Verification of Coloured Petri Nets Using 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 it does allow derivation of many verification results.
Other researchers have taken advantage of the symmetries in 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 a representative example a communication protocol is verified. Aided by a developed computer tool significant reductions of state spaces are exhibited, representing some first results on the practical use of state spaces with equivalence classes for Coloured Petri Nets.

Keywords: State space based approaches, efficient model checking, tools, Coloured Petri Nets, communication protocols, reduction by equivalence and symmetry.

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 [Eme96]. 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 [Jen94], 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 [Jen94] notes that the experiences with practical use of SSEs are rather limited and in the examples given, all equivalences are 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 (CP-nets or CPN) [Jen92], 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. 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. The reader is assumed to be familiar with CP-nets.

2 An Example — The Transport Protocol

In this section, we present a protocol from the transport layer of the ISO reference model (see e.g., [Com97]). This layer is concerned with protocols ensuring reliable transmission between sites. In the following this protocol will be referred to as 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 next, 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 CPN model of the transport protocol, shown in Fig. 1, is created with the Design/CPN tool [Jenb,CJK97], which supports CP-nets. The Design/CPN OE/OS-tool [JK96] which is used in this paper supports verification based on SSEs and is an integrated part of the Design/CPN tool. Below, we give a brief description of the CPN model of the transport protocol and introduce some notation used in subsequent sections.

The state of the sender is modelled by the two places `Send` and `NextSend`. `Send` contains all data packets, and `NextSend` contains the sequence 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 sequence 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. This is necessary in order to obtain a finite state space.

The actions of the sender correspond to the two transitions `SendData` and `RecAck`. `SendData` models sending of data packets and `RecAck` models reception of acknowledgements. The receiver has only 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. 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.

The initial markings of the places 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 (`""`). Multi-sets are functions from their domain 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.

Declarations

```
(* Colour sets *)
(* Sequence numbers*)
color INT = int;

(* Packet contents *)
color DATA = string;

(* Data packets *)
color INTxDATA =
       product INT * DATA;

color E = with e;
color BOOL = bool;

(* Variables *)
var n,k: INT;
var d,rcv: DATA
var success: BOOL;

(* Constants *)
val stop = "########";
```

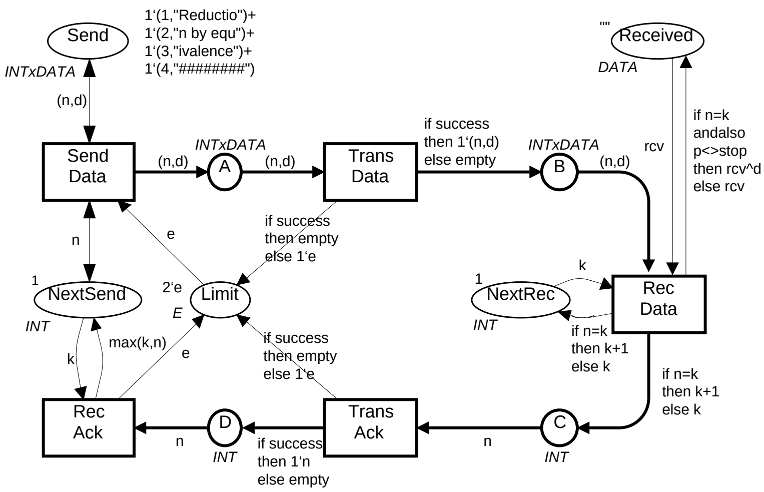

Fig. 1. CPN model of the Transport Protocol.

In a given marking M of a CP-net, the marking of a place p is denoted $M(p)$. $\mathbb{M}$ denotes the set of all markings. M_0 denotes the initial marking. When a *binding element* b (a pair consisting of a transition and a binding of data values to its variables) is enabled in a marking M_1 and the occurrence yields the marking M_2, we write $M_1[b\!>\!M_2$. The notation $M_1[b\!>$ means that b is enabled in M_1. A binding element of a transition t will be written on the form $(t, < v_1 = c_1, v_2 = c_2, \ldots v_n = c_n >)$, where $v_1 \ldots v_n$ are the variables of t and $c_1 \ldots c_n$ are data values. The set of all binding elements is denoted BE.

