Efficient Performance Analysis Techniques for Stochastic Well-Formed Nets and Stochastic Process Algebras

G. Franceschinis and M. Ribaudo

Dipartimento di Informatica
Università di Torino, Torino, Italy
Phone: +39-11-7429111, Telefax: +39-11-751603
E-mail: giuliana@di.unito.it, marina@di.unito.it

Abstract. *Stochastic Well Formed Nets and Stochastic Process Algebras are high level description languages for the specification and the performance evaluation of concurrent systems. In both formalisms the performance analysis of the modelled system can be performed by generating a continuous time Markov chain of the size of the model state space: this often leads to the so called state space explosion problem which can prevent the possibility of completing the desired analysis. In this chapter we will present two state space aggregation techniques, each working on one of the two formalisms, allowing efficient performance analysis. The advantages and disadvantages of the two techniques will be discussed and compared.*

Keywords: High Level Petri Nets, Stochastic Petri Nets, Stochastic Process Algebras, Performance Evaluation, Markov Chains, Lumpability.

1 Introduction

Stochastic Petri Nets (SPNs) are a timed flavour of Petri nets that have been used for the modelling and performance evaluation of concurrent system since the early eighties. The performance analysis methods available for SPNs are the Montecarlo simulation, and the solution of the stochastic process (continuous time Markov chain - CTMC) that can be derived from the SPN reachability graph (see Chapter [2]). As the (untimed) Petri net formalism evolved into a number of high level variants (Predicate/Transition Nets, Coloured Petri Nets, etc.), also SPNs have been enhanced to provide a more powerful modelling formalism.

The introduction of High Level (stochastic) Petri Net (HLPN) formalisms, on one hand has increased the appealing of this modelling language for the study of realistic systems, on the other hand it has exacerbated the so called state space explosion problem. Since the introduction of the HLPN formalisms, several research efforts have been devoted to the development of new analysis techniques capable of exploiting the possibility of parameterisation offered by these modelling languages, and in particular capable of drawing on the presence

of behavioural symmetries in the model [35, 42]. The exploitation of symmetries at the reachability graph construction level is often an effective method to cope with the state space explosion problem. Intuitively, markings that lead to similar future behaviour can be grouped into equivalence classes, and the system properties can thus be studied on a subset of reachable markings (one representative for each equivalence class).

In the first part of this chapter we describe a method for the construction of reduced reachability graphs (called Symbolic Reachability Graph - SRG) of Stochastic Well-formed Nets (SWNs), a high level SPN formalism in which behavioural symmetries can be automatically discovered and exploited through the concept of *symbolic marking*. The symbolic marking idea has been first introduced in [26] for the class of Regular Nets (RN), and was later extended to the less restrictive formalism of Well-formed Nets (WN) [10]. The basic idea behind these two classes of HLPN is to naturally lead the modeller to describe the system in such a way that the behavioural symmetries are implicitly expressed in the model and can thus be automatically exploited: this is achieved through a carefully chosen syntax for the definition of the places colour domains and the arc expressions. The other interesting idea which distinguishes the approach presented in this chapter from other similar approaches (e.g., the OS-graph presented in [38]) is that equivalence classes of markings are represented by a unique, symbolic representation, the *symbolic marking*. Moreover a symbolic firing rule is defined that allows to directly obtain the set of symbolic markings reachable from a given symbolic marking without ever generating the *ordinary* markings belonging to them. Finally, the chosen symbolic marking representation and the symbolic firing definition allow to group several transition firings into a single symbolic firing [13], hence achieving a reduction also in the number of RG arcs. Most interesting (qualitative) properties that can be proved on the complete RG, can be also proved on the SRG.

When RN and WN are extended to include also timing information (we are assuming here a definition of timing similar to that of Generalized Stochastic Petri Nets - GSPN - presented in Chapter [2]), the question that naturally arises is: can we exploit the reduction achieved on the RG also for performance analysis purposes? More precisely, can we aggregate the CTMC isomorphic to the RG and still obtain the same performance figures from it? Some preliminary ideas in this direction were presented in [9, 41, 51]. The stochastic extension of RN (without immediate transitions) was presented in [20]: in this paper it was shown that the CTMC associated with the RG of a RN model, is lumpable with respect to the aggregation induced by the SRG algorithm, moreover the lumped Markov chain can be directly derived from the SRG, and used to compute any performance measure that could be computed on the (usually much) larger CTMC. This result has been later shown to hold also for the SWN formalism, WNs extended with exponentially distributed stochastic timing and immediate transitions [12, 39], and will be presented in Section 3.4. A similar technique has been developed at the same time for the formalism of Stochastic Activity Networks [49].

In [15, 46] it has been shown that the use of symbolic marking and symbolic

firing can also improve the efficiency of event driven simulation of SWN models: this is due to the grouping of ordinary firings into symbolic firings that reduces the cost of event list management.

Tool support for the computation of the SRG and the lumped Markov chain of an SWN model, and for efficient simulation of SWN models through the use of the symbolic marking representation is available within the GreatSPN tool [1, 17] (versions 1.7 and 2.x).

In the second part of this chapter we will present another approach to efficient performance analysis based on a related formalism more recently developed: Stochastic Process Algebras (SPA) [5, 8, 29, 33], a timing extension of Process Algebras [34, 43] introduced for modelling and performance analysis of concurrent systems. These are abstract languages for the description of concurrent systems whose peculiar feature is compositionality: complex systems are built composing smaller subsystems by means of language operators. Like in SPNs, from the transition diagram underlying an SPA model it is possible to derive a CTMC, the starting point for the computation of performance figures.

SPA inherited from standard process algebras the compositional technique for model construction and also the possibility of exploiting equivalences between models for model verification and simplification, features which are not present in the SWN formalism. Moreover, equivalence notions have been extended to take into account also timing information [5, 29, 32] (in pure process algebras only functional aspects are considered). This extension has been proved to be very powerful for achieving aggregation, both at the state space level and at the Markov chain level. Like in the case of the SWN formalism, states which lead to the same future behaviour are grouped into equivalence classes which form the states of an aggregated state space. In this formalism, however, equivalence classes are computed a posteriori looking for equivalences within the state space of the model. Indeed it is possible to improve the performance of the generation of an aggregated state space by exploiting the compositional technique to model construction provided by the formalism, and by computing and aggregating partial state spaces underlying model subcomponents.

Compared to SPNs and their extensions, SPAs are a rather novel research topic however some tools [24, 30] already exist for the specification of SPA models, the generation of their state spaces, the computation of equivalence classes of states, the calculation of performance measures based on the underlying CTMCs.

For making more concrete the description of the SWN and the SPA formalisms and the corresponding efficient analysis techniques, throughout this chapter we will use a common example of a simple communication protocol, the same example already used in Chapter [37] for introducing Coloured Petri Nets.

The Chapter is organised as follows: in Section 2 SWNs are introduced by means of the protocol example, in Section 3 the state space aggregation and the CTMC lumping techniques based on the SRG are presented. In Section 4 SPAs are introduced by means of the same example of Section 2, and in Section 5 the corresponding aggregation techniques are illustrated showing that SPAs allow to find some aggregations that cannot be exploited by the SRG. Finally, in

Section 6 we indicate some new results that extend the work presented in the chapter, indicating possible future evolution of the research on this topic.

2 Stochastic Well-Formed Nets

In this section we define the SWN formalism. We first introduce SWNs informally through an example (we have chosen the same running example of Chapter [37] on Coloured Petri Nets - CPN), then we give its formal definition.

Let us recall that the extension of the PN formalism obtained by introducing *colours* has been motivated by the need of representing complex systems with compact models. Systems are made up of several components, and it often happens that several components in a system have the same characteristics: for example in a network of workstations we may have several workstations of the same type. A PN model of a system can be obtained by composing the models of the subsystems, and in case several copies of similar subsystems are present, several copies of the same submodel will also appear in the PN model. In this situation it is very convenient to include a single submodel of each type and then use distinguishable tokens and distinguishable transition firings to be able to maintain the same information; this approach has the further advantage of making the complete model parametric in the number of submodels of each type.

For example, in a model of a network of workstations, we may have the submodel representing the behaviour of a given workstation type: the possible states of the workstation are represented by some configuration of tokens in the places of the submodel. The states of different workstations of the same type can be embedded into a single submodel by assigning a colour (i.e., a unique identifier) to each workstation and representing the state of a given workstation through a configuration of tokens of the corresponding colour in the places of the submodel. A state change of a given workstation shall correspond to the firing of a transition in the submodel acting only on the tokens of proper colour: this leads to the definition of *transition instance* as a pair transition name, colour of tokens involved in the firing; functions associated with the arcs relate the colours associated with a transition instance and the colours of the tokens withdrawn form the transition input places and put into the transition output places.

The idea of *folding* outlined above can be extended further and in fact, using coloured tokens, coloured transition instances, and assigning functions to arcs that relate transition colours and place colours, it is possible to fold a given PN model in several different ways, up to the extreme case in which all places are folded into a single place, all transitions are folded into a single transition, and all the information about the model behaviour is embedded into the (very complex) function associated with the arcs connecting the only two nodes in the model.

Although there is no general rule allowing to decide which is the best *degree of compactness* of an high level PN model, in our opinion it is important to find a good trade off between the net size and the amount of information on the model behaviour that is conveyed by its graphical structure. The SWN formalism

presented in this section, naturally leads to the definition of models with rather simple arc functions, hence discouraging the modeller from hiding in the arc functions relevant aspects of the model behaviour.

2.1 A simple protocol example

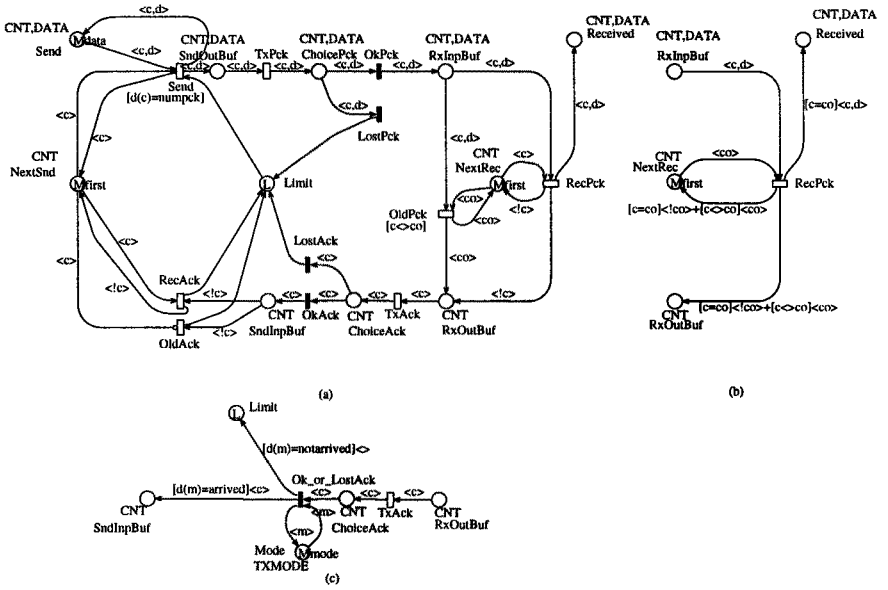

Fig. 1. SWN model of a communication protocol with one sender and one receiver.

Let us consider the following simple communication protocol involving one sender and one receiver exchanging messages over a net that may lose packets. The sender must send a pool of packets to the receiver, and the packets have to be received in a predefined order, hence they are numbered. The sender starts sending the first packet, and it keeps resending the same packet until it receives an acknowledge message from the receiver, saying that it is ready to receive the second packet; then the same procedure is repeated for each subsequent packet until every packet in the pool has been sent.

The receiver has an internal counter that keeps track of the next packet to be received: initially it is set to one. Each time the receiver gets a new packet from the network, if its number is equal to the internal counter value, then the packet is stored and the internal counter is increased by one, otherwise the packet is discarded; in any case the receiver sends back an acknowledge message to the sender containing the current value of the internal counter.

A first SWN model of this simple protocol is depicted in Fig. 1 (see the corresponding CPN model in Fig. 1 of Chapter [37]). An SWN model definition comprises a number of finite, non empty sets called *basic colour classes*, a set of places, each with an associated *colour domain*, a set of transitions, each with an associated guard, a set of (input, output or inhibitor) arcs connecting places and transitions, each with an associated expression used to define the enabling condition and state change of a *transition instance*. In addition, each transition has a priority function and a weight/rate function, whose semantics corresponds to that defined for GSPNs [2]. Transitions with priority 0 are *timed* and represent activities that take time, transitions with priority grater than zero are *immediate* and represent logic actions that take no time. Rates are assigned to timed transitions, while weights are assigned to immediate transitions. The rates are used to determine the duration of timed transition firings, while weights are used to perform probabilistic conflict resolution among immediate transitions that fire in zero time.

In our example, there are two basic colour classes, CNT and DATA representing the possible values of a packet counter and the actual data to be transmitted respectively. The class CNT is *ordered*, meaning that a successor function is defined on it, while class DATA is not. Basic colour classes may be partitioned into several *static subclasses*: in this example, CNT is composed of two static subclasses, numpck $= \{0, \ldots, max\text{-}1\}$, last $= \{max\}$ where max is the total number of packets that must be sent to the receiver. The successor function (denoted !) is defined as follows: $\forall c \in \mathrm{CNT}, !c = (c + 1) \bmod max + 1$. Class DATA instead contains a single static subclass data $= \{data_0, \ldots, data_{max-1}\}$.

The model in Fig. 1(a) is composed of three submodels: the sender submodel comprising places Send, NextSnd, SndOutBuf, and SndInpBuf, and transitions Send, RecAck, and OldAck, the receiver submodel comprising places Received, NextRec, RxInpBuf, and RxOutBuf, and transitions RecPck and OldPck, finally the network submodel comprising places Limit, SndOutBuf, SndInpBuf, RxInpBuf, RxOutBuf, ChoicePck, and ChoiceAck, and transitions TxPck, OkPck, LostPck, TxAck, OkAck, and LostAck; the complete model can be obtained by *fusion* of places with same name. Observe that the SWN formalism does not allow to define a model in a compositional way, the reason for describing the protocol model in terms of submodels is just to make it easier to understand, and to make it comparable with the corresponding model in Chapter [37].

Place Send represents the complete set of packets to be transmitted, place NextSnd represents the next packet to be sent, place SndOutBuf represents the output buffer of the sender, while place SndInpBuf represents the sender input buffer; place Received represents the set of packets already received, place NextRec represents the next packet expected by the receiver, place RxInpBuf and RxOutBuf represent the receiver input and output buffers respectively; place Limit represents the network capacity, i.e. the overall number of packets (either data or acknowledges) that can be simultaneously circulating in the network. Transition Send represents the network accepting a message from the sender, transitions RecAck and OldAck represent the reception of a new and old acknowledge packet

respectively (an acknowledge packet is old if the same acknowledge has been already received by the sender before). Transitions RecPck and OldPck represent the reception of a new and an old data packet respectively, transition TxPck represents the transportation of a data packet from the sender to the receiver, while transitions OkPck and LostPck represent the delivery of a data packet into the receiver input buffer and a data packet loss respectively, transitions TxAck, OkAck, and LostAck have similar meaning, but refer to acknowledge messages.

Place colour domains are defined as Cartesian product of basic colour classes, possibly with repetitions of the same class. For example the colour domain of place Send is CNT × DATA (in the picture of Fig. 1, printed out from the GreatSPN package, it is indicated as CNT,DATA next to the place name) meaning that the tokens contained into place Send are pairs $\langle sequence\ number, data \rangle$ modelling a packet to be sent. Place RxOutBuf instead has colour domain CNT, meaning that the tokens it may contain are identified by a number between 0 and max representing acknowledge messages carrying the sequence number of the next packet expected by the receiver.

A marking m of an SWN, is a place-indexed vector, with[1] $\forall p \in P, m[p] \in Bag(cdom(p))$. The initial marking is defined as follows:

$$m_0[\mathsf{Send}] = \mathsf{Mdata}, \text{ where } \mathsf{Mdata} = \{\langle i, data_i \rangle, i \in 0, \ldots, max - 1\}$$
$$m0[\mathsf{NextSnd}] = m0[\mathsf{NextRec}] = \mathsf{Mfirst}, \text{ where } \mathsf{Mfirst} = \{\langle 0 \rangle\}$$
$$m0[\mathsf{Limit}] = L \text{ where } L \in \mathbb{N}^+; \text{ all other places are initially empty.}$$

Observe that in this model the colours are used to represent data associated with tokens, hence a token can represent complex objects (e.g., messages); moreover we can observe that colours have been used to *fold* several similar subnets, each representing the possible evolution of a specific packet in the system.

Transitions in SWN models, as in CPN models, have an associated set of variables, each with a type among the basic colour classes. Intuitively this means that a transition represents a set of similar events, and the variables are event parameters: by assigning actual colour objects to the variables we can identify a specific event in the set. In SWN terminology, an assignment of values to the variables of a transition defines a *transition instance*. There is an analogy between transition variables and formal parameters of procedures: to actually execute a procedure (to fire a transition) we have to instantiate all its formal parameters with actual values (we have to assign colour objects to the variables).

For example the variables of transition OldPck are $c \in$ CNT, $co \in$ CNT, and $d \in$ DATA. Transition OldPck represents all possible events of type *reception of an old packet*; a specific event of this type is characterised by the number of the old data packet being received (variable c), the actual data contained in the packet (variable d), and the current value of the receiver's packet counter (variable co). The assignment $c \leftarrow 1$, $co \leftarrow 2, d \leftarrow data_1$ defines a possible

[1] Given a set A, $Bag(A)$ denotes the set of all possible *multisets* on A; a multiset is a set which may contain multiple copies of the same element. The colour domain of a place p is denoted $cdom(p)$.

instance of OldPck corresponding to the event: the receiver gets an old packet number 1 containing the data $data_1$, while its internal packet counter is 2.

