Chapter 8
Advanced State Space Methods

This chapter gives an overview and some concrete examples of state space reduction methods. The main limitation of using state spaces to verify behavioural properties of systems is the *state explosion problem* [106], i.e., that state spaces of systems may have an astronomical number of reachable states, which means that they are too large to be handled with the available computing power (memory and CPU speed). Methods for alleviating this inherent complexity problem are an active area of research, which has led to the development of a large collection of *state space reduction methods*. These methods have significantly broadened the class of systems that can be verified, and state spaces can now be used to verify systems of industrial size. Some of these methods [18, 61, 62, 108] have been developed in the context of the CPN modelling language. Other methods (e.g., [55, 87, 104, 110]) have been developed outside the context of the CPN modelling language. Most state space reduction methods are independent of the concrete modelling language used and hence are applicable to a large class of such languages.

Section 8.1 briefly introduces some representative classes of state space reduction methods and the associated paradigms. Section 8.2 presents the sweep-line method [18], which exploits a certain notion of progress in systems to delete states during state space exploration and thereby reduce the peak memory usage. Section 8.3 presents the symmetry method [62], which exploits inherent symmetries in systems to compute a condensed state space where each node represents an equivalence class of markings and each arc represents an equivalence class of binding elements. Finally, Sect. 8.4 presents the equivalence method [61], which is a generalisation of the symmetry method. The three advanced state space method presented in Sects. 8.2–8.4 have been used in the industrial application described in Sect. 14.3.

8.1 State Space Reduction Methods

State space reduction methods typically exploit certain characteristics of the system under analysis, and hence no single reduction method works well on all kinds of sys-

K. Jensen, L.M. Kristensen, *Coloured Petri Nets*, DOI 10.1007/b95112_8,

tems. Furthermore, the methods often limit the verification questions that can be answered. When verifying a concrete system one must therefore choose a method that exploits characteristics present in the system and preserves the behavioural properties to be verified. In many cases it is possible to use two or more state space reduction methods simultaneously, which typically leads to more reduction in CPU time and memory usage than when each method is used in isolation.

Many reduction methods are based on the paradigm of *on-the-fly verification*, which means that the verification question is stated *before* the exploration of the state space starts, and the state space exploration is done relative to the verification question provided. This makes it possible to ignore irrelevant parts of the state space and terminate the state space exploration as soon as the answer to the verification question has been obtained. Many advanced state space reduction methods uses linear-time temporal logic (LTL) [107] or computation tree temporal logic (CTL) [22] for stating the verification questions [37].

One class of methods aims at exploring only a subset of the state space. The *stubborn-set* [104], *ample-set* [87], and *persistent-set* [49] methods exploit the independence between transitions to construct a *reduced state space* which is a subset of the full state space. Many variants of these methods have been developed for different classes of behavioural properties, ranging from simple deadlock properties [104] to LTL [105] and CTL [47] model checking. A good survey of these variants can be found in [106]. It is also possible to use search heuristics known from the domain of artificial intelligence to guide the exploration of the state space such that only a subset of the state space is explored and the answer to the verification question is computed as soon as possible. An orthogonal approach is to delete states from memory during state space exploration. The *sweep-line method* [18] (to be discussed in Sect. 8.2) and the *state caching method* [58] delete states from memory during state space exploration to reduce the peak memory usage. These methods explore the full state space and may explore the same state several times. Belonging to this class of method is also the *to-store or not-to-store method* [5] which uses heuristics to decide whether a visited state must be stored in memory or not.

Another class of methods is based on computing a *condensed state space*, where each node represents an equivalence class of states (markings) and where each arc represents an equivalence class of events (binding elements). The idea is then to store only one representative for each such equivalence class and in this way consume less memory and obtain a faster construction of the state space. This class includes the *symmetry method* the [62], *equivalence method* [61], and the use of *time condensed-state spaces* [19]. Many reduction methods are also based on storing states in a compact manner in memory using special data structures. These include the *bit-state hashing method* [55], the *hash compaction method* [98, 110], and the *comback method* [108]. Belonging to this class are also methods [82] based on binary-decision diagrams (BDDs) [13] and the methods presented in [44, 55]. A related class of methods uses external storage to store the set of visited states [99].

