

CSE Centre

Computer Systems Engineering Centre
School of Electrical and Information Engineering
University of South Australia
AUSTRALIA

Towards the Parametric Verification of the Class of Stop-and-Wait Protocols over Ordered Channels

Guy Edward Gallasch
Jonathan Billington

March 2005, Revised June 2005

Summary

The correct operation of computer protocols is essential to the smooth operation of the distributed systems that facilitate our global economy. Formal techniques provide our best chance to ensure that protocol designs are free from errors. This report presents a Coloured Petri net (CPN) model of the class of Stop-and-Wait protocols (operating over lossy ordered channels) by using parameters for the maximum sequence number and the maximum number of retransmissions. This parameterisation produces an infinite number of instantiations of our CPN model and thus an infinite set of associated occurrence graphs (OGs) that must be analysed. To overcome this problem, we would like to represent this infinite set of occurrence graphs by a symbolic OG involving the model parameters. This report presents the first step towards establishing such a symbolic OG using algebraic expressions for the base case when there are no retransmissions. We firstly establish that the size of the occurrence graph is linear in the size of the sequence number space and quartic in the maximum number of retransmissions. We present formulas for the number of nodes and arcs over both parameters. We then consider occurrence graphs where the maximum number of retransmissions is set to zero. We define the symbolic OG (associated with the parameterised CPN) and derive algebraic expressions, both recursive and explicit, for its nodes and arcs in terms of the maximum sequence number parameter. We then prove the symbolic OG correct using induction over this parameter. This is a significant result as it allows us to represent the occurrence graph of the Stop-and-Wait CPN model for *any* positive maximum sequence number (with no retransmissions) without having to generate the occurrence graph for each value of the (unbounded) parameter. Further, we derive an expression from the symbolic OG for a symbolic Finite State Automaton (FSA) that embodies the protocol language. By applying automata reduction techniques to the symbolic FSA we obtain a single simple FSA of the protocol language for *any* positive value of the maximum sequence number. We compare this to the Stop-and-Wait service language of alternating send and receive events and thus verify that the protocol conforms to the Stop-and-Wait service for *all* positive values of the maximum sequence number parameter.

Contents

1	Introduction	1
2	Related Work	3
3	The Stop-and-Wait Protocol	5
3.1	Description	5
3.2	SWP CPN Model	5
3.2.1	Sender	5
3.2.2	Network	7
3.2.3	Receiver	7
4	Reachability Analysis	9
5	Algebraic Formulas for the OG	13
5.1	A Recursive Formula for the OG	14
5.1.1	Base case	15
5.1.2	Recursive Expression	15
5.2	An Explicit Algebraic Formula for the OG	17
5.3	Proof of Recursive and Algebraic Expressions	18
6	The Algebraic Protocol Language	22
6.1	Mapping from the Algebraic OG to an Algebraic FSA	22
6.2	Removing Empty Cycles	24
6.3	Removing Empty Moves	25
6.4	Determinisation and Minimisation	25
7	Conclusions and Future Work	28
	Acknowledgement	29
	References	30

List of Figures

3.1	A CPN of the Stop-and-Wait Protocol operating over an in-order medium.	6
3.2	Declarations of the CPN shown in Fig. 3.1.	6
4.1	The OGs of CPN_1 (OG_1) and CPN_2 (OG_2).	11
6.1	The fragment of FSA_{OG_n} corresponding to any given $i, 0 \leq i \leq n$	24
6.2	Refining the partition of states. (a) p and q are distinguishable on a , as p' and q' are in different subsets. (b) p and q are not distinguishable.	26
6.3	Illustration of ϵ removal and minimisation of FSA_{OG_1}	26

List of Tables

4.1	OG Statistics of the CPN in Figs. 3.1 and 3.2 for various parameter values.	10
5.1	$V_{(i,i)}^n$ for $0 \leq i \leq n$	14
5.2	$V_{(i,i \oplus_n 1)}^n$ for $0 \leq i \leq n$	14
5.3	$A_{(i,i)}^n$ for $0 \leq i \leq n$	15
5.4	$A_{(i,i \oplus_n 1)}^n$ for $0 \leq i \leq n$	15
5.5	V_1 - Nodes for the base case of MaxSeqNo=1.	16
5.6	A_1 - Arcs for the base case of MaxSeqNo=0.	16
5.7	V_n^{add} - Nodes to add to obtain V_n from V_{n-1}	17
5.8	V_n^{del} - Nodes to delete to obtain V_n from V_{n-1}	17
5.9	A_n^{add} - Arcs to add to obtain A_n from A_{n-1}	18
5.10	A_n^{del} - Arcs to delete to obtain A_n from A_{n-1}	18
6.1	FSA_{OG_n} where $0 \leq i \leq n$	24
6.2	FSA_{OG_n} after removal of ϵ moves and deletion of inaccessible states, where rows 2 and 3 are evaluated for $0 \leq i \leq n$	25
6.3	The minimised deterministic FSA_{OG_n} representing the protocol language of CPN_n	27