Transitions have an associated *guard* or *predicate* (by default it is *true*) and only the instances that satisfy the guard may be enabled. For example the above instance of OldPck satisfies the guard $[c <> co]$ while the instance $c \leftarrow 1,\ co \leftarrow 1, d \leftarrow data_1$ does not. This is consistent with the meaning of variables c and co: a packet is not old if its number is equal to the internal counter of the receiver. Another example of transition guard is $[d(c) = numpck]$ associated with transition Send: the meaning is that only instances of Send assigning a number in static subclass numpck to variable c can be enabled (the notation $d(c)$ is used within GreatSPN to denote the static subclass c belongs to).

The arc expressions in our running example are tuples of variables and of variable successors. In general an arc expression in SWNs is a weighted sum of (guarded) tuples, and each element in a tuple is in turn a weighted sum of variables (all of the same type), of the corresponding successors, and of constants denoting the set of all objects in a given static subclass. Given an assignment of values to variables, an expression tuple can be evaluated by first evaluating the expression contained in each element, and then composing the results through the Cartesian product operator. In general the evaluation of an arc expression gives a multiset of tuples: in our simple example all arc expressions evaluate to cardinality one multisets. For example, given the assignment $c \leftarrow 1, d \leftarrow data_1$, expression $\langle !c \rangle$ evaluates to $\{\langle 2 \rangle\}$, expression $\langle c, d \rangle$ evaluates to $\{\langle 1, data_1 \rangle\}$, expression $\langle !c, d \rangle$ evaluates to $\{\langle 2, data_1 \rangle\}$, expression $2\langle c, d \rangle + \langle !c, d \rangle = \langle 2c + !c, d \rangle$ evaluates to $\{2\langle 1, data_1 \rangle, \langle 2, data_1 \rangle\}$, and expression[2] $\langle c, S_{data} \rangle$ evaluates to $\{\langle 1, data_1 \rangle, \langle 1, data_2 \rangle, \ldots, \langle 1, data_{max} \rangle\}$. A more complex example of arc expression is $[co = c]\langle !co \rangle + [co <> c]\langle co \rangle$ where the tuples are guarded. A guarded tuple evaluates to the empty set if the associated guard is not satisfied, if instead the guard is satisfied it gets the value of the tuple following the guard. Hence the expression $[co = c]\langle !co \rangle + [co <> c]\langle co \rangle$ evaluates to $\langle 3 \rangle$ for the assignment $c \leftarrow 2, co \leftarrow 2$, while it evaluates to $\langle 2 \rangle$ for the assignment $c \leftarrow 1, co \leftarrow 2$. Observe that when a given transition instance (i.e., a given assignment) makes an arc expression evaluate to the empty set, this is equivalent to canceling that arc for that transition instance.

The use of arc expressions with guards allows to construct more compact models, however it decreases the amount of graphically conveyed information: for example in Fig. 1(b) it is shown an alternative representation of the receiver part with one transition less. In this version of the model, RecPck represents both cases of reception of a packet, to be stored or to be discarded (because it is out of sequence). In the model of Fig. 1(a) it is graphically evident that packets may be discarded, while in the variant of Fig. 1(b) this information is hidden in the arc expression from transition RecPck to place Received. This is a typical example of different possible choices in the degree of compactness of a coloured model.

[2] $S_{stat\_subcl\_name}$ is a constant function returning the set of elements in static subclass *stat_subcl_name*.

Another similar example is that of transitions OkAck and LostAck that could be merged as shown in Fig. 1(c), but in this case it would be less evident that acknowledge messages may get lost. Let us comment briefly Fig. 1(c): a new class TXMODE=arrived ∪ notarrived with arrived = $\{ok\}$, notarrived = $\{lost\}$ has been defined and a variable $m \in$TXMODE is used to decide whether a given instance of transition Ok_or_LostAck represents a delivery of acknowledge message or an acknowledge message loss. The initial marking of place Mode is the whole set TXMODE. Note that this place plays a role similar to that of place SA in the CPN of Fig. 3 in [37], however in our model it is not used to influence the probability of firing Ok_or_LostAck in either mode, since in SWNs the firing probability can be defined by assigning suitable *weights* to transitions.

In the initial marking, only the transition instance Send($c \leftarrow 0, d \leftarrow data_0$) is enabled due to the presence of token $\langle 0, data_0 \rangle$ in place Send, a token $\langle 0 \rangle$ in place NextSnd and some neutral token in place Limit. Its firing puts a new token of colour $\langle 0, data_0 \rangle$ into place SndOutBuf and withdraws one neutral token from place Limit; let us call this new marking m_1. Marking m_1 enables two transition instances: Send($c \leftarrow 0, d \leftarrow data_0$) and TxPck($c \leftarrow 0, d \leftarrow data_0$); both m_0 and m_1 are *tangible* markings, i.e. markings in which the model spends some time, in fact they enable only timed transitions, whose firing requires some time to be completed. By firing the TxPck instance, the token $\langle 0, data_0 \rangle$ is withdrawn from place SndOutBuf and a token of the same colour is put into place ChoicePck: the new reached marking, m_2, is *vanishing* since it enables two immediate transitions, namely OkPck and LostPck, and the model always passes through it without spending time. Observe that in m_2, also timed transition Send has enough tokens in its input places to be enabled, however immediate transitions have higher priority over timed ones.

Let us now discuss the definition of the timing and probabilistic conflict resolution of the model. All the activities of the system that take time are modeled as timed transitions (graphically represented as white boxes). Immediate transitions (graphically represented as thin bars) are instead used to model logical actions in the systems that take no time, or activities that take a negligible amount of time. In the model of Fig. 1(a) all transitions are timed, except OkPck, LostPck, OkAck, LostAck, that represent logical actions, i.e., the (random) choice of whether the network has correctly delivered or lost a message. In order to include in the model the information that messages get lost with probability 20/100 while they are correctly delivered with probability 80/100, weights 0.2 and 0.8 must be assigned to transitions LostPck and OkPck respectively (the same applies to LostAck and OkAck). Observe that in this example all instances of LostPck have the same weight 0.2, and all instances of OkPck have the same weight 0.8; in the example of Fig. 1(c) where transitions LostPck and OkPck are represented by a single transition Ok_or_LostPck, we should have assigned weight 0.2 to the transition instances with $m \in$ notarrived and weight 0.8 to the transition instances with $m \in$ arrived. As pointed out in [37], it might be useful to introduce in this model a delay between subsequent retransmissions of the same packet: to model this feature we need to refine place Send into a subnet

place-timed transition-place as shown in Fig. 2, and associate the required delay with transition wait.

To express the durations of the activities represented by timed transitions, the modeller has to assign a *delay* to each timed transition instance. Transition delays in SWNs are actually negative exponentially distributed random variables (see Chapter [2]), and the modeller provides the *average* delay (in the GreatSPN package the modeller actually provides the average transition rate, which is the parameter characterising the negative exponential distribution, and is the inverse of the average delay). It is often the case that all instances of a given transition in an SWN model have the same associated delay: in our case this is true for all timed transitions in the model. For example, the delay associated with transition Send represents the (average) time spent by the sender to prepare a message and make it available in its output buffer, and it is the same for all possible messages that the sender may want to send.

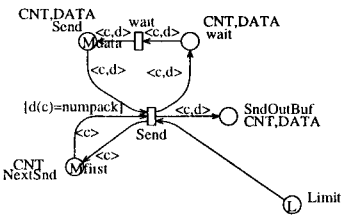

Fig. 2. Refinement of place Send to introduce a delay between subsequent retransmissions.

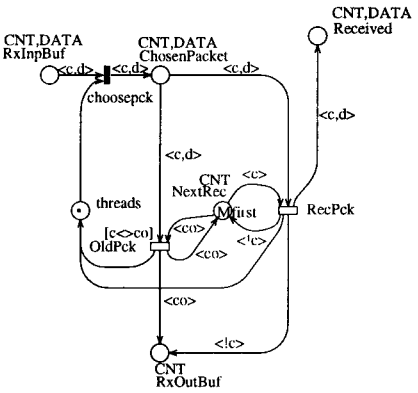

Fig. 3. Refinement of transition TxPck to represent sequential transmission of packets.

Let us discuss one more issue, related with the race conflict resolution policy for timed transitions adopted by the SWN formalism (in the same way as SPNs and GSPNs do: this is described in another chapter of this book [2]): when more than one token is concurrently present in place RxInpBuf, enabling one or more instances of transitions OldPck and one instance of transition RecPck, the default interpretation is that the activities represented by the different enabled instances run in parallel (as if the receiver had multiple threads running in parallel, able to process several packets at the same time). If we wish instead to represent a single threaded receiver, processing one packet at a time, the receiver submodel structure should be refined to select one packet from place RxInpBuf, process it, and only after sending the corresponding acknowledge message, process the next one (see Fig. 3). Similar considerations hold for the sender submodel that

in the current version represents a multithreaded sender, able to concurrently send a new packet and receive one or more acknowledge messages, and for the transmission submodel that models a network able to transport all the data and acknowledge packets in parallel.

Once the SWN model has been completely defined, its qualitative behaviour can be checked on the underlying[3] untimed model using the available analysis methods for[4] WNs. One such analysis method consists of generating the RG (provided it is finite) and then checking the desired behavioural properties on it (see Chapter [37] for a comprehensive list of interesting qualitative properties that can be checked on the RG of a CPN).

The performance evaluation phase instead can be performed through Monte Carlo simulation, or through the (numerical) analysis of the Markov chain associated with the RG. In this paper we shall focus on the latter type of analysis method, restricting ourselves to the case of steady state analysis of ergodic models with finite state spaces. If the initial state is an *home state*, the CTMC associated with the RG is ergodic: we shall therefore concentrate on models whose initial marking is an home state.

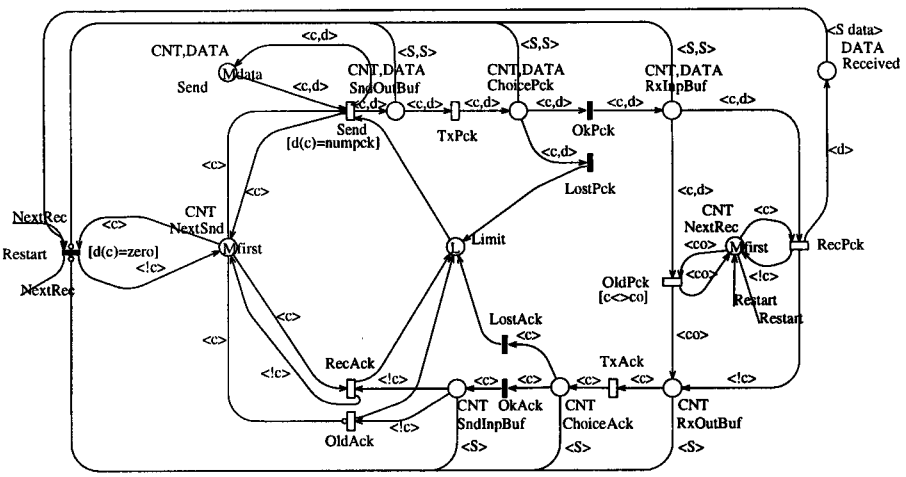

Fig. 4. Ergodic SWN model of a communication protocol with one sender and one receiver.

[3] The WN underlying an SWN is simply the model obtained by disregarding the transition rates/weights. Note that since in SWNs the delays are interpreted as negative exponentially distributed random variables, and since the negative exponential distribution has infinite support, the possible qualitative behaviours of the timed model are exactly the same as those of the corresponding untimed model.

[4] Since WNs can be considered as a *subclass* of CPNs, all the analysis methods available for CPNs, can be applied to WNs as well.

The model of the protocol in Fig. 1 is not ergodic: in fact it contains a deadlock corresponding to the state in which all messages have been successfully sent and received, and the network does not contain any message. This model can be made ergodic by adding a transition that *restarts* the system every time it reaches its (correct) final state, as shown in Fig. 4: in this way the steady state performance indices that are computed on this model represent its average behaviour considering an infinite number of possible executions of the protocol.

Let us now consider a different version of the protocol in which one sender has to broadcast the messages to several receivers. We assume that the network either delivers a given message to all receivers or none receiver gets the message. Each receiver has its own counter keeping track of the next message it expects. The sender has one counter for each receiver representing the next packet to be sent to that receiver; these counters are updated on the basis of the acknowledge messages received from the different receivers. Each time the sender decides to send a packet, it randomly chooses one receiver r, and broadcasts to all receivers the next message to be sent to r. This implies that a receiver may find in its buffer[5] the next expected message, an old message, or a new message out of sequence: only in the first case the message is accepted and the internal counter updated, while in the other two cases the message is discarded. In the new system, acknowledge messages contain a receiver identifier r and a message number n. If n is less than or equal to the value of the sender counter for receiver r, it is considered as an old acknowledge and discarded, otherwise, the counter for receiver r is updated to n. Observe that while in the first model the sender counter of the next packet to be sent can only be increased by one at each acknowledge reception, now the sender counter associated with a given receiver can be increased by more than one.

Since we are assuming that all receivers in the new system behave in the same way, we can take advantage of colours to avoid repeating the receiver submodel as many times as the number of receivers in the system. A new basic colour class REC is thus introduced, composed of a single static subclass $\mathsf{rec} = \{r_1, \ldots, r_{nrec}\}$, where $nrec$ is the number of receivers in the system.

In Fig. 5 an SWN model of the system with multiple receivers is depicted[6]. The colour structure of this model has been simplified on one hand, by eliminating the class DATA which was redundant[7], and enriched with the receivers colour class on the other hand. With the elimination of class DATA the place Send has become *implicit* and therefore it has been deleted from the model. All places but SndOutBuf have the new colour class REC in their colour domain, to distinguish messages destined to/arriving from a given receiver, and packet

[5] Note that messages are taken from the buffer in a random order, i.e., not necessarily in the same order of arrival.

[6] The model in Fig. 5 is not ergodic but it can be easily made ergodic in the same way as the SWN model of Fig. 4

[7] Note that the information on the data actually transmitted, at the detail level of this model is not relevant; algorithms have been defined for the automatic detection and elimination of redundant colour components in SWNs [14].

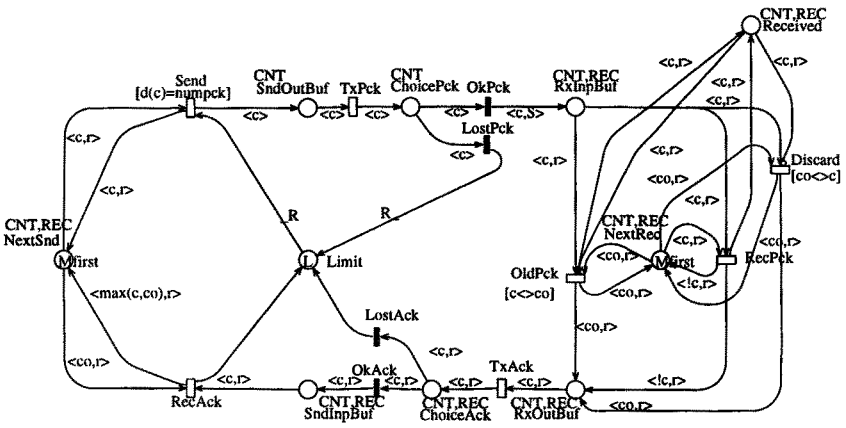

Fig. 5. SWN model of a communication protocol with one sender and *nrec* receivers.

counters (places NextRec and NextSnd that refer to a given receiver).

Other changes with respect to the single receiver model are the new transition Discard that represents a packet that was never received before, but is discarded by the receiver because it is out of sequence (these packets have an associated number greater than the current counter value in NextRec), and the transition RecAck representing the reception of an acknowledge which is stored into place NextSnd if it is greater than the current counter value in this place for the same receiver, while it is discarded otherwise. Observe that this behaviour is described in the model by using the function $max()$ in the expression labelling the arc from transition RecAck to place NextSnd, however this function is unfortunately not allowed in SWNs (the reason for this restriction will be clarified in Sec. 3). In the single receiver model we could simply overcome this problem by splitting transition RecAck into two transitions, RecAck and OldAck, one for new acknowledges, the other for old ones. This solution relies on the assumption that it is not possible to receive an acknowledge whose number is greater than the successor of the current value of the counter in NextSnd, hence it cannot work in the new model where this assumption is not satisfied. A possible solution is depicted in Fig. 6: it uses immediate transitions to process in several steps (which take no time) an acknowledge message, to properly update the marking of place NextSnd and of the additional place OldAcks. The latter place is used to store, for each receiver, the set of already acknowledged packet numbers. When a new acknowledge $\langle c, r\rangle$ arrives, all the acknowledges between the current counter value $\langle co, r\rangle$ in NextSnd and $\langle c, r\rangle$ are put into OldAcks. For example if place NextSnd contains the tokens $\langle 2, r_1\rangle, \langle 0, r_2\rangle$, OldAcks contains $\langle 0, r_1\rangle, \langle 1, r_1\rangle$, meaning that packets with number 0 and 1 have already been acknowledged by receiver r_1, while no acknowledges have been received yet from receiver r_2. If now an acknowledge arrives represented by the presence of token $\langle 2, r_2\rangle$ into place SndInpBuf (meaning that the next expected packet from receiver r_2 is the number 2), transition Many-

Ack is enabled: upon its firing, token $\langle 2, r_2 \rangle$ replaces $\langle 0, r_2 \rangle$ into NextSnd, and an immediate transition sequence firing is triggered by putting a token $\langle 1, r_2 \rangle$ into place UpdateAck. Hence, the immediate transition sequence following the firing of ManyAck is: update($c \leftarrow 1, r \leftarrow r_2, cp \leftarrow 0$), update($c \leftarrow 0, r \leftarrow r_2, cp \leftarrow max$), end1($c \leftarrow max, r \leftarrow r_2$). After the occurrence of this firing sequence (which has priority with respect to any other timed activity in the net), the marking of place OldAcks will be equal to $\langle 0, r_1 \rangle, \langle 1, r_1 \rangle, \langle 0, r_2 \rangle, \langle 1, r_2 \rangle$. Now, if an acknowledge of packet number 2 arrives from receiver r_2, represented by the presence of token $\langle 1, r_2 \rangle$ into place SndInpBuf, it will be discarded as an old one (firing of transition OldAck($c \leftarrow 0, r \leftarrow r_2$)) because it was implicitly contained in the previous acknowledge of packet number 2 from the same receiver.