A *reachable marking* is a marking which can be obtained from M_0 by a sequence of occurrences of binding elements. $[M_0\!>$ denotes the set of all reachable markings.

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

- *No improper termination*: If the protocol terminates, all data packets have been received in the right order and the network is empty.
- *Possibility of termination*: In any reachable state of the protocol, it is always possible within a finite number of steps to terminate the protocol.
- *Eventual termination*: If only finitely many packets are lost, then the protocol 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 marking and arcs corresponding to occurring binding elements. The source of an arc is the marking in which the associated binding element is enabled, and the destination is the marking 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 markings ($\approx_M$) and one on the set of binding elements ($\approx_{BE}$). The equivalence relations must capture an equivalence actually present in the considered system. This means that two equivalent markings must induce similar behaviours. This requirement is referred to as *consistency*, and resembles strong bisimulation [Mil89]. Consistency is formalised in Def. 1 below, which is equivalent to Def. 2.2. in [Jen94]. For two markings 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 $ES = (\approx_M, \approx_{BE})$ be an equivalence specification for a CP-net. ES is *consistent* if and only if for all markings $M_1, M_1^{'} \in [[M_0\rangle]$ and all binding elements $b \in BE$:

$$
\begin{array}{l}
\quad M_1^{'} \in [M_1] \wedge M_1[b\rangle M_2 \\
\Downarrow \\
\quad \exists b^{'} \in [b] \wedge \exists M_2^{'} \in [M_2] : M_1^{'}[b^{'}\rangle M_2^{'}
\end{array}
$$

The definition expresses, that equivalent markings must have equivalent sets of enabled binding elements, and equivalent sets of directly reachable markings. Intuitively, this justifies, that it is sufficient to explore the possible behaviours of the system for one marking of each equivalence class.

Given a consistent equivalence specification, the SSE has a node for each equivalence class containing a reachable marking. Moreover, the SSE has an arc between two nodes if and only if there is a marking in the equivalence class of the source node in which a binding element is enabled and leads to a marking 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 [Jen94]. The set of all equivalence classes of markings 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. The SSE can be computed directly, i.e., *without* first constructing the SSO and then groupings markings and binding elements into equivalence classes. This is done using a modified version of the standard algorithm for generating an SSO. Instead of testing whether the marking is already included in the SSE, it is tested whether an equivalent marking is already included. Similarly, there is a test of whether an equivalent binding element is already inserted between the two nodes before a new arc is created.

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 [Jen94].

The user of the OE/OS-tool supplies the equivalence specification by writing two predicates. One predicate expressing when two markings are equivalent and one expressing when two binding elements are equivalent. Using the supplied equivalence specification the OE/OS-tool then computes the corresponding SSE. Once the SSE has been computed the user can invoke suitable query functions to verify the CPN model. The OE/OS-tool contains a full implementation of the proof rules for SSEs from Sect. 2.3 in [Jen94].

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 sequence 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 markings. Let $M_1, M_2 \in \mathbb{M}$ be two markings. $M_1 \approx_M M_2$ 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. $M_1 \approx_M M_2$ 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 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 marking 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 of $\approx_M$:

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

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 SSOs are isomorphic (some binding elements are extended/renamed).

Let $b_1, b_2 \in BE$ be two binding elements. In general, $b_1 \approx_{BE} b_2$ 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 corresponds to reception of an old packet, then the other must also correspond to reception of an old packet. 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 element 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 of $\approx_{BE}$:

$$b_1 \approx_{BE} b_2 \Leftrightarrow \begin{cases} b_1 = b_2, & t(b_1) = t(b_2) = \texttt{SendData} \\ (is\_old(b_1) \wedge is\_old(b_2)) \vee (b_1 = b_2), & t(b_1) = t(b_2) \in \{\texttt{RecData}, \texttt{RecAck}\} \\ ((is\_old(b_1) \wedge is\_old(b_2) \ \wedge & \\ (is\_lost(b_1) = is\_lost(b_2))) \vee (b_1 = b_2), & t(b_1) = t(b_2) \in \{\texttt{TransData}, \texttt{TranAck}\} \end{cases}$$

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 parameters L and N. L is the capacity of the network as determined by the number of `e` tokens on the `Limit` place in the initial marking, and N is the number of data packets as determined by the initial marking of the `Send` place.

