof
Trivial. □

Now, the notion of symmetry can be defined. A system is symmetric if it is insensitive to permutation.

Definition 4.10 (symmetric labeled transition system)
A labeled transition system is *symmetric* with respect to label permutation π iff there exists a permutation automorphism based on π.

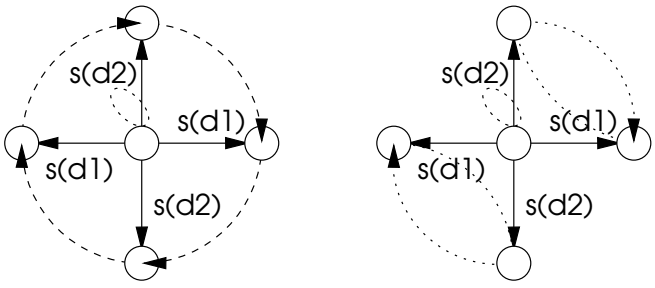
Example 2
Consider the running example introduced in Figure 1. Define the label permutation π to reflect that each action on datum d_1 corresponds to this same action with $d1$ replaced by $d2$, that is:

$$\begin{aligned}\pi = \{&\langle r1(d1), r1(d2)\rangle,\\ &\langle c2(d1,0), c2(d2,0)\rangle,\\ &\langle c2(d1,1), c2(d2,1)\rangle,\\ &\langle c3(d1,0), c3(d2,0)\rangle,\\ &\langle c3(d1,1), c3(d2,1)\rangle,\\ &\langle s4(d1), s4(d2)\rangle\}\end{aligned}$$

The permutation automorphism induced by π corresponds to the geometrical symmetry in Figure 1: each black state on the left corresponds to its white mirror image on the right, and vice versa.

Summarising, a label permutation gives rise to one or more permutation isomorphisms. The following example shows that the order of a permutation isomorphism can be higher than the order of the underlying label permutation. In some sense, the complexity of the symmetry of the transition system can be higher than that of the label permutation.

Example 3
The following labeled transition system represents a trivial system, which sends one of two data elements and, subsequently, halts. The obvious label permutation permutes the two data elements $d1$ and $d2$; it is a permutation of order 2. The permutation isomorphism, indicated by dotted arrows, on the left is of order 4, while the one on the right is of order 2.



This example gives rise to two observations. First, the labeled transition system has a non-trivial automorphism and, second, the permutation isomorphism of order 2 can be obtained by 'short-cutting' the orbits of the larger isomorphism. The following result states that these two hold in general.

Lemma 4.11
Suppose $\mathcal{S}$ is a labeled transition system that is symmetric with respect to a label permutation π of order n.

i if there is a permutation isomorphism for $\mathcal{S}$ of order greater than n then $\mathcal{S}$ has a non-trivial automorphism.
ii there is a permutation isomorphism for $\mathcal{S}$ which order does not exceed n.

Proof
i Suppose $\pi\sigma$ is a permutation isomorphism for $\mathcal{S}$ of order greater than n. Then $(\pi\sigma)^n\mathcal{S} = \pi^n\sigma^n\mathcal{S} = \sigma^n\mathcal{S} \neq \mathcal{S}$, so σ^n is a non-trivial automorphism.
ii Suppose $\pi\sigma$ is a permutation isomorphism for $\mathcal{S}$. It holds that $(\pi\sigma)^n\mathcal{S} = \pi^n\sigma^n\mathcal{S} = \sigma^n\mathcal{S}$. Proceed by induction on the number of orbits of σ^n which contain a state s with $\sigma^n s \neq s$. If there are no such orbits, then $\sigma^n\mathcal{S} = \mathcal{S}$ and it follows that the order of $\pi\sigma$ is at most n. Otherwise, consider $orb(s)$ for some s with $\sigma^n s \neq s$; suppose $orb(s)$ is of length l. It holds that for all $i \geqslant 0$ all states of the form $\sigma^{i+jn}s$ $(j \geqslant 0)$ are isomorph. So, $orb(s)$ can be partitioned into isomorph sequences of states, i.e. $orb(s) = \langle s, \cdots, \sigma^{k-1}s; \sigma^k s, \cdots, \sigma^{2k-1}s; \cdots; \sigma^{l-k}s, \cdots, \sigma^{l-1}s\rangle$ with $k = \gcd(n, l)$ (i.e. k is the greatest common divisor of n and l). Now decompose $orb(s)$ into orbits of shorter length by defining a isomorphism $\sigma' \colon S \to S$ as follows: $\sigma'\sigma^i s = \begin{cases} \sigma^{i+1}s & (i+1 \bmod k \neq 0) \\ \sigma^{i-k+1}s & (i+1 \bmod k = 0) \end{cases}$
For σ' the number of orbits of σ'^n which contain a state s with $\sigma^n s \neq s$ is smaller than for σ, so the hypothesis applies to σ', which completes the proof.

□

5. Symmetrical relations

In the context of equivalence relations and permutation isomorphisms, the question whether a permutation isomorphism leaves intact an equivalence is a natural one. More precise, the question is whether some equivalence relation is closed under permutation isomorphisms. To address this property, first it will be defined how a relation itself can be permuted.

Definition 5.1 (permuted relation)
Suppose π is a label permutation, σ_1, σ_2 are system permutations for S based on π and $R : Sys \times Sys$ is a relation, then the *permutation of R with respect to σ_1, σ_2*, denoted by R_{σ_1,σ_2}, is defined by

$$(\sigma_1\mathcal{S}_1)R_{\sigma_1,\sigma_2}(\sigma_2\mathcal{S}_2) \Leftrightarrow \mathcal{S}_1 R \mathcal{S}_2$$

It is easy to see that, generally, classes of equivalence relations are closed under permutation isomorphisms. For instance, if two systems are bisimilar, then permutations of these, based on the same label permutation, are also bisimilar. The underlying reason is that equivalences are typically defined without attaching special meaning to designated states or labels. These properties are defined as *state independence* and *label independence*.

Definition 5.2 (state independence)
A set of relations $\mathcal{R} : Sys \times Sys$ is *state independent* iff for each state permutation σ it holds that $\sigma\mathcal{R} = \mathcal{R}$.

Definition 5.3 (label independence)
A set of relations $\mathcal{R} : Sys \times Sys$ is *label independent* iff for each label permutation π it holds that $\pi\mathcal{R} = \mathcal{R}$.

Definition 5.4 (symmetrical relation)
A set of relations $\mathcal{R} : Sys \times Sys$ is *symmetrical* iff for each system permutation $\pi\sigma$ it holds that $\pi\sigma\mathcal{R} = \mathcal{R}$.

The following lemma follows directly from the definitions.

Lemma 5.5
A set of relations which is both label independent and state independent is symmetrical.

For the equivalences addressed in this report both types of independence hold trivially, with the one exception of weak bisimulation which attaches special meaning to the 'silent τ action'. If permutations are limited to those leaving intact this silent action, then also weak bisimulation is label independent.

Lemma 5.6
i Isomorphy is state independent.
ii Strong bisimulation is state independent.
iii Weak bisimulation is state independent.

Proof
Directly from the definitions. □

Lemma 5.7
i Isomorphy is label independent.
ii Strong bisimulation is label independent.
iii Weak bisimulation is label independent for labels different from τ.

Proof
Directly from the definitions. □

Corollary 5.8
i Isomorphy is symmetrical.
ii Strong bisimulation is symmetrical.
iii Weak bisimulation is symmetrical.

6. Symmetrical reductions

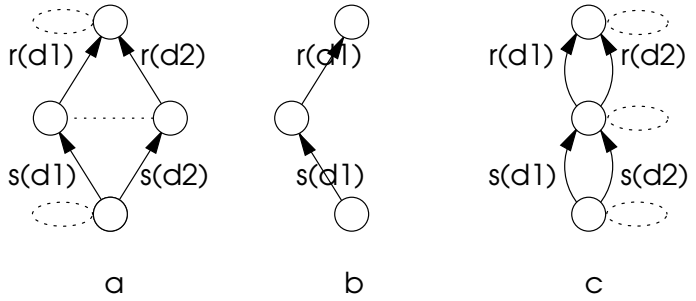
This section describes the definition of a 'symmetric half', or *symmetrical reduction*, of a symmetrical labeled transition system. A suitable definition must satisfy the requirement that equivalent symmetric halves originate from equivalent labeled transition systems, for the equivalence relation used. Another useful requirement is that equivalent labeled transition systems have equivalent symmetric halves.

The section is structured as follows. Section 6.1 gives the basic definitions of reduction and expansion. Section 6.2 outlines how to generate the original labeled transition system from a symmetrical reduction and how to calculate the dimensions of the original system from the reduction. Section 6.3 addresses one of the main issues in this report, i.e. whether a symmetrical reduction can represent the original. Finally, Section 6.4 shows how to represent a symmetrical reduction in a labeled transition system.

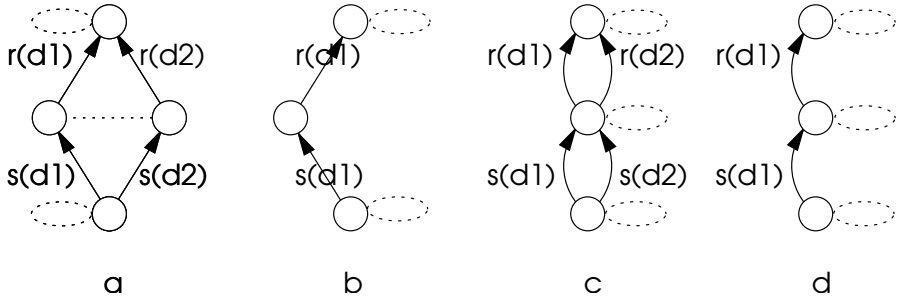
6.1 Definitions

To provide some intuition on the nature of symmetrical reductions, this section starts with an example.

Example 4
The labeled transition system (a) below represents a simple communication protocol which sends and, subsequently, receives a data element. This system is symmetrical for the label permutation $\pi = \{\langle r(d1), r(d2)\rangle, \langle s(d1), s(d2)\rangle\}$; the permutation isomorphism is shown in dotted lines. A naive candidate reduction is obtained by choosing one transition from each orbit, as shown in (b) below. However, the labeled transition system (c), representing a flawed version of the same protocol, has the same reduction.



The problem with this naive reduction is that although using the label permutation all symmetric halves can be generated, the information as to how these halves should be merged to form a complete symmetrical system is missing. The key to the solution is to add to the naive reduction shown above information on how states from the reduction have symmetric counterparts within this reduction. The picture below shows for both (a) and (c) from above the restriction of the symmetrical isomorphism to the naive reduction in (b) and (d), respectively. They are different.



The intuition from the above example is formalised in the definition of symmetrical reduction. A complication that arises is that states from the reduction may be mapped onto other states by repeated applications of the symmetrical reduction. So, for every number of applications this indirect mapping needs to be stored.

Definition 6.1 (symmetrical reduction)
A *symmetrical reduction* $\mathcal{C}_\rho$ of order n consists of a labeled transition system $\mathcal{C}$ and a partial function $\rho : \{1, \cdots, n-1\} \rightarrow (S \rightarrow S)$

Equivalence relations for labeled transition systems are extended to apply to symmetrical reductions by the following.

Definition 6.2 (equivalence for symmetrical reductions)
An equivalence relation for labeled transition systems R is extended to apply to symmetrical reductions of order n by: $C_\rho R C'_{\rho'}$ iff the following hold:

i $\mathcal{C} R \mathcal{C}'$

ii $sRs' \Rightarrow \rho(i) s R \rho'(i) s'$ for all s, s' and all $1 \leqslant i < n$

Now that the shape of a symmetrical reduction has been defined, it can be defined when a reduction is a reduction of some symmetrical system.

Definition 6.3 (symmetrical reduction)
Suppose $\mathcal{S} = \langle S, \mathcal{L}, \rightarrow_s, s_0 \rangle$ is a symmetrical system with respect to system permutation $\sigma\pi$ of order n and $\mathcal{C}_\rho$ is a symmetrical reduction with $\mathcal{C} = \langle C, \mathcal{L}, \rightarrow_c, c_0 \rangle$, then $\mathcal{C}_\rho$ is a *reduction of* $\mathcal{S}$ iff the following hold:

i $\rho(i) = \sigma^i \mid_{S \to S}$ for all $1 \leqslant i < n$.
ii for all $\langle s, a, t\rangle \in \to_s$ there exists exactly one $\langle s', a', t'\rangle \in \to_c$ which satisfies $\langle s', a', t'\rangle = \langle \sigma^i s', \pi^i a', \sigma^i t'\rangle$ for some $0 \leqslant i < n$.

Conversely, $\mathcal{S}$ is an *expansion of* $\mathcal{C}_\rho$.

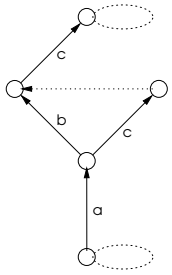
The definition of permutation is extended to symmetrical reductions in a straightforward way.

Definition 6.4 (permutation)
Suppose $\pi\sigma$ is a system permutation and $\mathcal{C}_\rho$ is a symmetrical component, then $\pi\sigma\mathcal{C}_\rho$ is defined as $\pi\sigma\mathcal{C}_\rho = (\pi\sigma\mathcal{C})_{\rho'}$ where ρ' is defined by $\rho'(i)(\sigma s) = \sigma\rho(i)(s)$.

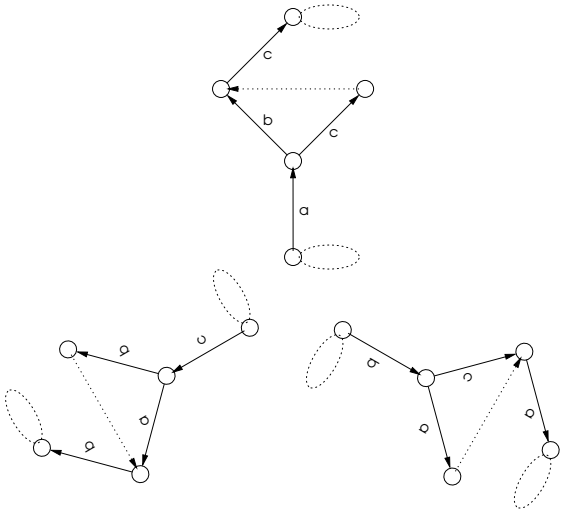
6.2 Generating symmetrical expansions
The first step in restoring an original labeled transition system from a symmetrical reduction is to generate the permutations of the labeled transition system with respect to the underlying label permutation. The next step is to string these permutations together, using reduction's ρ mapping.

The algorithm is outlined in Table 1. It is designed more for clarity than for efficiency. It leaves implicit the number of states which are actually defined equivalent in the equivalence relation. To clarify the underlying principles, the next example presents a sample expansion.

Example 5
Consider the following label permutation $\pi = \langle a, b, c\rangle$. A sample symmetrical reduction is presented by the following picture; here, all dotted arrows represent $\rho(1)$ mappings.