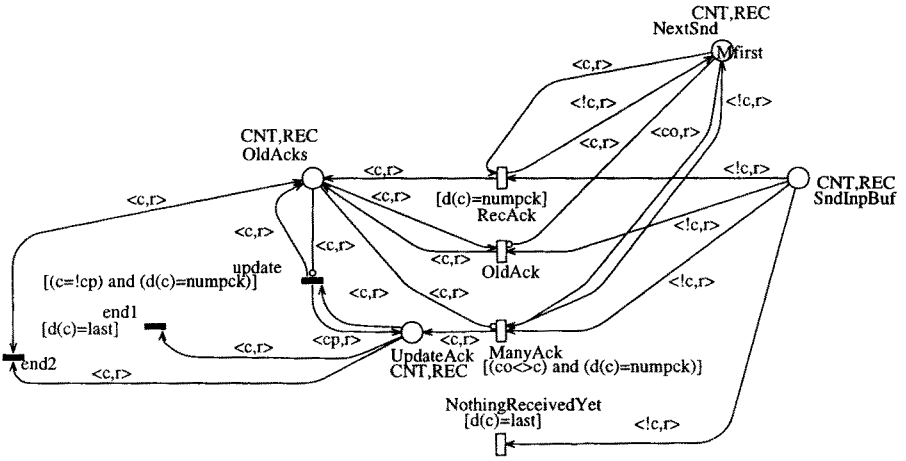

Fig. 6. Detailed view of the actual SWN implementation of the acknowledge packets reception in the multireceiver model.

2.2 SWN formal definition

In this section we give a formal definition of the concepts informally introduced in the previous section.

Definition1 Stochastic Well-formed Net. A Stochastic Well-formed Net, is a 10-tuple

$$\mathcal{N} = \langle P, T, C, cdom, \Phi, I, O, H, \Pi, W \rangle$$

where:

P is the finite set of places;
T is the finite set of transitions, $P \cap T = \emptyset$, $P \cup T \neq \emptyset$;
$\mathcal{C}$ is the family of basic colour classes: $\mathcal{C} = \{C_1, \ldots, C_n\}$; C_i is partitioned in static subclasses: $C_i = \bigcup_{q=1}^{n_i} D_{i,q}$; if $n_i = 1$ then no distinction is made between class C_i and the unique subclass $D_{i,1}$;
cdom: $P \cup T \rightarrow \bigotimes_{i=1,\ldots,n} C_i^{e_i}$, where $\bigotimes$ is a Cartesian product and e_i is the number of occurrences of C_i in the colour domain of a given place or transition.
$\Phi(t)$: $cdom(t) \rightarrow \{true, false\}$ is a *standard predicate* (see Definition 2) associated with transition t;
$I(p,t), O(p,t), H(p,t)$: $cdom(t) \rightarrow Bag(cdom(p))$ are the *arc expressions* (see Definition 3) associated respectively with an input, output, or inhibitor arc connecting transition t and place p;
$\Pi(t)$: $cdom(t) \rightarrow \mathbb{N}$ is a priority function, defining a priority (expressed as a natural number) for each instance of transition t; there is a restriction on the priority functions: two instances of a given transition may be assigned different priorities only if there exists a standard predicate capable of distinguishing the two.
$W(t)$: $cdom(t) \times (\bigotimes_{p \in P} Bag(cdom(p))) \rightarrow \mathbb{R}^+$ defining a *rate* for each (exponential) timed transition instance and a *weight* for each immediate transition instance. The transition rate/weight function may depend on the transition instance and it may be marking dependent, however, the type of dependence on the instance or on the marking is restricted in such a way that *no explicit reference to precise elements in colour classes* is made.

Definition 2 Standard Predicate. A standard predicate (or guard) associated with a transition t is a boolean expression of *basic predicates*. The allowed basic predicates are: $x = y$, $x = !y$, $d(x) = D_{i,j}$, $d(x) = d(y)$, where x, y are variables associated with transition t of the same type C_i, $!y$ denotes the successor of y (assuming that the type of y is an ordered class), and $d(x)$ denotes the static subclass x belongs to.

Definition 3 Arc Expressions. An arc expression has the following form:

$$\sum_k \delta_k . [pred_k] F_k$$

where δ_k is a positive integer, F_k is a function and $[pred_k]$ is a standard predicate. Each F_k : $cdom(t) \rightarrow Bag(cdom(p))$ is a function of the form

$$F = \bigotimes_{C_i \in \mathcal{C}} \bigotimes_{j=1,\ldots,e_i} f_i^j = \langle f_1^1, \ldots, f_1^{e_1}, \ldots, f_n^1, \ldots, f_n^{e_n} \rangle$$

with e_i representing the number of occurrences of class C_i in colour domain of place p

Each function f_i^j in turn is defined as:

$$f_i^j = \sum_{q=1}^{n_i} \alpha_{i,q} . S_{D_{i,q}} + \sum_{x \in \mathrm{var}_i(t)} (\beta_x . x + \gamma_x . !x)$$

where $S_{D_{i,q}}$, x and $!x$ are basic functions (defined hereafter), $\mathrm{var}_i(t)$ is the set of variables associated with transition t of type C_i, $\alpha_{i,q}$, β_x and γ_x are natural numbers.

The evaluation of function "$[pred]f$" is defined as:

$$[pred]f(c) \quad = \quad \text{If } pred(c) \text{ then } f(c) \text{ else } 0.$$

The multiset resulting from the evaluation of a tuple of basic functions is obtained by Cartesian product composition of the multisets resulting from the evaluation of the tuple elements. As it can be observed in the formal definition of arc expressions, there are three types of basic functions: the *projection* function, the *successor* function and the *diffusion/synchronisation* function. The syntax used for the projection function is x, where x is one of the transition variables; it is called projection because it selects one element from the tuple of value assignments defining the transition colour instance. The syntax used for the successor function is $!x$ where x is again one of the transition variables, it applies only to ordered classes and returns the successor of the object assigned to x in the transition colour instance. Finally, the syntax for the diffusion/synchronisation function is S_{C_i} (or $S_{D_{i,j}}$): it is a constant function that returns the whole set of objects of class C_i (of static subclass $D_{i,j} \subset C_i$). It is called synchronisation when used on a transition input arc because it implements a synchronisation among a set of coloured tokens contained into a place, while it is called diffusion when used on a transition output arc because it puts several tokens of different colour into a place.

Definition 4 Marking of an SWN. m is a place indexed vector which assigns to each place p a multiset over $cdom(p)$: $m[p] \in Bag(cdom(p))$.

Definition 5 SWN model. An SWN model is a pair

$$\{\mathcal{N}, m_0\}$$

where $\mathcal{N}$ is an SWN net and m_0 is the initial marking.

For reasons that will become clear later on, it is useful to define a *symmetric* initial marking of an SWN model.

Definition 6 Symmetric marking of an SWN model. A marking m of an SWN model is symmetric if it can be expressed as follows:

$$\forall p \in P, m[p] = \sum_{\tilde{c} \in \bigotimes_{C_i \in C} \tilde{C}_i^{e_i}} \alpha_{\tilde{c}}\, \tilde{c}$$

where $\tilde{c}$ is a tuple of static subclasses (consistent with the colour domain of the corresponding place), and $\alpha_{\tilde{c}} \in \mathbb{N}$ is the coefficient of tuple $\tilde{c}$.

As usual a tuple $\langle A_1, \ldots, A_k \rangle$ of sets represents the set of tuples obtained by composing the sets through the Cartesian product operator:

$$\langle A_1, \ldots, A_k \rangle := \bigotimes_{i=1,\ldots,k} A_i$$

Definition 7 Transition instances enabling and firing. A transition instance $t(c)$ has concession in marking m iff

- for each place p in the transition input set: $I(p,t)(c) \subseteq m[p]$
- for each place p in the transition inhibition set:
  $\forall c' \in cdom(p) : H(p,t)(c)(c') > 0, m[p](c') < H(p,t)(c)(c')$
- $\Phi(t)(c) = true$

A transition instance $t(c)$ is enabled in in marking m if it has concession in m and there is no higher priority transition instance that has concession:

- $\nexists t'(c') : t'(c')$ has concession in m and $\Pi(t')(c') > \Pi(t)(c)$.

A transition instance $t(c)$ which is enabled in marking m may fire yielding the marking m', denoted $m[t(c)\rangle m'$. The new marking m' is obtained as follows:

- $\forall p, m'[p] = m[p] - I(t,p)(c) + O(t,p)(c)$

Let us comment a little bit on the definition of weight/rate functions W that are part of the model specification allowing to derive a CTMC from the model RG. Observe that the restrictions on the definition of the priority and transition rate functions are needed to ensure that symmetry presented at a qualitative behaviour level is reflected also at a quantitative behaviour level: this is a mandatory requirement to apply the efficient performance analysis algorithms described later on.

In practice, the type of priority and rate functions used are even more restrictive than needed (but easier to specify and to deal with in an implementation), and are based on the concept of *static partition* of a transition colour domain and marking as defined in [11]. Intuitively, in this more restricted definition the rate/weight function can depend only on the static classes which contain the objects involved in a transition firing and that compose the marking.

We do not explain here how the CTMC underlying an SWN model is defined since it can be derived in a straightforward way from the definition of the CTMC underlying a GSPN model presented in Chapter [2]. Further details can be found in [11, 12].

3 The Symbolic Reachability Graph

We discuss in this section the notions of symbolic marking, symbolic firing and symbolic reachability graph. The discussion will be rather informal, mainly based on the running example of the communication protocol already presented in Section 2.1. The interested reader may find formal definitions in the papers [12, 13] suggested in the bibliography.

3.1 Symbolic marking

As briefly discussed in Chapter [37], it is possible to use some forms of equivalences to obtain a reduced, or condensed, state space underlying a CPN model. Also SWNs provide a modelling framework in which the equivalences between states, which are based on intrinsic symmetries, can be automatically detected and used naturally as a way for reducing the size of the underlying state space.

Let us explain what does it mean *automatically detect the intrinsic symmetries*. Intuitively, a symmetric system is composed of replicas of the same components, all sharing the same behaviour: we have seen a first example of a symmetric system in Section 2.1, when discussing the protocol example for the case of multiple receivers. The symmetry present in this system, allows us to represent the state in a more abstract form: instead of keeping track of the state of each receiver we could just keep track of how many receivers are in a given state, independently of their actual identity. This can be done only if the potential behaviour of a receiver does only depend on its current state and not on its identity, i.e. if all receivers behave *homogeneously*.

The idea of *homogeneous* behaviour is the basis for the definition of the so called *symbolic marking* and the consequent generation of a more compact state space. Let us fix the number of receivers in our example and consider the case of $nrec = 3$. In the initial marking of the net shown in Fig. 5 we can identify the following situation:

1. the sender is ready to send the first data packet
2. all the receivers are ready to receive the first data packet
3. all the buffers are empty
4. no messages are circulating in the network

Let us suppose that, after a certain number of transition firings, we reach the following new situation:

1. the first two receivers have already received and acknowledged the first two data packets and are now waiting for the third data packet
2. the third receiver is still waiting for the first data packet
3. the sender is ready to broadcast either the second data packet required by the first two receivers or the first data packet required by the third receiver
4. the buffer SndOutBuf contains the first data packet
5. the buffer RxInpBuf contains the first data packet for the third receiver
6. the buffer RxOutBuf contains two acknowledge messages expressing the fact that receivers r_1 and r_2 are now waiting packet number three
7. since there are five messages circulating in the network (three for the broadcast of packet number 1 and two acknowledges), there is only room left for other $L - 5$ messages.

The marking[8] m' that models such a situation is the following:

$\mathsf{NextSnd}(\langle 1, r_3\rangle, \langle 2, r_1\rangle, \langle 2, r_2\rangle)$ $\quad$ $\mathsf{SndOutBuf}(\langle 1\rangle)$
$\mathsf{RxInpBuf}(\langle 1, r_3\rangle)$ $\quad$ $\mathsf{NextRec}(\langle 1, r_3\rangle, \langle 3, r_1\rangle, \langle 3, r_2\rangle)$
$\mathsf{Received}(\langle 1, r_1\rangle, \langle 2, r_1\rangle, \langle 1, r_2\rangle, \langle 2, r_2\rangle)$ $\quad$ $\mathsf{RxOutBuf}(\langle 3, r_1\rangle, \langle 3, r_2\rangle)$
$\mathsf{Limit}(L-5)$

Now, notice that there are situations which are similar to the one just described and that can be obtained from it by *applying a permutation* on the receivers. For instance, if we exchange the first and third receiver we obtain the marking m'' below:

$\mathsf{NextSnd}(\langle 1, r_1\rangle, \langle 2, r_2\rangle, \langle 2, r_3\rangle)$ $\quad$ $\mathsf{SndOutBuf}(\langle 1\rangle)$
$\mathsf{RxInpBuf}(\langle 1, r_1\rangle)$ $\quad$ $\mathsf{NextRec}(\langle 1, r_1\rangle, \langle 3, r_2\rangle, \langle 3, r_3\rangle)$
$\mathsf{Received}(\langle 1, r_2\rangle, \langle 2, r_2\rangle, \langle 1, r_3\rangle, \langle 2, r_3\rangle)$ $\quad$ $\mathsf{RxOutBuf}(\langle 3, r_2\rangle, \langle 3, r_3\rangle)$
$\mathsf{Limit}(L-5)$

The same reasoning holds when we exchange the second and third receiver. In fact, when we permute the identities of the receivers we obtain markings which describe the same situation and which are characterised by an equivalent future behaviour in terms of possible transition firing sequences. This allows us to define an equivalence notion for markings that are equal up to the permutation of the receiver identities:

$$m_1 \sim m_2 \Leftrightarrow \exists s : m_2 = s.m_1$$

where $s : \mathsf{REC} \to \mathsf{REC}$ is a permutation operator on the class of receivers, and the application of a permutation s to a marking m, denoted $s.m$, consists of substituting each occurrence of any $r_i \in \mathsf{REC}$ in m with $s(r_i)$.

The following property holds for markings that are equal up to the permutation of the receiver identities:

$$m[t, c\rangle m' \iff s.m[t, s(c)\rangle s.m'$$

where $s(c)$ denotes the application of a permutation s to a transition colour instance c. Hence, whenever a transition instance $t(c)$ is enabled in a marking m, we are sure that there will be a corresponding instance $t(c')$ enabled in any marking m' equivalent to m. Moreover, the two markings reached are in turn equivalent. Notice that this property is the analogous of the notion of bisimulation which has been introduced for Process Algebras [43] and that will be discussed in Section 4.

The above property holds because the arc expressions, the transition predicates and the priorities of SWNs satisfy the following conditions [13]:

$$\forall s: \; s.I = I.s, \; s.O = O.s, \; s.N = H.s$$
$$\forall s: \; \Phi(c) = \Phi(s(c)) \text{ and } \Pi(c) = \Pi(s(c))$$

[8] We consider non empty places only and we use a slightly modified notation for the marking, enumerating in round brackets the tuples contained in each place.

Our objective now is to exploit this equivalence relation among markings to reduce the state space size: since equivalent markings lead to the same behaviour we want to avoid generating all the states in a given equivalence class. There are different possible approaches to achieve this goal:

1. when a marking m is reached while constructing the (reduced) RG, it is added into the RS only if the RS does not contain any marking in the same equivalence class yet;
2. the RS does not contain markings but rather symbolic representations of the equivalence classes (the *symbolic markings*), the set of reachable equivalence classes and the arcs connecting them is obtained by defining a symbolic firing rule working directly at the symbolic marking level.

The Symbolic Reachability Graph (SRG) adopts the latter approach, the advantages being a reduced computational cost for deciding whether a reached marking belongs to a new equivalence class never reached before, that requires a test for equivalence in the former approach rather than a test for equality, and a more abstract representation of the equivalence class [13].

In order to explain the concept of symbolic marking let us go back to our running example. We can start by defining a more abstract description of state m' which gives us the possibility of defining the whole set of markings equivalent to it. Such an abstract description could be informally expressed as follows:

1. (any) *two* receivers have already received two data packets each, and they have already acknowledged the receptions of the first data packet
2. the remaining receiver is still waiting for the first data packet
3. the sender is ready to broadcast the second data packet required by the two receivers (of item 1) and the first data packet to the remaining receiver
4. the buffer SndOutBuf contains the first data packet
5. the buffer RxInpBuf contains the first data packet for the receiver of item 2 waiting for it
6. the buffer RxOutBuf contains two acknowledges expressing the fact that two receivers (of item 1) are waiting for the third packet.

This informal description can be formally expressed by defining sets of receivers that in the current marking play the same role (i.e. receivers currently in the same state). For instance, in the case of marking m' it is possible to identify a partition of the colour class REC into two disjoint subsets $\mathsf{REC}_1 = \{r_3\}$ and $\mathsf{REC}_2 = \{r_1, r_2\}$: where REC_1 represents the set of receivers still waiting the first data packet and REC_2 represents the set of receivers that have already received two data packets each, and have already acknowledged the reception of the first data packet. In marking m'' we can identify the same partition of receivers into two subsets REC_1 and REC_2 of cardinality 1 and 2 respectively, the only difference being the actual identity of the receivers belonging to the subsets: $\mathsf{REC}_1 = \{r_1\}$ and $\mathsf{REC}_2 = \{r_2, r_3\}$.