A computer tool supporting state spaces must implement a wide range of state space reduction methods, since no single method works well on all systems. CPN

Tools supports a number of reduction methods, and the set of supported methods is continuously being expanded as new methods are developed and implemented.

8.2 Sweep-Line Method

The amount of main memory available is often the limiting factor in the practical use of state spaces. During construction of the state space, the set of markings encountered is kept in memory in order to recognise already visited markings and thereby ensure that the state space exploration terminates. The basic idea of the sweep-line method [18] is to exploit a certain kind of *progress* exhibited by many systems. Exploiting progress makes it possible to explore all of the reachable markings of a CPN model while storing only small fragments of the state space in main memory at any given time. This means that the peak memory usage is significantly reduced. The sweep-line method is aimed at on-the-fly verification of safety properties, such as verifying that all reachable states satisfy a given predicate on states or determining whether a reachable marking exists that satisfies a given predicate. Below, we illustrate the use of the sweep-line method using a variant of the CPN model of the protocol system described in Sect. 7.1, shown in Fig. 8.1. The difference compared with the model considered in Chap. 7 is that we have modified the arc expression on the arc from ReceiveAck to NextSend such that the sender never decreases the sequence number on NextSend. Furthermore, we are considering a configuration of the protocol where the network has a limit of six data packets.

For the protocol system, one source of progress is the sequence number of the receiver, i.e., the colour of the token on the place NextRec. The basic observation is that the receiver sequence number, modelled by the place NextRec has the property

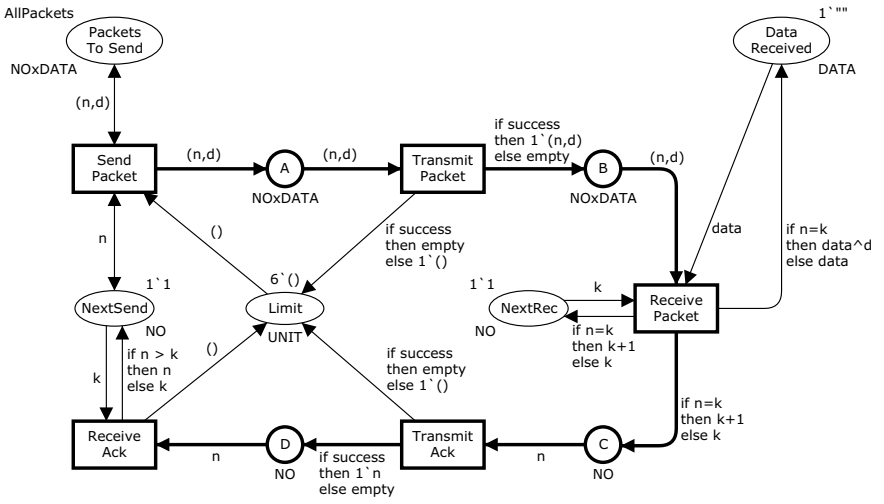

Fig. 8.1 CPN model used to illustrate the sweep-line method

that as the protocol executes, the value of this counter is increased and never decreased. This means that we can quantify how far the protocol system has progressed by considering the sequence number of the data packet expected by the receiver. This progress is also reflected in the state space of the protocol system. Figure 8.2 shows an initial fragment of the state space for the protocol system, where node 1 represents the initial marking. To simplify the drawing, we have omitted the detailed information about the markings corresponding to each node and the arc labels describing the binding elements. We have organised the nodes into layers (separated by thick horizontal lines) according to the value of the receiver sequence number on NextRec. Layer 1 contains the markings in which the receiver sequence number has the value 1, and layer 2 contains markings where the receiver sequence number is 2. This means that markings in higher-numbered layers are markings where the system has progressed further (in terms of receiver sequence number) than in markings in lower-numbered layers.