Acronyms

ABP	Alternating Bit Protocol
ARQ	Automatic Repeat reQuest
BDD	Binary Decision Diagram
BRP	Bounded Retransmission Protocol
CFFD	Chaos-Free-Failures-Divergences
CPN	Coloured Petri Net
CPN ML	Coloured Petri Net Meta-Language
FAST	Fast Acceleration of Symbolic Transition systems
FIFO	First-In-First-Out
FSA	Finite State Automaton
OG	Occurrence Graph
SML	Standard Meta-Language
SWP	Stop-and-Wait Protocol
TCP	Transmission Control Protocol
TReX	Tool for Reachability analysis of complex systems

Chapter 1

Introduction

With the reliance of the global economy on computers growing every year, it is imperative that the underlying computer communication infrastructure does not fail. Reliable communication between the networks that form the Internet is key to achieving this. It is necessary to understand how these systems work at a basic level and, more importantly, to understand how and why they may fail.

The Stop-and-Wait protocol (SWP) [33, 36] is a basic flow control protocol. The main idea is that the sender of data over a communication channel only sends one message, then stops and waits for that message to be acknowledged by its receiver, before proceeding to send the next message. The simplest Stop-and-Wait protocol incorporating sequence numbers for error recovery uses two sequence numbers (0 and 1) and is known as the Alternating Bit Protocol (ABP) [5]. The Stop-and-Wait mechanism is a fundamental mechanism on which many practical protocols are based, such as the Internet's Transmission Control Protocol (TCP) [31]. We have therefore embarked on a fundamental study of SWP mechanisms [8–10] in which we are attempting to verify the correctness of the Stop-and-Wait protocol operating over lossy and lossless, reordering and in-order media. The SWP must satisfy a set of properties defined for the service it is meant to provide (e.g. that data is neither lost nor duplicated and is delivered in sequence, and there are no deadlocks). Proving that a protocol satisfies the required properties is known as *protocol verification*.

Petri nets have proven to be a suitable formal method for protocol verification [6, 7, 19, 23, 24, 26, 30, 36, 41]. A protocol verification methodology using Coloured Petri nets (CPNs) [22, 25] is presented in [10]. In [9, 10] a CPN model of the Stop-and-Wait protocol was presented. The model was parameterised with the maximum sequence number (MaxSeqNo) and the maximum number of retransmissions (MaxRetrans). Correctness of the protocol with respect to absence of loss and duplication of data, and conformance to the Stop-and-Wait *property* of alternating send and receive events was investigated using automated language analysis. However, we could only prove correctness of the SWP for a limited range of parameter values. What makes our problem interesting is that the parameter values are unbounded, resulting in an infinite set of CPNs and their associated occurrence graphs (OGs).

We begin our investigations by presenting formulae for the number of nodes and arcs in the OG of the SWP CPN as a function of the MaxSeqNo and MaxRetrans parameters. We then present algebraic expressions, both recursive and explicit, representing the infinite set of occurrence graphs over the MaxSeqNo parameter. This result is significant as it embodies the OG for the SWP for *any* finite maximum sequence number, for the situation where $\text{MaxRetrans} = 0$. We verify conformance of the SWP to the Stop-and-Wait property of alternating send and receive events over these parameter values by deriving the *protocol language* (the set of all sequences of user-observable events called *service primitives*) from our algebraic expressions and comparing it to the *service language* of alternating sends and receives using *language equivalence*.

Related work on symbolic verification of the SWP operating over lossy, in order, channels [2] considers an infinite OG, rather than an infinite set of finite OGs, for unbounded retransmissions, with $\text{MaxSeqNo} = 1$ (i.e. the ABP) and also a variant of the ABP called the Bounded Retransmission Protocol (BRP), that considers arbitrary MaxRetrans , but $\text{MaxSeqNo} = 1$. The work we present in this report complements [2] by verifying the SWP for every positive value of MaxSeqNo (with $\text{MaxRetrans} = 0$.)

The authors are not aware of any previous attempts to verify a parameterised model of the class of Stop-and-Wait protocols by obtaining an explicit algebraic expression representation of the OGs over the parameter values.

The rest of this report is organised as follows. Chapter 2 describes related work in the area of verification of the SWP/ABP with a focus on parametric and symbolic verification of protocols. Chapter 3 provides some background on the Stop-and-Wait protocol and a description of our CPN model (derived from [8–10]). In Chapter 4 we present our formula for the size of the occurrence graph over both parameters based on empirical evidence, and provide the motivation for and intuition behind the formalisation of the parameterised occurrence graph which is presented in Chapter 5. In Chapter 6 we derive the protocol language from the symbolic OG and in doing so, verify the stop-and-wait property for all values of MaxSeqNo . Finally, conclusions and future work are presented in Chapter 7.

