Verification by State Spaces with Equivalence Classes

Jens Bæk Jørgensen and Lars Michael Kristensen

Computer Science Department, University of Aarhus
Ny Munkegade, Bldg. 540, DK–8000 Aarhus C, Denmark
E-mail: {jbj, kris}@daimi.aau.dk

Abstract. This paper demonstrates the potential of verification based on state spaces reduced by equivalence relations. The basic observation is that quite often, some states of a system are similar, i.e., they induce similar behaviours. Similarity can be formally expressed by defining equivalence relations on the set of states and the set of actions of a system under consideration. A state space can be constructed, in which the nodes correspond to equivalence classes of states and the arcs correspond to equivalence classes of actions. Such a state space is often much smaller than the ordinary full state space, but does allow derivation of many verification results.
Other researchers have taken advantage of the symmetries of systems, which induce a certain kind of equivalence. The contribution of this paper is to show that a more general notion of equivalence is useful. As example, a communication protocol modelled in the formalism of Coloured Petri Nets is verified. Aided by a computer tool supporting state spaces with equivalence classes, significant reductions of state spaces are exhibited.

Topics. State space reduction methods, equivalence vs. symmetry, High-level Petri Nets, communication protocols.

1 Introduction

In the research of verification of parallel and distributed systems, some attention has been given to take advantage of *symmetry* to alleviate the state explosion problem [2]. Symmetry appears when a system is composed of similar components, whose identities are immaterial with respect to state space verification. As an example, consider the well-known dining philosophers system. A state of this system in which philosophers 1 and 3 are eating, is symmetric to a state in which philosophers 3 and 5 are eating. The first state can be mapped to the second by the permutation which rotates philosopher i into philosopher $i + 2$ (modulo the number of philosophers). Symmetry is also present in many real-world systems.

In [8], state spaces with equivalence classes (SSEs) are presented under the name OE-graphs as a theoretical generalisation of state spaces based on symmetries. The author of [8] notes that the experiences with practical use of SSEs are rather limited, and the examples given are all equivalences defined using only the structure of the systems under consideration. In particular, symmetry is a structural, statical notion, based on permutation of similar components. The contribution of this paper is to recognise that sometimes, a more dynamic kind of equivalence is needed, and to demonstrate that SSEs are applicable for this purpose. SSEs are described and defined for the formalism of Coloured Petri Nets, but the idea generalises immediately to formalisms allowing an explicit representation of both states and actions of systems.

This paper is organised as follows: Section 2 introduces the communication protocol to be used as example and the properties that we are going to verify. Moreover, Sect. 2 informally outlines the formalism of Coloured Petri Nets. Section 3 presents the concept of SSEs and shows how to define appropriate equivalence relations on the states and actions of the considered example. In Sect. 4, the example is verified: It is first proved that the equivalence relations are consistent (well-defined), ensuring that the SSEs can actually be used to derive the desired verification results. Then it is described how the verification was carried out. Finally, statistics are presented to compare verification based on SSEs and based on ordinary full state spaces. Section 5 draws the conclusions including a discussion of related work.