Now, if we *forget about the identity of the receivers* belonging to the subsets REC_i, and only keep the information on the cardinality of the subsets (notice

that the subsets must form a partition of class REC, i.e. they are disjoint and their union must be the whole class REC), we obtain the desired symbolic representation of the equivalence class of markings:

$$\begin{array}{ll} \mathsf{NextSnd}(\langle 1, \mathsf{REC}_1\rangle, \langle 2, \mathsf{REC}_2\rangle) & \mathsf{SndOutBuf}(\langle 1\rangle) \\ \mathsf{RxInpBuf}(\langle 1, \mathsf{REC}_1\rangle) & \mathsf{NextRec}(\langle 1, \mathsf{REC}_1\rangle, \langle 3, \mathsf{REC}_2\rangle) \\ \mathsf{Received}(\langle 1, \mathsf{REC}_2\rangle, \langle 2, \mathsf{REC}_2\rangle) & \mathsf{RxOutBuf}(\langle 3, \mathsf{REC}_2\rangle) \\ \mathsf{Limit}(L-5) & |\mathsf{REC}_1| = 1, |\mathsf{REC}_2| = 2 \end{array}$$

Observe that in the symbolic marking representation the symbols REC_i, representing subsets of (unidentified) receivers, replace the receiver identities $r_j \in \mathsf{REC}$ in the tuples representing the tokens. Since subsets can have cardinality greater than one, the representation of a symbolic marking can be much more compact than the representation of the ordinary markings belonging to the corresponding equivalence class. For example in the above symbolic marking representation, places NextSnd and NextRec contain only two tuples and place RxOutBuf contains only one tuple while all these places contained one more tuple in the representation of the ordinary markings m' and m'' belonging to this equivalence class. The reason is that tuple $\langle 3, \mathsf{REC}_2\rangle$ actually represents any two tokens $\langle 3, r_x\rangle + \langle 3, r_y\rangle$ with $r_x, r_y \in \mathsf{REC}, r_x \neq r_y$, because the cardinality of REC_2 is two.

Let us now define more formally how it is possible to transform an ordinary marking into the symbolic marking representing the equivalence class it belongs to. Although intuitively we can say that the receivers grouped in the same subset are those with the same potential future behaviour, it is not necessary to actually check the future evolution of the net from the marking to decide how to group receivers, instead this can be done syntactically by observing the *state of the receivers*, i.e. their distribution in the model places. Informally, given a marking m the distribution of an object c_i into the places for that marking is defined by:

1. the set of places $p \in P$ such that c_i appears in $m[p]$;
2. for each place p in the above set, and for each tuple in $m[p]$ having c_i among its elements, the multiplicity of the tuple, the position of element c_i in the tuple, and the identity and position of the other elements in the tuple.

The distribution[9] of the two receivers r_1, r_2 in marking m' is the same, i.e.

$$\mathsf{NextSnd}(1\langle 2, .\rangle) + \mathsf{NextRec}(1\langle 3, .\rangle) + \mathsf{RxOutBuf}(1\langle 3, .\rangle) + \mathsf{Received}(1\langle 1, .\rangle, 1\langle 2, .\rangle)$$

therefore we can conclude that they have the same potential future behaviour and hence group them into subset REC_2. The remaining receiver instead has a different distribution, namely:

$$\mathsf{NextSnd}(1\langle 1, .\rangle) + \mathsf{NextRec}(1\langle 1, .\rangle) + \mathsf{RxInpBuf}(1\langle 1, .\rangle)$$

and hence is kept in a separate subset REC_1 of cardinality one.

[9] For an object c_i the expression $PlaceName(n\langle c_j, .\rangle)$ means that it appears in the place $PlaceName$ as second element a 2-tuple of multiplicity n, whose first element is c_j.

Given a partition of the receivers into subsets REC_i, the symbolic representation $\hat{m}$ of m is obtained as follows: for each place $p \in P$ with colour domain CNT, $\hat{m}[p] = m[p]$, for each place $p \in P$ with colour domain $\mathsf{CNT} \times \mathsf{REC}$, include in $\hat{m}(p)$ a tuple $n\langle c, \mathsf{REC}_j \rangle$ iff [10] $\sum_{r_h \in \mathsf{REC}_j} n\langle c, r_h \rangle \subseteq m[p]$.

On the other hand, given a symbolic marking representation all ordinary markings belonging to it can be obtained by enumeration of all possible assignments of receivers to subsets, satisfying the requirement that a given receiver must belong to one and only one subset, and the number of receivers assigned to a subset must be consistent with the subset cardinality (which is part of the symbolic marking definition).

In our example, there are three possible assignments of receivers to subsets REC_1 and REC_2, each corresponding to one of the three ordinary markings belonging to the equivalence class represented by the symbolic marking.

$$\begin{array}{ll} \mathsf{REC}_1 = \{r_1\} & \mathsf{REC}_2 = \{r_2, r_3\} \\ \mathsf{REC}_1 = \{r_2\} & \mathsf{REC}_2 = \{r_1, r_3\} \\ \mathsf{REC}_1 = \{r_3\} & \mathsf{REC}_2 = \{r_1, r_2\} \end{array}$$

In the SRG terminology, the subsets REC_i are called *dynamic subclasses*. The partitioning of a colour class into dynamic subclasses is marking dependent and must not be confused with the partition of the colour classes into static subclasses which is fixed and is part of the colour class definition. There is a further requirement about dynamic subclasses: each dynamic subclass must be a subset of only one static subclass; for each dynamic subclass, the indication of the symbolic subclass it belongs to, is part of the symbolic marking definition. In our running example we could omit this information because class REC comprises a single static subclass.

To summarise, the following steps are necessary to introduce the notion of symbolic marking

1. basic colour classes are partitioned into dynamic subclasses, each containing elements that have the same possible future behaviour;
2. the identities of the objects within dynamic subclasses are lost; dynamic subclasses are simply characterised by their cardinality;
3. the coloured tokens in the places of the net are replaced by symbolic tokens which are tuples of dynamic subclasses; each symbolic marking corresponds to a set of ordinary markings, depending on the number of different possible assignments of actual objects sets to the dynamic subclasses.

The definition of symbolic marking given so far, does not ensure that the representation of an equivalence class through a symbolic marking is unique. For instance, the assignment of names to dynamic subclasses is completely arbitrary: we thus need some additional constraint on the representation leading to a unique (canonical) representation of a symbolic marking.

[10] Observe that given our definition of partition into subsets of receivers with same distribution into places, if $n\langle c, r_h \rangle \subseteq m[p], r_h \in \mathsf{REC}_j \Rightarrow n\langle c, r_k \rangle \subseteq m[p], \forall r_k \in \mathsf{REC}_j$.

The two constraints on the symbolic marking giving a canonical representation are:

1. the partition of the basic colour classes into dynamic subclasses must be *minimal*,
2. the names assigned to dynamic subclasses must give a lexicographically minimal symbolic marking representation.

The minimality requirement of item 1, refers to the number of dynamic subclasses in each basic class; intuitively we want to have the smallest possible number of dynamic subclasses since this minimises the *size* of the symbolic marking representation. A symbolic marking is said to be minimal when it is not possible to find any pair of dynamic subclasses belonging to the same static class that have the same distribution over the places of the net.

Observe that there are several possible representations that minimise the number of dynamic subclasses, which are equal up to a renaming of the dynamic subclasses: here comes into play the second constraint. A canonical representation can be obtained by introducing an *ordering* relation among all possible representations of a symbolic marking that minimise the number of dynamic subclasses. We use the lexicographical ordering, and choose the lexicographically minimal representation (of course a lexicographic order can be defined only after defining a fixed - though arbitrary - order for places in the marking representation). The complete algorithm for the computation of the canonical representation is described in [10].

One remark is important about the initial symbolic marking: since in general symbolic markings can contain several ordinary markings, this can be the case also for the initial marking. In this case, the set of reachable symbolic markings would represent all ordinary markings reachable from any ordinary marking belonging to the initial symbolic marking. It is however frequent that the initial symbolic marking represents just one ordinary marking: this is true if each static subclass is initially completely grouped into a single dynamic subclass. This special type of marking is called *symmetric* (see Definition 6). The initial marking of our running example can indeed be represented by a symmetric symbolic marking. As we shall see later, having a symmetric initial marking simplifies the study of ergodicity which is an important property when the intended use of the model is the analysis of the steady state timed behaviour of the system.

3.2 Symbolic Enabling and Symbolic Firing

Starting from the notion of symbolic marking a *symbolic enabling rule* and a *symbolic firing rule* have been defined.

In a symbolic firing instance dynamic subclasses are assigned to the transition parameters instead of the actual objects. When we assign a dynamic subclass to a transition parameter we mean that *any* object of that subclass may be selected for assignment to the parameter. It does not actually matter *which* object is chosen since they all have the same potential future behaviour.

In order to define the symbolic firing rule we need to introduce the notion of *split symbolic marking*. From each dynamic subclass assigned to one of the transition variables, we need to pick an (arbitrary) object that will be actually involved in the transition firing. Each dynamic subclass involved in a symbolic transition firing is thus *split* into two new subclasses, one containing the object selected for the firing and the other containing all the remaining objects.

The first new dynamic subclass has cardinality one and represents the single object involved in the firing, the second subclass has one element less than the originating subclass, and contains all the other objects which are not involved in the firing. After splitting, the cardinality one dynamic subclass is assigned to the transition instance to be checked for enabling and possibly fired.

The symbolic enabling rule is exactly the same as the usual enabling rule with the only difference that now dynamic subclasses are assigned to the transition variables in a symbolic transition instance, and hence the arc expressions evaluation must be extended to dynamic subclasses. Since after splitting the involved dynamic subclasses have cardinality one, they can be treated as normal objects. Concerning the constant function S_{C_i}, in this case it returns the complete set of dynamic subclasses within class C_i rather than the set of objects of C_i. Once the input/inhibition arc functions are evaluated, the enabling test is the same as usual: the symbolic marking of each input place must contain the multiset obtained by evaluation of the corresponding arc expression, and each inhibitor place must not contain the multiset obtained by evaluation of the corresponding arc expression.

Hence, the symbolic firing comprises four steps:

1. *Splitting*: the dynamic subclasses assigned to the transition variables are split in two new dynamic subclasses, one containing the single object actually involved in the firing, the other containing the remaining objects. This step is obviously not performed in the case of subclasses of cardinality one.
2. *Firing*: the input/output arc expressions are evaluated by substituting the variables with the new cardinality one subclasses assigned to them: the corresponding multisets of *symbolic tokens* are withdrawn from/added to the transition input/output places. The new symbolic marking representation m' reached after the firing is usually not canonical.
3. *Minimality*: the minimal representation of $\hat{m}'$ is obtained by merging the dynamic subclasses that have the same distribution over the places of the net. The dynamic subclass resulting from the merge operation will have a cardinality equal to the sum of the cardinalities of the merged dynamic subclasses.
4. *Canonical representation*: the canonical representation $\hat{m}''$ of $\hat{m}'$ is obtained by applying a lexicographic ordering operation.

Let us make a remark: the implication of assigning a whole dynamic subclass to variables in symbolic transition instances is that a symbolic firing may represent several ordinary firings. Let us explain this on two examples of symbolic firings

enabled in the symbolic marking $\hat{m}_1$ below:

$$\begin{array}{ll}
\mathsf{NextSnd}(\langle 2, \mathsf{REC}_1\rangle, \langle 2, \mathsf{REC}_2\rangle) & \mathsf{RxInpBuf}(\langle 2, \mathsf{REC}_1\rangle) \\
\mathsf{NextRec}(\langle 2, \mathsf{REC}_1\rangle, \langle 3, \mathsf{REC}_2\rangle) & \mathsf{RxOutBuf}(\langle 3, \mathsf{REC}_2\rangle) \\
\mathsf{Received}(\langle 1, \mathsf{REC}_1\rangle, \langle 1, \mathsf{REC}_2\rangle, \langle 2, \mathsf{REC}_2\rangle) & \mathsf{Limit}(L-3) \\
|\mathsf{REC}_1| = 1, |\mathsf{REC}_2| = 2 &
\end{array}$$

The two symbolic firing instances we shall consider are: $\mathsf{RecPck}(c \leftarrow 2, r \leftarrow \mathsf{REC}_1)$ and $\mathsf{TxAck}(c \leftarrow 3, r \leftarrow \mathsf{REC}_2)$. The first transition instance involves dynamic subclass REC_1 of cardinality 1, hence splitting is not needed in this case. It is enabled because a token $\langle 2, \mathsf{REC}_1\rangle$ is contained both in place NextRec and in place RxInpBuf so that we can proceed in the actual firing pretty much in the same way as we perform an ordinary firing. The new symbolic marking $\hat{m}_2$ reached is:

$$\begin{array}{ll}
\mathsf{NextSnd}(\langle 2, \mathsf{REC}_1\rangle, \langle 2, \mathsf{REC}_2\rangle) & \\
\mathsf{NextRec}(\langle 3, \mathsf{REC}_1\rangle, \langle 3, \mathsf{REC}_2\rangle) & \mathsf{RxOutBuf}(\langle 3, \mathsf{REC}_1\rangle, \langle 3, \mathsf{REC}_2\rangle) \\
\mathsf{Received}(\langle 1, \mathsf{REC}_1\rangle, \langle 2, \mathsf{REC}_1\rangle, \langle 1, \mathsf{REC}_2\rangle, \langle 2, \mathsf{REC}_2\rangle) & \mathsf{Limit}(L-3) \\
|\mathsf{REC}_1| = 1, |\mathsf{REC}_2| = 2 &
\end{array}$$

It is easy to see that this representation does not minimise the number of dynamic subclasses, indeed now both subclasses REC_1 and REC_2 have the same distribution in the places and hence can be merged leading to the new symbolic marking representation:

$$\begin{array}{ll}
\mathsf{NextSnd}(\langle 2, \mathsf{REC}_1\rangle) & \\
\mathsf{NextRec}(\langle 3, \mathsf{REC}_1\rangle) & \mathsf{RxOutBuf}(\langle 3, \mathsf{REC}_1\rangle) \\
\mathsf{Received}(\langle 1, \mathsf{REC}_1\rangle, \langle 2, \mathsf{REC}_1\rangle) & \mathsf{Limit}(L-3) \\
|\mathsf{REC}_1| = 3 &
\end{array}$$

Since this symbolic firing involved only a subclass of cardinality one, it represented just one ordinary firing. The second symbolic firing instance instead, represents two ordinary firings, one for each object that can be chosen within the dynamic subclass REC_2 (see Fig. 7 for a pictorial representation of the example). In this case we need to first split dynamic subclass REC_2 into two subclasses REC_2 and REC_3 representing respectively the object chosen for the firing and the remaining objects in the subclass. We thus obtain the split symbolic marking representation below:

$$\begin{array}{ll}
\mathsf{NextSnd}(\langle 2, \mathsf{REC}_1\rangle, \langle 2, \mathsf{REC}_2\rangle, \langle 2, \mathsf{REC}_3\rangle) & \mathsf{RxInpBuf}(\langle 2, \mathsf{REC}_1\rangle) \\
\mathsf{NextRec}(\langle 2, \mathsf{REC}_1\rangle, \langle 3, \mathsf{REC}_2\rangle, \langle 3, \mathsf{REC}_3\rangle) & \mathsf{RxOutBuf}(\langle 3, \mathsf{REC}_2\rangle, \langle 3, \mathsf{REC}_3\rangle) \\
\mathsf{Received}(\langle 1, \mathsf{REC}_1\rangle, \langle 1, \mathsf{REC}_2\rangle, \langle 2, \mathsf{REC}_2, \rangle\langle 1, \mathsf{REC}_3\rangle, \langle 2, \mathsf{REC}_3\rangle) & \mathsf{Limit}(L-3) \\
|\mathsf{REC}_1| = 1, |\mathsf{REC}_2| = 1, |\mathsf{REC}_3| = 1 &
\end{array}$$

The transition instance $\mathsf{TxAck}(c \leftarrow 3, r \leftarrow \mathsf{REC}_2)$, with the new subclass REC_2 assigned to variable r, is enabled since token $\langle 3, \mathsf{REC}_2\rangle$ is contained into place

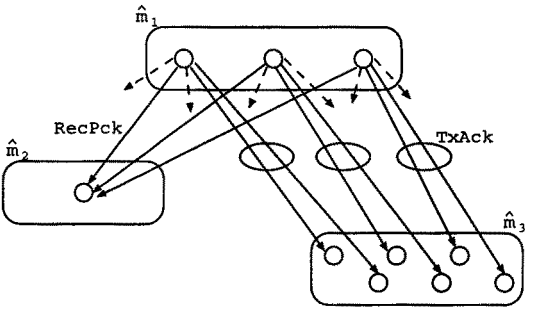

Fig. 7. Marking and symbolic firing aggregation in the SRG.

RxOutBuf. After performing the firing, we obtain the new (vanishing) symbolic marking $\hat{m}_3$

$$\begin{array}{l}\mathsf{NextSnd}(\langle 2,\mathsf{REC}_1\rangle,\langle 2,\mathsf{REC}_2\rangle,\langle 2,\mathsf{REC}_3\rangle)\quad \mathsf{RxInpBuf}(\langle 2,\mathsf{REC}_1\rangle)\\ \mathsf{NextRec}(\langle 2,\mathsf{REC}_1\rangle,\langle 3,\mathsf{REC}_2\rangle,\langle 3,\mathsf{REC}_3\rangle)\quad \mathsf{RxOutBuf}(\langle 3,\mathsf{REC}_3\rangle)\\ \mathsf{Received}(\langle 1,\mathsf{REC}_1\rangle,\langle 1,\mathsf{REC}_2\rangle,\langle 2,\mathsf{REC}_2,\rangle\langle 1,\mathsf{REC}_3\rangle,\langle 2,\mathsf{REC}_3\rangle)\\ \mathsf{ChoiceAck}(\langle 3,\mathsf{REC}_2\rangle)\quad \mathsf{Limit}(L-3)\\ |\mathsf{REC}_1|=1,|\mathsf{REC}_2|=1,|\mathsf{REC}_3|=1\end{array}$$

This representation is already minimal with respect to the grouping into dynamic subclasses, however it is not yet canonical: in fact by exchanging (the names of) dynamic subclasses REC_2 and REC_3 we obtain a representation that is lexicographically less than that obtained after the firing.