Chapter 2

Related Work

The ABP [5] and its extensions (e.g. [14]) have been used extensively in the literature as case studies (e.g. [32]). Often such papers demonstrate in various ways whether the ABP works as expected over (lossy or lossless) First-In-First-Out (FIFO) channels [12, 35], investigate performance [28, 34], demonstrate new tools [12], or illustrate verification methodologies [16], the application of formal description techniques [38], new modelling languages and derivatives of existing languages [28, 34, 35]. Some illustrate the development of the ABP incrementally [32, 36]. However, none of these papers address the issue of parametric verification of the ABP (i.e. for arbitrary values of `MaxRetrans`.)

More recently there has been work in the area of symbolic verification of the ABP. Valmari and his co-workers (e.g. [40]) promote a behavioural fixed point method and compositional techniques for the verification of parametric systems. In [40] a variant of the ABP using limited retransmission, i.e. where there is an arbitrary bound (e.g. `MaxRetrans`) on the number of retransmissions, is verified using Valmari's Chaos-Free-Failures-Divergences (CFFD) equivalence. There are several differences with our work. Perhaps the most significant is that the channels are limited to holding only one message or acknowledgement at a time, whereas ours are unbounded FIFO queues. Valmari [40] concedes this to be a much more difficult problem. Valmari's method relies on defining a separate counter process which needs to be synchronised (using parallel composition) with the sender logic, which has 18 states. The counter itself is a recursive parallel composition of counter cells. The receiver is a relatively straightforward 6 state process. The ack channel is given as a 3 state process, but the data channel is more complex and not given explicitly in the paper. To obtain the model, all these processes need to be synchronised with parallel composition. In contrast our CPN model integrates all these aspects in the one model, and extends the model to include unbounded FIFO queues and sequence numbers with an arbitrary maximum sequence number as a parameter. However, our model does not have explicit communication with the users (but relies on the send and (non-duplicate) receive transitions to be considered as synchronised communication with the user) and does not consider reporting errors to the user. We see no problem in extending our model to include these features in the future.

Both the ABP (with unbounded retransmissions) and another variant called the Bounded Retransmission Protocol are used in [2] to demonstrate a symbolic verification methodology [1]. TReX (Tool for Reachability analysis of compleX systems) [37] was used to implement this methodology in [2]. The content of unbounded lossy FIFO channels is modelled by (a restricted class of) regular expressions, thus providing a symbolic representation of the channels. A process to examine the effect of an arbitrary number of iterations of loops in the model behaviour (acceleration) is used to construct these regular expressions. This allows a small symbolic state space to be calculated based on the states of the sender and receiver processes. They verify that the ABP conforms to the property of alternating send and receive events using the Aldebaran tool [13] for finite state automata. The maximum number

of retransmissions was considered to be unlimited giving rise to a single, infinite state model. This differs in a subtle way from our approach of modelling `MaxRetrans` as a parameter and thus having an infinite number of finite-state models, one for each parameter value. This has the advantage of allowing detection of e.g. a counter exceeding the parameter value, something not possible with the infinite state approach. The cost is an infinite set of models to analyse (for an unbounded parameter) rather than a single infinite state model.

In our own previous work [10] we summarised a protocol verification methodology based on CPNs [21] and finite state automata. CPNs are an executable modelling language with a formal semantics, based on Petri nets and the SML functional programming language. This methodology uses state space methods and has been applied successfully for finite state systems, for small values of parameters. Techniques (such as partial orders, Binary Decision Diagrams (BDDs), and the sweep-line method) for alleviating the state space explosion problem [39] help to extend the method to larger ranges of parameters, but cannot handle large or unbounded values.

In [10], the methodology is illustrated using a stop-and-wait Protocol (SWP) [33, 36] which involves two parameters: the maximum sequence number, `MaxSeqNo`; and the maximum number of retransmissions, `MaxRetrans`. From a modelling point of view, the values of these parameters may be chosen arbitrarily. We would thus like to prove that the SWP class is correct for any values of $\text{MaxSeqNo} \geq 1$ and $\text{MaxRetrans} \geq 0$. This becomes impossible using finite state techniques, as we need to consider an infinite number of increasingly larger finite state spaces. For FIFO channels (either lossy or lossless), a hand proof is given in [10] that shows that the number of messages in the message channel (and the number of acks in the acknowledgement channel) has a least upper bound of $2\text{MaxRetrans} + 1$, for any positive value of `MaxSeqNo`, and any non-negative value of `MaxRetrans`. For other properties, such as verifying that the protocol conforms to its service of alternating send and receive events, the standard methodology was used for a range of parameter values ($0 < \text{MaxSeqNo} < 1024$, $0 \leq \text{MaxRetrans} \leq 4$), but no general result was obtained.