The progress present in the protocol system manifests itself in the state space in that a marking in a given layer has successor markings either in the same layer or in layers that represent further progress, but never in layers that represent less progress. Markings in layer 1 can thus never be reached from markings in layer 2 (because the value of the token on NextRec is never decremented). If we process the markings (i.e., calculate successor markings) one layer at a time, moving from one layer to the next when all markings in the first layer have been processed and not before, we can think of a 'sweep-line' moving through the state space. At any given point during state space exploration, the sweep-line corresponds to a single layer – all of the markings in the layer are 'on' the sweep-line – and all new markings calculated are either on the sweep-line or in front of the sweep-line.

The progress in the protocol system can be captured by a *progress measure* which is a function that maps each marking into a *progress value*. In this case the function

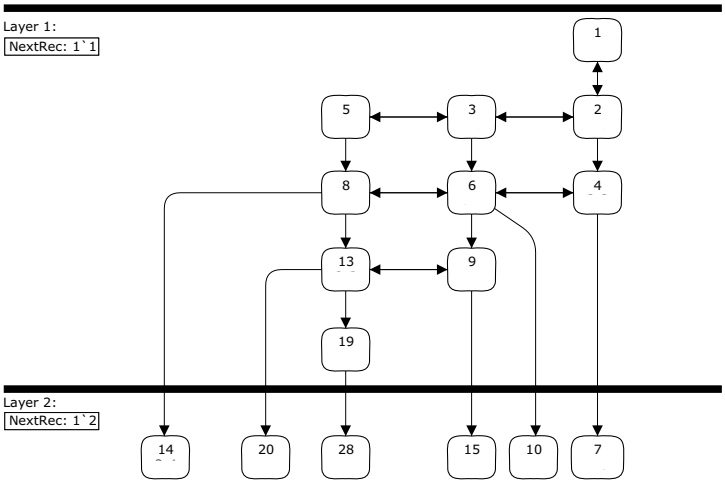

Fig. 8.2 Initial fragment of state space, arranged by progress

maps a marking into the receiver sequence number. This progress measure can be implemented as the following CPN ML function, where the structure `Mark` is used to obtain the marking of the place NextRec:

```
fun ProtocolPM n =
       ms_to_col (Mark.Protocol'NextRec 1 n);
```

The fundamental property that the sweep-line method requires from a progress measure is that for a given marking M, the progress value of any successor marking M' of M must be greater than or equal to the progress value of M. This property can be checked during state space exploration by simply checking that the property holds for all markings encountered.

Table 8.1 gives statistics for the application of the sweep-line method to the protocol system using the receiver sequence number as the progress measure. The column 'Limit' specifies the capacity of the network, and the column 'Packets' specifies the number of data packets to be sent. The column 'Nodes' gives the number of nodes in the ordinary state space, and the column 'Arcs' column gives the number of arcs. The column 'Sweep-line peak' gives the peak number of nodes stored simultaneously in main memory during state space exploration using the sweep-line method. The column 'Node ratio' specifies the number of nodes in the state space divided by the peak number of nodes stored when the sweep-line method is used, and hence specifies the reduction in memory usage obtained using the sweep-line method. There is no reduction factor for arcs, since the sweep-line method does not store any arcs during exploration of the state space. The column 'Time ratio' specifies the CPU time used to generate the state space using ordinary state space exploration divided by the CPU time used to explore the state space using the sweep-line method, and hence specifies the reduction in time usage obtained using the sweep-line method. The time ratio of 1.0 for the first three configurations (the three smallest) is due to the fact that the CPU times become identical for the two methods when rounded to two digits.