The reader may have observed that in the above example, the objects of class REC have been treated in a symbolic way, while the objects of class CNT have been represented in the usual way. This is because there is no exploitable symmetry within class CNT since we need to identify the first packet and all its successors (the behaviour of the system is not independent on their actual identity). The example above concerns the exploitation of symmetries of a non ordered class, treatment of ordered classes is slightly more complicated than that of non ordered ones, and will not be explained in this chapter: the reader may refer to [11] for more details on this topic.

Finally, the symbolic marking and the symbolic firing rule can deal also with several simultaneous classes with symmetries.

3.3 Generation and analysis of the Symbolic Reachability Graph

The Symbolic Reachability Graph (SRG) describes the evolution of an SWN model through a set of macro-states (the equivalence classes represented by symbolic markings). Each macro-state represents a set of more detailed states which are equivalent. The nodes of the SRG are labelled with symbolic markings in canonical form and the arcs are labelled with symbolic firing instances. The algorithm for the SRG generation has the same structure of the algorithm for

the generation of the RG with marking and firings replaced by their symbolic counterparts: a detailed description of this algorithm may be found in [13].

The strength of the SRG approach is due to the possibility of reducing the state space size, while still being able to study at the SRG level most of the properties that can be studied on the complete RG. We mention here the most relevant properties omitting their proofs (that can be found in [13]).

The SRG and RG are equivalent with respect to reachability, i.e., all ordinary markings reachable from the set of initial ordinary markings belonging to the initial symbolic marking are represented by some symbolic marking in the symbolic RS and vice-versa. This means for example that if the RG contains a deadlock, this can be found also in the SRG. Another important property is that the RG is finite if and only if the SRG is finite. Also liveness of transitions in the SRG can be related to liveness of transitions in the RG since any firing sequence in the RG has a counterpart in the SRG and vice-versa.

As mentioned before, a property which is very relevant in the context of SPNs is ergodicity: the problem is to check whether the RG is strongly connected. If the RG of an SWN model is strongly connected, the corresponding SRG is strongly connected too but, in general, the vice-versa does not hold. In the particular case of a symmetric initial marking however, if the SRG is strongly connected, also the RG is so. In [11] a less restrictive sufficient condition for RG ergodicity has been defined.

Another set of important properties, called numerical properties, allows one to compute both the number of ordinary markings belonging to a given equivalence class and the number of ordinary firings represented by each symbolic firing (see [12] for the details): as we shall see later, these properties are very important for the performance analysis of SWN models through the SRG.

If arbitrary qualitative properties have to be checked on the model state space, as for example any kind of property that can be expressed through temporal logic formulae, then it might be necessary to partially *unfold* the SRG, depending on the type of formula expressing the property. In [36] it is shown how the SRG can be used to prove SWN models qualitative properties expressed as CTL formulas by applying model checking techniques. It is worthwhile highlighting that recently some papers have appeared on the possibility of exploiting symmetries in the framework of temporal logic model checking [18, 21] which are based on ideas similar to those presented in this chapter for SWNs.

We conclude this section showing some numerical results we obtained using the GreatSPN tool [17]: we have computed the SRG of the SWN model of Fig. 5 varying the network capacity, the number of receivers and of data packets to be sent, and computed the size of the corresponding RG by using a formula that allows to compute the number of ordinary markings represented by a symbolic marking. The SRG and RG sizes are listed in Table 1: they coincide with the number of states in the ordinary and condensed state spaces computed for the multiple receivers CPN model of Chapter [37].

It can be observed that in the case of two receivers the size of the RG is close twice the size of the corresponding SRG. In the case of three receivers the ratio

Recs	Lim	Packets	RG	SRG	RG/SRG
2	2	2	245	131	1,870
2	2	3	529	277	1,910
2	2	4	921	477	1,931
2	3	2	3.609	1.819	1,984
2	3	3	14.025	7.037	1,993
2	3	4	35.909	17.991	1,996
3	3	3	9.775	1.903	5,137
3	3	4	22.317	4.195	5,320
3	4	2	104.258	18.253	5,712
4	4	2	39.617	2.559	15,481
4	4	3	172.581	9.888	17,454
5	5	2	486.767	8.387	58,038
6	6	2	5.917.145	24.122	245,301

Table 1. Sizes of RGs and SRGs of the net model of Fig. 5.

between the RG and the SRG sizes is almost 5, while with four receivers we have a ratio of about 16, ... Observe that the aggregation which can be achieved when using the SRG algorithm is bounded from above by the product of the factorial of colour classes cardinalities (in the protocol example |REC|!).

3.4 SRG aggregation and lumpability of Markov chains

In this section we discuss the relation between the state space aggregation due to the application of the SRG algorithm and the *lumpability* of the CTMC that can be derived from the RG. The aim is to reduce the cost of performance analysis exploiting again the model symmetries. We first give the definition of ordinary, exact and strict lumpability (according to the definitions in [7]), then we discuss the lumpability of the CTMC underlying an SWN model.

Definition 8 Ordinary Lumpability of MC. Let $S = \{s_1, \ldots, s_n\}$ be the set of states of a CTMC, Q be the corresponding infinitesimal generator and $A = \{A_1, \ldots, A_k\}$ be a partition of S into *aggregates*. The strong lumpability condition is defined as:

$$\forall A_i, A_j \in A, \ \forall s_{i1}, s_{i2} \in A_i, \ \sum_{s_k \in A_j} q_{i1,k} = \sum_{s_k \in A_j} q_{i2,k}$$

If ordinary lumpability (also called strong lumpability in the literature) holds, then a lumped CTMC can be constructed with rates between aggregate A_i and A_j equal to:

$$\hat{q_{i,j}} = \sum_{s_k \in A_j} q_{h,k}, \text{ where } s_h \in A_i$$

and steady state analysis of the CTMC can be performed on the reduced MC.

Definition 9 Exact Lumpability of MC. Let $S = \{s_1, \ldots, s_n\}$ be the set of states of a CTMC, Q be the corresponding infinitesimal generator and $A = \{A_1, \ldots, A_k\}$ be a partition of S into *aggregates*. The exact lumpability condition is defined as:

$$\forall A_i, A_j \in A, \ \forall s_{i1}, s_{i2} \in A_i, \ \sum_{s_k \in A_j} q_{k,i1} = \sum_{s_k \in A_j} q_{k,i2}$$

If exact lumpability holds, then all states in an aggregate have the same steady state probability, hence a lumped CTMC can be built whose inter aggregate transition rates are given by:

$$\hat{q_{i,j}} = \frac{1}{|A_i|} \sum_{s_h \in A_i} \sum_{s_k \in A_j} q_{h,k}$$

If both ordinary and exact lumpability conditions are satisfied, we can compute the aggregate MC with any formula above since in this case they are equivalent. Moreover, we know that the states within an aggregate are equiprobable. The name *strict* lumpability has been introduced in [7] meaning that both exact and strong lumpability conditions are met.

In [11] it has been shown that strict lumpability holds for the CTMC corresponding to an SWN model with respect to the SRG aggregation. Since it is easy to compute the number of ordinary markings belonging to a symbolic marking, we can compute any performance index that could be computed on the complete CTMC by generating and solving only the aggregated CTMC.

Proposition 10. *The CTMC corresponding to the RG of an SWN model satisfies the strict lumpability conditions with respect to the aggregation of (ordinary) markings into symbolic markings.*

The above result tells us that we can exploit the SRG aggregation also for performance evaluation purposes. This result can be fully exploited because not only we can solve a lumped MC to obtain performance indices, but we are able to build the lumped MC directly from the SRG, without ever computing the complete MC.

Proposition 11. *The lumped CTMC corresponding to the SRG of an SWN model can be directly computed using only the information on the transition instance rates/weights and the SRG.*

The rates between two symbolic markings $\hat{m}$ and $\hat{m}'$ can be obtained as the sum over all symbolic transition instances $t(\hat{c})$ leading from $\hat{m}$ to $\hat{m}'$ of the product $|t(\hat{c})|\theta(t(\hat{c}))$ where $|t(\hat{c})|$ represents the number of ordinary firings represented by symbolic firing $|t(\hat{c})|$ and $\theta(t(\hat{c}))$ is the rate of any ordinary firing represented by symbolic firing $t(\hat{c})$. Observe that the constraints imposed in the definition of the transition firing rates (see Definition 1) guarantees that all ordinary firings grouped into a symbolic firing are assigned the same rate.

In our example of $\hat{m}_1[TxAck(c \leftarrow 3, r \leftarrow \mathrm{REC}_2))\hat{m}_3$, the rate is given by 2η where $1/\eta$ is the average delay experienced by an acknowledge packet traversing the network from the receiver to the sender, while the factor 2 accounts for the fact that the above symbolic firing instance represents 2 ordinary firings (as shown in Figure 7).

3.5 Computation of performance indices from the lumped CTMC

Let us spend a few words on what kind of performance indices can be defined on an SWN model: as in GSPN models, starting from the steady state probability distribution of all the markings in the reachability set, it is possible to define more high level indices like, for example, the probability distribution of the number of tokens into places, transition throughputs, or other user defined performance indices. A convenient way of defining these performance indices is through *reward functions* (see Chapter [2]).

However, since now tokens are coloured, and the firing refers to transition instances, one may choose whether to compute the distribution of *coloured* tokens into places, or rather the distribution of tokens into places independently on the colour, or even something in between the two, like for example the probability distribution of tokens into a given place, partitioned on static subsets. Similarly, one may be interested in the overall throughput of a given transition, independently on the instance, or may want to know the throughput of a specific instance or of subsets of instances with common characteristics.

Examples of *high-level* performance indices in our running examples are: average time required to send the whole pool of packets to the receiver(s) (this is a function of the throughput of transition Restart), utilisation of the network (function of the token probability distribution of place Limit), ratio between the number of useless (i.e., old) and useful received packets (function of the throughputs of transitions OldPck and RecPck).

In the model with $n > 1$ receivers, one may want to ask questions on the performance of a specific receiver, e.g. the average number of packets into the receiver input buffer: due to the constraint in the definition of transition delays it is possible to efficiently compute this kind of indices exploiting the fact that the receivers behave *homogeneously*, both from a qualitative and from a quantitative point of view, and that the average number of packets in the input buffer is the same for all receivers.

3.6 Discussion

We conclude this section by summarising the main ideas and advantages of the SRG technique: given an SWN model, comprising one or more colour classes, the technique allows to automatically exploit the behavioural symmetries of the system through the use of the symbolic marking representation and the symbolic firing rule. This can be done thanks to the particular way of defining places colour domain and arc expressions peculiar of SWNs, which guarantee that any pair of markings which are equal up to permutation of objects within basic

colour classes are equivalent, i.e. lead to the same future behaviour. Symbolic markings represent equivalence classes of ordinary markings. The symbolic firing rule allows to generate all symbolic markings reachable from a given symbolic marking, so that the RG generation algorithm can be easily adapted to directly generate the SRG, whose size is usually much smaller than that of the RG, while retaining enough information to prove most interesting qualitative properties. The other important feature of this technique concerns the performance analysis of SWN systems, in fact a lumped CTMC can be *directly* derived from the SRG allowing to compute the same performance indices that could be computed for the complete CTMC. Moreover, the notion of symbolic marking and symbolic firing have also been used to improve the efficiency of event driven simulation of SWN models [15, 46].

Finally, observe that the SRG captures only those equivalences which are due to permutations (rotations) of objects belonging to the same colour class while it cannot capture other forms of equivalences, like for example those based on the idea of making all old packets and old acknowledges in the protocol example indistinguishable, presented in Chapter [37] and in [40].

Moreover, it might be the case that the way of composing colour classes to model a system can in some not very intuitive case lead to less aggregation than a more clever model could do (see [16] for examples on this topic).

4 Stochastic Process Algebras

Stochastic Process Algebras (SPA) [5, 8, 29, 33] are a timed extension of Process Algebras [43, 34] introduced for the specification, understanding, and performance analysis of concurrent systems composed of entities that execute independently and cooperate through communication.

We will describe here a *generic* SPA language, which has most of the basic ingredients of the stochastic algebraic languages proposed in the literature. Systems are described as interactions of *components* that can perform a set of activities described as pairs (α, r), where α is the *type* of the activity and $r \in \mathbb{R}^+$ is the parameter of the negative exponential distribution governing its duration. Whenever a process P can perform an action, a duration is sampled: the resulting number specifies how long it will take to *complete* the action. If there are several concurrently enabled actions which are in conflict, the conflict is solved by means of a *race* policy, like was the case of SPNs [2].

SPA terms may be written considering a core set of language operators. The syntax for language terms can be defined by the grammar below.

$$P ::= Nil \mid (\alpha, r).P \mid P + Q \mid P \|_S Q \mid P/H \mid A$$

The names and the intuitive meanings of the operators are the following:

- **Inactivity:** Nil represents a process that cannot perform any action.
- **Prefix:** $(\alpha, r).P$ is the process that after a certain delay (negative exponentially distributed) performs an action of type α and then behaves like

P. Actions can be *active* or *passive*, the second being executable only when synchronised with active actions; the rate of passive actions is unspecified. Some languages [5, 29] allow also actions that happens in zero time with priority over timed actions.

- **Choice:** The component $P + Q$ enables activities of P and Q. A race policy governs the dynamic behaviour of this language expression. All the enabled activities are attempting to proceed, but only that with the *shorter* duration will succeed and determine the future behaviour of $P + Q$.
- **Cooperation:** The component $P \|_S Q$ represents a system where the processes P and Q are executing some of their actions independently and sometimes work together to perform some *common* activities. The cooperation between components follows the CSP style of communication [34]. The set S is called the *synchronisation* or *cooperation* set and defines the action types on which the components must synchronise.
  Activities with action types in the set S are assumed to require the simultaneous involvement of both components. The resulting activity will have the same action type as the two contributing activities and a rate computed as a function of the rates of the activities participating in the synchronisation. An interesting discussion about the different proposals for the computation of synchronisation rate may be found in [31].
  When the set S is empty, $\|_S$ has the effect of parallel composition, allowing components to proceed concurrently without any interaction between them (in this case the concise notation $P \| Q$ is usually used).
- **Hiding:** P/H is a process that cannot perform any action belonging to the set H. These actions appear as the *invisible* action τ (they can be regarded as internal delays) and they are not accessible for cooperation.
- **Constant:** Constants are components whose meaning is given by defining an equation such as $A \stackrel{\text{def}}{=} P$ that gives the constant A the behaviour of P. Constants are used to define recursive behaviours.

The formal semantics for the operators is given following the structural operational semantics style of Plotkin [45], the interested reader can refer to the literature for further details about this topic.

Each component P is characterised by the set of reachable states. Applying the semantic rules a *transition diagram* may be associated with each SPA term. This transition diagram can be viewed as an alternative way for describing the behaviour of the system: the nodes are labelled by elements of the set of reachable states (i.e., language expressions), and the arcs are labelled by pairs (*action type*, r) describing the actions that caused the state change. Notice the analogy existing between the transition diagram underlying an SPA model and the reachability graph underlying an SPN model.

The Markov process underlying any finite SPA component can be obtained directly from the transition diagram: a state of the Markov process is associated with each node of the graph and the transitions between states are defined by the arcs of the graph. Since all activity durations are exponentially distributed, the total transition rate between two states will be the sum of the activity rates

labelling arcs connecting the corresponding nodes in the transition diagram.

One of the most important aspect of process algebras, and of SPAs, is the possibility of defining equivalence relation between processes [43] which can be used to compare process specifications and to replace one specification by another one which exhibits an *equivalent* behaviour, but possibly has a different representation. Several notions of equivalence have been defined for SPAs. Some of them consider only the functional aspects of the components, some consider only the temporal aspects, and other consider both aspects.

When we consider the functional aspects only, we can apply to SPA specifications a well known notion of equivalence: *strong bisimulation*; it was first introduced in [44] and formally defined for CCS in [43] and it is fundamental in the untimed process algebra theory. Two components P and Q are considered equivalent if they cannot be distinguished by an external observer: essentially anything either one can do is matched by something the other can do and afterwards they remain equivalent. Hence, if P and Q are strongly bisimilar, any action, including the invisible action τ, performed by one must be matched by the other. The notion of strong bisimulation is very strict and makes distinction between components even if they differ in their internal behaviours only. A weaker notion of equivalence which disregards from τ actions (i.e. from the internal behaviour) and considers only the observable actions of the components (i.e. their external behaviour) has also been defined [43].

Another fundamental notion in the process algebras theory is the notion of *congruence*. An equivalence relation is a congruence when it is preserved by all combinators of the language. This means that, whenever two components P and Q are equivalent with respect to a given equivalence notion which has been proved to be a congruence, we can compose them with a third component obtaining two models that are still equivalent. Alternatively, we can say that the two components P and Q can be interchanged in any complex system S, with confidence that its behaviour remains the same.

Again we refer to the literature for more information on equivalence notions and congruences while in Section 5.2 we will briefly discuss one equivalence notion which takes into account both functional and temporal aspects and has been defined for model aggregation.