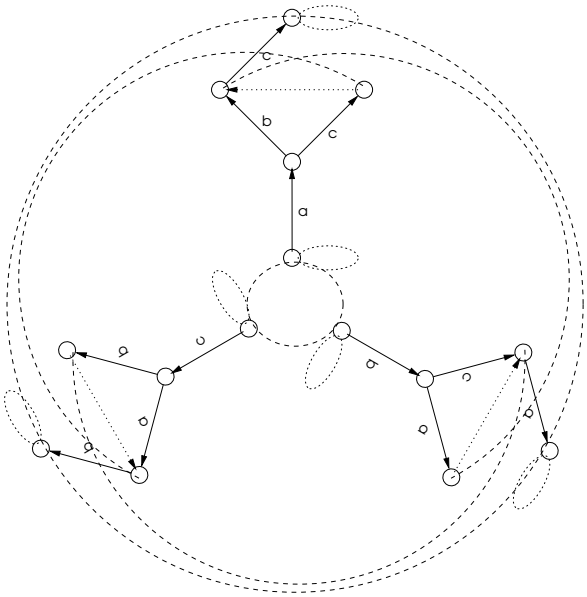
The first step is the generation of the three permutation of the above reduction.



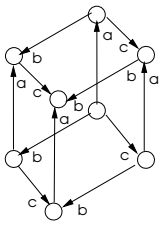
The equivalence relation generated from the dotted $\rho(1)$ steps are as follows.

```
/* Extend equivalence relation 'R' with 'sRt' */

procedure relate(R,s,t);


/* Add to labeled transition system 'S' all states and transition from 'T' */

procedure unite(S,T);


/* Generate a symmetrical expansion 'S' for the reduction given by 'C' and 'rho'
   with respect to label permutation 'pi' of order 'n' */

procedure expand(C, rho, pi, n, S)
   var LTS C[n];
       int i;

   /* Generate the permutations of 'C' */

   C[0]=C;
   for i=1 to n-1 do
      C[i]=pi(C[i-1]);
   done;

   /* String the permutations together */

   for i=0 to n-1 do
      for each state s of C[i] do
         if   rho(i)(s) is defined
         then relate(R,<i,rho(i)(s)>,<i+1 mod n,s>);
      done;
   done;

   /* Construct the expansion */

   S=C[0];
   for i=1 to n-1 do
      unite(S,C[i]);
   done;

   S=S/R;

end procedure
```

Table 1: An algorithm for generating symmetrical expansions



Finally, 'dividing out' this equivalence relation and dropping the dotted and dashed links yields the following 'cube'.



It is not trivial to calculate the dimensions of the expansion from the dimensions of the reduction. The rest of this section is geared towards this calculation.

The key to the calculation is in *ρ cycles* which determine how many permutations share one single state.

Definition 6.5 (ρ cycle)
Suppose C_ρ is a symmetrical reduction.

A *ρ cycle* of length $l > 0$ is a set of states $\{s_0, \cdots, s_{l-1}\}$ where for all $0 \leqslant i < l$ there exists some k for which $\rho(k)s_i = s_{i+1 mod l}$.

The ρ cycles from the symmetrical reduction are permuted themselves with the reduction. A complication here is that these ρ cycles may have a period that is smaller than the order n of the permutation. That is, the n permutations of the reduction may show duplicate cycles.

Definition 6.6 (order of a ρ cycle)
Suppose $\mathcal{S}$ is a labeled transition system with symmetrical reduction C_ρ. The *order of a ρ cycle* $\{s_0, \cdots, s_{l-1}\}$ is the smallest $k > 0$ for which $\sigma^k s_i = s_i$ for all $0 \leqslant i < l$.

The sharing of transitions among permutations of the reductions is a bit more complex, since the action labels are involved here. To this end, the period of a transition is defined.

Lemma 6.7
Suppose $\mathcal{S}$ is a labeled transition system with symmetrical reduction C_ρ of the same order n.

i $\mathrm{per}_\sigma(s) = \begin{cases} k & \text{if } \rho(k)(s) = s \\ n & \text{otherwise} \end{cases}$
ii $\mathrm{per}_{\sigma\pi}(\langle s, a, t\rangle) = \mathrm{lcm}(\mathrm{per}_\sigma(s), \mathrm{per}_\pi(a), \mathrm{per}_\sigma(t))$
Here, $\mathrm{lcm}(x, y, z)$ denotes the least common multiple of three natural numbers x, y and z.

Proof
Suppose $\pi\sigma$ is the symmetrical automorphism on which ρ is based.
i Consider the smallest $0 < k \leqslant n$ for which $\sigma^k s = s$. Then $s \in C \cap \sigma^k C$. If $k < n$ then $\rho(k)s = s$.
ii Consider the smallest $0 < k \leqslant n$ for which $\sigma\pi^k\langle s, a, t\rangle = \langle s, a, t\rangle$. That is, $\sigma^k s = s$, $\sigma^k t = t$ and $\pi^k a = a$ and it follows that $k = \mathrm{lcm}(\mathrm{per}_\sigma(s), \mathrm{per}_\pi(a), \mathrm{per}_\sigma(t))$.

□

Now that some insight is gained in the sharing of states and transitions, the calculation can be completed.

Lemma 6.8 (counting lemma)
Suppose $\mathcal{S}$ is a labeled transition system with symmetrical reduction C_ρ of the same order n, the number of cycles of period i of length j is denoted by $n_{i,j}$ and the number of transitions of period i is m_i. Then the following hold:
i $\mid S \mid = n \cdot \mid C \mid - \sum_{k=1}^{n} \sum_{l=1}^{n} n_{k,l}(nl - k)$
ii $\mid\!\rightarrow\!\mid = n \cdot \mid\!\rightarrow_C\!\mid - \sum_{k=1}^{n} m_k \cdot (k - 1)$

Proof
i Let $\sigma\pi$ be a permutation automorphism for $\mathcal{S}$. For notational convenience, write $C_i = (\sigma\pi)^i C$. Now $\mathcal{S} = C_0 \cup \cdots \cup C_{n-1}$; however, the C_i are not mutually disjoint. Suppose there is a state $s \in C_{i_0} \cap \cdots \cap C_{i_{l-1}} (0 < l, 0 \leqslant i_0 < \cdots < i_{l-1} < n)$ and $s \notin C_i$ for each $i \notin \{i_0, \cdots, i_{l-1}\}$. From $s \in C_{i_1}$ it follows that $s = \sigma^{i_1 - i_0} s'$ for some $s' \in C_{i_0}$. Since $s \in C_{i_0}$ it holds that $\rho(i_1 - i_0)(s')$ is defined. By this argument, it holds $\rho(i_{j+1} - i_j)(s')$ is defined for all $1 \leqslant j < l$. So, each state shared by l permutations of C_{i_0} corresponds to a ρ cycle in C_{i_0} of length l and it can be easily proven that the converse correspondence also holds.
By symmetry, both the shared state and the corresponding ρ cycle have a permutation in C. Now this ρ cycle has k distinct permutations in the n permutations of C, where k is the period of the cycle.
Summarising, there are n permutations of $\mid C \mid$ states each. Each of the $n_{k,l}$ ρ cycles of length l and period k in C corresponds to k classes of $\frac{nl}{k}$ states, all of which have the same permutation.
ii There are n permutations of $\mid\!\rightarrow\!\mid$ transitions each. However, each of the m_k transitions of period k yields only k distinct permutations.

□

At first sight, the calculation of the dimensions of the expansion is complex. However, as will be shown in Section 8, in practice this calculation is feasible, since occurrences of transitions with a period smaller than the order of the permutations are rare.

6.3 Soundness and completeness
The question of whether a symmetrical reduction represents the original system is formalised in terms of soundness and completeness.

Definition 6.9 (soundness/completeness)
Suppose $\mathcal{C}_\rho$ is a symmetrical reduction of labeled transition system $\mathcal{S}$ and $\mathcal{R}$ is a set of relations.
i $\mathcal{C}_\rho$ is a *sound reduction of $\mathcal{S}$ with respect to $\mathcal{R}$* iff for all symmetrical reductions $\mathcal{C}'_{\rho'}$ of labeled transition systems $\mathcal{S}'$ it holds that $\mathcal{C}_\rho \mathcal{R} \mathcal{C}'_{\rho'} \Rightarrow \mathcal{S}\mathcal{R}\mathcal{S}'$.
ii $\mathcal{C}_\rho$ is a *complete reduction of $\mathcal{S}$ with respect to $\mathcal{R}$* iff for all symmetrical reductions $\mathcal{C}'_{\rho'}$ of labeled transition systems $\mathcal{S}'$ it holds that $\mathcal{S}\mathcal{R}\mathcal{S}' \Rightarrow \mathcal{C}_\rho \mathcal{R} \mathcal{C}'_{\rho'}$.

The proof that symmetrical reduction is sound for some equivalence relation has the following structure. First, it needs to be shown that application of the permutation isomorphism to two equivalent reductions yields another pair of equivalences; this requires that the equivalence relation is closed under permutation. Second, repeated applications of the permutation isomorphism yields two sequences of reductions, which are element-wise equivalent, and which are to be merged into two equivalent labeled transition systems; this requires that the equivalence relation is *compositional* with respect to the merge operator. Merging permutations of symmetrical reductions consists of taking all states and transitions of the reductions.

Definition 6.10 (sum of labeled transition systems)
Suppose $\mathcal{S}_1 = \langle S_1, \mathcal{L}, \rightarrow_1, s_1 \rangle$ and $\mathcal{S}_2 = \langle S_2, \mathcal{L}, \rightarrow_2, s_1 \rangle$ are labeled transition systems with the same initial state, then the sum of $\mathcal{S}_1$ and $\mathcal{S}_2$, denoted by $\mathcal{S}_1 + \mathcal{S}_2$ is defined by $\mathcal{S}_1 \cup \mathcal{S}_2 = \langle S_1 \cup S_2, \mathcal{L}, \rightarrow_1 \cup \rightarrow_2, s_1 \rangle$.

As expected, this sum definition has the properties of idempotency, commutativity and associativity, so in notations brackets can be dropped.

Lemma 6.11
Suppose $\mathcal{S}_1, \mathcal{S}_2$ and $\mathcal{S}_3$ are labeled transition systems.
i $\mathcal{S}_1 + \mathcal{S}_1 = \mathcal{S}_1$
ii $\mathcal{S}_1 + \mathcal{S}_2 = \mathcal{S}_2 + \mathcal{S}_1$
iii $\mathcal{S}_1 + (\mathcal{S}_2 + \mathcal{S}_3) = (\mathcal{S}_1 + \mathcal{S}_2) + \mathcal{S}_3$

Proof
Trivial. □

Definition 6.12 (soundness/completeness)
Suppose $\mathcal{C}_\rho$ is a symmetrical reduction and R is a relation.
i $\mathcal{C}_\rho$ is *sound with respect to* R iff for all system permutations $\pi\sigma$, symmetrical reductions $\mathcal{C}'_{\rho'}$ and all system permutations $\pi\sigma'$ it holds that $(\sigma^i\mathcal{C}_\rho)R_{\sigma^i\sigma'^i}(\sigma^i\mathcal{C}'_{\rho'})$ for all $0 \leqslant i < n \Rightarrow (\sigma^0\mathcal{C} + \cdots + \sigma^{n-1}\mathcal{C})(R_{\sigma^0\sigma'^0} \cup \cdots \cup R_{\sigma^{n-1}\sigma'^{n-1}})(\sigma^0\mathcal{C}' + \cdots + \sigma^{n-1}\mathcal{C}')$.
ii $\mathcal{C}_\rho$ is *complete with respect to* R iff for all symmetrical reductions $\mathcal{C}'_{\rho'}$ it holds that $(\sigma^0\mathcal{C} + \cdots + \sigma^{n-1}\mathcal{C})(R_{\sigma^0\sigma'^0} \cup \cdots \cup R_{\sigma^{n-1}\sigma'^{n-1}})(\sigma^0\mathcal{C}' + \cdots + \sigma^{n-1}\mathcal{C}') \Rightarrow \sigma^i\mathcal{C}_\rho R_{\sigma^i\sigma'^i}\sigma^i\mathcal{C}'_{\rho'}$ for all $0 \leqslant i < n$.

With the previous definitions, one of the main results of this sections can be formulated.

Theorem 6.13
Suppose $\mathcal{C}_\rho$ is a reduction of labeled transition system $\mathcal{S}$ and $\mathcal{R}$ is a symmetrical set of relations.
i if $\mathcal{C}_\rho$ is sound with respect to $\mathcal{R}$ then $\mathcal{C}_\rho$ is a sound reduction of $\mathcal{S}$ with respect to $\mathcal{R}$.
ii if $\mathcal{C}_\rho$ is complete with respect to $\mathcal{R}$ then $\mathcal{C}_\rho$ is a complete reduction of $\mathcal{S}$ with respect to $\mathcal{R}$.

Proof
i Suppose $\mathcal{C}_\rho$ is sound with respect to $\mathcal{R}$, $\mathcal{S}$ is symmetrical with respect to system permutation $\pi\sigma$, $\mathcal{S}'$ is a labeled transition system which is symmetrical with respect to system permutation $\pi'\sigma'$ and $\mathcal{C}'_{\rho'}$ is a symmetrical reduction of $\mathcal{S}'$ for which $\mathcal{C}_\rho\mathcal{R}\mathcal{C}'_{\rho'}$. For notational convenience, write $\mathcal{C}^i_\rho = \sigma^i\mathcal{C}_\rho$, $\mathcal{C}'^i_{\rho'} = \sigma'^i\mathcal{C}'_{\rho'}$ and $R^i = R_{\sigma^i\sigma'^i}$. Since $\mathcal{R}$ is symmetrical, it holds that $\mathcal{C}^i_\rho R^i \mathcal{C}'^i_{\rho'}$ for all $\leqslant i < n$. Since $\mathcal{C}_\rho$ is sound, it follows that $(\mathcal{C}^0 + \cdots + \mathcal{C}^{n-1})(R^0 \cup \cdots \cup R^{n-1})(\mathcal{C}'^0 + \cdots + \mathcal{C}'^{n-1})$ and $R^0 \cup \cdots \cup R^{n-1} \in \mathcal{R}$. From the definition of symmetrical reduction it follows that $\mathcal{C}^0 + \cdots + \mathcal{C}^{n-1} = \mathcal{S}$ and $\mathcal{C}'^0 + \cdots + \mathcal{C}'^{n-1} = \mathcal{S}'$. Hence, $\mathcal{S}\mathcal{R}\mathcal{S}'$.
ii Analogous.

□

The rest of this section proves compositionality for the three equivalences studied in this report.

Lemma 6.14

i Symmetrical reductions are sound with respect to isomorphy.
ii Symmetrical reductions are sound with respect to strong bisimulation.
iii Symmetrical reductions are sound with respect to weak bisimulation.