Table 8.1 Statistics for application of the sweep-line method

		State space		Sweep-line	Node	Time
Limit	Packets	Nodes	Arcs	peak	ratio	ratio
1	4	33	44	33	1.00	1.00
2	4	293	764	134	2.19	1.00
3	4	1 829	6 860	758	2.41	1.00
4	4	9 025	43 124	4 449	2.03	1.78
5	4	37 477	213 902	20 826	1.80	1.65
6	4	136 107	891 830	82 586	1.65	1.51
4	5	20 016	99 355	8 521	2.35	1.95
4	6	38 885	198 150	14 545	2.67	2.19
4	7	68 720	356 965	22 905	3.00	2.27
4	8	113 121	596 264	33 985	3.33	2.41

It can be seen that the sweep-line method yields a reduction in both space and time. The former is expected, since markings are deleted during state space exploration. The saving in time is because the deletion of states implies that there are fewer markings to be compared during state space exploration when determining whether a marking has already been encountered.

In the above, we have introduced the basic sweep-line method, which relies on the use of a *monotonic progress measure*, i.e., a progress measure for which the successor markings of a given marking M have progress values which are always greater than or equal to the progress value of M. This property ensures that it is safe to delete markings and that the sweep-line method terminates after all reachable markings have been processed once. A generalised version of the sweep-line method [70] also exists, which can deal with progress measures that are not monotonic. The basic idea of the generalised sweep-line method is to perform multiple sweeps of the state space and to make certain markings persistent, which means that they cannot be deleted from memory. In addition, the basic sweep-line method has been extended with respect to the properties that can be verified with it [81]. It has also been extended [72] to use external storage such that counterexamples and diagnostic information can be obtained, which is not possible with the basic method, since it deletes the markings from memory.

For timed CP-nets the global clock can be used as a progress measure. We shall give further examples of the use of the basic and generalised sweep-line methods in Sect. 14.3.

8.3 Symmetry Method

Many concurrent systems possess a certain degree of symmetry. For example, many concurrent systems are composed of similar components whose identities are interchangeable from the point of view of verification. This symmetry is also reflected in the state spaces of such systems. The basic idea in the symmetry method [23, 39, 57, 62] is to represent symmetric markings and symmetric binding elements using *equivalence classes*. State spaces can be condensed by factoring out this symmetry, and the symmetry-condensed state space is typically orders of magnitude smaller than the full state space. A symmetry-condensed state space can be constructed directly without first constructing the full state space and then grouping nodes and arcs into equivalence classes. Furthermore, behavioural properties can be verified directly using symmetry-condensed state spaces without unfolding to the full state space. Below we explain the use of the symmetry method using a variant of the hierarchical CPN model of the protocol system with multiple receivers described in Sect. 5.4. Figure 8.3 shows the Protocol module of the CPN model. Compared with the CPN model presented in Sect. 5.4, we have added a place Limit to obtain a finite state space in a way similar to what was done in Sect. 7.1.

It can be observed that the receivers in the protocol system are symmetric, in the sense that they behave in the same way. They are distinguishable only by their

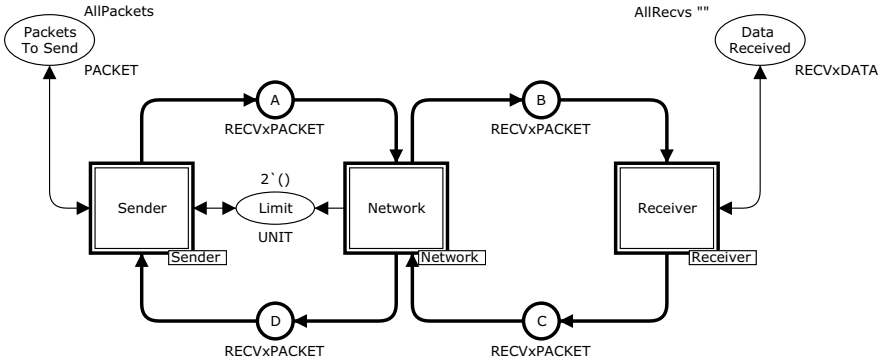

Fig. 8.3 Protocol module for protocol, used to illustrate the symmetry method

identities. This symmetry is also reflected in the state space. Figure 8.4 shows an initial fragment of the state space for the CPN model shown in Fig. 8.3, with two receivers. We have used the same notation in the arc labels as in Chap. 7 and have additionally appended the identity of the receiver which a given binding element corresponds to. As an example, the arc label TP1+ (Recv(2)) on the arc from node