4.1 The protocol example

We describe now the SPA model of the protocol example already presented in Section 2.1. We will use timed actions only but for readability we will not introduce rates in the specification. The actions will be represented by their types only and we assume that those with the same type have the same rate.
The compositional technique for the model definition requires to specify first the components in isolation and subsequently to define their interactions. Several processes have been identified: the *sender*, which is responsible for sending the data packets, the *receiver*, which is responsible of the reception of the packets and of the sending of acknowledge messages, and the *network*, which is composed of several *buffers* for the storing of different types of messages. The

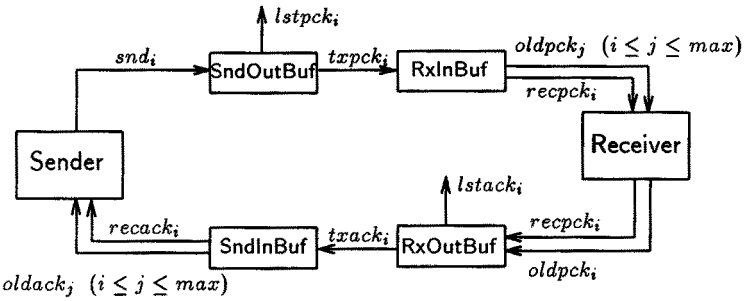

Fig. 8. Abstract view of the protocol example.

network component is in turn composed of smaller subcomponents, each one modelling a single buffer. Moreover we have a component responsible for limiting the number of messages simultaneously circulating in the network. Fig. 8 shows an abstract view of the components involved in the system except the one which is responsible of limiting the number of messages concurrently transmitted (we have omitted it to avoid cluttering the schema with too many arcs).

Each component is *sequential* i.e., written by means of prefix, choice and constant operators. The interactions among the components are specified making use of the parallel composition operator and proper synchronisation sets. In order to facilitate the understanding of the components described below the have used action names similar[11] to transitions labels in the net shown in Fig. 4.

Let us now go into the details of the sender: it must send a pool of packets and keeps sending the same packet until it has received a proper acknowledge message from the receiver. A possible specification of such a behaviour in the case a pool of max data packets could be the following:

$$NextSnd_0 \stackrel{\text{def}}{=} snd_1.NextSnd_0 + recack_1.NextSnd_1$$
$$NextSnd_1 \stackrel{\text{def}}{=} snd_2.NextSnd_1 + oldack_1.NextSnd_1 + recack_2.NextSnd_2$$
$$\ldots$$
$$NextSnd_i \stackrel{\text{def}}{=} snd_{i+1}.NextSnd_i + oldack_i.NextSnd_i + recack_{i+1}.NextSnd_{i+1}$$
$$\ldots$$
$$NextSnd_{max} \stackrel{\text{def}}{=} oldack_{max}.NextSnd_{max} + restart.NextSnd_0$$

In the initial state, $NextSnd_0$, two possible actions are enabled: snd_1, modelling the sending of the first data packet, and $recack_1$, modelling the reception of the acknowledge message for the first data packet. The pessimistic strategy requires that the sender keeps transmitting a packet until it receives an acknowledge for it. This behaviour is obtained by keeping the sender in the state $NextSnd_0$ until the action $recack_1$ is performed. After the execution of $recack_1$, the component moves to the next state $NextSnd_1$. Notice that the choice between the two

[11] We have chosen to write action names using small letters only, to distinguish them from the names of the components.

actions snd_1 and $recack_1$ is random at this level of the component definition, while in the complete model it will depend on which action is offered by the other components. These two actions in fact (and many others) will form the cooperation sets on which components must synchronise. We will return to this notion when forming the complete model of the system.

Starting from the state $NextSnd_1$ three actions are enabled: snd_2, modelling the sending of the second data packet, $oldack_1$, modelling the reception of old acknowledge messages, and $recack_2$ representing the reception of the acknowledge message for the second data packet. When this new acknowledge message is received the component evolves into $NextSnd_3$, and so on, until it reaches the state $NextSnd_{max}$ in which only old acknowledges can be received until a *restart* action, that moves back to the initial state, is performed. Notice that the execution of action $oldack_i$ represents a situation in which the sender is receiving an acknowledge for a message of index j with $1 \leq j \leq i$, i.e. an old acknowledge, while its internal state is $NextSnd_i$.

The receiver is modelled by a component, called $NextRec_0$, which has the following specification:

$$
\begin{aligned}
NextRec_0 &\stackrel{\text{def}}{=} recpck_1.NextRec_1 \\
NextRec_1 &\stackrel{\text{def}}{=} recpck_2.NextRec_2 + oldpck_1.NextRec_1 \\
&\ldots \\
NextRec_i &\stackrel{\text{def}}{=} recpck_{i+1}.NextRec_{i+1} + oldpck_i.NextRec_i \\
&\ldots \\
NextRec_{max} &\stackrel{\text{def}}{=} oldpck_{max}.NextRec_{max} + restart.NextRec_0
\end{aligned}
$$

The first action that the receiver can perform, $recpck_1$, represents the reception of the first data packet. Afterwards the second data packet ($recpck_2$) or old data packets ($oldpck_1$) can be received. This behaviour is repeated until all data packets have been received and the receiver evolves into $NextRec_{max}$, state in which only old packets can be received ($oldpck_{max}$) until a *restart* action is performed. Notice that any action $oldpck_i$ represents the reception of old packets numbered $1, 2, \ldots, i$, while the receiver was expecting a packet numbered $i + 1$.

Several components are used to model the buffers of the network; we will describe in detail the $SndOutBuf$ buffer connecting the sender to the network, all the other buffers having a similar specification.

$$
\begin{aligned}
&SndOutBuf_0^i \stackrel{\text{def}}{=} snd_i.SndOutBuf_1^i \\
&SndOutBuf_1^i \stackrel{\text{def}}{=} snd_i.SndOutBuf_2^i + txpck_i.SndOutBuf_0^i + lstpck_i.SndOutBuf_0^i \\
&\ldots \\
&SndOutBuf_k^i \stackrel{\text{def}}{=} txpck_i.SndOutBuf_{k-1}^i + lstpck_i.SndOutBuf_{k-i}^i
\end{aligned}
$$

The variable i may assume values $i = 1, \ldots, max$ corresponding to the number associated with data packets that can circulate in the network: a buffer that can contain packets of max different numbers is specified as the parallel composition of max buffers, one for each packet number.

The state $SndOutBuf_0^i$ offers a snd_i action, representing a situation in which the buffer can accept a data packet number i. When one packet is stored in the buffer, the component evolves into $SndOutBuf_1^i$ where three actions are enabled. A new packet number i can be stored in the buffer (snd_i), the packet can be transmitted through the network ($txpck_i$) or the packet can be lost ($lstpck_i$). The choice of the action to be performed will be determined by what is offered by cooperating components. The same actions are offered in the other states until the buffer is full ($SndOutBuf_k^i$). Notice that, until we do not fix the value of k, we are modelling buffers with an unknown number of positions.

All the other buffers have similar specifications which are written below without any comment.

$$RxInBuf_0^i \stackrel{\text{def}}{=} txpck_i.RxInBuf_1^i$$

$$RxInBuf_1^i \stackrel{\text{def}}{=} txpck_i.RxInBuf_2^i + \sum_{j=i}^{max} oldpck_j.RxInBuf_0^i + recpck_i.RxInBuf_0^i$$

...

$$RxInBuf_k^i \stackrel{\text{def}}{=} \sum_{j=i}^{max} oldpck_j.RxInBuf_{k-1}^i + recpck_i.RxInBuf_{k-1}^i$$

$$RxOutBuf_0^i \stackrel{\text{def}}{=} recpck_i.RxOutBuf_1^i + oldpck_i.RxOutBuf_1^i$$

$$RxOutBuf_1^i \stackrel{\text{def}}{=} recpck_i.RxOutBuf_2^i + oldpck_i.RxOutBuf_2^i + lstack_i.RxOutBuf_0^i + txack_i.RxOut.Buf_0^i$$

...

$$RxOutBuf_k^i \stackrel{\text{def}}{=} lstack_i.RxOutBuf_{k-1}^i + txack_i.RxOut.Buf_{k-1}^i$$

$$SndInBuf_0^i \stackrel{\text{def}}{=} txack_i.SndInBuf_1^i$$

$$SndInBuf_1^i \stackrel{\text{def}}{=} txack_i.SndInBuf_2^i + recack_i.SndInBuf_0^i + \sum_{j=i}^{max} oldack_j.SndInBuf_0^i$$

...

$$SndInBuf_k^i \stackrel{\text{def}}{=} recack_i.SndInBuf_{k-1}^i + \sum_{j=i}^{max} oldack_j.SndInBuf_{k-1}^i$$

The Lim component is responsible of limiting the maximum number of messages simultaneously present in the network. In the net of Fig. 5 the initial marking of place Lim is equal to two, meaning that we admit at most two messages. This is obtained in SPA using the following specification:

$$\begin{aligned}
Lim_2 &\stackrel{\text{def}}{=} snd_1.Lim_1 + \ldots + snd_{max}.Lim_1 + restart.Lim_2 \\
Lim_1 &\stackrel{\text{def}}{=} snd_1.Lim_0 + \ldots + snd_{max}.Lim_0 + \\
&\quad lstpck_1.Lim_2 + \ldots + lstpck_{max}.Lim_2 + lstack_1.Lim_2 + \ldots + \\
&\quad lstack_{max}.Lim_2 + recack_1.Lim_2 + \ldots + recack_{max}.Lim_2 + \\
&\quad oldack_1.Lim_2 + \ldots + oldack_{max}.Lim_2 \\
Lim_0 &\stackrel{\text{def}}{=} lstpck_1.Lim_1 + \ldots + lstpck_{max}.Lim_1 + lstack_1.Lim_1 + \ldots + \\
&\quad lstack_{max}.Lim_1 + recack_1.Lim_1 + \ldots + recack_{max}.Lim_1 + \\
&\quad oldack_1.Lim_1 + \ldots + oldack_{max}.Lim_1
\end{aligned}$$

Notice that, once we have fixed the number of concurrently circulating messages,

we can also fix the minimum size of the buffers ($k = 2$ in this case). Of course we could also have buffers with more positions but we can be sure that these positions will never be occupied.

Finally, we can obtain the model of the whole system by specifying the interactions among all the components; the complete protocol specification is thus defined as:

$$Protocol \stackrel{\text{def}}{=} Lim_2 \|_{S_1} (NextSnd_0 \|_{S_2} ((SndOutBuf_0^1 \| \ldots \| SndOutBuf_0^{max}) \|_{S_3} (RxInBuf_0^1 \| \ldots \| RxInBuf_0^{max}) \|_{S_4} NextRec_0 \|_{S_4} (RxOutBuf_0^1 \| \ldots \| RxOutBuf_0^{max}) \|_{S_5} (SndInBuf_0^1 \| \ldots \| SndInBuf_0^{max})))$$

$$S_1 = \{snd_1, \ldots, snd_{max}, lstack_1, \ldots, lstack_{max}, lstpck_1, \ldots, lstpck_{max}, recack_1, \ldots, recack_{max}, oldack_1, \ldots, oldack_{max}, restart\}$$
$$S_2 = \{snd_1, \ldots, snd_{max}, recack_1, \ldots, recack_{max}, oldack_1, \ldots, oldack_{max}\}$$
$$S_3 = \{txpck_1, \ldots, txpck_{max}\}$$
$$S_4 = \{recpck_1, \ldots, recpck_{max}, oldpck_1, \ldots, oldpck_{max}\}$$
$$S_5 = \{txack_1, \ldots, txack_{max}\}$$

We have computed the ordinary state space underlying this model for different numbers of data packets and different numbers of messages concurrently sent in the net using the TIPP tool [30]. Some results are listed in Table 2.

Lim	Packets	States	Transitions
2	2	83	282
2	3	172	608
2	4	293	1058
2	5	446	1632
3	2	299	1282
3	3	846	3876
3	4	1829	8690

Table 2. Number of ordinary states for the *Protocol* specification.

We end this section inviting the reader to observe that there exists a close relation between places and transitions in the net model of Fig. 5 and (subcomponent) derivatives and actions in the SPA model presented here. For example, place NextSnd, which models the sender, is the input place of transition Send[12], which models the sending of a data packet. After its firing, a data packet is stored into the buffer connecting the sender to the network, modelled by place SndOutBuf. The same behaviour is obtained in the SPA specification by imposing a synchronisation on action type snd_1 between the processes $NextSnd_0$ and $SndOutBuf_0^1$.

[12] We do not consider place Send whose marking never changes during the net evolution.

The mapping of nets models into algebraic specification, or vice-versa, requires some ingenuity. However, rules have been identified to guide the translation process and the interested reader may refer to the literature for major details about this topic [4, 48].

5 Aggregation in SPA

SPAs, as well as SPNs, suffer from the so-called state space explosion problem: to cope with this problem aggregation techniques have been defined in this framework too.

We will describe in this section an aggregation technique based on a notion of equivalence which has been firstly defined in Performance Evaluation Process Algebra (PEPA) [33] where has been called *strong equivalence* [32]. We will use the notation proposed for PEPA but most of the considerations we will discuss can also be extended to other languages [5, 29] by just adapting the notation.

First we need some definitions to understand this aggregation technique. We recall that in PEPA the reachable states (the language terms) are called *derivatives*, the set of all reachable states is the *derivative set* (DS), the transition diagram is called *derivation graph* (DG), and the aggregated transition diagram is called *lumped* derivation graph (LDG).

The *transition rate* between two components C_i and C_j, denoted by $q(C_i, C_j)$, is the sum of the activity rates labelling arcs connecting node C_i to node C_j. The *conditional transition rate* from C_i to C_j via an action type α is denoted by $q(C_i, C_j, \alpha)$. This is the sum of the rates corresponding to actions of type α labelling arcs connecting the corresponding nodes in the DG. The conditional transition rate is thus the rate at which a system behaving as component C_i evolves to the behaviour of component C_j after having completed an activity of type α.

If we consider a *set* of possible derivatives S, the *total conditional transition rate* from C_i to S, denoted $q[C_i, S, \alpha]$, is equal to the sum of the conditional transition rates from C_i to components C_j belonging to S:

$$q[C_i, S, \alpha] = \sum_{C_j \in S} q(C_i, C_j, \alpha)$$

The concept of total conditional transition rate is the basis for the definition of strong equivalence since two PEPA components are considered strongly equivalent if for any action type α, the total conditional transition rates from those components to any equivalence class, via activities of this type, are the same.

The notion of strong equivalence has been used in [32] to define a procedure for the generation of aggregated Markov processes underlying SPA models. The state space of an SPA model is partitioned into equivalence classes. Once the equivalence classes have been identified, they form the states of the aggregated state space and they label the nodes of an LDG. Instead of deriving the CTMC from the DG underlying the model, it is possible to derive a lumped CTMC starting from the LDG which has been built. The LDG and the aggregated

CTMC are isomorphic: each state in the LDG corresponds to a state in the Markov process, and the transition rates between nodes are the sum of the total conditional transition rates attached to the arcs connecting them.

Unfortunately, the partitioning of the DS still requires its generation for the complete model. In some cases this set can be so large that even the aggregation of the model is infeasible. However, all the benefits of this technique can be obtained by taking advantage of the fact that strong equivalence is a congruence, and by exploiting the compositional structure of the model [32]. Instead of constructing the complete DS of the entire model and then partitioning it into equivalence classes, it is possible to generate *partial* DSs by considering pairs of cooperating components in turn. These partial DSs can be aggregated and the corresponding lumped components can be computed by considering the associated LDGs. Furthermore, each pair of components can be replaced by the lumped one into the original model, obtaining a new model which is strongly equivalent to the original (i.e. its behaviour is maintained) but, in general, has a smaller DS.

Let us informally explain this procedure by considering the expression $Sys \stackrel{\text{def}}{=} P \|_{S_1} Q \|_S R \|_{S_2} T$, where P, Q, R and T are the model components. We can for example compute the partial DSs underlying $P \|_{S_1} Q$ and $R \|_{S_2} T$, partition them into equivalence classes, compute the underlying LDGs, and form the new (aggregated) components[13], say $\hat{PQ}$ and $\hat{RT}$. These new components exhibit exactly the same behaviour of $P \|_{S_1} Q$ and $R \|_{S_2} T$, respectively, and can be substituted in Sys having confidence that its behaviour will be maintained. Hence we obtain the new specification $Sys' \stackrel{\text{def}}{=} \hat{PQ} \|_S \hat{RT}$ which has the same behaviour of Sys but less states.

Notice that the DS of the original model does not need to be constructed, and that no CTMC is derived until the aggregation procedure is complete. Details of the procedure that realises the compositional aggregation technique may be found in [32, 33].

The aggregated Markov chain which can be obtained by means of strong equivalence satisfies the lumpability conditions and therefore all the states belonging to a single aggregate move towards the other aggregates with the same probabilities. However, the lumpability is not strict [7] (see Section 3.4) and therefore equivalent states within the same aggregate are not equiprobable. This means that, once we know the probability of being into a single aggregate, we cannot compute the probability of being into a single ordinary state by simply dividing this probability by the number of states within the macro-state; also the performance indices that can be computed can refer to macro-states only.

5.1 Horizontal and vertical aggregation

We discuss in this section some forms of aggregation which are possible when using the strong equivalence relation; further details may be found in [47].

[13] An aggregated component is derived from the LDG in such a way that its (ordinary) state space has exactly the same states of the LDG.