In [11] we have used a symbolic verification tool called FAST (Fast Acceleration of Symbolic Transition systems) [17] to analyse the SWP. Like TRex, FAST also uses accelerations (meta-transitions) to perform symbolic analysis of infinite state systems, to encode an arbitrary number of iterations of sequences of actions within the system. Models are represented in FAST using *counter systems*. These are automata extended with vectors of integer variables and whose transitions are labelled with Presburger-linear functions. In essence, FAST views the problem as a labelled transition system that uses tuples of integers to represent the state and *Presburger functions* to represent changes of state [3, 18]. In [11] a methodology was described for converting CPNs into counter systems and the SWP CPN model from [8, 10] was revised to make it easier to convert to counter systems and analyse using FAST. We were able to symbolically verify a number of properties of our SWP model for arbitrary `MaxRetrans` and for `MaxSeqNo` from 1 to 5, including: conformance to the property of alternating send and receive events; in-sequence delivery of data; no loss or duplication of data; no deadlocks; and a lowest upper bound on the total number of messages and acknowledgements of $2\text{MaxRetrans}+1$ thus validating the hand proof given in [10] for this range of parameter values.

Symbolic analysis of the SWP using FAST (and the ABP and BRP using TRex [2]) was successful when modelling `MaxRetrans` as the parameter (or as an arbitrary value), but were only successful for small values of `MaxSeqNo`. In contrast, the approach presented in this report is able to verify the SWP for every explicit value of $\text{MaxSeqNo} > 0$, but for $\text{MaxRetrans}=0$.

Chapter 3

The Stop-and-Wait Protocol

3.1 Description

Stop-and-Wait is an elementary form of flow control [33, 36] between a sender and a receiver. Essentially, the sender may send a message to the receiver but must then stop and wait for an acknowledgement indicating that the receiver is ready for the next message.

Stop-and-Wait protocols often operate over channels that may corrupt or lose messages so that acknowledgements are used for both flow control and transmission error recovery. A timeout and retransmission scheme known as Automatic Repeat ReQuest (ARQ) [36] is used to recover from message loss. Corruption is detected by including a checksum in messages and acknowledgements. If the received message or acknowledgement fails the checksum, it is discarded, converting message corruption into message loss. Finally, sequence numbers are used to differentiate between new messages and duplicates of previous messages.

3.2 SWP CPN Model

The SWP is modelled using Coloured Petri nets (CPNs) [22, 25]. CPNs are a form of high-level net [20] where the tokens are arbitrarily complex data values. The software tool Design/CPN [15] was used for the construction and analysis of the model. We assume the reader is familiar with general Petri net concepts. We give an informal introduction to CPNs through our description of our SWP CPN model, however for a thorough introduction to CPNs the reader is referred to [22, 25].

The CPN of our SWP is shown in Figs. 3.1 and 3.2. Figure 3.1 presents the CPN graph of the system, while Fig. 3.2 defines all the colour sets, variables, constants, and functions used in the annotations of the CPN graph. The model in Fig. 3.1 is divided into three parts called **Sender**, **Network** and **Receiver**. We discuss the operation of each part below.

3.2.1 Sender

The Sender comprises three *places* (ellipses) and four *transitions* (rectangles). Places are typed by a *colour set* (by convention written below the place), defining all allowable data values (tokens) that may be present on that place. The *marking* of a place is the multiset of tokens present on that place. The marking of a CPN is the distribution of tokens over all places of the CPN. The place `sender_state` models the two states of the sender. It is typed by the colour set `Sender` comprising two values representing the two possible states (ready or waiting for an acknowledgement) using CPN ML (Coloured Petri Net Meta-Language) notation, an extension of SML (Standard Meta-Language) [29] and can be seen in Fig. 3.2. The initial marking of one `s_ready` token on `sender_state` indicates that the sender