Proof

i Suppose $\mathcal{C}_\rho$ and $\mathcal{C}'_{\rho'}$ are symmetrical reductions, σ, σ' are system permutations and $R_1, \cdots, R_n$ are isomorphisms with $(\sigma^i\mathcal{C})R_i(\sigma'^i\mathcal{C}')$, for all $0 \leqslant i < n$. For notational convenience, write $\mathcal{C}_i = \sigma^i\mathcal{C}$ and $\mathcal{C}'_i = \sigma'^i\mathcal{C}'$. It will be proven that $R = R_1 \cup \cdots \cup R_n$ is an isomorphism for $\mathcal{C}_0 + \cdots + \mathcal{C}_{n-1}$ and $\mathcal{C}'_0 + \cdots + \mathcal{C}'_{n-1}$.
First, it will be proven that R is bijective; it will be proven that R is functional, surjective and injective. For functionality, suppose sRt and sRu for states s, t, u. That is:
(a) $s \in \mathcal{C}_i$, $t \in \mathcal{C}'_i$ and sR_it for some $0 \leqslant i < n$.
(b) $s \in \mathcal{C}_j$, $u \in \mathcal{C}'_j$ and sR_ju for some $0 \leqslant j < n$.
Without loss of generality, assume $i \leqslant j$. Now, the following hold:
(a) $s = \sigma^{j-i}s'$, for some $s' \in \mathcal{C}_i$.
(b) $u = \sigma'^{j-i}u'$ for some $u' \in \mathcal{C}'_i$.
From sR_ju it follows that $s'R_iu'$. Since $s, s' \in \mathcal{C}_i$ and $s = \sigma^{j-i}s'$ it follows that $s = \rho(j-i)s'$. From $(\mathcal{C}_\rho)_jR_j(\mathcal{C}'_{\rho'})_j$ and $s'R_iu'$ it follows that $u = \rho(j-i)u'$ and sR_iu. By functionality of R_i, from sR_iu and sR_it it follows that $t = u$.
The proof of injectivity is analogous. Surjectivity follows directly from the definition of symmetrical reduction.
Second, it will be proven that R is an isomorphism between $\mathcal{C} \cup \cdots \cup \sigma^{n-1}\mathcal{C}$ and $\mathcal{C}' \cup \cdots \cup \sigma^{n-1}\mathcal{C}'$. For convenience, the notation $R(s)$ will be used for the unique t which satisfies sRt. Suppose $s \xrightarrow{a} t$ for $s, t \in \mathcal{C} \cup \cdots \cup \sigma^{n-1}\mathcal{C}$. That is, $s, t \in \sigma^i\mathcal{C}$ and $s \xrightarrow{a}_i t$ for some $0 \leqslant i < n$. From isomorphy of R_i it follows that $R_i(s) \xrightarrow{a}_i R_i(t)$, hence, $R(s) \xrightarrow{a} R(t)$.

ii Suppose $s \xrightarrow{a} t$ for $s, t \in \mathcal{C} \cup \cdots \cup \sigma^{n-1}\mathcal{C}$ and sRu. That is, $s, t \in \sigma^i\mathcal{C}$, $s \xrightarrow{a}_i t$ and sR_ju for some $0 \leqslant i, j < n$; without loss of generality, assume $i < j$. By symmetry, from sR_ju it follows that $s'R_iu'$, for $s', u' \in \sigma^i\mathcal{C}$ with $s = \sigma^{j-i}s'$ and $u = \sigma'^{j-i}u'$. From the definition of symmetrical reduction, $s = \rho(j-i)s'$ and $u = \rho'(j-i)u'$. So, since $s'R_iu'$, $s = \rho(j-i)s'$ and $\sigma^i\mathcal{C}_\rho R_i\sigma'^i\mathcal{C}'_{\rho'}$ it holds that sR_iu. Since R_i is a bisimulation, from sR_iu and $s \xrightarrow{a}_i t$ it follows that $u \xrightarrow{a}_i v$ and tR_iv for some v. The other half of the proof is analogous.

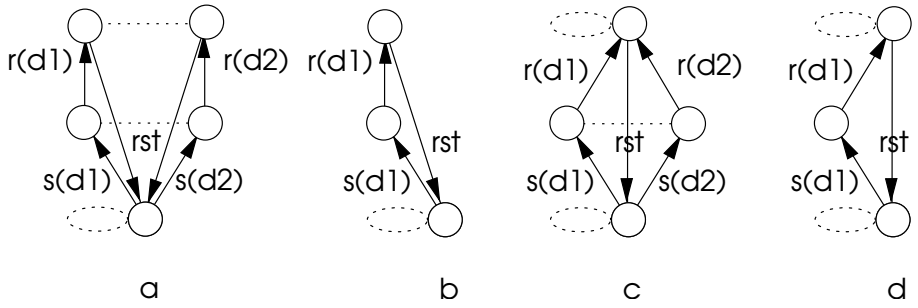
iii Similar.

□

Corollary 6.15
Symmetrical reduction is sound with respect to isomorphism, strong bisimulation and weak bisimulation.

The property of completeness appears to be more complex than soundness. Some intuition is provided by the following example.

Example 6
The simple communication protocol from the previous example is extended with a 'reset' action which returns the system to its initial state after each send and receive. The label permutation leaves the reset label intact, i.e. $\pi = \{\langle r(d1), r(d2)\rangle, \langle s(d1), s(d2)\rangle\}$. The transition system is (a) and a symmetrical reduction is (b) below.



The transition system is not minimal with respect to strong bisimulation, in the sense that the two states from which a reset is possible are bisimilar. The minimalisation is shown in (c) with a reduction in (d). So, although (a) and (c) are equivalent, the symmetrical reductions (c) and (d) are not. The cause of the problem is that there are states which are both permutation isomorphic and bisimilar.

The example suggests that the construction of complete reductions is hampered by a twisted interaction between symmetry and equivalence. The motivation for developing symmetrical reductions includes the intuition that operations like bisimulation reduction are cheaper after symmetrical reduction, followed by symmetrical expansion. While soundness guarantees that bisimulation reduction thus obtained are indeed bisimilar, lack of completeness leaves open the possibility that there are cases in which the maximal reduction is out of reach. This incompleteness can be addressed trivially by including bisimulation equivalence in the definition of reduction; the cost of this naive approach cannot outweigh the benefits of symmetrical reduction. So, complete reductions will remain a subject of further study.

6.4 Encoding symmetrical reductions

What has received no attention yet, is how to effectively check equivalence for symmetrical reductions. The remainder of this section aims at translating symmetrical reductions to labeled transition systems such that equivalent symmetrical reductions translate to equivalent labeled transition systems.

Definition 6.16 (encoding)

Suppose $\mathcal{C}_\rho$ is a symmetrical reduction with $\mathcal{C} = \langle C, \mathcal{L}, \rightarrow, s_0 \rangle$ and $\mathcal{L} \cap \mathbb{N} = \emptyset$.

i The *encoding of ρ with respect to $\mathcal{C}$*, denoted by $\mathrm{enc}(\mathcal{C}, \rho)$, is defined by $\mathrm{enc}(\mathcal{C}_\rho) = \langle C, \mathbb{N}, \rightarrow', s_0 \rangle$, where the transition relation $\rightarrow'$ is defined by $s_1 \overset{a}{\rightarrow'} s_2$ iff one of the following holds:
(a) $a = \tau$ and $s_1 \overset{a}{\rightarrow} s_2$
(b) $a \in \mathbb{N}$ and $\rho(a)(s_1) = s_2$

ii The *encoding of $\mathcal{C}_\rho$*, denoted by $\mathrm{enc}(\mathcal{C}_\rho)$, is defined by $\mathrm{enc}(\mathcal{C}_\rho) = \mathrm{enc}(\mathcal{C}, \rho) + \mathcal{C}$.

It will be proven that an encoding indeed represents a ρ function, if the relevant equivalence relation satisfies two requirements. The first requirement is that equivalent systems can be partitioned into two subsystems which have no labels in common.

Definition 6.17 (compositionality with respect to labels)

An equivalence relation R is *compositional with respect to labels A* iff for all systems $\mathcal{S}_1 = \langle S, \mathcal{L}_1, \rightarrow_1, s_0 \rangle, \mathcal{S}_2' = \langle S, \mathcal{L}_2, \rightarrow_2, s_0 \rangle$ $\mathcal{S}_1' = \langle S', \mathcal{L}_1, \rightarrow_1', s_0' \rangle, \mathcal{S}_2' = \langle S', \mathcal{L}_2, \rightarrow_2', s_0' \rangle$ with $A = \mathcal{L}_1 \cap \mathcal{L}_2$ and for all $a \in A$ it holds that $s \overset{a}{\rightarrow}_1 t \Leftrightarrow s \overset{a}{\rightarrow}_2 t$ and $s \overset{a}{\rightarrow}_1' t \Leftrightarrow s \overset{a}{\rightarrow}_2' t$ it holds that $(\mathcal{S}_1 + \mathcal{S}_2) R (\mathcal{S}_1' + \mathcal{S}_2') \Leftrightarrow \mathcal{S}_1 R \mathcal{S}_1'$ and $\mathcal{S}_2 R \mathcal{S}_2'$. If $A = \emptyset$ then R is said to be *compositional with respect to labels*.

The second requirement is that the encodings of equivalent systems are equivalent by themselves.

Definition 6.18

An equivalence relation R *respects encodings* iff for all symmetrical reductions $C_\rho, C'_{\rho'}$ it holds that $C_\rho R C'_{\rho'} \Rightarrow \mathrm{enc}(C, \rho) R \mathrm{enc}(C', \rho')$.

Theorem 6.19
Suppose $\mathcal{C}_\rho, \mathcal{C}'_{\rho'}$ are symmetrical reductions, $\mathcal{R}$ respects encodings and is compositional with respect to labels, then $\mathcal{C}_\rho \mathcal{R} \mathcal{C}'_{\rho'} \Leftrightarrow \text{enc}(\mathcal{C}_\rho) \mathcal{R} \text{enc}(\mathcal{C}'_{\rho'})$

Proof
Suppose $\mathcal{C}_\rho \mathcal{R} \mathcal{C}'_{\rho'}$. Since R respects encodings, it follows that $\text{enc}(\mathcal{C}, \rho) R \text{enc}(\mathcal{C}', \rho')$. Compositionality completes the proof. □

It remains to be proven that the three equivalences addressed in this report are compositional and respect encodings.

Lemma 6.20
- **i** Isomorphism is compositional with respect to labels.
- **ii** Strong bisimulation is compositional with respect to labels.
- **iii** Weak bisimulation is compositional with respect to labels other than τ.

Proof
Let $\mathcal{S}_1 = \langle S, \mathcal{L}_1, \rightarrow_1, s_0 \rangle, \mathcal{S}'_2 = \langle S, \mathcal{L}_2, \rightarrow_2, s_0 \rangle$, $\mathcal{S}'_1 = \langle S', \mathcal{L}_1, \rightarrow'_1, s'_0 \rangle, \mathcal{S}'_2 = \langle S', \mathcal{L}_2, \rightarrow'_2, s'_0 \rangle$ with $\mathcal{L}_1 \cap \mathcal{L}_2 = \emptyset$ be arbitrarily given, and let $\rightarrow_{12}, \rightarrow'_{12}$ denote the transition relation of $\mathcal{S}_1 + \mathcal{S}_2, \mathcal{S}'_1 + \mathcal{S}'_2$, respectively

- **i** "⇒" Suppose $(\mathcal{S}_1 + \mathcal{S}_2) R (\mathcal{S}'_1 + \mathcal{S}'_2)$ for some isomorphism R. Now suppose $s_1 \xrightarrow{a}_1 t_1$, $s_1 R s'_1$, $t_1 R t'_1$ for states s_1, t_1, s'_1, t'_1. It follows that $s_1 \xrightarrow{a}_{12} t_1$ and $a \in \mathcal{L}_1$. Since R is an isomorphism, it follows that $s'_1 \xrightarrow{a}_{12} t'_1$. Since $a \in \mathcal{L}_1$, also $s'_1 \xrightarrow{a}_1 t'_1$. The other half of the proof is analogous.
  "⇐" Suppose $\mathcal{S}_1 R \mathcal{S}'_1$ and $\mathcal{S}_2 R \mathcal{S}'_2$. Suppose $s_1 \xrightarrow{a}_{12} t_1$, $s_1 R s'_1$, $t_1 R t'_1$ for states s_1, t_1. Without loss of generality, assume $s_1 \xrightarrow{a}_1 t_1$ and since R is an isomorphism it follows that $s'_1 \xrightarrow{a}_1 t'_1$. Hence, $s'_1 \xrightarrow{a}_{12} t'_1$.
- **ii** "⇒" Suppose $(\mathcal{S}_1 + \mathcal{S}_2) R (\mathcal{S}'_1 + \mathcal{S}'_2)$ for some strong bisimulation R. Now suppose $s_1 \xrightarrow{a}_1 t_1$ and $s_1 R s'_1$ for states s_1, s'_1, t_1. Then $s_1 \xrightarrow{a}_{12} t_1$ and $a \in \mathcal{L}_1$. From R being a strong bisimulation it follows that $s'_1 \xrightarrow{a}_{12} t'_1$ and $t_1 R t'_1$ for some state t'_1. Since $a \in \mathcal{L}_1$ it follows that $s'_1 \xrightarrow{a}_1 t'_1$. The other half, as well as the proof of $\mathcal{S}_2 R \mathcal{S}'_2$, is analogous.
  "⇐" Suppose $\mathcal{S}_1 R \mathcal{S}'_1$ and $\mathcal{S}_2 R \mathcal{S}'_2$. Suppose $s_1 \xrightarrow{a}_{12} t_1$, $s_1 R s'_1$ for states s_1, s'_1, t_1. Without loss of generality assume $s_1 \xrightarrow{a}_1 t_1$ and from R being a strong bisimulation it follows that $s'_1 \xrightarrow{a}_1 t'_1$. Hence, $s'_1 \xrightarrow{a}_{12} t'_1$. The other half of the proof is analogous.
- **iii** "⇒" Suppose $(\mathcal{S}_1 + \mathcal{S}_2) R (\mathcal{S}'_1 + \mathcal{S}'_2)$ for some weak bisimulation R. Now suppose $s_1 \xrightarrow{a}_1 t_1$ and $s_1 R s'_1$ for states s_1, s'_1, t_1. Then $s_1 \xrightarrow{a}_{12} t_1$ and $a \in \mathcal{L}_1$. From R being a weak bisimulation it follows that $t_1 R t'_1$ and $s'_1 \xrightarrow{\tau}_{12}{}^* \cdot \xrightarrow{a}_{12} \cdot \xrightarrow{\tau}_{12}{}^* t'_1$ for a state t'_1 or $a = \tau$ and $t_1 R t'_1$. Since $a \in \mathcal{L}_1$ and $\tau \in \mathcal{L}_2$ it holds that $s'_1 \xrightarrow{\tau}_1{}^* \cdot \xrightarrow{a}_1 \cdot \xrightarrow{\tau}_1{}^* t'_1$. The other half of the proof is analogous.
  "⇐" Suppose $\mathcal{S}_1 R \mathcal{S}'_1$ and $\mathcal{S}_2 R \mathcal{S}'_2$. Suppose $s_1 \xrightarrow{a}_{12} t_1$, $s_1 R s'_1$ for states s_1, s'_1, t_1. Without loss of generality assume $s_1 \xrightarrow{a}_1 t_1$ and from R being a weak bisimulation it follows that $s'_1 \xrightarrow{\tau}_1{}^* \cdot \xrightarrow{a}_1 \cdot \xrightarrow{\tau}_1{}^* t'_1$. Hence, $s'_1 \xrightarrow{\tau}_{12}{}^* \cdot \xrightarrow{a}_{12} \cdot \xrightarrow{\tau}_{12}{}^* t'_1$. The other half of the proof is analogous.