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

$b' \approx_{BE} b \wedge M_2' \approx_M M_2 \wedge M_1'[b'>M_2'$.

We do so by a case analysis on the transition of b. In this paper, we sketch the proof by considering one of the five transitions, namely `TransData`. The other cases are similar. 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 $p \in \{\mathtt{A},\mathtt{B}\}$ and a marking M, let $young(M(p))$ be the marking of p in M 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 $M_1 \approx_M M_1^{'}$ we have $(n',d') \in young(M_1^{'}(\mathtt{A}))$. Hence b is also enabled in $M_1^{'}$, and we choose $b^{'} = b$. Let $M_2^{'}$ be such that $M_1^{'}[b > M_2^{'}$. 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 $M_1 \approx_M M_1^{'}$, we therefore have: $young(M_2(\mathtt{A})) = young(M_1(\mathtt{A})) - \{(n',d')\} = young(M_1^{'}(\mathtt{A})) - \{(n',d')\} = young(M_2^{'}(\mathtt{A}))$ and $|old(M_2(\mathtt{A}))| = |old(M_1(\mathtt{A}))| = |old(M_1^{'}(\mathtt{A}))| = |old(M_2^{'}(\mathtt{A})|$. 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 $M_2^{'} \approx_M M_2$.

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 $M_1 \approx_M M_1^{'}$ there is also an old data packet on `A` in $M_1^{'}$. Choose $b^{'}$ corresponding to a successful transmission of this old data packet. Clearly, $b^{'} \approx_{BE} b$ and $b^{'}$ is enabled in $M_1^{'}$. Let $M_2^{'}$ be such that $M_1^{'}[b > M_2^{'}$. The proof that $M_2^{'} \approx_M M_2$ 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 [Jen92], and informally described in the following. For *No improper termination*, we need the concept of a *dead marking*, which is a marking in which no binding elements are enabled. The *No improper termination* property is established if we can verify that in all dead markings of the CP-net, all data packets have been properly received and the network is empty. For *Possibility of termination*, we need the concept of a *home space*, which is a set of markings with the property that from any reachable marking, it is possible to reach a marking of the home space. The *Possibility of termination* property is established if we can verify that there exists a home space containing only dead markings. 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 occurrence sequence starting in the initial marking. The *Eventual termination* property is established if the set of binding elements corresponding to loss of packets is impartial.

The proof rules for SSEs (OE-graphs) in [Jen94] and their implementation as query functions in the OE/OS-tool, allowed us to verify the protocol. It turned out that for each investigated value of the system parameters L and N, the generated SSE had exactly one terminal node (a node with no outgoing arcs). The corresponding equivalence class had only one member, namely the marking 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 dead marking was a *home marking.*

Table 1 contains the sizes of the SSOs and SSEs for different values of L and a fixed number of data packets $N = 4$. 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.

The sizes of the SSOs and SSEs from Table 1 has also been depicted in Fig. 2. 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.

L	Number of Nodes			Number of Arcs			Generation Time			Query Time		
	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 (4 data packets).