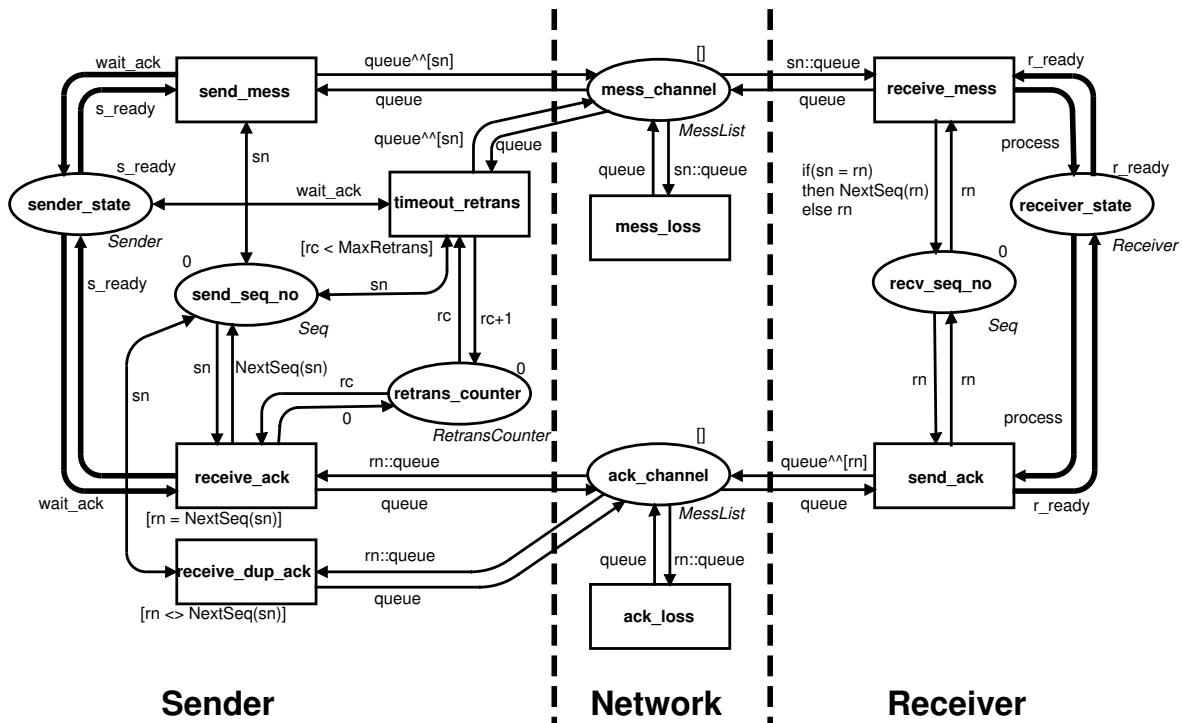


Figure 3.1: A CPN of the Stop-and-Wait Protocol operating over an in-order medium.

```

val MaxRetrans = 0;
val MaxSeqNo = 1;

color Sender = with s_ready | wait_ack;
color Receiver = with r_ready | process;
color Seq = int with 0..MaxSeqNo;
color RetransCounter = int with 0..MaxRetrans;
color Message = Seq;
color MessList = list Message;

var sn, rn : Seq;
var rc      : RetransCounter;
var queue   : MessList;

fun NextSeq(n) = if (n = MaxSeqNo) then 0 else n+1;

```

Figure 3.2: Declarations of the CPN shown in Fig. 3.1.

is in the ready state initially. The `seq_no` place is typed by `Seq` and stores the sender sequence number (an integer ranging over $[0, \text{MaxSeqNo}]$ - see Fig. 3.2) and the `retrans_counter` place records the number of retransmissions (ranging from 0 to `MaxRetrans`) of an unacknowledged message.

Transition `send_mess` models the sending of a message. Places are connected to transitions via *input arcs* and transitions are connected to places via *output arcs*. Every arc has an inscription that must evaluate to a multiset over the colour set of the associated place. A transition is *enabled*

when, for a given *binding* of variables, the input arc inscriptions evaluate to multisets that are present in the corresponding input places. When a transition *occurs*, these multisets are removed from the input places and the multisets resulting from evaluation of the output arc inscriptions are placed in the corresponding output places. When `send_mess` occurs it writes the sequence number (as the message) to the message channel and changes the sender state to `wait_ack`. Message content is not modelled as it does not influence the operation of the protocol. The same is true of the address of the sender and receiver, as we only have one of each in this model.

The `timeout_retrans` transition models the expiry of a timer and the retransmission of the currently unacknowledged message. The guard `[rc < MaxRetrans]` prevents this transition from occurring when the `MaxRetrans` parameter is set to 0 and the bidirectional arc connected to place `sender_state` ensures retransmissions only occur when the sender is waiting for an acknowledgement. Transition `receive_ack` models the receipt of expected acknowledgements, those containing a sequence number one greater than that of the sender. This transition removes the acknowledgement from the channel, returns the sender to the ready state, resets the retransmission counter to 0 and increments (modulo `MaxSeqNo+1`) the sender sequence number stored in `seq_no` (using the function `NextSeq` defined in Fig. 3.2.) Duplicate acknowledgements are received and discarded by the `receive_dup_ack` transition.

The sender differs cosmetically from similar models presented in earlier publications [8–10] as the two states of the sender are modelled by a single place, rather than being represented explicitly by two separate places.

3.2.2 Network

The underlying communication medium is modelled as an in-order bidirectional channel consisting of one place and one transition for each direction of communication. The `mess_channel` place models the message channel. It is typed by the colour set `MessList` (see Fig. 3.2) and has an initial marking of the empty list `[]`. All arcs incident on this place have inscriptions written to manipulate this list as a First-In-First-Out (FIFO) queue. (The operator `^^` concatenates two lists while `::` concatenates an element with a list of elements and is used in this model to remove an element from the beginning of a list.)

The two transitions `mess_loss` and `ack_loss` model loss of messages and acknowledgements respectively. This models both loss in the network (congestion or buffer overflow in a router) and loss caused by discarding corrupt messages and acknowledgements (checksum failures).

3.2.3 Receiver

The receiver consists of two places and two transitions. The two states of the receiver (`r_ready` and `process`) are modelled by the `receiver_state` place. The receiver sequence number (the sequence number of the message expected next by the receiver) is recorded in the place `recv_seq_no` typed by the colour set `Seq`. The initial marking of `r_ready` on place `receiver_state` and 0 on `recv_seq_no` indicates that the receiver is initially ready and expecting a message with sequence number 0.