□

Lemma 6.21
- **i** Isomorphism respects encodings.
- **ii** Strong bisimulation respects encodings.
- **iii** Weak bisimulation respects encodings.

Proof

Let symmetrical reductions $C_\rho, C'_{\rho'}$ be arbitrarily given.

i Suppose $C_\rho R C'_{\rho'}$. Now suppose $s \xrightarrow{i} t$ ($i \in \mathbb{N}$), sRs' and tRt'. By definition of encoding, $\rho(i)(s) = t$ and by $C_\rho R C'_{\rho'}$ also $\rho(i)(s') = t'$. The other half of the proof is analogous.

ii Suppose $C_\rho R C'_{\rho'}$. Now suppose $s \xrightarrow{i} t$ ($i \in \mathbb{N}$) and sRs'. By definition of encoding, $\rho(i)(s) = t$ and by $C_\rho R C'_{\rho'}$ also $\rho(i)(s') = t'$ and tRt'.

iii Suppose $C_\rho R C'_{\rho'}$. The encoding contains two types of transitions. For those labeled with natural numbers, suppose $s \xrightarrow{i} t$ ($i \in \mathbb{N}$) and sRs'. By definition of encoding, $\rho(i)(s) = t$ and by $C_\rho R C'_{\rho'}$ also $\rho(i)(s') = t'$ and tRt'. For those labeled with τ labels, suppose $s \xrightarrow{\tau} t$ and sRs'. From the definition of weak bisimulation, tRt' for some t'.

□

7. Symmetry in specifications

The notion of symmetry which has been explored so far applies to concrete labeled transition systems, which is a severe drawback when it comes to applications. In practice, labeled transitions typically contain at least millions of states, the generation of which takes a decent amount of processor time. Even worse, with existing algorithms for graph isomorphism checking being NP, the existence of efficient symmetry checkers is at least unlikely. However, the theory developed in this report is not meant to be directly applied at concrete labeled transition systems.

Labeled transition systems are usually generated from specifications of a high level of abstraction. At this level, there often exists a strong intuition that symmetry has a conceptual basis. For example, data flowing through a communication protocol, in no way influencing the course of the system, typically gives rise to symmetry. Also, systems consisting of a number of parallel processes, which are identical except for their process identifier, are often symmetrical. It is expected that this intuition can be recognised in specifications of systems.

The specification language used to support the conceptual base for symmetry in this report is μCRL [BFG$^+$01], a language based on process algebra with data, with connectives for sequential composition '$\cdot$', non-deterministic choice '+', parallel composition '$\|$', alternative quantification over data and handshaking as native communication model. A simple example μCRL fragment is given by the following.

```
proc  Producer = sum(d:D, produce(d).give(d)).Producer
      Consumer = sum(d:D, take(d)).delta
```

The example shows a producer and a consumer process. The former repeatedly produces one element from a data sort D and gives it, while the latter takes an arbitrary data element once. It is specified elsewhere that the producer's give corresponds with the consumer's take action, the resulting communication action is called 'consume'. The producer and the consumer are executed in parallel with the following construct, under the proviso that the actions of give and take cannot be executed in isolation.

```
init encap({give,take},  Producer||Consumer)
```

It is intuitively clear that the example system is symmetrical in the data handled. However, with no explicit states or transitions, it is unclear how this symmetry could be verified. Moreover, simple as it may seem, the example shows a number of non-trivial constructs like parallel composition and communication, which have not been addressed in the context of symmetry yet.

The problem has been acknowledged before, in that it is is a well-known problem that expressiveness of specifications is at odds with conciseness of automated tools. The solution chosen in μCRL context is the *linear process equation* format, introduced in [BG94], which plays the role of machine code in the μCRL tool set [BFG$^+$01]. This linear format specifies a system in terms of a finite number of conditional transitions of the form $\sum_{e \in E} a(f(x,e)) \cdot P(g(x,e)) \lhd c(x,e) \rhd \delta$, which means that

if condition $c(x, e)$ holds in state x for some datum e of sort E, then an action a with parameters depending on state s and datum e can be executed after which the system moves to another state which depends on x and e. An automated *lineariser* tool for the generation of linear process equations from μCRL specifications is included in the tool set [Use02].

The strong point of this linear format is that it lends itself to automated analysis, since the format is simple and lacking complicated constructs like parallel composition and communication. Its weakness is that it is less suited for human understanding. So, the linear process equations are presented 'as is' with no attempt to provide intuition on what this or that state variable means. The linear process equation generated for this section's examples, with some minor editing to improve legibility, is given below.

```
proc X(s:State,d:D) =

sum(d0:D,produce(d0).
 X(2,d0)
 <|s=3|>delta)

consume(d).
 X(1,d1)
 <|s=2|>delta+

init X(3,d1)
```

States are represented by a $\langle s, d\rangle$ pair, where s is a local state which determines whether the producer or the consumer is active, and d is the data element being handled. As it is the data which generates the symmetry, it makes sense to fix some permutation $\delta : D \to D$. From this data permutation a label permutation π is defined by $\pi produce(d) = produce(\delta d)$ and $\pi consume(d) = consume(\delta d)$. A candidate state permutation is σ defined by $\sigma\langle s, d\rangle = \langle s, \delta d\rangle$.

Verifying that $\pi\sigma$ indeed defines a system automorphism proceeds as follows. Consider an arbitrary state $\langle s, d\rangle$. Now, consider the first summand and assume that for some arbitrary data element d_0 the condition $s = 3$ holds in this state, which implies that action $consume(d_0)$ can be executed after which the system moves to state $\langle 2, d_0\rangle$. This implies that in the permuted state $\langle s, \delta d\rangle$ there is a data element δd_0 for which condition $s = 3$ holds, after which action $consume(\delta d_0)$ moves the system to $\langle 2, \delta d_0\rangle = \sigma\langle 2, d_0\rangle$. The second summand is to be verified in the same way, but here is a complication.

The second summand leads to a successor state $\langle 1, d_1\rangle$ where d_1 is a constant data element. At first sight, this assignment of a 'special' data element breaks the symmetry, in that it moves the system to a state which is not the permuted destination state which corresponds to the permuted source state. A closer exmination reveals that the constant data element d_1 plays the role of a dummy placeholder, which is not used in states where $s = 2$. So, instead of assigning the dummy d_1 the original value of the d parameter can be maintained, leaving intact the symmetry. This process of removing dummy assignments to state variables was independently discovered by Jaco van de Pol as an optimisation for linear process equations. As *dummy elimination* it is mentioned in [BGL+03]. So, the second summand can be replaced by an equivalent summand.

```
consume(d).
 X(1,d)
 <|s=2|>delta+
```

What remains to be verified is that the state permutation does not change the initial state $\langle 3, d_1\rangle$. Here, again the data element d_1 is used as a dummy element which breaks the symmetry. A solution is to extend the data sort D with an explicit dummy element nil, with the drawback that this element is to be excluded from regular communication. The resulting linear process equation follows below.

```
proc X(s:State,d:D) =

sum(d0:D,produce(d0).
 X(2,d0)
 <|s=3 and not(d0=nil)|>delta)

consume(d).
 X(1,d1)
 <|s=2|>delta+

init X(3,nil)
```

Summarising, the linear format of μCRL makes it possible to check symmetry in a relatively easy way, which is likely to be amenable to an automated theorem proving approach. However, the lineariser introduces some asymmetrical artefacts, which might obscure symmetry in practice. Further study of optimisations in the style of dummy elimination is a necessary provision here.

8. Cases

This section studies three cases. First, the *Positive Acknowledgement with Retransmission (PAR)* protocol [Def81], also dubbed *alternating bit protocol*, which was used as an example in Section 3 is studied (Section 8.1). Second, the generalised Peterson's mutual exclusion algorithm [Pet81] is addressed Section 8.2 and, third and final, the Dining philosophers [Dij71] case is studied (Section 8.3). The section ends with a discussion of the statistics for the three cases (Section 8.4).

8.1 The alternating bit protocol

The alternating bit protocol has a sender transmit a datum to a receiver, and repeats this until this receiver acknowledges the receipt of this datum. In order to be prevent problems with lost data and acknowledgements, both are accompanied by a sequence number which is checked on reception: data and acknowledgements which are out of sequence are ignored. Because of this scheme it suffices to have two sequence numbers, 0 and 1, which explains the name of the protocol.

The corresponding linear process equation is given below. In this equation, sort D defines the data elements that can be sent. It is this D which is the key to symmetry, as permutations of the elements of this data sort lead to isomorph labeled transition systems.

$X(s_0 : State, d : D, s_1 : State, n : bit, d_{12} : D, s_7 : State, n_2 : bit, s_8 : State, d_{11} : D) =$

1 $\sum_{b:Bool} \sum_{d':D} r_1(d') \cdot X(case(b, 5, 2), d', s_1, n, d_{12}, s_7, n_2, s_8, d_{11})$
$\quad \lhd d \neq d_0 \wedge case(b, s_0 = 6, s_0 = 3) \rhd \delta+$

2 $\sum_{e:Enum_6} c_3(d_{12}, n) \cdot X(s_0, d, 4, 0, d_{12}, s_7, n_2, case(e, 6, 8, 6, 8, 3, 2), d_{12})$
$\quad \lhd n = case(e, 0, 1, 0, 1, 0, 1) \wedge s_1 = 2 \wedge case(e, s_8 = 9, s_8 = 9, s_8 = 7, s_8 = 7, s_8 = 4, s_8 = 4) \rhd \delta+$

3 $\sum_{b:Bool} c_3(error) \cdot X(s_0, d, 4, 0, d_{12}, s_7, n_2, case(b, 3, 8), d_{11})$
$\quad \lhd s_1 = 1 \wedge case(b, s_8 = 4, s_8 = 9 \vee s_8 = 8) \rhd \delta+$

4 $\tau \cdot X(s_0, d, s_1, n, d_{12}, 2, n_2, s_8, d_{11})$
$\quad \lhd s_7 = 3 \rhd \delta+$

5 $\tau \cdot X(s_0, d, s_1, n, d_{12}, 1, 0, s_8, d_{11})$
$\quad \lhd s_7 = 3 \rhd \delta+$

6 $\sum_{b:Bool} s_4(d_{11}) \cdot X(s_0, d, s_1, n, d_{12}, s_7, n_2, case(b, 5, 1), d_{11})$
$\quad \lhd case(b, s_8 = 6, s_8 = 2) \rhd \delta+$

7 $\sum_{e:Enum_4} c_5(case(e, 1, 1, 0, 0)) \cdot X(s_0, d, s_1, n, d_{12}, 3, case(e, 1, 1, 0, 0), case(e, 7, 9, 4, 4), d_{11})$
$\quad \lhd s_7 = 4 \wedge case(e, s_8 = 8, s_8 = 1, s_8 = 3, s_8 = 5) \rhd \delta+$

8 $\tau \cdot X(s_0, d, 1, 0, d_{12}, s_7, n_2, s_8, d_{11})$
$\quad \lhd s_1 = 3 \rhd \delta+$

$$
\begin{array}{ll}
9 & \tau \cdot X(s_0, d, 2, n, d_{12}, s_7, n_2, s_8, d_{11}) \\
& \quad \lhd s_1 = 3 \rhd \delta + \\
10 & \sum_{b:Bool} c_6(error) \cdot X(case(b, 5, 2), d, s_1, n, d_{12}, 4, 0, s_8, d_{11}) \\
& \quad \lhd case(b, s_0 = 4, s_0 = 1) \wedge s_7 = 1 \rhd \delta + \\
11 & \sum_{e:Enum_4} c_6(case(e, 0, 1, 0, 1)) \cdot X(case(e, 3, 5, 2, 6), d, s_1, n, d_{12}, 4, 0, s_8, d_{11}) \\
& \quad \lhd case(e, 0, 1, 0, 1) = n_2 \wedge case(e, s_0 = 4, s_0 = 4, s_0 = 1, s_0 = 1, s_7 = 2) \rhd \delta + \\
12 & \sum_{b:Bool} c_2(d, case(b, 0, 1)) \cdot X(case(b, 4, 1), d, 3, case(b, 0, 1), d, s_7, n_2, s_8, d_{11}) \\
& \quad \lhd case(b, s_0 = 5, s_0 = 2) \wedge s_1 = 4 \rhd \delta \\
& initX(6, d_0, 4, 0, d_0, 4, 0, 9, d_0)
\end{array}
$$

In all fairness it has to be admitted that the above linear process equation needed some manual tweaking in order to make it a candidate for symmetry analysis. The reason for this is that the state variables of sort D represent the contents of some of the channels used for data and acknowledgements, and which are initially empty. The μCRL lineariser initialises these with some arbitrary data element and only a combination of other state variables guarantees that in some cases the specific data element is interpreted as a default dummy element. As a result, the linear process equation generated does not look symmetrical at all, since one data element seems to have a special meaning. To repair this, an explicit dummy datum d_0 was introduced, which is only used for initialisation and not for regular transmission.

Lemma 8.1
Suppose δ is a permutation of data sort D with $\delta d_0 = d_0$, π is a mapping on labels defined by $\pi\tau = \tau, \pi r_1(d) = r_1(\delta d), c_2(d, n) = c_2(\delta d, n), c_3(d, n) = c_3(\delta d, n), \pi c_3(error) = c_3(error), \pi s_4(d) = s_4(\delta d), \pi c_5(n) = c_5(n), \pi c_6(n) = c_6(n)$ and σ is a mapping on states defined by $\sigma\langle s_0, d, s_1, n, d_{12}, s_7, n_2, s_8, d_{11}\rangle = \langle s_0, \pi(d), s_1, n, \pi(d_{12}), s_7, n_2, s_8, \pi(d_{11})\rangle$.

Then $\sigma\pi$ is a permutation isomorphism for X.