Let us consider a simple model $P \stackrel{\text{def}}{=} P_1 \| P_1$ where P_1 is specified as follows:

$$P_1 \stackrel{\text{def}}{=} (\alpha, r_1).P_2 \qquad\qquad P_2 \stackrel{\text{def}}{=} (\beta, r_2).P_3$$

The DS underlying the model is:

$$DS(P) = \{P_1\|P_1, P_1\|P_2, P_2\|P_1, P_2\|P_2, P_1\|P_3, P_3\|P_1, P_2\|P_3, P_3\|P_2, P_3\|P_3\}$$

The DG of P has the so called *diamond* structure and it is shown in Fig. 9(a). The states $P_1\|P_2$ and $P_2\|P_1$ are strongly equivalent because they enable actions

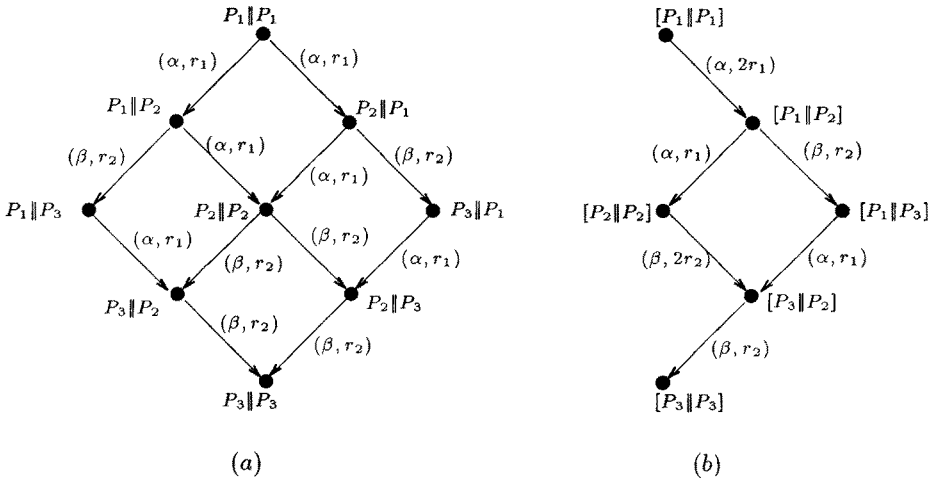

Fig. 9. DG and LDG of $P \stackrel{\text{def}}{=} P_1\|P_1$.

of the same type (α, resp. β) with the same total conditional transition rate (r_1, resp. r_2) and, afterwards, they reach equivalent states ($P_2\|P_2$, resp. $P_1\|P_3$ and $P_3\|P_1$). Hence $P_1\|P_2$ and $P_2\|P_1$ belong to the same equivalence class which we denote as $[P_1\|P_2]$. The same reasoning holds for the pairs $P_1\|P_3$ and $P_3\|P_1$, $P_2\|P_3$ and $P_3\|P_2$, i.e. for those derivatives that are parallel composition of the same set of components but in different order. Therefore we can partition the DS of P into equivalence classes as follows:

$$\begin{array}{ll} [P_1\|P_1] = \{P_1\|P_1\} & [P_1\|P_2] = \{P_1\|P_2, P_2\|P_1\} \\ [P_1\|P_3] = \{P_1\|P_3, P_3\|P_1\} & [P_2\|P_3] = \{P_2\|P_3, P_3\|P_2\} \\ [P_3\|P_3] = \{P_3\|P_3\} & \end{array}$$

If we consider only the equivalence classes instead of the ordinary derivatives we obtain the LDG depicted in Fig. 9(b) and the corresponding aggregated

component[14] could be written as follows:

$$Q_1 \stackrel{\text{def}}{=} (\alpha, 2r_1).Q_2 \qquad Q_2 \stackrel{\text{def}}{=} (\alpha, r_1).Q_3 + (\beta, r_2).Q_4$$
$$Q_3 \stackrel{\text{def}}{=} (\beta, 2r_2).Q_5 \qquad Q_4 \stackrel{\text{def}}{=} (\alpha, r_1).Q_5$$
$$Q_5 \stackrel{\text{def}}{=} (\beta, r_2).Q_6$$

This kind of reduction in the derivation graph takes into account only one among different, but equivalent, interleavings and graphically it can be seen as an *horizontal* aggregation in which equivalent nodes which are at the same level within the graph are folded together. It is easy to see such folding comparing the two graphs in Fig. 9. We invite the reader to observe that this idea of considering equivalent those derivatives that can be obtained by permutation of the components within the parallel composition operator closely reminds the idea of permutation of similar objects within dynamic subclasses discussed in Section 3 for the SWN formalism.

The strong equivalence relation allows also another kind of reduction of the state space which does not consider permutations of the same components within derivatives but repeated patterns of behaviour within the specification of the components. We call this reduction *vertical* aggregation in contrast to the horizontal one just discussed. In this case we consider as belonging to the same equivalence class those states characterised by the same pattern of behaviour, i.e. those states that execute actions of the same type, with the same total transition rates, and afterwards reach states which are still equivalent.

Let us again explain this fact on the following simple example whose DG is shown in Fig. 10(a).

$$P_1 \stackrel{\text{def}}{=} (\alpha, r_1).P_2 \qquad P_2 \stackrel{\text{def}}{=} (\beta, r_2).P_1 + (\beta, r_2).P_3$$
$$P_3 \stackrel{\text{def}}{=} (\alpha, r_1).P_4 \qquad P_4 \stackrel{\text{def}}{=} (\beta, 2 \cdot r_2).P_1$$

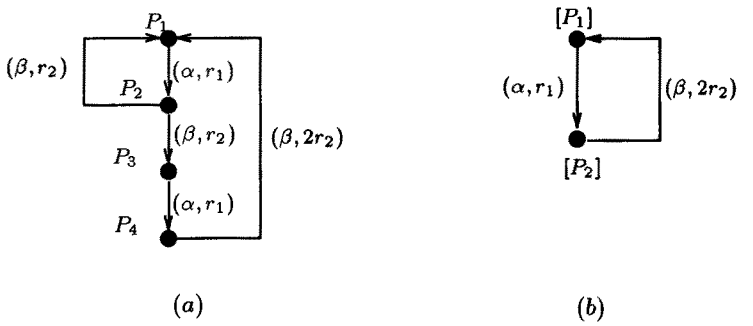

Fig. 10. DG and LDG of P_1.

[14] When we write an aggregated component starting from an LDG we associate arbitrary names with the derivatives.

In the component described above we can identify two equivalence classes $[P_1] = \{P_1, P_3\}$ and $[P_2] = \{P_2, P_4\}$ which form the states of the LDG drawn in the left part of Fig. 10. In this case we have identified a repeated pattern of behaviour within a single component and we have added to the same equivalence class those states which generate such a repetitive behaviour. Hence we can write a new model whose underlying transition diagram is exactly the LDG of Fig. 10(b); a possible specification is:

$$Q_1 \stackrel{\text{def}}{=} (\alpha, r_1).Q_2 \qquad Q_2 \stackrel{\text{def}}{=} (\beta, 2r_2).Q_1$$

Notice that this type of equivalence is in general difficult to capture with the SRG technique because it is difficult to characterise it syntactically on the state description since it refers to symmetries which are hidden in the behaviour of the model.

5.2 Aggregation of the protocol example

We discuss in the following sections different forms of aggregation which can be computed on the SPA model of the protocol example we have introduced in Section 4.1. First we show how it is possible to find equivalent states within the state space when we ignore the identities of data packets and acknowledges that are *old*. In Section 5.2 we consider also timing information and we discuss the aggregation which can be obtained by applying strong equivalence. Finally, in Section 5.2 we modify our model to obtain the protocol example with multiple receivers and we discuss the new form of aggregation which can be obtained with this modified specification.

Aggregation due to old packets and old acknowledges Let us explain the aggregation due to old packets and old acknowledges [40] through an example: suppose that the receiver is in state $NextRec_j$ expecting a data packet number $j+1$. Any data packet of number i less that j will not change its state and will be considered an old packet. When the receiver gets an old packet, it executes the action $oldpck_j$ producing an acknowledge message for the sender to let it know that the next expected packet is that numbered $j+1$.

As we have just said, the reception of an old packet does not change the future behaviour of the receiver, independently of the actual identity of the old packet itself. This means that we can consider as equivalent all those states that differ only for the identity of the old packets circulating in the network since they have the same future behaviour. Therefore we can group them into the same equivalence class.

The same reasoning holds when we consider old acknowledges instead of old packets and the sender instead of the receiver. Here we simply recall that an acknowledge is old when the sender is in a state $NextSnd_j$ waiting for the acknowledge for packet j ($recack_j$) and acknowledges of messages $i < j$ are on their way to the sender.

The key point of the model that allows the exploitation of this equivalence is in the $RxInBuf$ and $SndInBuf$ definitions. Let us explain the $RxInBuf$ buffer, the case of old acknowledges being similar. If we consider the case of two data packets and Lim equal to two, the specification is the following:

$$
\begin{aligned}
RxInBuf_0^1 &\stackrel{\text{def}}{=} txpck_1.RxInBuf_1^1 \\
RxInBuf_1^1 &\stackrel{\text{def}}{=} txpck_1.RxInBuf_2^1 + \\
&\quad recpck_1.RxInBuf_0^1 + oldpck_1.RxInBuf_0^1 + oldpck_2.RxInBuf_0^1 \\
RxInBuf_2^1 &\stackrel{\text{def}}{=} recpck_1.RxInBuf_1^1 + oldpck_1.RxInBuf_1^1 + oldpck_2.RxInBuf_1^1
\end{aligned}
$$

$$
\begin{aligned}
RxInBuf_0^2 &\stackrel{\text{def}}{=} txpck_2.RxInBuf_1^2 \\
RxInBuf_1^2 &\stackrel{\text{def}}{=} txpck_2.RxInBuf_2^2 + \\
&\quad recpck_2.RxInBuf_0^2 + oldpck_2.RxInBuf_0^2 \\
RxInBuf_2^2 &\stackrel{\text{def}}{=} recpck_2.RxInBuf_1^2 + oldpck_2.RxInBuf_1^2
\end{aligned}
$$

We have two instances of the buffer, one for each type of packet. When the buffer for packets number i is empty and there are less than two circulating messages, the network can transmit a packet and the buffer moves to a new state where only one position is available. In this situation eventually the network can transmit a second packet of type i, filling the buffer, or the receiver can get a message which can be new ($recpck_i$) or can be old ($oldpck_j$).

The buffer for data packets number 1 can perform both actions $oldpck_1$ and $oldpck_2$ which have the following meaning: *the receiver is in state 1 or 2 and one old packet number 1 is received.* The buffer for packets number 2 can perform only the action $oldpck_2$ with the following meaning: *the receiver is in state 2 and one old packet number 2 is received.*

When the receiver is in state $NextRec_2$ it performs the same action $oldpck_2$ for all old packets number 1 and number 2, and afterwards the same state is reached in both cases. Therefore these states that differ only in the number associated with old packets are equivalent.

If now we hide actions $txpck_i, lstpck_i, txack_i, lstack_i, i = 1, 2$ we obtain a model whose abstract view is shown in Fig. 11. The network is now a *black box*: the only visible actions involve the interactions between the network and the sender, and the network and the receiver while the interactions between the buffers within the network are completely hidden.

By applying strong bisimulation to this model we obtained the aggregated results listed in Table 3 which coincide with those of the CPN example presented in Chapter [37].

Before ending this section let us show in detail one example of equivalence class in the case of $Lim = 2$ and two data packets. The following two states[15] resulted

[15] We have chosen to write only the components which are important for the discussion, the missing ones, indicated by (.), being exactly the same in both states.

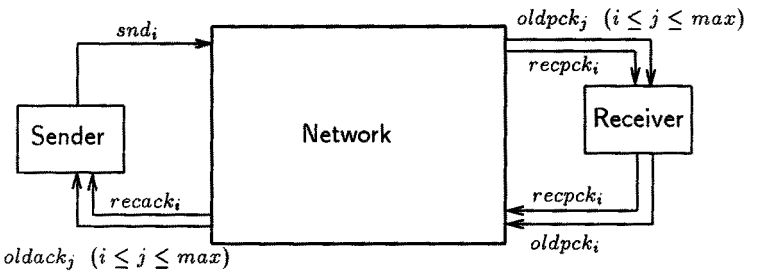

Fig. 11. Abstract view of the protocol example with hiding.

Lim	Packets	Ord. states	Equiv. classes
2	2	83	69
2	3	172	112
2	4	293	155
2	5	446	198
3	2	299	204
3	3	846	348
3	4	1829	492

Table 3. Condensed state space of the *Protocol* specification.

to be equivalent:

$$Lim_0 \|_{S_1} (NextSnd_1 \|_{S_2} (.) \|_{S_3} (\mathbf{RxInBuf_1^1} \|_{S_4} RxInBuf_0^2) \|_{S_4} NextRec_2 \|_{S_4} (RxOutBuf_0^1 \|_{S_5} RxOutBuf_1^2) \|_{S_5} (.))$$
$$Lim_0 \|_{S_1} (NextSnd_1 \|_{S_2} (.) \|_{S_3} (RxInBuf_0^1 \|_{S_4} \mathbf{RxInBuf_1^2}) \|_{S_4} NextRec_2 \|_{S_5} (RxOutBuf_0^1 \|_{S_5} RxOutBuf_1^2) \|_{S_5} (.))$$

In both cases, in fact, there are two circulating messages (Lim_0), the receiver has already got both data packets ($NextRec_2$) and the sender is still waiting for the acknowledges for the second data packet ($NextSnd_1$). The only difference between these two states is the content of the buffer $RxInBuf$. However, since in both cases the packets in this buffer are old, there is no distinction between them, i.e. they belong to the same equivalence class.

Adding timing specification We have discussed so far an algebraic specification without considering any timing information. When we add proper rates to our actions we obtain the corresponding stochastic model suitable for the computation of performance measures.

Moreover, when we apply the notion of strong equivalence discussed in Section 5 to our specification, we obtain less aggregation on the state space with respect to that obtained with strong bisimulation. This is due to the fact that

in this case we are taking into account not only action types but also the total conditional transition rates exiting from the derivatives.

Let us consider the following three states which enable actions $oldpck_2$:

$$Lim_0 \| _{S_1} (NextSnd_2 \| _{S_2} (.) \| _{S_3} (\mathbf{RxInBuf_1^1} \| \mathbf{RxInBuf_1^2}) \| _{S_4} NextRec_2 \| _{S_4} (.) \| _{S_5} (.))$$
$$Lim_0 \| _{S_1} (NextSnd_2 \| _{S_2} (.) \| _{S_3} (\mathbf{RxInBuf_1^1} \| \mathbf{RxInBuf_1^2}) \| _{S_4} NextRec_2 \| _{S_4} (.) \| _{S_5} (.))$$
$$Lim_0 \| _{S_1} (NextSnd_2 \| _{S_2} (.) \| _{S_3} (RxInBuf_0^1 \| \mathbf{RxInBuf_2^2}) \| _{S_4} NextRec_2 \| _{S_4} (.) \| _{S_5} (.))$$

If all $oldpck_i$ actions have the same rate λ_{oldpck}, the global rate for action type $oldpck_2$ exiting from the first two states is equal to $2 \cdot \lambda_{oldpck}$, since two instances of $oldpck_2$ are enabled. In the case of the third state instead the global rate for $oldpck_2$ is equal to λ_{oldpck} since only one instance is enabled. For this reason, the third state is kept separate from the first two when considering both functional and temporal aspects while, from a purely functional point of view, these three states are considered to be equivalent.

Notice that the distinction among these states is due to our choice of modelling every buffer as the parallel composition of several instances of the same component, one for each packet number, and to the adoption of the race policy for the solution of actions conflicts. This could not happen in a model with FIFO sequential buffers. As was discussed in Section 2.1, we are indeed modelling a multi threaded receiver able to process several packets at the same time, a more complex specification is required to represent a single threaded receiver.

The case of more receivers We have presented in Section 2.1 a different version of the protocol in which the sender has to broadcast its messages to several receivers. We have also shown that it is possible to detect the symmetries within this system due to the presence of replicas of the same component (the receiver).

Here we will discuss the same modified example in terms of an algebraic specification (without going into all the details of the model). In the net of Fig. 5 a new colour class REC has been introduced to distinguish among the receivers without the necessity of adding new subnets in the complete model. This colour class allows to distinguish among the packets received and the acknowledges sent by each receiver, and to keep track of the state of each single receiver.

In SPA there is no notion of *coloured* components so that we need to replicate several components when passing from the single receiver to the multiple receivers case. Since we need to distinguish not only the state of the different receivers (modelled by components $NextRec_i$) but also which receiver got a message or sent an acknowledge, we had to replicate also the sender component and most of the buffers. More precisely, we have grouped all the sequential components corresponding to portions of the net in which the new class REC has been introduced, into a *macro*-component that we have called $SndNetRec$ and then

replicated this component *nrec* times.

$$SndNetRec \stackrel{\text{def}}{=} NextSnd_0 \| (RxInBuf_0^1 \| \ldots \| RxInBuf_0^{max}) \|_{R_1} NextRec_0 \|_{R_1} (RxOutBuf_0^1 \| \ldots \| RxOutBuf_0^{max}) \|_{R_2} (SndInBuf_0^1 \| \ldots \| SndInBuf_0^{max})$$
$$R_1 = \{recpck_1, \ldots, recpck_{max}, oldpck_1, \ldots, oldpck_{max}, dscpck_0, \ldots, dscpck_{max}\}$$
$$R_2 = \{txack_1, \ldots, txack_{max}\}$$

Actually, the specification of the sender in this new model is slightly different from the previous one since now acknowledge messages can be received out of sequence due to the way the sender broadcasts the data packets to all the receivers. Hence, when the sender is in state i, it can receive an acknowledge for a packet of type j with j bigger that i. This models a correct situation in which some of the receivers already got their first j packets and acknowledged them while others received the j packets but some of the corresponding acknowledges have been lost. When this case is recognised the sender must *jump* forward into state j. A possible specification of such a behaviour could be the following:

$$\begin{aligned}
NextSnd_0 &\stackrel{\text{def}}{=} snd_1.NextSnd_0 + \\
&\quad recack_1.NextSnd_1 + \ldots + recack_{max}.NextSnd_{max} \\
NextSnd_1 &\stackrel{\text{def}}{=} snd_2.NextSnd_1 + oldack_1.NextSnd_1 + \\
&\quad recack_2.NextSnd_2 + \ldots + recack_{max}.NextSnd_{max} \\
&\ldots \\
NextSnd_i &\stackrel{\text{def}}{=} snd_{i+1}.NextSnd_i + oldack_i.NextSnd_i + \\
&\quad recack_{i+1}.NextSnd_{i+1} + \ldots + recack_{max}.NextSnd_{max} \\
&\ldots \\
NextSnd_{max} &\stackrel{\text{def}}{=} oldack_{max}.NextSnd_{max} + restart.NextSnd_0
\end{aligned}$$

Also the receivers can get packets which are out of sequence, i.e. *future* packets. On the receiver side, however, future packets must be discarded. This may happen when the packets are received in the wrong order (recall that the network does not guarantee that the packets are received in the same order as they are sent). The component belows model such a behaviour: we have introduced the actions $dscpck_i$ representing the discarding of out of sequence messages, i.e. the discarding of messages of type j which are received when the receiver is in a state i smaller than j. After having discarded an out of sequence packet, the receiver keeps waiting for the correct one.

$$\begin{aligned}
NextRec_0 &\stackrel{\text{def}}{=} recpck_1.NextRec_1 + dscpck_0.NextRec_0 \\
NextRec_1 &\stackrel{\text{def}}{=} recpck_2.NextRec_2 + oldpck_1.NextRec_1 + dscpck_1.NextRec_1 \\
&\ldots \\
NextRec_i &\stackrel{\text{def}}{=} recpck_{i+1}.NextRec_{i+1} + oldpck_i.NextRec_i + dscpck_i.NextRec_1 \\
&\ldots \\
NextRec_{max} &\stackrel{\text{def}}{=} oldpck_{max}.NextRec_{max} + restart.NextRec_0
\end{aligned}$$

To obtain the specification of the whole system with *nrec* receivers, *nrec* replicas of the macro-component must be composed by parallel composition. Moreover,

since the sending of a message implies a broadcast to all receivers, the macro-components must synchronise on actions $txpck_i$. Finally, the macro-components must synchronise with the *Lim* component and with the $SndOutBuf_0^i$ buffers (notice that these are the only components whose corresponding subnets do not make use of the REC colour class in the net of Fig. 5).