Transition `receive_mess` models the receipt of both expected and unexpected messages from the sender. When an expected message is received (`sn=rn`) the receiver sequence number is incremented (using `NextSeq`) and the receiver changes state to `process`. If the message is unexpected (`sn≠rn`) a duplicate message is detected, discarded and the receiver sequence number remains unchanged. Transition `send_ack` models the sending of an acknowledgement (containing the receiver sequence number) and returns the receiver to the ready state.

The receiver model differs from [8–10] in that the receiver sequence number and receiver state are represented separately and stored in separate places. This aligns the modelling style used in the

Receiver with that of the Sender.

Chapter 4

Reachability Analysis

Using Design/CPN [15], the occurrence graphs of instantiations of the CPN for a range of parameter values (MaxSeqNo from 1 to 1023, MaxRetrans from 0 to 4) were generated [9]. Selected statistics are shown in Table 4.1. The first two columns show the values of the MaxRetrans and MaxSeqNo parameters. The next two columns show the number of nodes and arcs in each occurrence graph. The last column shows the number of dead markings present in each OG.

For a given MaxRetrans the number of nodes and arcs increases linearly with MaxSeqNo . For example, when $\text{MaxRetrans} = 0$ the number of nodes and arcs increase by 6 for each unit increase in MaxSeqNo and when $\text{MaxRetrans} = 1$, the increase is 40 nodes and 97 arcs. The following conjecture gives the size of the occurrence graphs for $\text{MaxSeqNo} > 0$ and $\text{MaxRetrans} \geq 0$.

Conjecture 1. *For the Stop-and-Wait CPN of Figs. 3.1 and 3.2, the number of nodes in the occurrence graph is given by*

$$|V| = ((\text{MaxSeqNo} + 1)/3)(\text{MaxRetrans}^4 + 13\text{MaxRetrans}^3 + 41\text{MaxRetrans}^2 + 47\text{MaxRetrans} + 18)$$

and the number of arcs is given by

$$|A| = \begin{cases} 6(\text{MaxSeqNo} + 1) & \text{for } \text{MaxRetrans} = 0, \\ ((\text{MaxSeqNo} + 1)/6)(10\text{MaxRetrans}^4 + 115\text{MaxRetrans}^3 + 266\text{MaxRetrans}^2 + 167\text{MaxRetrans} + 24) & \text{for } \text{MaxRetrans} > 0. \end{cases}$$

The formulae are derived from a more extensive version of Table 4.1 using standard techniques for fitting polynomials to data. The large range of parameter values investigated in [9] give us a high degree of confidence in this result.

Intuitively, such a linear increase in the number of nodes and arcs in MaxSeqNo for a static MaxRetrans suggests that there is some regularity to the occurrence graphs. To test this assertion we examine the structure of the occurrence graphs, starting with the simplest case of $\text{MaxRetrans} = 0$.

Let us denote the CPN from Figs. 3.1 and 3.2 with $\text{MaxSeqNo} = n$ as CPN_n and its occurrence graph as OG_n . OG_1 is shown in Fig. 4.1(a) and consists of 12 nodes and 12 arcs (as indicated in Table 4.1). It has a main loop of 8 nodes with four dead markings branching from it. The nodes are numbered from 1 to 12 and each node has been annotated with its sender and receiver sequence number (written as a pair) for reasons that will become evident. The sender and receiver sequence numbers are the marking of places send_seq_no and recv_seq_no respectively. For example, node 4 is annotated with (0,1) indicating that the sender sequence number is 0 and the receiver sequence number is

Table 4.1: OG Statistics of the CPN in Figs. 3.1 and 3.2 for various parameter values.

MaxRetrans	MaxSeqNo	Nodes	Arcs	Dead Nodes
0	1	12	12	4
0	2	18	18	6
0	3	24	24	8
0	4	30	30	10
0	9	60	60	20
0	10	66	66	22
1	1	80	194	4
1	2	120	291	6
1	3	160	388	8
1	4	200	485	10
1	9	400	970	20
1	10	440	1067	22
2	1	264	834	4
2	2	396	1251	6
2	3	528	1668	8
2	4	660	2085	10
2	9	1320	4170	20
2	10	1452	4587	22
3	1	640	2278	4
3	2	960	3417	6
3	3	1280	4556	8
3	4	1600	5695	10
3	9	3200	11390	20
3	10	3520	12529	22
4	1	1300	4956	4
4	2	1950	7434	6
4	3	2600	9912	8
4	4	3250	12390	10
4	9	6500	24780	20
4	10	7150	27258	22

1. Node 4 represents the marking in which $M(\text{sender_state}) = \text{wait_ack}$, $M(\text{retrans_counter}) = 0$, $M(\text{send_seq_no}) = 0$, $M(\text{receiver_state}) = \text{process}$, $M(\text{recv_seq_no}) = 1$, $M(\text{mess_channel}) = []$, $M(\text{ack_channel}) = []$. We omit the unit coefficient (e.g. 1 'wait_ack) from the marking of each place.

Each arc is labelled with the corresponding action from the CPN, including the sequence number of the message or acknowledgement where necessary for clarity. For example, the label `send_mess 0` on the arc between node 1 and node 2 represents the occurrence of transition `send_mess` with `sn` bound to 0 (the sending of a message with sequence number 0).