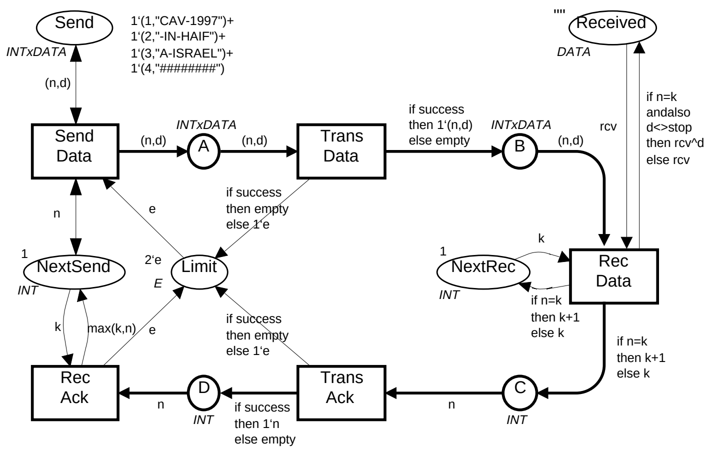
2 An Example — The Transport Protocol

In this section, we present a protocol from the transport layer of the OSI reference model, the *transport protocol.* The system consists of a *sender*, which wants to transfer some data to a *receiver.* Communication takes place on an unreliable *network*, with risk of loss and overtaking. The data is a text string, split into substrings of length eight, and each assigned a sequence number. A pair consisting of a sequence number and a string is called a *data packet.* Data packets must be received in the right order. Whenever a data packet is received, an *acknowledgement* is sent. The protocol is a stop-and-wait protocol: The sender keeps sending copies of the data packet that the receiver expects, until the sender gets a proper acknowledgement from the receiver. Then, the sender starts sending the next data packet. The term *packet* is used generically for both data packets and acknowledgements.

The model of the transport protocol, shown in Fig. 1, is created in the formalism of Coloured Petri Nets (CP-nets) [5,7] which is a graphical language for design, specification, validation, and verification of parallel and distributed systems. CP-nets belong to the class of High-level Petri Nets, a generalisation of ordinary Petri Nets, allowing succinct descriptions of models. The model in Fig. 1 is created with the Design/CPN tool [6,7], which supports CP-nets. The tool support for verification based on SSEs [9], which is used in this paper, is an integrated part of Design/CPN.

Below, we give a brief, informal description of CP-nets, sufficient to understand the model of the transport protocol and subsequently the idea of SSEs.

In Fig. 1, the state of the sender is modelled by the two ellipse-shaped *places* `Send` and `NextSend`. `Send` contains all data packets, and `NextSend` contains the number of the next data packet to be sent. The state of the receiver is, in a similar way, modelled by the two places `Received` and `NextRec`. `Received` contains the data received until now, and `NextRec` contains the number of the next data packet expected. The state of the network is modelled by the circular *network places*, `A` and `B` which may contain data packets, and `C` and `D` which may contain acknowledgements. The place `Limit` is used to model that the network has a certain capacity, i.e., that the network can maximally contain a certain number of packets at a time.



```
color INT = int;                          var n,k: INT;
color DATA = string;                      var d,rcv: DATA;
color INTxDATA = product INT * DATA;
color E = with e;
color BOOL = bool;                        var success: BOOL;
val stop = "########";
```

Fig. 1. The transport protocol.

A place has a *marking* describing its contents. A *state* of a CP-net is a function which maps each place to its marking. The marking of a place is a *multi-set*[1], i.e., it may contain more copies of the same *token*. The tokens carry data values (called colours), and each place is assigned a *type* (colour set) which determines the kind of tokens which the place may contain. The type of a place is written in italics close to the place. E.g., the place `Send` has the type `INTxDATA`, which in the *declaration box* shown at the bottom of Fig. 1, is defined as a (Cartesian) product, where the first component is of type `INT` (an integer) and the second component is of type `DATA` (a string). A data packet is represented as a token of type `INTxDATA`. The *initial state* of the CP-net is determined by the *initial markings* of the places, which are written close to the places (and omitted when empty). E.g., `Send` initially contains all data packets, all four network places are empty, and `Received` contains the empty string (`""`).

[1] Multi-sets are functions from their domains into the set of natural numbers. A multi-set ms over a domain X is written as a formal sum like $\sum_{x \in X} ms(x)'x$. `empty` denotes the empty multi-set.

In Fig. 1, the actions of the sender correspond to the two box-shaped *transitions* `SendData` and `RecAck`. `SendData` models sending of data packets and `RecAck` models reception of acknowledgements. The receiver only has one action, corresponding to the transition `RecData` which models reception of data packets and sending of acknowledgements. The actions of the network correspond to the two transitions `TransData` and `TransAck` modelling transfer of packets.