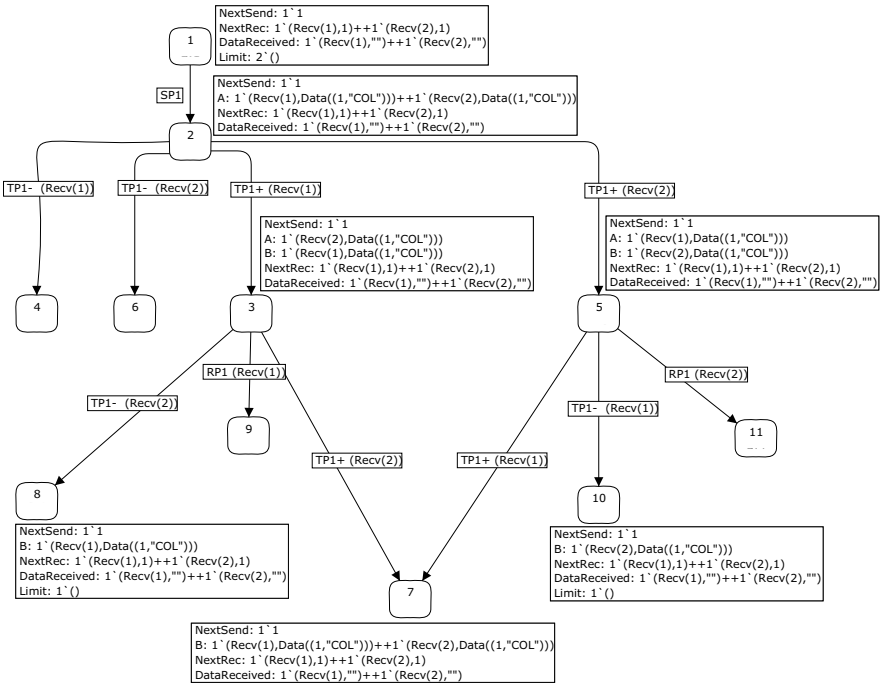

Fig. 8.4 Initial fragment of state space

2 to node 5 specifies that this arc represents an occurrence of TransmitPacket with a binding corresponding to a successful transmission of data packet 1, intended for receiver 2.

Consider now the two nodes 3 and 5, which correspond to markings in which exactly one of the two data packets has been transmitted successfully across the network. These two markings are symmetric in the sense that the marking represented by node 3 can be obtained from the marking represented by node 5 by swapping the identities of receivers 1 and 2. Similarly, the two markings represented by nodes 4 and 6 can be obtained from each other by interchanging the identities of the receivers. Nodes 4 and 6 correspond to markings in which one of the two data packets was lost on the network. It can also be observed that two symmetric markings such as marking 3 and marking 5 have symmetric sets of enabled binding elements, and symmetric sets of successor markings. As an example, markings 7, 8, and 9, which are successor markings of marking 3, are symmetric to markings 7, 10, and 11, respectively, which are successor markings of marking 5. This property can be extended to finite and infinite occurrence sequences, i.e., for any occurrence sequence starting in a marking M and for all markings M' symmetric with M, there exists a symmetric occurrence sequence starting in M'.

Figure 8.5 shows an initial fragment of the symmetry-condensed state space for the protocol system obtained by considering two markings equivalent if one of them can be obtained from the other by a permutation of the identities of the receivers. The nodes and arcs now represent equivalence classes of markings and binding elements, respectively. The equivalence class of markings represented by a node is listed in curly brackets in the inscription of the node; for example, node 3 represents nodes 3 and 5 in Fig. 8.4. A similar notation is used for binding elements.