The main loop represents the behaviour of the Stop-and-Wait protocol when no messages are lost. As there are no retransmissions the dead markings arise when either a message or an acknowledgement is lost. The message or the acknowledgement may be lost for each sequence number, so we have a total of four dead markings (as seen in Table 4.1).

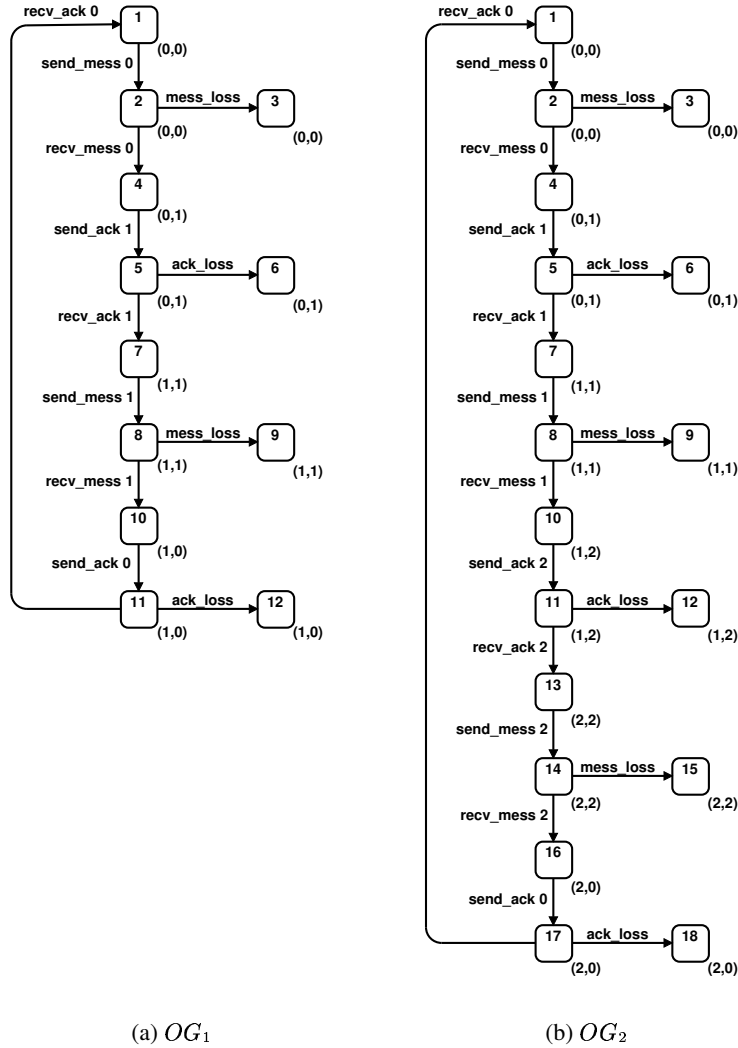


Figure 4.1: The OGs of CPN_1 (OG_1) and CPN_2 (OG_2).

OG_2 given in Fig 4.1(b) contains 18 nodes and 18 arcs, with 6 dead markings. A main loop similar to that in OG_1 is present, but consisting of 12 nodes instead of 8 because $MaxSeqNo$ now equals 2, thus the message with sequence number 2 must be sent and acknowledged before the protocol can return to the initial state. This new message and acknowledgement may be lost, resulting in the two additional dead markings.

There is a regular structure to the OGs. The annotation of the nodes with the sender/receiver sequence number pair allows us to see that for each message sent (i.e. for each sequence number) there are 6 markings generated that can be partitioned into two sets of 3 markings each. The sender and receiver sequence numbers are identical for the first set of three markings which comprise the markings where: 1. the acknowledgement has been received by the sender (or it is the initial marking); 2. the message has been sent; and 3. the message has been lost. The set of nodes $\{1, 2, 3\}$ (for sequence number 0) is an example of this first set of 3 markings. The second set of 3 markings corresponds to the receive sequence number being one more than the send sequence number modulo $MaxSeqNo+1$ ($n+1$), and comprises the markings where: 1. the message has been received; 2. the

acknowledgement has been sent; and 3. the ack has been lost. The set of nodes $\{4, 5, 6\}$ (for sequence number 0) is an example of this second set of 3 markings.

The first 9 nodes of OG_1 and OG_2 are identical. For example, node 8 from both OGs is the marking $M(\text{sender_state}) = \text{wait_ack}$, $M(\text{retrans_counter}) = 0$, $M(\text{send_seq_no}) = 1$, $M(\text{receiver_state}) = \text{r_ready}$, $M(\text{recv_seq_no}) = 1$, $M(\text{mess_channel}) = [1]$, $M(\text{ack_channel}) = []$. The first 9 nodes in each OG correspond to the three sets of nodes with sender and receiver sequence numbers of (0,0), (0,1) and (1,1). We denote these subsets of three nodes by $V_{(0,0)}^1$, $V_{(0,1)}^1$ and $V_{(1,1)}^1$ for OG_1 and by $V_{(0,0)}^2$, $V_{(0,1)}^2$ and $V_{(1,1)}^2$ for OG_2 . The subscript is the sender/receiver sequence number pair and the superscript is the value of **MaxSeqNo** of the corresponding CPN.