Transitions and places are connected by *arcs*. The behaviour of a CP-net consists of transitions removing tokens from the places connected by incoming arcs (input places) and adding tokens to the places connected by outgoing arcs (output places). The tokens removed and added are determined by *arc expressions*, which are positioned next to the arcs. A transition that is ready to remove and add tokens is said to be *enabled* and may *occur*. The conditions on a transition to be enabled is that appropriate tokens are present on the input places. More precisely, it must be possible to bind data values to the *variables* appearing on input arcs such that the arc expressions evaluate to tokens available on the input places. The condition on enabling of, e.g., `SendData`, is that it is possible to bind the variables `n` and `d` such that the place `Send` contains a data packet `(n,d)` and the integer `n` is on the `NextSend` place. Moreover, there must be an `e` token on the `Limit` place. A *binding element* is a pair consisting of a transition and a binding of data values to its variables. In the initial state, the binding element `(SendData,<n=1,d="CAV-1997">)` is enabled. An occurrence of the transition `SendData` models sending of a data packet. Because a double arc is a shorthand for two arcs, one arc in each direction with the same arc expression, the net effect of an occurrence of this binding element is that an `e` token is removed from the place `Limit` and a `(1,"CAV-1997")` token is added to the `A` place.

The possibility of losing packets on the network is modelled using the boolean variable `success`. In occurrences of the transitions `TransData` and `TransAck`, `success` can be bound to either `true` or `false`. The former case corresponds to successful transmission, the latter to loss of a packet.

In a given state M, the marking of a place p is denoted $M(p)$. $\mathbb{M}$ denotes the set of all states. M_0 denotes the initial state. When a binding element b is enabled in a state M_1 and the occurrence yields the state M_2, we write $M_1[b\!>\!M_2$. The notation $M_1[b\!>$ means that b is enabled in M_1. A *reachable state* is a state which can be obtained from M_0 by a sequence of occurrences of binding elements. $[M_0\!>$ denotes the set of all reachable states. The set of all binding elements is denoted BE.

The properties which we want to verify for the transport protocol are:

– *No improper terminal state*: In all terminal states (i.e., states with no enabled transitions), all data packets have been received in the right order.
– *Possibility of termination*: From any reachable state, it is always possible to reach a terminal state.
– *Eventual termination*: If only finitely many packets are lost, then the system does eventually terminate.

3 State Spaces with Equivalence Classes

An *ordinary state space (SSO)* for a CP-net is a directed graph with a node for each reachable state and arcs corresponding to occurring binding elements. The source of an arc is the state in which the associated binding element occurs, and the destination is the state resulting from the occurrence.

The definition of a *state space with equivalence classes (SSE)* for a CP-net requires that an *equivalence specification* is given. An equivalence specification consists of two equivalence relations — one on the set of states and one on the set of binding elements. The equivalence relations must capture an equivalence actually present in the considered system. This means that two equivalent reachable states must induce similar behaviours. This requirement is referred to as *consistency*, and resembles strong bisimulation in the process algebra CCS [10]. Consistency is formalised in Def. 1 below, which is equivalent to Def. 2.2. in [8]. For two states or two binding elements x and y, if x is equivalent to y, we write $x \approx y$, and the equivalence class of x is written $[x]$. For a set X, $[X]$ denotes the set of elements equivalent with some element in X.

Definition 1. Let ES be an equivalence specification. ES is *consistent* if and only if for all states $M_1, M_2 \in [[M_0\rangle]$ and all binding elements $b \in BE$:

$$\begin{array}{l} M_1 \approx M_2 \wedge M_1[b\rangle M_1^{'} \\ \Downarrow \\ \exists b^{'} \in BE, M_2^{'} \in \mathbb{M} : b^{'} \approx b \wedge M_2^{'} \approx M_1^{'} \wedge M_2[b^{'}\rangle M_2^{'}. \end{array}$$

Given a consistent equivalence specification, the SSE has a node for each equivalence class containing a reachable state. Moreover, the SSE has an arc between two nodes if and only if there is a state in the equivalence class of the source node in which a binding element is enabled and leads to a state in the equivalence class of the destination node. There is exactly one arc for each equivalence class of binding elements with this property. This is formalised in Def. 2 below, which, disregarding differences in terminology, is identical to Def. 2.3 in [8]. The set of all equivalence classes of states is denoted $\mathbb{M}_{\approx}$, and the set of all equivalence classes of binding elements is denoted $BE_{\approx}$.

Definition 2. A *state space with equivalence classes (SSE)* is a triple (V, A, N) satisfying the requirements below:

1. $V = \{C \in \mathbb{M}_{\approx} \mid C \cap [M_0\rangle \neq \emptyset\}$.
2. $A = \{(C_1, B, C_2) \in V \times BE_{\approx} \times V \mid \exists (M_1, b, M_2) \in C_1 \times B \times C_2 : M_1[b\rangle M_2\}$.
3. $\forall a = (C_1, B, C_2) \in A : N(a) = (C_1, C_2)$.