Symmetry-condensed state spaces can be represented by storing a representative marking (or binding element) for each equivalence class of markings (or binding elements), and the calculation of the condensed state space is typically based on

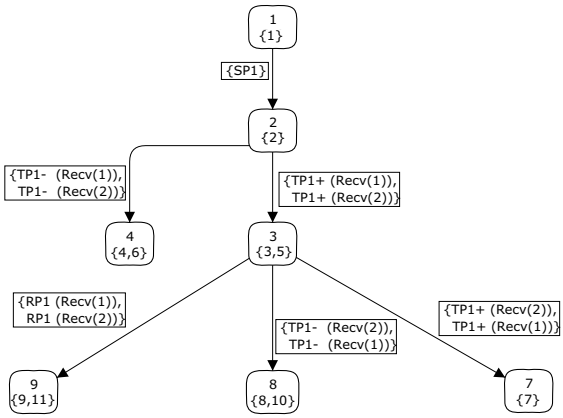

Fig. 8.5 Initial fragment of symmetry-condensed state space

calculating canonical representatives of each equivalence class [79]. This means that whenever a new marking is generated during state space exploration, this marking is transformed into a canonical representative of its equivalence class. It is then checked whether this canonical representative is already included in the state space. A similar processing is done for binding elements.

The symmetries used to reduce the state space are required to be symmetries actually present in the CPN model. This amounts to checking the following three properties:

- All initial marking inscriptions must be symmetric, i.e., applying a permutation to the initial marking does not change the initial marking.
- All guard expressions must be symmetric, i.e., evaluating the guard in a binding must give the same result as first permuting the binding and then evaluating the guard.
- All arc expressions must be symmetric, i.e., evaluating the arc expression in a binding and then applying a permutation must give the same result as first permuting the binding and then evaluating the arc expression.

These requirements can be checked prior to state space exploration by examining the inscriptions of the CPN model one at a time. This means that they can be checked statically without considering occurrence sequences. The specification of the symmetries themselves is done by associating an algebraic group of permutations with each atomic colour set of the CPN model. The atomic colour sets are those colour sets defined without reference to other colour sets, i.e., those colour sets defined without using colour set constructors such as unions, products, and records. Examples of permutation groups are the set of all permutations of colours in the colour set, the set of all rotations for an ordered colour set, and the set that consists of just the identity element, i.e., the group that allows no permutation of the colours in the colour set. For the present protocol system, we associate the set of all permutations with the atomic colour set `RECV`, which models the identities of receivers. For the other atomic colour sets (such as `NO`, modelling the sequence numbers), we assign the trivial permutation group consisting of the identity element only, since we do not allow permutation of colours in these colour sets.

Table 8.2 shows some statistics obtained using the symmetry method on the protocol system for different configurations. The column L lists the capacity of the network, the column P lists the number of data packets, and the column R lists the number of receivers in the configuration. The table gives the numbers of nodes and arcs in the full and in the symmetry-condensed state spaces. The columns 'Node ratio', 'Arc ratio', and 'Time ratio' give the reduction obtained in terms of nodes, arcs, and CPU time, respectively. The column $R!$ lists the factorial of the number of receivers in the configuration. When there are R receivers in the configuration, there are $R!$ possible permutations of the receivers. Hence, $R!$ is the theoretical upper limit on the reduction factor that can be obtained for a configuration with R receivers. For the configurations with five and six receivers, it was not possible to compute the full state space; its size was instead computed by computing the sizes of the equivalence

Table 8.2 Statistics for application of the symmetry method

			State space		Symmetry		Node	Arc	Time	
L	*P*	*R*	Nodes	Arcs	Nodes	Arcs	ratio	ratio	ratio	*R*!
2	3	2	921	1 832	477	924	1.93	1.98	0.7	2
3	3	3	22 371	64 684	4 195	11 280	5.33	5.73	2.0	6
4	3	4	172 581	671 948	9 888	32 963	17.45	20.38	23.9	24
5	2	5	486 767	2 392 458	8 387	31 110	58.04	76.90	–	120
6	2	6	5 917 145	35 068 448	24 122	101 240	245.30	346.39	–	720

classes represented by the nodes and arcs in the condensed state space. This is why no time reduction ratio has been provided for these configurations.