Proof
From the fact that δ is a permutation of data sort D it follows that π is a label permutation and σ is a permutation of states. It can be easily verified that σ leaves intact the initial state. Now, it remains to be proven that $\sigma\pi$ is a permutation isomorphism, i.e. for each transition $\langle s, a, t\rangle$ there exists a transition $\langle \sigma s, \pi a, \sigma t\rangle$, so consider an arbitrary transition $\langle s, a, t\rangle$. There are 12 cases to be considered, corresponding with the 12 summands of the above linear process equation. Here only the first and most interesting case will be addressed.

$$
\begin{array}{ll}
1 & \sum_{b:Bool} \sum_{d':D} r_1(d') \cdot X(case(b, 5, 2), d', s_1, n, d_{12}, s_7, n_2, s_8, d_{11}) \\
& \quad \lhd d \neq d_0 \wedge case(b, s_0 = 6, s_0 = 3) \rhd \delta +
\end{array}
$$

So, s is of the form $s = \langle s_0, d, s_1, n, d_{12}, s_7, n_2, s_8, d_{11}\rangle$, the condition $d \neq d_0 \wedge case(b, s_0 = 6, s_0 = 3)$ is satisfied, $a = r_1(d')$ for some datum d' and $t = \langle case(b, 5, 2), d', s_1, n, d_{12}, s_7, n_2, s_8, d_{11}\rangle$. Now, considering $\sigma s = \langle s_0, \pi(d), s_1, n, \pi(d_{12}), s_7, n_2, s_8, \pi(d_{11})\rangle$ it follows that condition $\delta d' \neq d_0 \wedge case(b, s_0 = 6, s_0 = 3)$ is satisfied, and allows action $r_1(\delta d') = \pi a$ to be executed leading, to successor state $\langle case(b, 5, 2), \delta d', s_1, n, \delta d_{12}, s_7, n_2, s_8, \delta d_{11}\rangle = \sigma t$. □

It is instructive to see that, departing from the assumption that permutations of data are the cause of symmetry, the definition of a candidate permutation isomorphism is highly intuitive and that the verification of the candidate is simple, in the sense that it is more a matter of pattern matching than of theorem proving. The next step is the definition of a symmetric reduction.

Lemma 8.2
Let C be defined by the linear process equation for X with the one difference that the first summand is defined as follows:

1 $\sum_{b:Bool} r_1(d) \cdot C(case(b,5,2),d,s_1,n,d_{12},s_7,n_2,s_8,d_{11})$
$\lhd case(b, s_0 = 6, s_0 = 3) \rhd \delta+$

Let ρ be defined by $\rho(1)(\langle s_0,d,s_1,n,d_{12},s_7,n_2,s_8,d_{11}\rangle) = \langle s_0,d,s_1,n,d_{12},s_7,n_2,s_8,d_{11}\rangle$ iff $d = d_{12} = d_{11} = d_0$.

Then C_ρ is a symmetrical reduction of X.

Proof
First it needs to be proven that all transitions in X are in the orbit of exactly one transition from C; this is by an easy induction on the distance of a transition from the initial state. Second, it needs to be proven that for all $s \in C, 1 \leqslant i < n$ with $\sigma^i(s) \in C$ it holds that $\rho(i)(s) = \sigma^i(s)$. By an easy induction, it can be shown that C contains no other data than d and dummy element d_0, so for any $s \in C, 1 \leqslant i < n$ with $\sigma^i(s) \in C$ it holds that s contains only the dummy d as data, so $\sigma^i(s) = s$, hence $\rho(i)(s) = \sigma^i(s)$. □

This case shows a number of features that make symmetry reduction feasible in practice. First, symmetry is firmly rooted in the intuition that the alternating bit protocol does not depend on the data that flows through it. That is, the system is insensitive to permutations of the data set. Second, given any data permutation, the definition of a candidate label permutation and a candidate permutation isomorphism is an easy task, and the actual verification of these candidates is simple enough to be subjected to mechanical analysis. Third, the definition of a candidate symmetrical reduction is intuitive, although its verification requires some level of human wit.

8.2 Peterson's mutual exclusion algorithm
The algorithm for mutual exclusion by Peterson [Pet81] was initially formulated for two parallel processes claiming exclusive access to a *critical region* in their code, and later generalised for arbitrary numbers of processes. The generic code for a process, parametrised by its process identifier, is shown below.

```
procedure P(i):
  for j := 1 to n-1 do
     Q[i] := j;
     Turn[j] := i;
     wait until (for all k<>i: Q(k)<j) or Turn[j]<>i;
  done
  Critical Section;
  Q[i]:=0;
end procedure
```

The underlying principle is to have all processes enter a queue where each slot is occupied by zero or more processes. A process in a slot may only advance one slot upstream in the queue if all other processes are further downstream or some other process entered the slot later. The position of a process i in the queue is stored in element i of shared array Q, and the process id of the process which entered slot j last is stored in element j of shared array $Turn$. The corresponding linear process equation is given below.

$X(s\colon State, pid\colon \mathbb{N}, x_0\colon \mathbb{N}, x_1\colon \mathbb{N}, x_2\colon \mathbb{N}, x_3\colon \mathbb{N}, x_4\colon \mathbb{N}, x_5\colon \mathbb{N}, x_6\colon \mathbb{N}, x_7\colon \mathbb{N}, x_8\colon \mathbb{N}, x_9\colon \mathbb{N}, x_{10}\colon \mathbb{N},$
$s'\colon State, pid'\colon \mathbb{N}, x'_0\colon \mathbb{N}, x'_1\colon \mathbb{N}, x'_2\colon \mathbb{N}, x'_3\colon \mathbb{N}, x'_4\colon \mathbb{N}, x'_5\colon \mathbb{N}, x'_6\colon \mathbb{N}, x'_7\colon \mathbb{N}, x'_8\colon \mathbb{N}, x'_9\colon \mathbb{N}, x'_{10}\colon \mathbb{N},$
$turns\colon NatArray, flags\colon NatArray) =$
1 $\sum_{e:Enum_2} setflag(case(e,pid,x_{10}), case(e,1,0)) \cdot$
$X(case(e,6,7), pid, case(e,1,0), case(e,pid,0), case(e,pid,0), case(e,pid,0),$

$case(e,1,0),0,0,case(e,2,0),case(e,pid,0),case(e,pid,0),case(e,pid,0),$
$s',pid',x'_0,x'_1,x'_2,x'_3,x'_4,x'_5,x'_6,x'_7,x'_8,x'_9,x'_{10},$
$turns,set(flags,case(e,pid,x_{10}),case(e,1,0)))$
$\lhd case(e,s=7,s=1)\rhd\delta+$

2 $\tau\cdot$
$X(s,pid,x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10},$
$3,pid',0,0,0,0,0,0,0,0,x'_8,x'_9,x'_{10}),$
$turns,flags)$
$\lhd s'=4\wedge x'_7=2\rhd\delta+$

3 $wait(x_3,x_4)\cdot$
$X(4,pid,0,0,0,0,0,0,x_5,x_7,x_8,x_9,x_{10},$
$s',pid',x'_0,x'_1,x'_2,x'_3,x'_4,x'_5,x'_6,x'_7,x'_8,x'_9,x'_{10},$
$turns,flags)$
$\lhd s=5\wedge(x_3=2\vee flags[2]<x_4)\wedge(x_3=1\vee flags[1]<x_4),turns[x_4]\neq x_3))\rhd\delta+$

4 $setturn(x_0,x_1)\cdot$
$X(5,pid,0,0,0,x_3,x_4,x_2,0,x_7,x_8,x_9,x_{10},$
$s',pid',x'_0,x'_1,x'_2,x'_3,x'_4,x'_5,x'_6,x'_7,x'_8,x'_9,x'_{10},$
$set(turns,x_0,x_1),flags)$
$\lhd s=6\rhd\delta+$

5 $criticalin(x'_8)\cdot$
$X(s,pid,x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10},$
$2,pid',0,0,0,0,0,0,0,0,0,x'_9,x'_{10},$
$turns,flags)$
$\lhd s'=3\rhd\delta+$

6 $criticalout(x'_9)\cdot$
$X(s,pid,x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10},$
$1,pid',0,0,0,0,0,0,0,0,0,0,x'_{10},$
$turns,flags)$
$\lhd s'=2\rhd\delta+$

7 $\sum_{e:Enum_2} setflag(case(e,pid',x'_{10}),case(e,1,0))\cdot$
$X(s,pid,x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10},$
$case(e,6,7),pid',case(e,1,0),case(e,pid',0),case(e,pid',0),case(e,pid',0),$
$case(e,1,0),0,0,case(e,2,0),case(e,pid',0),case(e,pid',0),case(e,pid',0),$
$turns,set(flags,case(e,pid',x'_{10}),case(e,1,0)))$
$\lhd case(e,s'=7,s'=1)\rhd\delta+$

8 $\tau\cdot$
$X(3,pid,0,0,0,0,0,0,0,0,x_8,x_9,x_{10},$
$s',pid',x'_0,x'_1,x'_2,x'_3,x'_4,x'_5,x'_6,x'_7,x'_8,x'_9,x'_{10},$
$turns,flags)$
$\lhd s=4\wedge x_7=2\rhd\delta+$

9 $wait(x'_3,x'_4)\cdot$
$X(s,pid,x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10},$
$4,pid',0,0,0,0,0,0,0,x'_5,x'_7,x'_8,x'_9,x'_{10},$
$turns,flags)$
$\lhd s'=5\wedge(x'_3=2\vee flags[2]<x'_4)\wedge(x'_3=1\vee flags[1]<x'_4)\wedge turns[x'_4]\neq x'_3\rhd\delta+$

10 $setturn(x'_0,x'_1)\cdot$
$X(s,pid,x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10},$
$5,pid',0,0,0,x'_3,x'_4,x'_2,0,x'_7,x'_8,x'_9,x'_{10},$
$set(turns,x'_0,x'_1),flags)$
$\lhd s'=6\rhd\delta+$

11 $criticalin(x_8)\cdot$

$$
\begin{aligned}
&\quad X(2, pid, 0, 0, 0, 0, 0, 0, 0, 0, 0, x_9, x_{10},\\
&\qquad s', pid', x'_0, x'_1, x'_2, x'_3, x'_4, x'_5, x'_6, x'_7, x'_8, x'_9, x'_{10},\\
&\qquad turns, flags)\\
&\quad \lhd s = 3 \rhd \delta +\\
12 \quad &criticalout(x_9)\cdot\\
&\quad X(1, pid, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, x_{10},\\
&\qquad s', pid', x'_0, x'_1, x'_2, x'_3, x'_4, x'_5, x'_6, x'_7, x'_8, x'_9, x'_{10},\\
&\qquad turns, flags)\\
&\quad \lhd s = 2 \rhd \delta\\
&X(7, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, nil(1), nil(0))
\end{aligned}
$$

Symmetry occurs in Peterson's algorithm because it is a parallel composition of parallel processes which only differ in their process identifiers. Intuitively, a permutation isomorphism is defined in terms of a permutation of process ids by permuting a state by moving the elements of the state vector corresponding to one process to the elements of the permutation of this process, simultaneously permuting all process identifiers occurring in the state vectors. The latter might be more complex than it looks at first, since not all process ids are simply assigned to process variables.

The 'TURNS' and 'Q' arrays are used to store information about process identifiers: the former assigns process identifiers to positions in the queue, while the latter assigns positions in the queue to process identifiers. It is essential that these data structures are permuted in the sense that, say, a permuted process has the same position in the permuted queue as the original process in the original queue.

Lemma 8.3
Suppose $\delta : Pid \to Pid$ is a permutation of process identifiers.
i The mapping $\delta_1 : (\mathbb{N} \to Pid) \to (\mathbb{N} \to Pid)$ defined by $\delta_1(f)(n) = \delta(f(n))$ is a permutation.
ii The mapping $\delta_2 : (Pid \to \mathbb{N}) \to (Pid \to \mathbb{N})$ defined by $\delta_2(f)(\delta(i)) = f(i)$ is a permutation.

The proof of the above lemma is omitted because of its simplicity; however, the actual μCRL specification needs more complex definitions. In μCRL, data is represented by terms and operations on data is specified as a rewrite system. Thus, the two above mappings δ_1, δ_2 are to be defined on terms, and the proof of their being a permutation is to be formulated in terms of rewrite systems. This report relies on an idealised mathematical data representation, but it is good to note here that the step towards exact theorem proving is less trivial than the reader might expect.

Lemma 8.4
Mapping $\pi : \mathcal{L} \to \mathcal{L}$ defined by: $\pi setflag(x, y) = setflag(\delta x, y)$, $\pi\tau = \tau$, $\pi wait(x, y) = wait(\pi x, y)$, $\pi setturn(x, y) = setturn(x, \delta y)$, $\pi criticalin(x) = criticalin(\delta x)$, $\pi criticalout(x) = criticalout(\delta x)$ is a label permutation.

Lemma 8.5
Mapping $\sigma : S \to S$ defined by $\sigma[s, pid, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, s', pid', x'_0, x'_1, x'_2, x'_3, x'_4, x'_5, x'_6, x'_7, x'_8, x'_9, x'_{10}, turns, flags] = [s', pid', x'_0, \sigma x'_1, \sigma x'_2, \sigma x'_3, x'_4, x'_5, x'_6, x'_7, \sigma x'_8, \sigma x'_9, \sigma x'_{10}, s, pid, x_0, \sigma x_1, \sigma x_2, \sigma x_3, x_4, x_5, x_6, x_7, \sigma x_8, \sigma x_9, \sigma x_{10}, \delta_1 turns, \delta_2 flags]$ is a state permutation.

Lemma 8.6
The mapping $\sigma\pi$ is a permutation isomorphism.

Proof
Now suppose $s \xrightarrow{a} t$. It is to be proven that $\sigma s \xrightarrow{\pi a} \sigma t$. To this end, there are twelve cases to be considered, matching the twelve summands of the linear process equation. Here, only the first case will be treated, the other eleven are analogous. In this case, s satisfies the guard $case(e, s = 7, s = 1, s =$

$4 \wedge x_7 \neq 2$) for some e, $a = setflag(case(e, pid, x_{10}, x_6), case(e, 1, 0, x_7))$ and $t = \langle case(e, 6, 7, 6), pid, case(e, 1, 0, x_7), case(e, pid, 0, x_6), case(e, pid, 0, x_6), case(e, pid, 0, x_6), case(e, 1, 0, x_7), 0, 0, case(e, 2, 0, x_7{+}1), case(e, pid, 0, x_8), case(e, pid, 0, x_9), case(e, pid, 0, x_{10}), s', pid', x'_0, x'_1, x'_2, x'_3, x'_4, x'_5, x'_6, x'_7, x'_8, x'_9, x'_{10}, turns, set(flags, case(e, pid, x_{10}, x_6), case(e, 1, 0, x_7))\rangle$ It can be easily verified that σs satisfies the guard of the 7th summand, with action $\pi a = setflag(case(e, \delta pid, \delta x_{10}, \delta x_6), case(e, 1, 0, x_7))$ and successor state $t' = \langle s', \delta pid, x'_0, \delta x'_1, \delta x'_2, \delta x'_3, x'_4, x'_5, x'_6, x'_7, \delta x'_8, \delta x'_9, \delta x'_{10}, case(e, 6, 7, 6), \delta pid, case(e, 1, 0, x_7), case(e, \delta pid, 0, \delta x_6), case(e, \delta pid, 0, \delta x_6), case(e, \delta pid, 0, \delta x_6), case(e, 1, 0, x_7), 0, 0, case(e, 2, 0, x_7 + 1), case(e, \delta pid, 0, \delta x_8), \delta case(e, \delta pid, 0, \delta x_9), case(e, \delta pid, 0, \delta x_{10}, \delta_1 turns, set(\delta_2 flags, case(e, \sigma pid, \sigma x_{10}, \sigma x_6), case(e, 1, 0, x_7))\rangle$. Hence, $t' = \sigma t$. □

Peterson's algorithm misses the pleasant feature that a definition of a symmetrical reduction can be distilled from the linear process equation; for this, the interaction between the processes is too complex. In Section 8.4 it will be mentioned that the reduction can be generated on-the-fly, so a symmetrical reduction can be computed, but a symbolical reduction remains an interesting research issue.