Items 1 and 2 define the sets of nodes and arcs, respectively. Item 3 defines a function which for each arc designates its source and destination. Item 3 is necessary to allow multiple arcs between two nodes which may appear in SSEs.

An SSE is often much smaller than the corresponding SSO, of course depending on the equivalence specification, and can be computed on-the-fly, i.e., without first constructing the SSO. When the SSE is finite, it can be used directly to prove many dynamic properties of the CP-net. In this paper, we will not discuss these properties exhaustively, but just note that the ones to be used for verification of the transport protocol can be proved from the SSE. For more details, the reader is encouraged to consult [8].

The equivalence specification for the transport protocol is dynamic, and based on the observation that certain packets on the network may become similar as the system executes. Suppose that the receiver expects data packet number three next. Arrival of any data packet with a number less than three does not change the state of the receiver. Such a data packet on the network will be called *old*. Arrival of any old data packet has the effect that an acknowledgement asking for data packet number three is sent. Thus two old data packets arriving at the receiver have exactly the same effect. Similar observations and terminology apply to acknowledgements arriving at the sender. The purpose of the equivalence specification is to capture that old data packets and old acknowledgements, respectively, are equivalent. The equivalence specification will be explained and defined below.

First, we consider the equivalence relation on the set of states. Let $M_1, M_2 \in \mathbb{M}$ be two states. $M_1 \approx M_2$ requires that the markings of all places but the network places `A`, `B`, `C`, and `D` are identical in M_1 and M_2. For each of the network places, the marking of the place is partitioned into two multi-sets, one containing the old packets, and one containing the other packets. $M_1 \approx M_2$ requires that the number of old packets are the same in M_1 and M_2, and that the multi-sets of the other packets are identical in M_1 and M_2. Below, $|ms|$ denotes the size of the multi-set ms, i.e., the number of elements appearing with their multiplicity taken into account. For a multi-set with only one element, ms also denotes that element. The definition uses a function *old* which takes a state M in which $|M(\texttt{NextRec})| = 1$ and one of the network places $p \in \{\texttt{A},\texttt{B}\}$ as arguments, and yields the multi-set of old packets on that place:

$$old(M,p) = \sum_{(n,d)\in M(p):n<M(\texttt{NextRec})} ((M(p))(n,d))'(n,d). \tag{1}$$

A similar function, also called *old*, coping with old acknowledgements on the places $p \in \{\texttt{C},\texttt{D}\}$ is used, where the condition $n < M(\texttt{NextRec})$ in (1) is replaced by $n \leq M(\texttt{NextSend})$. Now the definition:

$$\begin{aligned}
&M_1 \approx M_2 \Leftrightarrow \\
&(|M_1(\texttt{NextSend})| = |M_2(\texttt{NextSend})| = |M_1(\texttt{NextRec})| = |M_2(\texttt{NextRec})| = 1)\wedge \\
&(\forall p \in \{\texttt{Send},\texttt{NextSend},\texttt{Limit},\texttt{Received},\texttt{NextRec}\} : M_1(p) = M_2(p))\wedge \\
&(\forall p \in \{\texttt{A},\texttt{B},\texttt{C},\texttt{D}\} : \\
&\quad |old(M_1(p))| = |old(M_2(p))| \wedge M_1(p) \Leftrightarrow old(M_1(p)) = M_2(p) \Leftrightarrow old(M_2(p))).
\end{aligned}$$

We now consider the equivalence relation on the set of binding elements. A problem to solve first is that the transition `TransData` (see Fig. 1) cannot tell whether a data packet that it is going to transmit is old or not. It only knows the binding of its variables. Whether a data packet is old or not depends on the marking of the place `NextRec`. The solution is to extend the model with a double arc between `TransData` and `NextRec` with arc expression `k`. Similarly, a double arc is added between `TransAck` and `NextSend` with arc expression `k` to cope with old acknowledgements. This modification changes concurrency properties of the CP-net, but the SSO and SSE are preserved up to isomorphism (some binding elements are extended/renamed).