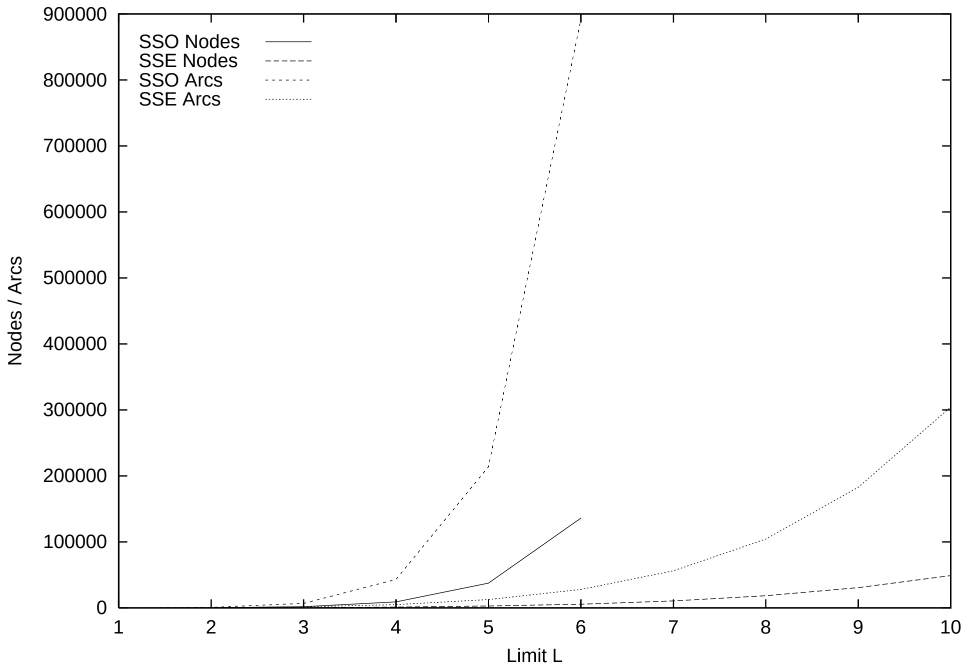

Fig. 2. Sizes (Nodes and Arcs) of SSOs and SSEs (4 data packets).

The generation and query times from Table. 1 is depicted in Fig. 3. It can be seen, that from a certain point, generation of the SSE was faster than generation of the SSO. E.g., for $L = 6$ the generation time is reduced from approximately 2 hours and 5 minutes to 9 minutes.

The query time, i.e., the actual verification of the three considered properties of the transport protocol, was, again from a certain point, significantly faster on the SSE than on the SSO. E.g., for $L = 6$ the query time is reduced from approximately 28 minutes to 2 minutes. This is because the queries are made directly on the SSE, and the size of the graph is the critical factor in the time complexity.

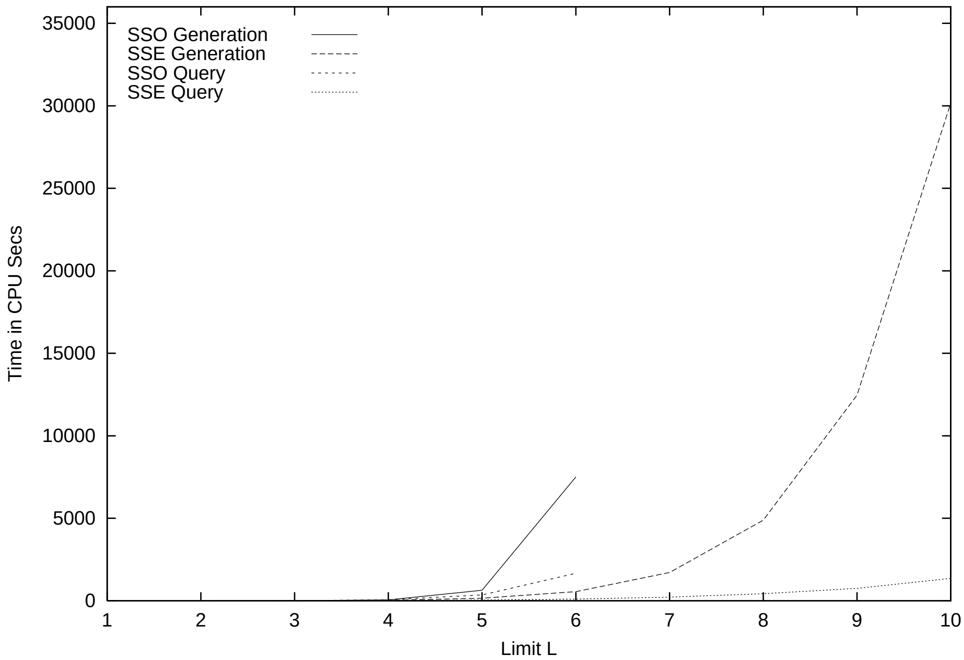

Fig. 3. Generation and Query Time for SSOs and SSEs (4 data packets).