8.3 The Dining Philosophers

The dining philosophers were introduced by Dijkstra [Dij71] and have since then performed their act to enliven many publications on concurrency. The problem is that of a number of philosophers sitting around a table, each of which facing a dish filled with some undefined substance which satisfies the somewhat distasteful property that it requires two forks to be eaten. However, each of these philosophers has access to two forks, which he needs to share with his neighbours on the left and right. The problem is to design a protocol which allows these people to acquire the two forks every once in a while to eat for a while, without deadlock or starvation. The following solution is from [Tan03].

```
procedure Philosopher(i)
  while(true) do
    think;
    take_forks(i);
    eat;
    put_forks(i);
  done;
end procedure

procedure take_forks(i)
  down(mutex);
  state[i]=HUNGRY;
  test(i);
  up(mutex);
  down(s[i]);
end procedure

procedure put_forks(i)
  down(mutex);
  state[i]=THINKING;
  test((i-1)%N);
  test((i+1)%N);
  up(mutex);
end procedure

procedure test(i)
```

```
  if(state[i]==HUNGRY & state[(i-1)%N]!=EATING && state[(i+1)%N]!=EATING) do
     state[i]=EATING;
     up(s[i]);
  done;
end procedure
```

The linear process equation is as follows.

$$
\begin{aligned}
&\mathrm{X}(s\colon State, x_0\colon \mathbb{N}, x_1\colon \mathbb{N}, x_2\colon \mathbb{N}, pid\colon \mathbb{N},\\
&\quad s'\colon State, x'_0\colon \mathbb{N}, x'_1\colon \mathbb{N}, x'_2\colon \mathbb{N}, pid'\colon \mathbb{N},\\
&\quad mutex\colon \mathbb{N}, v\colon NatArray, state\colon StateArray) =\\
1\quad &\textstyle\sum_{e:Enum5}\sum_{b:Bool}\sum_{j:\mathbb{N}} up(case(e, MUTEX, S(left(pid)), S(right(pid)), MUTEX, S(pid)))\cdot\\
&\qquad X(case(b, 15, 6, 1, 24, 16), 0, 0, 0, pid,\\
&\qquad\quad s', x'_0, x'_1, x'_2, pid',\\
&\qquad\quad case(b, mutex, mutex+1), case(b, set(v, j, v[j]+1), v), state)\\
&\qquad \lhd case(e, MUTEX, S(left(pid)), S(right(pid)), MUTEX, S(pid)) = case(b, S(j), MUTEX) \wedge\\
&\qquad\quad case(e, s = 16, s = 7, s = 2, s = 1, s = 17) \wedge\\
&\qquad\quad case(b, T, j = 0) \rhd \delta +\\
2\quad &\textstyle\sum_{e:Enum3}\sum_{b:Bool}\sum_{i:\mathbb{N}}\sum_{j:\mathbb{N}} down(case(e, MUTEX, MUTEX, S(pid)))\cdot\\
&\qquad X(case(e, 22, 12, 14), 0, 0, 0, pid,\\
&\qquad\quad s', x'_0, x'_1, x'_2, pid',\\
&\qquad\quad case(b, MUTEX, j), case(b, set(v, i, j), v), state)\\
&\qquad \lhd case(e, MUTEX, MUTEX, S(pid)) = case(b, S(i), MUTEX) \wedge\\
&\qquad\quad case(e, s = 23, s = 13, s = 15) \wedge\\
&\qquad\quad case(b, T, i = 0) \wedge\\
&\qquad\quad case(b, j+1 = v[i], j+1 = mutex) \rhd \delta +\\
3\quad &\textstyle\sum_{e:Enum9}\sum_{j:\mathbb{N}} get(STATE(case(e, pid, right(pid), left(left(pid)), pid,\\
&\qquad\qquad\qquad\qquad right(right(pid)), right(pid), pid, left(pid), left(pid))), state[j])\cdot\\
&\qquad X(case(e, 20, 18, 9, 4, 3, 5, 8, 10, 19),\\
&\qquad\quad case(e, 0, state[j], 0, 0, state[j], 0, state[j], 0, 0),\\
&\qquad\quad case(e, 0, x_1, state[j], state[j], x_1, 0, x_1, 0, state[j]),\\
&\qquad\quad case(e, state[j], x_2, x_2, x_2, x_2, state[j], x_2, state[j], x_2),\\
&\qquad\quad pid,\\
&\qquad\quad s', x'_0, x'_1, x'_2, pid',\\
&\qquad\quad mutex, v, state)\\
&\qquad \lhd case(e, pid, right(pid), left(left(pid)), pid, right(right(pid)),\\
&\qquad\qquad right(pid), pid, left(pid), left(pid)) = j \wedge\\
&\qquad\quad case(e, s = 21, s = 19, s = 10, s = 5, s = 4, s = 6, s = 9, s = 11, s = 20) \rhd \delta +\\
4\quad &\textstyle\sum_{e:Enum5}\sum_{i:\mathbb{N}} set(STATE(case(e, pid, left(pid), right(pid), pid, pid),\\
&\qquad\qquad\qquad case(e, hungry, eating, eating, thinking, eating)))\cdot\\
&\qquad X(case(e, 21, 7, 2, 11, 17), 0, 0, 0, pid,\\
&\qquad\quad s', x'_0, x'_1, x'_2, pid',\\
&\qquad\quad mutex, v, set(state, i, case(e, hungry, eating, eating, thinking, eating)))\\
&\qquad \lhd case(e, pid, left(pid), left(pid), pid, pid) = i \wedge\\
&\qquad\quad case(e, s = 22,\\
&\qquad\qquad s = 8 \wedge x_2 = 1 \wedge x_1 \neq 0 \wedge x_0 \neq 0,\\
&\qquad\qquad s = 3 \wedge x_2 = 1 \wedge x_1 \neq 0 \wedge x_0 \neq 0,\\
&\qquad\qquad s = 12,\\
&\qquad\qquad s = 18 \wedge x_2 = 1 \wedge x_1 \neq 0 \wedge x_0 \neq 0) \rhd \delta +\\
5\quad &\textstyle\sum_{e:Enum3} \tau\cdot\\
&\qquad X(s, x_0, x_1, x_2, pid, case(e, 16, 1, 6), 0, 0, 0, pid', mutex, v, state)
\end{aligned}
$$

$\lhd case(e, s' = 18, s' = 3, s' = 8) \wedge x'_2 \neq 1 \wedge x'_1 \neq 0 \wedge x'_0 = 0 \rhd \delta)+$

6 $eat(pid')\cdot$
$X(s, x_0, x_1, x_2, pid, 13, 0, 0, 0, pid', mutex, v, state)$
$\lhd s' = 14 \rhd \delta+$

7 $think(pid')\cdot$
$X(s, x_0, x_1, x_2, pid, 23, 0, 0, 0, pid', mutex, v, state)$
$\lhd s' = 24 \rhd \delta+$

8 $\sum_{e:Enum5} \sum_{b:Bool} \sum_{j:\mathbb{N}} up(case(e, MUTEX, S(left(pid')), S(right(pid'))), MUTEX, S(pid')))\cdot$
$X(s, x_0, x_1, x_2, pid,$
$case(e, 15, 6, 1, 24, 16), 0, 0, 0, pid',$
$case(b, mutex, mutex + 1), case(b, set(v, j, v[j] + 1), v), state)$
$\lhd case(e, MUTEX, S(left(pid')), S(right(pid')), MUTEX, S(pid')) = case(b, S(j), MUTEX) \wedge$
$case(e, s' = 16, s' = 7, s' = 2, s' = 1, s' = 17) \wedge case(b, T, j = 0)) \rhd \delta+$

9 $\sum_{e:Enum3} \sum_{b:Bool} \sum_{i:\mathbb{N}} \sum_{j:\mathbb{N}} down(case(e, MUTEX, MUTEX, S(pid')))\cdot$
$X(s, x_0, x_1, x_2, pid,$
$case(e, 22, 12, 14), 0, 0, 0, pid',$
$case(b, mutex, j), case(b, set(v, i, j), v), state)$
$\lhd case(e, MUTEX, MUTEX, S(pid')) = case(b, S(i), MUTEX) \wedge$
$case(e, s' = 23, s' = 13, s' = 15) \wedge case(b, T, i = 0) \wedge case(b, j + 1 = v[i], j + 1 = mutex) \rhd \delta+$

10 $\sum_{e:Enum9} \sum_{j:\mathbb{N}} get(STATE(case(e, pid', right(pid'), left(left(pid')), pid', right(right(pid'),$
$right(pid'), pid', left(pid'))), left(pid'))), state[j])\cdot$
$X(s, x_0, x_1, x_2, pid,$
$case(e, 20, 18, 9, 4, 3, 5, 8, 10, 19),$
$case(e, 0, state[j], 0, 0, state[j], 0, state[j], 0, 0),$
$case(e, 0, x'_1, state[j], state[j], x'_1, 0, x'_1, 0, state[j]),$
$case(e, state[j], x'_2, x'_2, x'_2, x'_2, state[j], x'_2, state[j], x'_2), pid',$
$mutex, v, state)$
$\lhd case(e, pid', right(pid'), left(left(pid')), pid', right(right(pid')),$
$right(pid'), pid', left(pid'), left(pid')) = j \wedge$
$case(e, s' = 21, s' = 19, s' = 10, s' = 5, s' = 4, s' = 6, s' = 9, s' = 11, s' = 20) \rhd \delta+$

11 $\sum_{e:Enum5} \sum_{i:\mathbb{N}} set(STATE(case(e, pid', left(pid'), right(pid'), pid', pid')),$
$case(e, hungry, eating, eating, thinking, eating))\cdot$
$X(s, x_0, x_1, x_2, pid,$
$case(e, 21, 7, 2, 11, 17), 0, 0, 0, pid',$
$mutex, v, set(state, i, case(e, hungry, eating, eating, thinking, eating)))$
$\lhd case(e, pid', left(pid'), left(pid'), pid', pid') = i \wedge$
$case(e, s' = 22,$
$s' = 8 \wedge x'_2 = 1 \wedge x'_1 \neq 0 \wedge x'_0 \neq 0,$
$s' = 3 \wedge x'_2 = 1 \wedge x'_1 \neq 0 \wedge x'_0 \neq 0,$
$s' = 12,$
$s' = 18 \wedge x'_2 = 1 \wedge x'_1 \neq 0 \wedge x'_0 \neq 0) \rhd \delta+$

12 $\sum_{e:Enum3} \tau\cdot$
$X(case(e, 16, 1, 6), 0, 0, 0, pid, s', x'_0, x'_1, x'_2, pid', mutex, v, state)$
$\lhd case(e, s = 18, s = 3, s = 8) \wedge x_2 \neq 1 \wedge x_1 \neq 0 \wedge x_0 = 0 \rhd \delta)+$

13 $eat(pid)\cdot$
$X(13, 0, 0, 0, pid, s', x'_0, x'_1, x'_2, pid', mutex, v, state)$
$\lhd s = 14 \rhd \delta+$

14 $think(pid)\cdot$
$X(23, 0, 0, 0, pid, s', x'_0, x'_1, x'_2, pid', mutex, v, state)$
$\lhd s = 24 \rhd \delta$

$X(24,0,0,0,1,24,0,0,0,2,1,nil(1),nil(0))$

This is another example of similar parallel processes, so permutations of process identifiers are expected to be the key to symmetry. However, not every permutation gives rise to a correct permutation isomorphism, which follows from the observation that no two neighbours, having to share one fork, can be eating at the same time. So, only the permutations that respect 'neighbourship' are possible candidates. In the simple two-philosophers case, this plays no role.

Definition 8.7
Let δ be a permutation of process identifiers.

Lemma 8.8
The mapping $\pi : \mathcal{L} \to \mathcal{L}$ defined by $\pi up(MUTEX) = up(MUTEX)$, $\pi up(S(pid)) = up(S(\delta pid))$, $\pi down(MUTEX) = down(MUTEX)$, $\pi down(S(pid)) = down(S(\delta pid))$, $\pi\tau = \tau$, $\pi get(STATE(pid)) = get(STATE(\delta pid))$, $\pi set(STATE(pid), s) = set(STATE(\delta pid), s)$.

Lemma 8.9
The mapping σ defined by
$\sigma\langle s, x_0, x_1, x_2, pid, s', x'_0, x'_1, x'_2, pid', mutex, v, state\rangle = \langle s', x'_0, x'_1, x'_2, pid', s, x_0, x_1, x_2, pid, mutex, v, state\rangle$
is a state permutation.

Lemma 8.10
The mapping $\sigma\pi$ is a permutation isomorphism.

Proof
The proof is similar to the proof for Peterson's algorithm. □

8.4 Statistics
With symmetrical reductions having been encoded as a labeled transition system, with the permutation isomorphism stored in the ρ transitions, a natural question is what the dimensions of these encodings are. Also, with soundness having been established for a number of equivalence relations and completeness rendered complicated, the question remains to what extent symmetrical reductions are complete. This section presents the statistics for the case studies presented in the previous section, with the extension that the statistics are generated for symmetry of both order 2 and order 3.

The approach taken to generate statistics on the dimensions is to modify the existing *instantiator*, which generates labeled transition systems from μCRL specifications, in such a way that it generates symmetrical reductions. The modifications contain hard-coded definitions of the permutation isomorphisms verified in the previous section, so the resulting code is only applicable for the case studies addressed. However, the rationale of these 'hacks' may guide the development of a general symmetrical reduction tool.

Table 2 presents the dimensions of the labeled transition system and the encoding of the symmetrical reduction for the alternating bit protocol, with 2 and 3 data elements, and Peterson's mutual exclusion protocol and the Dining Philosophers, both for 2 and 3 processes. The dimensions tabulated are the number of states and transitions, the number of $\rho(i)$ transitions ($1 \leqslant i < N$), and the number of cycles of ρ transitions of length i ($1 \leqslant i \leqslant N$). The latter two sequences of numbers are presented as a comma separated list.