Let $b_1, b_2 \in BE$ be two binding elements. In general, $b_1 \approx b_2$ requires that b_1 and b_2 are binding elements for the same transition. For `SendData`, b_1 and b_2 themselves must be identical. For `RecData` and `RecAck`, if either b_1 or b_2 is old, the other must also be old. Otherwise, b_1 and b_2 must be identical, i.e., correspond to reception of the same packet. For `TransData` and `TransAck`, there are the same requirements, with the addition that a loss of packet in either b_1 or b_2 must be matched by the other. In the definition below, we use a predicate *is_old* which given a binding elements for the transition `TransData` yields true if the binding element corresponds to an old packet, and false otherwise. We similarly use a predicate *is_lost* signalling a loss:

$$is\_old(\texttt{TransData}, <\texttt{n} = \texttt{n}', \texttt{d} = \texttt{d}', \texttt{success} = \texttt{success}', \texttt{k} = \texttt{k}'>) = (\texttt{n}' < \texttt{k}')$$
$$is\_lost(\texttt{TransData}, <\texttt{n} = \texttt{n}', \texttt{d} = \texttt{d}', \texttt{success} = \texttt{success}', \texttt{k} = \texttt{k}'>) = \texttt{success}'.$$

We use similar predicates *is_old* which are defined for the transitions `RecData`, `TransAck`, and `RecAck`; and a predicate *is_lost* which is defined for `TransAck`. The transition of a binding element $b \in BE$ is denoted $t(b)$. Now the definition (for clarity written in a functional style):

$$b_1 \approx b_2 =$$

$$\begin{cases} b_1 = b_2, & t(b_1) = t(b_2) = \texttt{SendData} \\ (is\_old(b_1) \land is\_old(b_2)) \lor (b_1 = b_2), & t(b_1) = t(b_2) \in \{\texttt{RecData}, \texttt{RecAck}\} \\ ((is\_old(b_1) \land is\_old(b_2)\ \land & \\ (is\_lost(b_1) = is\_lost(b_2))) \lor (b_1 = b_2), & t(b_1) = t(b_2) \in \{\texttt{TransData}, \texttt{TranAck}\} \\ false, & \text{otherwise.} \end{cases}$$

4 Verification of the Transport Protocol

In this section, we describe verification of the transport protocol. First, we prove that the equivalence specification defined above is consistent. Then, we translate the properties listed at the end of Sect. 2 into dynamic properties of the CP-net, and outline how these properties can be proved from the SSE. Finally, we present statistics to compare the verification based on SSEs and SSOs for different values of the system parameter L. L is the capacity of the network as determined by the number of `e` tokens on the `Limit` place in the initial state.

We first prove that the equivalence specification defined in Sect. 3 is consistent according to Def. 1. Let $M_1, M_2 \in [[M_0>]$. Let $b \in BE$. Assume that $M_1 \approx M_2$ and that $M_1[b>M_1^{'}$. We prove the existence of $b^{'}$ and $M_2^{'}$ such that:

$$b^{'} \approx b \land M_2^{'} \approx M_1^{'} \land M_2[b^{'}>M_2^{'}.$$

We do so by a case analysis on the transition of b. In this paper, we only sketch the proof, and we only consider one of the five transitions, namely `TransData`. We split the proof into four cases according to whether b corresponds to an old data packet or not, and whether b corresponds to a loss or not. For $p \in \{\mathtt{A}, \mathtt{B}\}, i \in \{1, 2\}$, let $young(M_i(p))$ be the marking of p in M_i in which all old data packets are removed from p.