The advantage of the symmetry method is that significant reductions can be obtained, as was illustrated above, and that the method can be used to check all behavioural properties that are invariant under symmetry. The main limitation of the symmetry method is that computing the canonical representations of markings and binding elements is computationally expensive. It has been shown [20] that computing canonical representatives for equivalence classes is at least as hard as the *graph isomorphism problem*, for which no polynomial-time algorithm is known. The current available algorithms for computing canonical representatives, which exploit a number of advanced algebraic techniques, can, however, in practice deal efficiently with systems where the number of permutation symmetries is below 10! [79].

8.4 Equivalence Method

The symmetry method presented in the previous section is based on symmetries in the system inducing an equivalence relation on the markings and binding elements. The *equivalence method* is a generalisation of the symmetry method, where the constraint that the equivalence relations are induced by symmetries is removed. Instead, arbitrary equivalence relations on the markings and binding elements can be used provided that they are *consistent*, i.e., equivalent markings must have equivalent sets of enabled binding elements and equivalent sets of successor markings. Below, we illustrate the use of the equivalence method using the CPN model from Fig. 8.1.

The equivalence relation for this protocol system is based on the observation that certain packets on the network become similar (equivalent) as the protocol executes. As an example, consider the marking M_1 in Fig. 8.6 and the arrival of the retransmitted data packet with sequence number 2 at the receiver. The arrival of this data packet does not change the state of the receiver. The sequence number is smaller than the expected number, and the data packet is said to be *old*. The arrival of this old data packet has the effect that an acknowledgement asking for data packet number 3 is sent. Generalising this, the arrival of any old data packet (with a sequence number

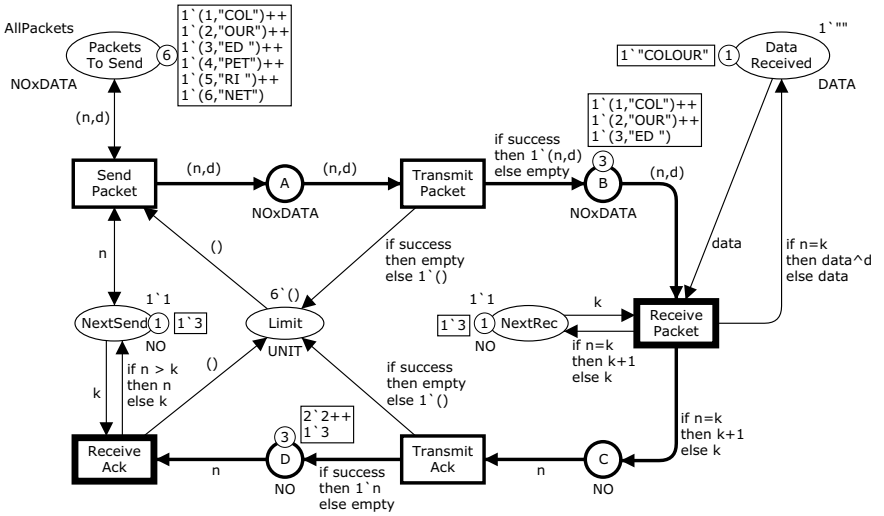

Fig. 8.6 Example marking M_1 for the equivalence method

less than the one expected) has the effect that an acknowledgement is sent indicating which packet is actually expected. For instance, reception of an old data packet with sequence number 1 on place A has exactly the same effect as the reception of an old data packet with sequence number 2. Similar observations and terminology apply to acknowledgements arriving at the sender, as we have changed the CPN model such that the sender never decreases its sequence number when an old acknowledgement is received.

The intuition behind the equivalence relation for the transport protocol is that it captures that old data packets are equivalent, and, similarly, old acknowledgements. The equivalence relation considers two markings to be equivalent if three conditions hold:

- The markings of the places PacketsToSend, NextSend, Limit, NextRec, and DataReceived must be identical.
- The markings of the network places A, B, C, and D must be identical when only non-old packets are considered.
- The markings of the network places A, B, C, and D must have the same numbers of old packets.