For the alternating bit protocol, the size of the reduction is independent from N, as was already suggested in Section 8. For Peterson's algorithm and the dining philosophers, things are more complicated, as is to be expected from their nature of being a parallel composition of similar processes of C states: while the size of the labeled transition systems is $\mathcal{O}(C^N)$ the size of the symmetrical reduction is $\mathcal{O}(\frac{C^N}{N})$

		original		encoded reduction		
labeled transition system	N	states	trans.	states	trans.	rho cycles [length, period]
alternating bit protocol	2	97	122	50	61	3 [2,1]
Peterson's algorithm	2	42	72	24	36	2 [2,1], 2 [3,2]
dining philosophers	2	980	1578	502	789	6 [2,1], 9 [3,2]
alternating bit protocol	3	144	183	50	61	3 [1,1]
Peterson's algorithm	3	668	1500	259	500	7 [2,3], 2 [1,1], 14 [3,3]
dining philosophers	3	14916	32160	5548	10720	558[2,3], 6 [1,1], 7 [3,3]

Table 2: The dimensions of the reductions for the case studies

		strong bisimulation minimisation						
		original		reduction		expansion		
labeled transition system	N	states	trans.	states	trans.	states	trans.	%
alternating bit protocol	2	68	86	36	46	70	88	93
Peterson's algorithm	2	34	56	24	36	42	72	0
dining philosophers	2	976	1572	497	786	976	1572	0
alternating bit protocol	3	100	128	36	46	106	132	86
Peterson's algorithm	3	347	753	259	558	668	1500	0
dining philosophers	3	14856	32058	5536	11841	14916	32106	0

Table 3: The relative completeness for strong bisimulation for the case studies

The statistics for relative completeness were generated through use of bisimulation minimisation tools from the Cæsar/Aldébaran [FGK+96] tool suite. Both the original labeled transition system and the encoded symmetrical reduction were minimised modulo strong bisimulation and for the minimised encoding the dimensions of the symmetrical expansion were calculated through Lemma 6.8. The procedure was generated for weak bisimulation, with the difference that all internal actions were hidden.

The results are presented in Table 3 and Table 4. For strong bisimulation (Table 3) the results are quite disappointing for Peterson's algorithm and the Dining Philosophers, in that no bisimulations are symmetrical. The cause for this is probably in the complex interaction between the parallel processes and the fact that most of the action labels contain the process identifier as parameter. For weak bisimulation (Table 4), after hiding the internal actions, the relative completeness approaches satisfactory levels.

The conclusion that can be drawn from these statistics are that weak bisimulation minimisation could benefit from symmetrical reductions, if an efficient algorithm for generating symmetrical reductions is found. For some cases, like the alternating bit protocol, where the generation of symmetrical reduction is trivial, and can be even done symbolically, symmetrical reduction could facilitate strong bisimulation minimisation.

9. Conclusions and future work

This report introduced the notion of symmetry for labeled transition systems, based on the notion of permutation of action labels. It was shown that a symmetrical labeled transition system can be reduced to a 'symmetrical fragment' with sufficient information to restore the original from its reduction.

The most interesting question that is left to be answered is what manipulations of labeled transition systems could benefit from the research reported here, in the sense that they can be performed more efficiently on the symmetrical reduction, as opposed to the original labeled transition system. The type of manipulation touched upon in Section 8.4, i.e. reduction modulo some form of bisimulation, is a possible candidate, although the statistics are not conclusive. The second application of symmetry

		weak bisimulation minimisation						
		original		reduction		expansion		
labeled transition system	N	states	trans.	states	trans.	states	trans.	%
alternating bit protocol	2	3	4	2	2	3	4	100
Peterson's algorithm	2	16	28	17	31	26	50	69
dining philosophers	2	37	70	51	76	68	152	98
alternating bit protocol	3	4	6	2	2	3	4	100
Peterson's algorithm	3	164	375	185	358	635	1074	49
dining philosophers	3	291	774	544	1286	807	3006	96

Table 4: The relative completeness for weak bisimulation for the case studies

is the generation of labeled transition systems. In cases like the alternating bit protocol, where symmetrical reductions can be symbolically generated, it might be more efficient to generate a labeled transition system as the expansion of its symmetrical reduction. The third application is visualisation. On the one hand, existing algorithms might benefit from symmetrical reduction and expansion. On the other hand, it is good to make explicit the symmetry inherent in a system. The fourth application is model checking. For some properties, like absence of deadlocks, it can be easily understood that these hold for a labeled transition system if and only if these hold for the symmetrical reduction. This understanding can be elaborated upon, by defining *symmetrical properties* which are insensitive to permutations of action labels in the same way as labeled transaction systems, and some notion of *reduction* of which can be checked in the symmetrical reduction of the system.

More technical is the development of efficient algorithms and implementations of the concepts developed. Section 8.4, suggests that symmetry checking is amenable to an automated theorem proving approach and relies on an 'ad hack' modification of an on-the-fly generator of symmetrical reductions. Also, generating symmetrical expansions from reductions is not complex, but an implementation has not been developed yet.

On the theoretical plane, it is interesting to study the freedom of choice in the generation of symmetrical reduction. With relative completeness in mind, it is obvious that some reductions are better than others, although it is not always clear how to effectively generate better reductions. Finally, it is worthwhile to pursue symbolical reductions, which can be efficiently generated as modified linear process equations.

ACKNOWLEDGEMENT

Wan Fokkink and Jun Pang have read and commented upon a draft version of this report

References

[AKS83] S. Aggarwal, R. P. Kurhan, and K. Sabnani. A calculus for protocol specification and validation. In H. Rudin and C. West, editors, *Protocol Specification, Testing and Verification*, volume III, pages 19–34. Elsevier Science Publishers B.V., 1983.

[BDH00] D. Bošnački, D. Dams, and L. Holenderski. Symmetric Spin. In *SPIN 2000*, volume 1885 of *Lecture Notes in Computer Science*, pages 1–19, 2000.

[BFG+01] S. C. C. Blom, W. J. Fokkink, J. F. Groote, I. A. van Langevelde, B. Lisser, and J. C. van de Pol. μCRL: A toolset for analysing algebraic specifications. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification (CAV 2001)*, volume 2102 of *Lecture Notes in Computer Science*, pages 250–254, 2001.

[BG94] M. A. Bezem and J. F. Groote. Invariants in process algebra with data. In B. Jonsson and J. Parrow, editors, *Proceedings CONCUR'94*, volume 836 of *Lecture Notes in Computer Science*, pages 401–416. Springer Verlag, 1994.

[BGL+03] S. C. C. Blom, J. F. Groote, I. A. v. Langevelde, B. Lisser, and J. v. d. Pol. New developments around the μCRL tool set. In *Proceedings International ERCIM Workshop on Formal Methods for Industrial Critical Systems (FMICS 2003)*, volume 80 of *Electronic Notes in Theoretical Computer Science*, June 2003.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[CFJ93] E. M. Clarke, T. Filkorn, and J. Jha. Exploiting symmetry in temporal logic model checking. In *Proceedings of the 5th International Conference on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 244–263, June 1993.

[DDHY92] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang. Hardware verification as a hardware design aid. In *Proceedings of the IEEE International Conference on Computer Design*, pages 522–525. IEEE Computer Society, 1992.

[Def81] Defense Advanced Research Projects Agency. *Transmission Control Protocol*, September 1981.

[Dij71] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1:115–138,

1971.

[ES96] F. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design: An International Journal*, 9(1/2):105–131, August 1996.

[FGK+96] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP (Cæsar/Aldébaran development package): A protocol validation and verification toolbox. In R. Alur and T. A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification*, volume 1102 of *LNCS*, pages 437–440. Springer Verlag, August 1996.

[Fok00] W. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science (an EATCS Series). Springer Verlag, January 2000.

[Gla90] R. v. Glabbeek. The linear time – branching time spectrum. In J. Baeten and J. Klop, editors, Proceedings *CONCUR '90, Theories of Concurrency: Unification and Extension,* Amsterdam, August 1990, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer Verlag, 1990.

[God90] P. Godefroid. Using partial orders to improve automatic verification methods. In *Proceedings of CAV'90*, pages 321–340. ACM, DIMACS volume 3, 1990.

[HJJJ84] P. Huber, A. M. Jensen, L. O. Jepsen, and K. Jensen. Towards reachability trees for high-level petri nets. In G. Rozenberg, editor, *Advances in Petri Nets 1984*, volume 188 of *Lecture Notes in Computer Science*, pages 215–233, 1984.

[Hof82] C. M. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism*, volume 136 of *Lecture Notes in Computer Science*. Springer Verlag, 1982.

[Hol91] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[ID96] C. N. Ip and D. L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9:41–75, 1996.

[LN00] R. S. Lazić and D. Nowak. A unifying approach to data independence. In *Proceedings of the 11th International Conference on Concurrency Theory*, volume XXX of *Lecture Notes in Computer Science*. Springer Verlag, 2000.

[Lub84] B. D. Lubachevski. An approach to automating the verification of compact parallel coordination programs I. *Acta Informatica*, 21:125–169, 1984.

[Pet81] G. L. Peterson. Myths about the mutual exclusion problem. *Information Processing Letters*, 12(3):115–116, June 1981.

[RS98] J. M. T. Romijn and J. G. Springintveld. Exploiting symmetry in protocol testing. In S. Budkowski, A. Cavelli, and E. Najm, editors, *Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE XI/PSTV XVII '98)*, pages 337–352. Kluwer Academic Publishers, 1998.

[Sta91] P. H. Starke. Reachability analysis of Petri nets using symmetries. *Systems Analysis, Modelling, Simulation*, 8(4/5):293–303, 1991.

[Tan03] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.

[Use02] Y. S. Usenko. Linearization of μCRL specifications. In M. Leuschel and U. Ultes-Nitsche, editors, *Proc. 3rd International Workshop on Verification and Computational Logic (VCL2002)*. Department of electronics and computer science, University of Southampton, October 2002. Tech. Report DSSE-TR-2002-5.

[Wol86] P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In unknown, editor, *Proceedings of the 13th ACM Symposium on Principles of Programming Languages (POPL'86)*, pages 184–193, 1986.

Appendix I
Alternating Bit Protocol in μCRL

```
%% $Id: abp,v 1.1.1.1 2000/03/30 12:08:31 mcrl Exp $
%% This file contains the alternating bit protocol,
%% as described in J.C.M. Baeten and W.P. Weijland,
%% Process Algebra, Cambridge Tracts in Theoretical
%% Computer Science 18, Cambridge University Press, 1990.
%% The only exception is that the domain
%% D to consist of two data elements d1 and d2 to facilitate
%% simulation.

sort D
func nil,d1,d2,d3: -> D
map  eq:D#D -> Bool

rew  eq(d1,d1)=T
     eq(d2,d2)=T
     eq(d3,d3)=T
     eq(d1,d2)=F
     eq(d1,d3)=F
     eq(d2,d1)=F
     eq(d2,d3)=F
     eq(d3,d1)=F
     eq(d3,d2)=F
     eq(nil,nil)=T
     eq(nil,d1)=F
     eq(nil,d2)=F
     eq(nil,d3)=F
     eq(d1,nil)=F
     eq(d2,nil)=F
     eq(d3,nil)=F


sort Bool

map  and,or:Bool#Bool -> Bool
     not:Bool -> Bool
     eq:Bool#Bool->Bool

func T,F:-> Bool

var  x:Bool
rew  and(T,x)=x
     and(x,T)=x
     and(x,F)=F
     and(F,x)=F
     or(T,x)=T
     or(x,T)=T
     or(x,F)=x
     or(F,x)=x

     not(F)=T
     not(T)=F

     eq(x,T)=x
     eq(T,x)=x
     eq(F,x)=not(x)
     eq(x,F)=not(x)


sort  error
func  e:-> error
map   eq:error#error -> Bool

var   x:error
rew   eq(x,x)=T

sort  bit
func  0,1:-> bit
map   invert:bit -> bit
      eq:bit#bit-> Bool

var   x:bit
rew   eq(x,x)=T
      eq(0,1)=F
      eq(1,0)=F

rew   invert(1)=0
      invert(0)=1

act   r1,s4    : D
      s2,r2,c2 : D#bit
      s3,r3,c3 : D#bit
      s3,r3,c3 : error
      s5,r5,c5 : bit
      s6,r6,c6 : bit
      s6,r6,c6 : error

      tau_s3dn tau_s3e tau_s6n tau_s6e

comm  r2|s2 = c2
      r3|s3 = c3
      r5|s5 = c5
      r6|s6 = c6

proc  S0(d:D,n:bit) = S1(nil,0).S1(nil,1).S0(nil,0)
      S1(d':D,n:bit)= sum(d:D,delta<|eq(d,nil)|>r1(d).S2(d,n))
      S2(d:D,n:bit) = s2(d,n).((r6(invert(n))+r6(e)).S2(d,n)+r6(n))

      R             = R(1).R(0).R
      R(n:bit)      = (sum(d:D,r3(d,n))+r3(e)).s5(n).R(n)+
                        sum(d:D,r3(d,invert(n)).s4(d).s5(invert(n)))

      K             = sum(d:D,sum(n:bit,r2(d,n).
                            (tau_s3dn.s3(d,n)+tau_s3e.s3(e)))) .K

      L             = sum(n:bit,r5(n).(tau_s6n.s6(n)+tau_s6e.s6(e))).L


init
  hide({tau_s3dn,tau_s3e,tau_s6n,tau_s6e},
         encap({r2,r3,r5,r6,s2,s3,s5,s6}, S0(nil,0) || K || L ||  R))
```