case 1: Assume $\neg is\_old(b) \wedge \neg is\_lost(b)$, i.e., b corresponds to a successful transmission of a data packet $(n', d') \in young(M_1(\mathtt{A}))$. Since $M_1 \approx M_2$ we have $(n', d') \in young(M_2(\mathtt{A}))$. Hence b is also enabled in M_2, and we choose $b^{'} = b$. Let $M_2^{'}$ be such that $M_2[b > M_2^{'}$. Since an occurrence of b cannot change the marking of `NextRec`, an occurrence of b cannot convert a data packet which is not old to an old, and vice versa. Since $M_1 \approx M_2$, we therefore have: $young(M_1^{'}(\mathtt{A})) = young(M_1(\mathtt{A})) \Leftrightarrow \{(n', d')\} = young(M_2(\mathtt{A})) \Leftrightarrow \{(n', d')\} = young(M_2^{'}(\mathtt{A}))$ and $|old(M_1^{'}(\mathtt{A}))| = |old(M_1(\mathtt{A}))| = |old(M_2(\mathtt{A}))| = |old(M_2^{'}(\mathtt{A})|$. A similar argument goes through for the place `B`. Since all places but `A` and `B` are left unchanged by an occurrence of b, we conclude that $M_1^{'} \approx M_2^{'}$.

case 2: Assume $\neg is\_old(b) \wedge is\_lost(b)$, i.e., b corresponds to a loss of a data packet, which is not old. This case is similar to case 1 above.

case 3: Assume $is\_old(b) \wedge \neg is\_lost(b)$, i.e., b corresponds to a successful transmission of an old data packet. Since $M_1 \approx M_2$ there is also an old data packet on `A` in M_2. Choose $b^{'}$ corresponding to a successful transmission of this old data packet. Clearly, $b^{'} \approx b$ and $b^{'}$ is enabled in M_2. Let $M_2^{'}$ be such that $M_2[b > M_2^{'}$. The proof that $M_1^{'} \approx M_2^{'}$ is similar to the corresponding part of case 1.

case 4: Assume $is\_old(b) \wedge is\_lost(b)$, i.e., b corresponds to a loss of an old data packet. This case is similar to case 3 above.

Let us now consider the three properties listed at the end of Sect. 2 which we want to verify for the protocol. We want to translate them into appropriate dynamic properties of the CP-net, which are formally defined in [7], and informally described in the following. For *No improper terminal state*, we need the concept of a *dead state*, which is a state in which no binding element is enabled. The *No improper terminal state* property is established if we can verify that in all dead states of the CP-net, all data packets have been properly received. For *Possibility of termination*, we need the concept of a *home space*, which is a set of states with the property that from any reachable state, it is possible to reach a state of the home space. The *Possibility of termination* property is established if we can verify that there exists a home space containing only dead states. For *Eventual termination*, we need the concept of a set of binding elements being *impartial*, meaning that elements from the set occur infinitely often in any infinite sequence of occurrences. The *Eventual termination* property is established if we can prove that the set of binding elements corresponding to loss of packets is impartial.

The proof rules for SSEs (OE-graphs) in [8] and their implementation as query functions [9], allowed us to verify the protocol. It turned out that for each investigated value of the system parameter L, the generated SSE had exactly one terminal node. The corresponding equivalence class had only one member, namely the state in which all data packets had been properly received, and all four network places were empty. This equivalence class was a home space, thus the only terminal state was a *home state*.

Table 1 contains the sizes of the SSOs and SSEs for different values of L. In addition, the generation and query times (in CPU seconds) are shown. The Ratio columns hold the savings factors, i.e., the figure for the SSO divided by the figure for the SSE. The measures were obtained on a Sun Ultra Sparc Enterprise 3000 computer with 512 MB RAM. An empty entry (-) signals that it was not possible to obtain that measure.

	Number of Nodes			Number of Arcs			Generation Time			Query Time		
L	SSO	SSE	Ratio	SSO	SSE	Ratio	SSO	SSE	Ratio	SSO	SSE	Ratio
1	33	33	1.0	44	44	1.0	1	1	1.0	1	1	1.0
2	293	155	1.9	764	383	2.0	1	1	1.0	1	1	1.0
3	1,829	492	3.7	6,860	1,632	4.2	6	7	0.9	8	6	1.3
4	9,025	1,260	7.1	43,124	5,019	8.6	56	36	1.6	63	19	3.3
5	37,477	2,803	11.2	213,902	12,685	16.9	642	157	4.1	358	48	7.5
6	136,107	5,635	24.2	891,830	28,044	31.8	7,507	553	13.6	1,666	112	14.9
7	-	10,488	-	-	56,203	-	-	1,716	-	-	225	-
8	-	18,366	-	-	104,442	-	-	4,885	-	-	432	-
9	-	30,605	-	-	182,754	-	-	12,461	-	-	755	-
10	-	48,939	-	-	304,445	-	-	30,169	-	-	1,359	-

Table 1. Verification statistics.