The sizes of the SSOs and SSEs for different values of N (the number of data packets) is listed in Table 2 for a fixed value of $L = 4$. The sizes for the SSOs and SSEs has also been depicted in Fig. 4.

Again, it can be seen that the use of SSEs yielded remarkable reductions in the numbers of nodes and arcs, and that SSEs enabled us to analyse configurations of the protocol that we could not handle using SSOs. In particular, it is worth noting that the number of nodes and arcs in the SSE grows linear in the number of data packets.

5 Conclusions

The motivation to write this paper came from our work with developing tool support for SSEs [JK96]. Our prime focus was on state spaces with symmetries (OS-graphs with permutations for CP-nets as defined in [Jen94]), because their usefulness was already recognised. The recent journal [Eme96] contains four papers [CEFJ,ES,ID,Jena] 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 user-supplied equivalence relations. During these experiments we realised the new perspective of SSEs. 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. To our knowledge, the general equivalence notion provided by SSEs has not been used by others. Furthermore, the use of SSEs allowed us to verify configurations of the considered protocol,

	Number of Nodes			Number of Arcs			Generation Time			Query Time		
N	SSO	SSE	Ratio	SSO	SSE	Ratio	SSO	SSE	Ratio	SSO	SSE	Ratio
1	120	120	1.0	375	375	1.0	1	1	1.0	1	1	1.0
2	885	500	1.7	3,570	1,923	1.8	2	11	0.2	7	6	1.0
3	3,336	880	3.8	15,009	3,471	4.3	14	23	0.6	45	15	3.0
4	9,025	1,260	7.1	43,124	5,019	8.6	56	36	1.6	63	19	3.3
5	20,016	1,640	12.2	99,355	6,567	15.1	187	48	3.9	70	26	2.7
6	38,885	2,020	19.3	198,150	8,115	24.4	576	62	9.2	114	30	3.8
7	68,720	2,400	28.6	356,965	9,663	36.9	1,681	87	29.3	240	35	6.9
8	-	2,780	-	-	11,211	-	-	142	-	-	37	-
9	-	3,160	-	-	12,759	-	-	178	-	-	49	-
10	-	3,540	-	-	14,307	-	-	254	-	-	54	-

Table 2. Verification statistics (Limit 4).

which could not be handled using ordinary full state spaces. One of the main potentials of verification by means of SSEs is that it allows verification of larger configurations of the system under consideration. This is in particular important in systems which have several parameters. Due to state explosion it may only be possible to verify the system when almost all parameters have small values. By applying SSEs it is possible to verify configurations of the system where several of the parameters have large values.

An interesting question is of course, whether the kind of equivalence specification presented in this paper generalise, i.e., apply to other systems. We believe they do. We believe that the notion of being *old* of the example can be found in various disguises in many systems in particular for protocols on the transport layer of the ISO reference model. Such communication protocols are often designed using standard techniques like timers, retransmissions, and sequence numbers, thereby inherently introducing the concept of an old packet. Future work includes more investigation on the use of SSEs within this domain, which includes sliding window protocols.

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. Verification based on SSEs is semi-automatic since a manual mathematical proof of consistency have to be conducted, with the risk of making mistakes. Future work includes investigating how this proof can be computer-aided. In state spaces with symmetries (OS-graphs with permutations of [Jen94]), 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. In the approaches to symmetry of [CEFJ,ES], it is the responsibility of the user to define the symmetries of the system and ensure their consistency. In contrast, [ID] 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 only symmetric constructions are expressible. Similar ideas has been pursued in the context of Petri Nets in [CDFH91].