As an example, the marking M_1 shown in Fig. 8.6 is equivalent to the marking M_2 shown in Fig. 8.7. The markings of places B and D are different in the two markings, but there is the same number of old data packets on B and the same number of old acknowledgements on D in the two markings. Furthermore, the non-old data packets and non-old acknowledgements on B and D are identical in the two markings.

Two bindings of a transition are considered equivalent if they both involve old data packets (or old acknowledgements), whereas bindings involving non-old data

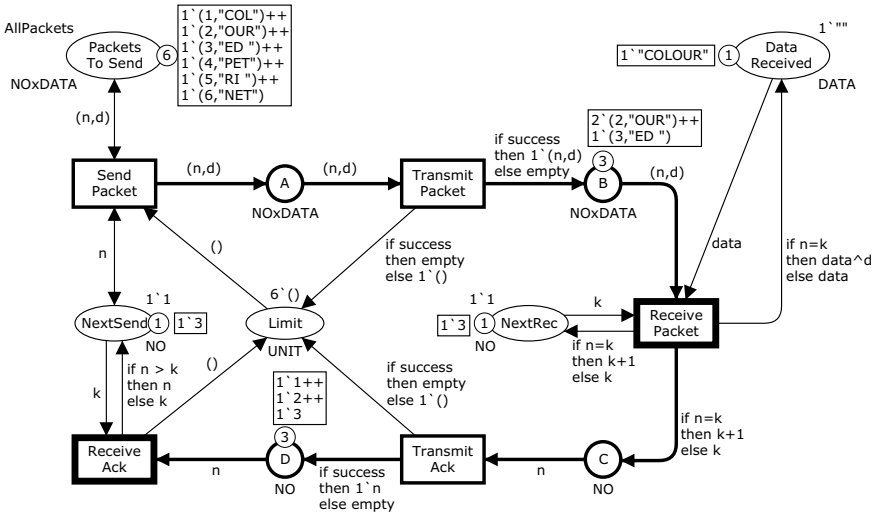

Fig. 8.7 Example marking M_2 for the equivalence method

packets (or acknowledgements) are equivalent only to themselves. The specification of the equivalence relations is provided to CPN Tools by implementing two predicates in CPN ML. The predicate on markings takes two markings and returns true if and only if the two markings are equivalent. Analogously, the predicate on binding elements takes two binding elements and returns true if and only if the two binding elements are equivalent. A formalisation of the above equivalence relation can be found in [65], including a proof that it is consistent. Table 8.3 gives some statistics obtained using the equivalence method on the protocol system for several different limits on the network.

The advantage of the equivalence method is that it allows a more general notion of equivalence compared with the symmetry method. The disadvantage is that the proof of consistency of the equivalence relation must be done manually, and this can be a difficult task for complex CPN models, Furthermore, it is often cumbersome to write the CPN ML code specifying the two equivalence relations. These are the

Table 8.3 Statistics for application of the equivalence method

		State space		Equivalence		Node	Arc	Time
Limit	Packets	Nodes	Arcs	Nodes	Arcs	ratio	ratio	ratio
1	4	33	44	33	44	1.00	1.00	1.00
2	4	293	764	155	383	1.89	1.99	1.00
3	4	1 829	6 860	492	1 632	3.72	4.20	0.90
4	4	9 025	43 124	1 260	5 019	7.16	8.59	1.56
5	4	37 477	213 902	2 803	12 685	13.37	16.86	4.09
6	4	136 107	891 830	5 635	28 044	24.15	31.80	13.58

main reasons why the equivalence method in its full generality has not been used very much in practice.

Section 10.4 presents a special version of the equivalence method that can be used to obtain a finite state space for any timed CPN model where the underlying untimed CPN model has a finite state space. This special version is fully automatic to use, as it requires no manual consistency proof and manual implementation of the equivalence predicates.