It can be seen that SSEs yielded remarkable reductions in the numbers of nodes and arcs, and that SSEs enabled us to analyse capacities of the network that we could not handle using SSOs. Moreover, from a certain point, generation of the SSE was faster than generation of the SSO. The query time, i.e., the actual verification of the three considered properties of the transport protocol, was, again from a certain point, much faster on the SSE than on the SSO. This is because the queries were made directly on the SSE, and the size of the graph is the critical factor in the time complexity.

5 Conclusions

The motivation to write this paper came from our work with developing tool support for SSEs [9]. Our prime focus was on state spaces with symmetries (OS-graphs with permutations for CP-nets as defined in [8]), because their usefulness was already recognised. The recent journal [2] contains four papers [1,3–5] that all demonstrate the potential of using symmetry in state space verification. A common denominator for the four papers is that symmetry is conceived as a structural property, described by permutations of similar components.

The generality of SSEs allowed us to experiment with different kinds of equivalence relations. During these experiments, we realised the usefulness of SSEs, not based on permutations. In this paper, we saw that SSEs allow equivalences that are dynamic, in the sense that they express that some information becomes irrelevant as the execution of a system progresses. An interesting question is of course, whether the results presented generalise, i.e., apply to other systems. We believe they do. We believe that the notion of *old* of the example can be found in various disguises in many systems.

State space verification methods are often touted as being automatic and thus quite reliable. For verification based on SSEs, a qualification must be made: Proving the consistency of a proposed equivalence specification may, as seen in this paper, be a non-trivial task. A manual mathematical proof had to be conducted, with the risk of making mistakes. In state spaces with symmetries (OS-graphs with permutations of [8]), proving consistency of a proposed specification can be done by a trivial analysis of all static inscriptions of the CP-net. No ingenuity is required, and the proof can be highly computer-aided. Also, in the approaches to symmetry of [1,3], it is the responsibility of the user to define the symmetries of the system and ensure their consistency. In contrast, [4] presents a procedure which automatically detects the symmetries of a system. The basic idea is to impose narrow syntactical restrictions on the modelling language ensuring that non-symmetry is not expressible.

The complexity of the consistency proof is a drawback of verification based on SSEs. However, the confidence in the consistency of an equivalence specification can be highly increased with the aid of the tool supporting SSEs as described in [8]. In conclusion, we do believe that the observations made and the reductions exhibited in this paper are very encouraging for verification based on SSEs.

Acknowledgements. We thank Kurt Jensen and Søren Christensen for help in writing this paper. We thank Rikke D. Andersen and Jesper G. Henriksen for comments and proof-reading. The work has been supported by grants from the Research Foundation and the Faculty of Science at University of Aarhus.

References

1. E.M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting Symmetries in Temporal Model Logic Model Checking. In [2]. A version also in CAV 93, LNCS 697.
2. E.A. Emerson (editor). Formal Methods in System Design, Volume 9, Numbers 1/2 — Special Issue on Symmetry in Automatic Verification, 1996.
3. E.A. Emerson and A.P. Sistla. Symmetry and Model Checking. In [2]. A version also in CAV 93, LNCS 697.
4. C.N. Ip and D.L. Dill. Better Verification Through Symmetry. In [2].
5. K. Jensen. Condensed State Spaces for Symmetrical Coloured Petri Nets. In [2].
6. K. Jensen. Design/CPN Online, Computer Science Department, University of Aarhus, Denmark. Online: `http://www.daimi.aau.dk/designCPN/`.
7. K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts.* Monographs in Theoretical Computer Science. Springer-Verlag, 1992.
8. K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Vol. 2, Analysis Methods.* Monographs in Theoretical Computer Science. Springer-Verlag, 1994.
9. J.B. Jørgensen and L.M. Kristensen. *Design/CPN OE/OS Graph Manual.* Computer Science Department, University of Aarhus, Denmark. Online: `http://www.daimi.aau.dk/designCPN/`.
10. R. Milner. *Communication and Concurrency.* Prentice-Hall International Series in Computer Science. Prentice-Hall, 1989.