In the case of two receivers, two data packets, and two concurrent messages we obtain the following model:

$$TwoRec \stackrel{\text{def}}{=} Lim_2 \|_{S_1} (SndOutBuf_0^1 \| SndOutBuf_0^2) \|_{S_2} (SndNetRec \|_{S_3} SndNetRec)$$

$$\begin{aligned} S_1 &= \{snd_1, snd_2, lstack_1, lstack_2, lstpck_1, lstpck_2, recack_1, recack_2, \\ &\qquad oldack_1, oldack_2, restart\} \\ S_2 &= \{snd_1, snd_2, txpck_1, txpck_2\} \\ S_3 &= \{txpck_1, txpck_2\} \end{aligned}$$

By running the TIPP tool we have again computed the ordinary and the ag-

Recs	Lim	Packets	Ord. states	Equiv. classes
2	2	2	245	111
2	2	3	529	217
2	2	4	921	357
2	3	2	3609	-
2	3	3	14025	-
3	3	3	9775	-

Table 4. Condensed state space with multiple receivers.

gregated state spaces underlying different models obtaining the results listed in Table 4. If we analyse the numbers we obtained we can immediately observe that in the case of two receivers the aggregated state spaces have less than half of the states of the ordinary state spaces. This is due to the fact that we captured different types of aggregations: the horizontal aggregation, due to the interchanging of the receivers (corresponding to the aggregation by symmetry exploitation of the SRG), and two different vertical aggregations, one due to equivalences among old packets and old acknowledges (the same discussed in Section 5.2 for the case of a single receiver), the other due the reception of acknowledges which are out of sequence.

Let us explain this third form of aggregation considering the following states

which resulted to be equivalent:

$$
\begin{array}{ll}
Lim_1 \|_{S_1}(.) & \|_{S_2} NextSnd_2 \;\; \|(.)\|_{R_1} NextRec_2 \|_{R_1}(.)\|_{R_2}(SndInBuf_0^1 \| SndInBuf_1^0) \\
& \|_{S_3} \mathbf{NextSnd_0} \|(.)\|_{R_1} NextRec_2 \|_{R_1}(.)\|_{R_2}(SndInBuf_0^1 \| SndInBuf_1^2) \\
Lim_1 \|_{S_1}(.) & \|_{S_2} \mathbf{NextSnd_0} \|(.)\|_{R_1} NextRec_2 \|_{R_1}(.)\|_{R_2}(SndInBuf_0^1 \| SndInBuf_1^2) \\
& \|_{S_3} NextSnd_2 \;\; \|(.)\|_{R_1} NextRec_2 \|_{R_1}(.)\|_{R_2}(SndInBuf_0^1 \| SndInBuf_1^0) \\
Lim_1 \|_{S_1}(.) & \|_{S_2} NextSnd_2 \;\; \|(.)\|_{R_1} NextRec_2 \|_{R_1}(.)\|_{R_2}(SndInBuf_0^1 \| SndInBuf_1^0) \\
& \|_{S_3} \mathbf{NextSnd_1} \|(.)\|_{R_1} NextRec_2 \|_{R_1}(.)\|_{R_2}(SndInBuf_0^1 \| SndInBuf_1^2) \\
Lim_1 \|_{S_1}(.) & \|_{S_2} \mathbf{NextSnd_1} \|(.)\|_{R_1} NextRec_2 \|_{R_1}(.)\|_{R_2}(SndInBuf_0^1 \| SndInBuf_1^2) \\
& \|_{S_3} NextSnd_2 \;\; \|(.)\|_{R_1} NextRec_2 \|_{R_1}(.)\|_{R_2}(SndInBuf_0^1 \| SndInBuf_1^0)
\end{array}
$$

The first two states are obviously equivalent being one the permutation of the other, analogously for the last two states. Less obvious is the fact that they all belong to the same equivalence class. In all the states the receivers already got both packets ($NextRec_2$) but in the first two states the acknowledges for the first packets have been lost (the sender is still in state $NextSnd_0$) while in the last two states they reached the sender ($NextSnd_1$). If we consider the enabled actions, however, we can observe that the four states enable action $recack_2$ only ($SndInBuf_1^2$). By executing this action, they all evolve to the same state since in the first two states the loss of the acknowledges is recognised and the sender correctly jumps to the final state $NextSnd_2$, like in the case of the last two states.

5.3 Discussion

The methodology we have used in the previous sections to compute an aggregated state space underlying an SPA model requires first to generate its underlying state space and then to investigate it looking for sets of equivalent states. The advantage of this technique with respect to the SRG algorithm is that in this case it is possible to detect more general forms of aggregation (see Section 5.2), not only those due the permutations of symmetric objects, like was the case of the protocol example with multiple receivers.

Unfortunately, we have experienced that the computation of equivalence classes is time consuming, specially when compared with the time required to compute the SRG.

A possible way for improving the performance of the aggregation step requires to take advantage of the fact that the equivalences we have used are congruences. Instead of deriving the whole state space and then looking for equivalent states, we could apply the procedure briefly introduced in Section 5, i.e. we could compute partial state spaces considering groups of cooperating components, aggregate them, derive the corresponding aggregated submodels and substitute them into the model again.

Another improvement considers the fact that we did not make use of any symbolic representation of the state, like it was the case for the SRG based technique. Instead one could think of defining sort of *macro-derivatives* representing sets of equivalent derivatives: this idea of macro-derivatives has been exploited in [25]

where an algorithm for the computation of an aggregated state space underlying a PEPA model which automatically recognises the symmetries within the model, in the style of the SRG algorithm, has been defined. The algorithm avoids the derivation of equivalent ordinary derivatives since a canonical form for language expressions, analogous to the canonical symbolic marking, has been proposed; the algorithm has already been implemented in the PEPA workbench [24].

6 More advanced research topics

The work on state space reduction presented in this paper has been extended in several ways in more recent papers.

In [14] a reduction technique based on structural decolorisation of SWN models has been proposed. This technique consists of detecting structurally redundant parts of the colour specification of an SWN model. When applied before the computation of the SRG technique, it can lead to further reduction on the SRG size, moreover, even in case the SRG algorithm itself is able to achieve the same reduction with and without decolorisation, the time required for the SRG computation is reduced if redundant colour components have been eliminated beforehand. In [3] the decolorisation technique has been extended to work also for performance evaluation purposes.

In [22, 23] an extension of the SRG lumping technique has been proposed, that allows to obtain bounds on SWN models performance indices, when the qualitative behavioural symmetries are not completely reflected at the quantitative level (hence leading to *quasi-lumpable* MC instead of a strictly lumpable MC).

In [27] an extension of the SRG has been proposed, that allows to get more reduction than the original technique in models that do not completely satisfy the conditions that ensure the equivalence among all ordinary markings belonging to a symbolic marking. This for example could be due to the use of more general arc expressions or transitions predicates than allowed by the SWN formalism. The work on Extended SRG, however, does not extend the results concerning the lumpability of the underlying CTMC.

Finally, some works have been proposed to combine the symmetry exploitation technique with other techniques developed to cope with the state space explosion problem. In [50] it has been shown that the *stubborn set* method and the symmetry exploitation method for CPNs are orthogonal and can be applied simultaneously to achieve more reduction in the state space size. In [6] it has been proposed an efficient method for the computation of stubborn sets of SWNs. Hence, a promising future research direction could be the combination of this method with the SRG technique. In [28] it has been shown how the SRG technique can be combined with the decomposition techniques based on *Kronecker algebra* operators [19].

References

1. Greatspn home page. URL:http://www.di.unito.it/WWW/PEgroup/GreatSPN/.
2. M. Ajmone Marsan, A. Bobbio, and S. Donatelli. *Petri Nets in Performance Analysis: an Introduction.* In this book.
3. M. Ajmone Marsan, S. Donatelli, G. Franceschinis, and F. Neri. Reductions in Generalized Stochastic Petri Nets and Stochastic Well-formed Nets: An Overview and an Example of Application. In J. Walrand, K. Bagchi, and G. Zobrist, editors, *Network Performance Modeling and Simulation.* Gordon and Breach Publishers INC, 1997.
4. M. Bernardo, L. Donatiello, and R. Gorrieri. Giving a Net Semantics to Markovian Process Algebras. In *Proc. 6th International Workshop on Petri Nets and Performance Models*, Durham, NC, 1995.
5. M. Bernardo and R. Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theoretical Computer Science*, 1998. to appear.
6. R. Brgan and D. Poitrenaud. An efficient algorithm for the computation of stubborn sets of well formed petri nets. In *Proceedings of 16th Int. Conference on Application and Theory of Petri Nets, ICATPN '95*, pages 121–140, Torino, Italy, June 1995.
7. P. Buchholz. Exact and ordinary lumpability in finite markov chains. *Journal of Appl. Prob.*, 31.59–75, 1994.
8. P. Buchholz. Markovian Process Algebra: Composition and Equivalence. In U. Herzog and M. Rettelbach, editors, *Proc. 2^{nd} Workshop on Process Algebra and Performance Modelling*, Erlangen, 1994.
9. G. Chiola, G. Bruno, and T. Demaria. Introducing a color formalism into generalized stochastic Petri nets. In *Proc. 9^{th} Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.
10. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On Well-Formed coloured nets and their symbolic reachability graph. In *Proc. 11^{th} Intern. Conference on Application and Theory of Petri Nets*, Paris, France, June 1990. Reprinted in *High-Level Petri Nets. Theory and Application*, K. Jensen and G. Rozenberg (editors), Springer Verlag, 1991.
11. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic Well-Formed coloured nets and multiprocessor modelling applications. In K. Jensen and G. Rozenberg, editors, *High-Level Petri Nets. Theory and Application.* Springer Verlag, 1991.
12. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11), Nov. 1993.
13. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A Symbolic Reachability Graph for Coloured Petri Nets. *Theoretical Computer Science B (Logic, semantics and theory of programming)*, 176(1&2):39–65, April 1997.
14. G. Chiola and G. Franceschinis. A structural colour simplification in Well-Formed coloured nets. In *Proc. 4^{th} Intern. Workshop on Petri Nets and Performance Models*, pages 144–153, Melbourne, Australia, Dec. 1991. IEEE-CS Press.
15. G. Chiola, G. Franceschinis, and R. Gaeta. A symbolic simulation mechanism for Well-Formed coloured Petri nets. In *Proc. 25^{th} SCS Annual Simulation Symposium*, Orlando, Florida, April 1992.

16. G. Chiola, G. Franceschinis, and R. Gaeta. Modelling symmetric computer architectures by SWNs. In *Proc. of the* 15^{th} *Intern. Conference on Applications and Theory of Petri Nets*, number 815 in Lecture Notes in Computer Science. Springer-Verlag, 1994.
17. G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation, special issue on Performance Modeling Tools*, 24(1&2):47–68, 1995.
18. E.M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal methods in system design*, 9:77–104, 1996.
19. S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–36, 1993.
20. C. Dutheillet and S. Haddad. Regular stochastic Petri nets. In *Proc. 10th Intern. Conf. Application and Theory of Petri Nets*, Bonn, Germany, June 1989.
21. E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal methods in system design*, 9:105–131, 1996.
22. G. Franceschinis and R.R. Muntz. Bounds for quasi-lumpable markov chains. *Performance Evaluation*, 20(1), May 1994. (Performance '93. 16th IFIP Working Group 7.3 International Symposium on Computer Performance Modeling, Measurement and Evaluation, Rome, Italy, 27 Sept.-1 Oct. 1993).
23. G. Franceschinis and R.R. Muntz. Computing bounds for the performance indices of quasi-lumpable stochastic well-formed nets. *IEEE Transactions on Software Engineering*, 20(7), July 1994.
24. S. Gilmore and J. Hillston. The PEPA workbench: A tool to support a Process Algebra based approach to performance modelling. In *Proc. Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Vienna, 1994.
25. S. Gilmore, J. Hillston, and M. Ribaudo. An Efficient Algorithm for Aggregating PEPA Models. Technical report, University of Edinburgh, 1998.
26. S. Haddad. *Une Categorie Regulier de Reseau de Petri de Haut Niveau: Definition, Proprietes et Reductions.* PhD thesis, Lab. MASI, Universite P. et M. Curie (Paris 6), Paris, France, Oct 1987. These de Doctorat, RR87/197 (in French).
27. S. Haddad, J-M Ilie, M. Taghelit, and B. Zouari. Symbolic marking graph and partial symmetries. In *Proceedings of 16th Int. Conference on Application and Theory of Petri Nets, ICATPN '95*, pages 238–257, Torino, Italy, June 1995.
28. S. Haddad and P. Moreaux. Evaluation of High Level Petri nets by means of aggregation and decomposition. In *Proc.* 6^{th} *Intern. Workshop on Petri Nets and Performance Models*, Durham, NC, USA, Oct. 1995.
29. H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic Process Algebras - Between LOTOS and Markov Chains. *Computer Networks and ISDN Systems*, 1998. to appear.
30. H. Hermanns, V. Mertsiotakis, and M. Rettelbach. A Construction and Analysis Tool Based on the Stochastic Process Algebra TIPP. In *Proc. of 2nd Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96)*, volume 1055 of *LNCS*. Springer, 1996.
31. J. Hillston. The Nature of Synchronization. In U. Herzog and M. Rettelbach, editors, *Proc.* 2^{nd} *Workshop on Process Algebra and Performance Modelling*, Erlangen, 1994.
32. J. Hillston. Compositional Markovian modelling using a process algebra. In *Proc. 2nd International Workshop on the Numerical Solution of Markov Chains*, Raleigh, North Carolina, Jan. 1995.

33. Jane Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
34. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
35. P. Huber, A.M. Jensen, L.O. Jepsen, and K. Jensen. Towards reachability trees for high-level Petri nets. In G. Rozenberg, editor, *Advances on Petri Nets '84*, volume 188 of *LNCS*, pages 215–233. Springer Verlag, 1984.
36. J-M. Ilie and K. Ajami. Model checking through the symbolic reachability graph. In *Proc. of TapSoft'97 Theory and Practice of Software Development - 7th CAAP*, LNCS 1214, pages 213–224, Lille, France, April 1997. Springer-Verlag. Extended version in *An Automatique Technique for CTL* Model Checking* , LIP6 Internal Report n.017, 1997.
37. K. Jensen. *An introduction to the practical use of Coloured Petri nets.* In this book.
38. K. Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use. Volume 2.* Springer Verlag, 1995.
39. K. Jensen and G. Rozenberg, editors. *High-Level Petri Nets. Theory and Application.* Springer Verlag, 1991.
40. J. B. Jorgensen and L. M. Kristensen. Verification of Coloured Petri Nets Using State Space Equivalneces. In *Petri Nets in System Engineering (PSNE '97) Modelling, Verification, and Validation - FBI-HH-B-205/97*, Hamburgh, 1997.
41. C. Lin and D.C. Marinescu. On stochastic high level Petri nets. In *Proc. Intern. Workshop on Petri Nets and Performance Models*, Madison, WI, USA, August 1987. IEEE-CS Press.
42. M. Linqvist. Parametrized reachability trees for predicate transition nets. In *Proc.* 11^{th} *Intern. Conference on Application and Theory of Petri Nets*, Paris, France, June 1990.
43. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
44. D. Park. Concurrency and Automata on Infinite Sequences. In *Proc.* 5^{th} *GI Conf. on Theoretical Computer Science*, volume 104 of *LNCS*, 1981.
45. G.D. Plotkin. An operational semantics for CSP. Technical Report CSR-114-82, The University of Edinburgh, May 1982.
46. R.Gaeta. Efficient discrete-event simulation of colored petri nets. *IEEE Transaction on Software Engineering*, 22(9):629–639, Sept. 1996.
47. M. Ribaudo. On the Aggregation Techniques in Stochastic Petri Nets and Stochastic Process Algebras. *The Computer Journal*, 38(6), 1995.
48. M. Ribaudo. Stochastic Petri nets semantics for stochastic process algebras. In *Proc.* 6^{th} *Intern. Workshop on Petri Nets and Performance Models*, Durham, NC, USA, Oct. 1995.
49. W. H. Sanders and J. F. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, Jan. 1991. Special Issue on Computer-Aided Modeling, Analysis and Design of Communication Networks.
50. A. Valmari. Stubborn sets of coloured petri nets. In *Proceedings of 12th Int. Conference on Application and Theory of Petri Nets, ICATPN '91*, pages 102–121, Gjern, Denmark, June 1991.
51. A. Zenie. Colored stochastic Petri nets. In *Proc. Intern. Workshop on Timed Petri Nets*, pages 262–271, Torino, Italy, July 1985. IEEE-CS Press.