Appendix II
Peterson's mutual exclusion protocol in μCRL

```
% The N-proc Peterson algorithm for mutual exclusion
% Note: the constant N is defined as a natural map

% The obligatory Bool sort

sort Bool
func T,F:->Bool
map  and,or,eq:Bool#Bool->Bool
     not:Bool->Bool
var  b: Bool
rew  and(T,b)=b
     and(F,b)=F
%    and(b,T)=b
%    and(b,F)=F
     or(T,b)=T
     or(F,b)=b
     eq(F,F)=T
     eq(F,T)=F
     eq(T,F)=F
     eq(T,T)=T
     not(T)=F
     not(F)=T



% The Natural numbers

sort Nat
func 0:->Nat
     S:Nat->Nat
map  eq,lt:Nat#Nat->Bool
     sub:Nat#Nat->Nat
     N:->Nat
var  n1,n2:Nat
rew  eq(0,0)=T
     eq(S(n1),0)=F
     eq(0,S(n2))=F
     eq(S(n1),S(n2))=eq(n1,n2)
     lt(n1,0)=F
     lt(0,S(n2))=T
     lt(S(n1),S(n2))=lt(n1,n2)
     sub(n1,0)=n1
     sub(S(n1),S(n2))=sub(n1,n2)
     N=S(S(S(0)))

% Process ids

sort Pid
func zero:->Pid
     n:Pid->Pid
map  eq,lt:Pid#Pid->Bool
     sub:Pid#Pid->Pid
%    N:->Pid
var  n1,n2:Pid
rew  eq(zero,zero)=T
     eq(n(n1),zero)=F
     eq(zero,n(n2))=F
     eq(n(n1),n(n2))=eq(n1,n2)
     lt(n1,zero)=F
     lt(zero,n(n2))=T
     lt(n(n1),n(n2))=lt(n1,n2)
     sub(n1,zero)=n1
     sub(n(n1),n(n2))=sub(n1,n2)
%    N=n(n(zero))



% Mappings Pid->Nat
% Representation: cons(f(0),cons(f(1), ... cons(f(m),nil(n))...))
%                 where f(m)<>n and for all l>m: f(l)=n

sort Pid2Nat
func nil:Nat->Pid2Nat
     cons:Nat#Pid2Nat->Pid2Nat
map  eq:Pid2Nat#Pid2Nat->Bool
     if:Bool#Pid2Nat#Pid2Nat->Pid2Nat
     set:Pid2Nat#Pid#Nat->Pid2Nat
     get:Pid2Nat#Pid->Nat
     normalise:Pid2Nat->Pid2Nat
var  l1,l2:Pid2Nat
     n1,n2:Nat
rew  eq(nil(n1),nil(n2))=eq(n1,n2)
     eq(nil(n1),cons(n2,l2))=F
     eq(cons(n1,l1),nil(n2))=F
     eq(cons(n1,l1),cons(n2,l2))=and(eq(n1,n2),eq(l1,l2))
var  l1,l2:Pid2Nat
     b:Bool
rew  if(T,l1,l2)=l1
     if(F,l1,l2)=l2
     if(b,l1,l1)=l1
var  tail:Pid2Nat
     pos:Pid
     head,value1,value2:Nat
rew  set(nil(value1),n(zero),value2)= cons(value2,nil(value1))
     set(nil(value1),n(n(pos)),value2)= cons(value1,set(nil(value1),n(pos),value2))
     set(cons(head,tail),n(zero),value1)=cons(value1,tail)
     set(cons(head,tail),n(n(pos)),value1)=cons(head,set(tail,n(pos),value1))
     get(nil(value1),pos)=value1
     get(cons(head,tail),n(zero))=head
     get(cons(head,tail),n(n(pos)))=get(tail,n(pos))
     normalise(nil(value1))=nil(value1)
%    normalise(cons(value1,nil(value2)))=if(eq(value1,value2),
%                                           nil(value2),
%                                           cons(value1,nil(value2)))
%    normalise(cons(value1,cons(value2,tail)))=cons(value1,normalise(cons(value2,tail)))
     normalise(cons(value1,tail))=if(eq(nil(value1),normalise(tail)),
                                     nil(value1),
                                     cons(value1,normalise(tail)))


% Mappings Nat->Pid
% Representation: cons(f(0),cons(f(1), ... cons(f(m),nil(pid))...))
%                 where f(m)<>pid and for all n>m: f(n)=pid

sort Nat2Pid
func nil:Pid->Nat2Pid
     cons:Pid#Nat2Pid->Nat2Pid
map  eq:Nat2Pid#Nat2Pid->Bool
     if:Bool#Nat2Pid#Nat2Pid->Nat2Pid
     set:Nat2Pid#Nat#Pid->Nat2Pid
     get:Nat2Pid#Nat->Pid
     normalise:Nat2Pid->Nat2Pid
var  l1,l2:Nat2Pid
     n1,n2:Pid
rew  eq(nil(n1),nil(n2))=eq(n1,n2)
     eq(nil(n1),cons(n2,l2))=F
     eq(cons(n1,l1),nil(n2))=F
     eq(cons(n1,l1),cons(n2,l2))=and(eq(n1,n2),eq(l1,l2))
var  l1,l2:Nat2Pid
     b:Bool
rew  if(T,l1,l2)=l1
     if(F,l1,l2)=l2
     if(b,l1,l1)=l1
var  tail:Nat2Pid
     pos:Nat
     head,value1,value2:Pid
rew  set(nil(value1),S(0),value2)= cons(value2,nil(value1))
     set(nil(value1),S(S(pos)),value2)= cons(value1,set(nil(value1),S(pos),value2))
     set(cons(head,tail),S(0),value1)=cons(value1,tail)
     set(cons(head,tail),S(S(pos)),value1)=cons(head,set(tail,S(pos),value1))
     get(nil(value1),pos)=value1
     get(cons(head,tail),S(0))=head
     get(cons(head,tail),S(S(pos)))=get(tail,S(pos))
     normalise(nil(value1))=nil(value1)
```

```
%    normalise(cons(value1,nil(value2)))=if(eq(value1,value2),
%                                           nil(value2),
%                                           cons(value1,nil(value2)))
%    normalise(cons(value1,cons(value2,tail)))=cons(value1,normalise(cons(value2,tail)))
     normalise(cons(value1,tail))=if(eq(nil(value1),normalise(tail)),
                                     nil(value1),
                                     cons(value1,normalise(tail)))



% Peterson's wait condition: turn[j]<>i  or  for all k<>i => flag(k)<j

map  test:Nat2Pid#Pid2Nat#Pid#Nat->Bool
     testturns:Nat2Pid#Pid#Nat->Bool
     testflags:Pid2Nat#Pid#Nat#Pid->Bool
var  turns:Nat2Pid
     flags:Pid2Nat
     i,k:Pid
     j:Nat
rew  test(turns,flags,i,j)=or(testflags(flags,i,j,n(n(n(n(n(zero))))))),testturns(turns,i,j))
     testturns(turns,i,j)=not(eq(get(turns,j),i))
     testflags(flags,i,j,zero)=T
     testflags(flags,i,j,n(k))=and(or(eq(n(k),i),lt(get(flags,n(k)),j)),
                                   testflags(flags,i,j,k))



act  getturn1:Nat#Pid
     setturn1:Nat#Pid
     getflag1:Pid#Nat
     setflag1:Pid#Nat
     wait1:Pid#Nat
     getturn2:Nat#Pid
     setturn2:Nat#Pid
     getflag2:Pid#Nat
     setflag2:Pid#Nat
     wait2:Pid#Nat
     getturn:Nat#Pid
     setturn:Nat#Pid
     getflag:Pid#Nat
     setflag:Pid#Nat
     wait:Pid#Nat
     criticalin:Pid
     criticalout:Pid

comm getturn1|getturn2=getturn
     setturn1|setturn2=setturn
     getflag1|getflag2=getflag
     setflag1|setflag2=setflag
          wait1|   wait2=wait



% The memory manager

proc memory(turns:Nat2Pid,flags:Pid2Nat)=
     sum(i:Nat,getturn1(i,get(turns,i)).memory(turns,flags))+
     sum(i:Nat,sum(j:Pid,setturn1(i,j).memory(normalise(set(turns,i,j)),flags)))+
     sum(i:Pid,getflag1(i,get(flags,i)).memory(turns,flags))+
     sum(i:Pid,sum(j:Nat,setflag1(i,j).memory(turns,normalise(set(flags,i,j)))))+
     sum(i:Pid,sum(j:Nat,wait1(i,j).memory(turns,flags)<|test(turns,flags,i,j)|>delta))

% Entering the critical section

proc enter(i:Pid,j:Nat)=
     tau
        <| eq(j,N) |>
     setflag2(i,j).
     setturn2(j,i).
     wait2(i,j).
     enter(i,S(j))

% Leaving the critical section

proc leave(i:Pid)=
     setflag2(i,0)

% The client process

proc P(pid:Pid)=
     enter(pid,S(0)).
     criticalin(pid).
     criticalout(pid).
     leave(pid).
     P(pid)



% Launch the clients and the memory manager

init %hide({getturn,setturn,getflag,setflag,wait},
           encap({getturn1,getturn2,
                  getflag1,getflag2,
                  setturn1,setturn2,
                  setflag1,setflag2,
                  wait1,   wait2},
           P(n(zero))||P(n(n(zero))) ||P(n(n(n(zero))))
           ||memory(nil(zero),nil(0)))%)
```

Appendix III
Dining Philosophers in μCRL

```
% The N-proc Dining Philosophers solution
% Note: the constant N is defined as a natural map

% The obligatory Bool sort

sort Bool
func T,F:->Bool
map  and,or,eq:Bool#Bool->Bool
     not:Bool->Bool
var  b: Bool
rew  and(T,b)=b
     and(F,b)=F
     and(b,T)=b
     and(b,F)=F
     or(T,b)=T
     or(F,b)=b
     eq(F,F)=T
     eq(F,T)=F
     eq(T,F)=F
     eq(T,T)=T
     not(T)=F
     not(F)=T



% The Natural numbers

sort Nat
func 0:->Nat
     S:Nat->Nat
map  eq,lt:Nat#Nat->Bool
     sub:Nat#Nat->Nat
     N:->Nat
     Eating:->Nat
     Hungry:->Nat
     Thinking:->Nat
     left:Nat->Nat
     right:Nat->Nat
     if:Bool#Nat#Nat->Nat
var  n1,n2:Nat
     b:Bool
rew  eq(0,0)=T
     eq(S(n1),0)=F
     eq(0,S(n2))=F
     eq(S(n1),S(n2))=eq(n1,n2)
     lt(n1,0)=F
     lt(0,S(n2))=T
     lt(S(n1),S(n2))=lt(n1,n2)
     sub(n1,0)=n1
     sub(S(n1),S(n2))=sub(n1,n2)
     N=S(S(0))
     Eating=0
     Hungry=S(0)
     Thinking=S(S(0))
     left(S(0))=N
     left(S(S(n1)))=S(n1)
     right(n1)=if(eq(n1,N),S(0),S(n1))
     if(T,n1,n2)=n1
     if(F,n1,n2)=n2
     if(b,n1,n1)=n1



% Unbounded arrays of Naturals

sort NatArray
func nil:Nat->NatArray
     cons:Nat#NatArray->NatArray
map  eq:NatArray#NatArray->Bool
     if:Bool#NatArray#NatArray->NatArray
     set:NatArray#Nat#Nat->NatArray
     get:NatArray#Nat->Nat
     normalise:NatArray->NatArray
var  l1,l2:NatArray
     n1,n2:Nat
rew  eq(nil(n1),nil(n2))=eq(n1,n2)
     eq(nil(n1),cons(n2,l2))=F
     eq(cons(n1,l1),nil(n2))=F
     eq(cons(n1,l1),cons(n2,l2))=and(eq(n1,n2),eq(l1,l2))
var  l1,l2:NatArray
     b:Bool
rew  if(T,l1,l2)=l1
     if(F,l1,l2)=l2
     if(b,l1,l1)=l1
var  tail:NatArray
     pos,value1,value2,head:Nat
rew  set(nil(value1),S(0),value2)= cons(value2,nil(value1))
     set(nil(value1),S(S(pos)),value2)= cons(value1,set(nil(value1),S(pos),value2))
     set(cons(head,tail),S(0),value1)=cons(value1,tail)
     set(cons(head,tail),S(S(pos)),value1)=cons(head,set(tail,S(pos),value1))
     get(nil(value1),pos)=value1
     get(cons(head,tail),S(0))=head
     get(cons(head,tail),S(S(pos)))=get(tail,S(pos))
var  tail:NatArray
     pos,value1,value2,head:Nat
rew  normalise(nil(value1))=nil(value1)
     normalise(cons(value1,nil(value2)))=if(eq(value1,value2),
                                            nil(value2),
                                            cons(value1,nil(value2)))
     normalise(cons(value1,cons(value2,tail)))=cons(value1,normalise(cons(value2,tail)))



sort Variable
func s:Nat->Variable
     mutex:->Variable
     state:Nat->Variable
map  eq:Variable#Variable->Bool
var  v1,v2:Variable
     n1,n2:Nat
rew  eq(mutex,s(n2))=F
     eq(s(n1),mutex)=F
     eq(mutex,state(n2))=F
     eq(state(n1),mutex)=F
     eq(s(n1),state(n2))=F
     eq(state(n1),s(n2))=F
     eq(mutex,mutex)=T
     eq(s(n1),s(n2))=eq(n1,n2)
     eq(state(n1),state(n2))=eq(n1,n2)




act  get1:Variable#Nat
     set1:Variable#Nat
     up1:Variable
     down1:Variable
     think:Nat
     eat:Nat
     get2:Variable#Nat
     set2:Variable#Nat
     up2:Variable
     down2:Variable
     get:Variable#Nat
     set:Variable#Nat
     up:Variable
     down:Variable

comm get1|get2=get
     set1|set2=set
     up1 |up2=up
     down1|down2=down
```

```
% The memory manager

proc memory(mutexValue:Nat, sValue:NatArray, stateValue: NatArray)=
                          up1(mutex).memory(S(mutexValue),sValue,stateValue)+
               sum(j:Nat, up1(s(j)).memory(mutexValue,normalise(set(sValue,j,S(get(sValue,j)))),stateValue))+
               sum(j:Nat, get1(state(j),get(stateValue,j)).memory(mutexValue,sValue,stateValue))+
               sum(j:Nat, down1(mutex).memory(j,sValue,stateValue) <| eq(S(j),mutexValue) |> delta)+
     sum(i:Nat,sum(j:Nat, down1(s(i)).memory(mutexValue,normalise(set(sValue,i,j)),stateValue) <| eq(S(j),get(sValue,i))|>delta)+
     sum(i:Nat,sum(j:Nat, set1(state(i),j).memory(mutexValue,sValue,normalise(set(stateValue,i,j)))))

% The client process

proc philosopher(pid:Nat)=
     think(pid).
     takeForks(pid).
     eat(pid).
     putForks(pid).
     philosopher(pid)

proc takeForks(pid:Nat)=
     down2(mutex).
     set2(state(pid),Hungry).
     test(pid).
     up2(mutex).
     down2(s(pid))

proc putForks(pid:Nat)=
     down2(mutex).
     set2(state(pid),Thinking).
     test(left(pid)).
     test(right(pid)).
     up2(mutex)

proc test(pid:Nat)=
     sum(s1:Nat, get2(state(pid),s1).
     sum(s2:Nat, get2(state(left(pid)),s2).
     sum(s3:Nat, get2(state(right(pid)),s3).
        (set2(state(pid),Eating).
         up2(s(pid))
            <| and(eq(s1,Hungry),and(not(eq(s2,Eating)),not(eq(s3,Eating)))) |>
         tau))))

% Launch the clients and the memory manager

init %hide({get,set,up,down, think},
           encap({get1,get2,
                  set1,set2,
                  up1, up2,
                  down1, down2},
            philosopher(S(0))||philosopher(S(S(0)))||memory(S(0),nil(S(0)),nil(0)))%)
```