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 OE/OS-tool using the technique sketched in [Jen94]. In conclusion, we

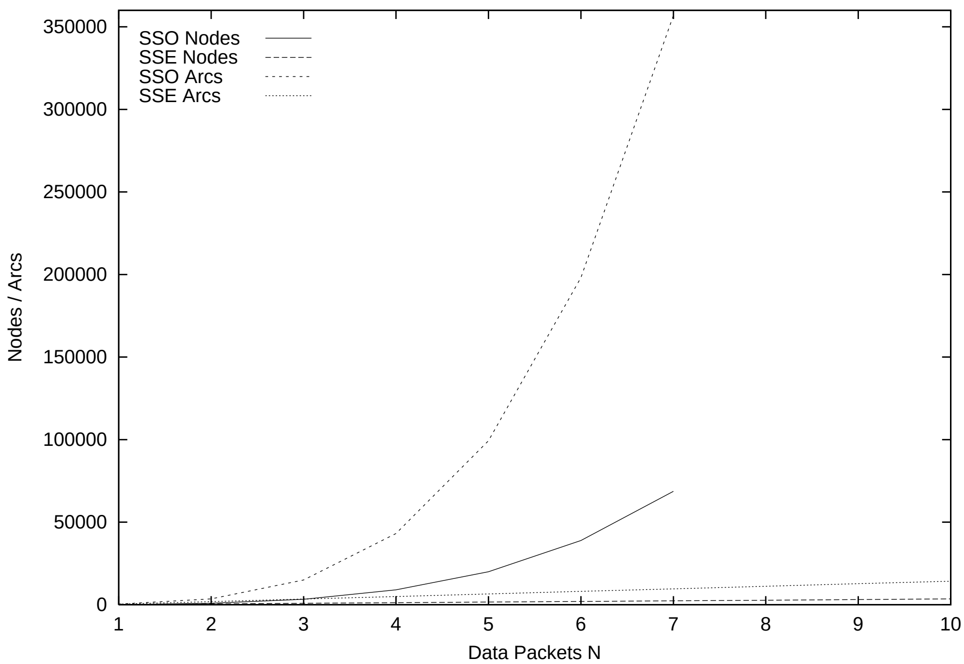

Fig. 4. Sizes (Nodes and Arcs) of SSOs and SSEs (Limit 4).

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, Jesper G. Henriksen, and Kjeld H. Mortensen for comments and proof-reading. The work has been supported by grants from University of Aarhus Research Foundation, the Faculty of Science at University of Aarhus, and the Danish National Science Research Council.

References

[CDFH91] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On Well-Formed Coloured Nets and Their Symbolic Reachability Graph. In K. Jensen and G. Rozenberg, editors, *High-level Petri Nets*. Springer Verlag, 1991.

[CEFJ] E.M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting Symmetries in Temporal Model Logic Model Checking. In [Eme96]. A version also in CAV 93, LNCS 697.

[CJK97] S. Christensen, J.B. Jørgensen, and L.M. Kristensen. Design/CPN — A Computer Tool for Coloured Petri Nets. In E. Brinksma, editor, *Proceedings of the Third International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Twente, The Netherlands*, volume 1217 of *Lecture Notes in Computer Science*, pages 209–223. Springer-Verlag, 1997. Also available as DAIMI PB-511, ISSN 0105-8517, February 1997.

[Com97] D. E. Comer. *Computer Networks and Internets*. Prentice-Hall International, Inc., 1997.

[Eme96] E.A. Emerson. Formal Methods in System Design, Volume 9, Numbers 1/2 — Special Issue on Symmetry in Automatic Verification, 1996.

[ES] E.A. Emerson and A.P. Sistla. Symmetry and Model Checking. In [Eme96]. A version also in CAV 93, LNCS 697.

[ID] C.N. Ip and D.L. Dill. Better Verification Through Symmetry. In [Eme96].

[Jena] K. Jensen. Condensed State Spaces for Symmetrical Coloured Petri Nets. In [Eme96].

[Jenb] K. Jensen. Design/CPN Online, Computer Science Department, University of Aarhus, Denmark. Online: `http://www.daimi.aau.dk/designCPN/`.

[Jen92] K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts.* Monographs in Theoretical Computer Science. Springer-Verlag, 1992.

[Jen94] K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Vol. 2, Analysis Methods.* Monographs in Theoretical Computer Science. Springer-Verlag, 1994.

[JK96] J.B. Jørgensen and L.M. Kristensen. *Design/CPN OE/OS Graph Manual.* Computer Science Department, University of Aarhus, Denmark, 1996.
Online: `http://www.daimi.aau.dk/designCPN/`.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall International Series in Computer Science. Prentice-Hall, 1989.