Nodes 10, 11 and 12, however, differ between OG_1 and OG_2 as the occurrence of **recv_mess 1** at node 8 increments the receiver sequence number to 2 in OG_2 (rather than wrapping it to 0 as in OG_1). Thus nodes $\{10, 11, 12\}$ in OG_2 have receiver sequence number equal to 2 (not 0) and in the same way as above we denote these nodes by $V_{(1,0)}^1$ in OG_1 and $V_{(1,2)}^2$ in OG_2 . The protocol behaviour is identical over these three nodes, the only difference being the acknowledgement sequence number of 2, not 0.

Nodes 13 to 18 in OG_2 are new, as the message with sequence number 2 must be sent and acknowledged before sequence numbers wrap and the protocol returns to the initial state. The same pattern of behaviour exhibited for sequence numbers 0 and 1 is repeated for sequence number 2. These nodes correspond to the sets $V_{(2,2)}^2$ and $V_{(2,0)}^2$, i.e. the nodes with annotation (2,2) and (2,0) respectively.

To obtain OG_2 from OG_1 we delete 3 nodes (10, 11 and 12 in OG_1) and add 9 nodes (10-18 in OG_2) giving a net increase of 6 nodes. Thus the set of nodes for OG_2 , V_2 , is given by $V_2 = (V_1 \cup (V_{(1,2)}^2 \cup V_{(2,2)}^2 \cup V_{(2,0)}^2)) \setminus V_{(1,0)}^1$, where $V_1 = V_{(0,0)}^1 \cup V_{(0,1)}^1 \cup V_{(1,1)}^1 \cup V_{(1,0)}^1$ is the set of nodes for OG_1 . Similarly for the arcs, we delete 4 (all the arcs associated with the nodes in $V_{(1,0)}^1$) and add 10 (all the arcs associated with the nodes in $V_{(1,2)}^2$, $V_{(2,2)}^2$ and $V_{(2,0)}^2$) giving a net increase of 6 arcs. Thus the set of arcs for OG_2 , A_2 , is given by $A_2 = A_1 \cup A_2^{add} \setminus A_2^{del}$, where A_1 is the set of arcs in OG_1 , A_2^{add} is the set of 10 new arcs and A_2^{del} is the set of 4 arcs deleted from OG_1 .

Chapter 5

Algebraic Formulas for the OG

In this chapter we derive algebraic formulas for the OG when $\text{MaxRetrans} = 0$, based on the intuition presented in Chapter 4. We firstly present a recursive expression for the OG for an arbitrary $\text{MaxSeqNo} = n - 1$ as a formalisation of the observations made in Chapter 4. Following this, we present a *closed form* expression explicitly stating the OG for an arbitrary MaxSeqNo .

Let us begin by defining the OG of a CPN, based on [23].

Definition 1. *The OG of a CPN with initial marking M_0 and a set of binding elements BE , is a labelled directed graph $OG = (V, A)$ where*

1. $V = [M_0]$ is the set of reachable markings of the CPN; and
2. $A = \{(M, b, M') \in V \times BE \times V \mid M[b]M'\}$ is the set of labelled directed arcs, where $M[b]M'$ denotes that the marking of the CPN changes from M to M' on the occurrence of binding element $b \in BE$.

The Stop-and-Wait protocol CPN in Figs. 3.1 and 3.2 is parameterised by MaxSeqNo (we set MaxRetrans to 0). Thus we have the following definition of an instantiation of the CPN and its OG for a given parameter value:

Definition 2. *For $n \in \mathbb{N}^+$ (the positive integers), CPN_n is defined as the Stop-and-Wait Protocol CPN of Figs. 3.1 and 3.2 with $\text{MaxSeqNo} = n$ and $\text{MaxRetrans} = 0$. The occurrence graph of CPN_n is denoted by $OG_n = (V_n, A_n)$.*

Based on our observations, we define the following notation for markings and arcs. Let $V_{(sn, rn)}^n \subset V_n$ denote the subset of nodes in OG_n with the given sender and receiver sequence numbers $sn = M(\text{send_seq_no})$ and $rn = M(\text{recv_seq_no})$. As previously described, for each message sent with sequence number $i \in \{0, 1, \dots, n\}$, there are 6 markings that can be partitioned into two sets of 3 markings each. The first set of 3 markings corresponds to those in which $sn=rn$ and is denoted by $V_{(i,i)}^n$. The second set corresponds to $sn=rn \oplus_n 1$ (where \oplus_n is modulo $(n+1)$ addition) and is denoted by $V_{(i, i \oplus_n 1)}^n$. Because of the Stop-and-Wait operation of the protocol over a FIFO medium, it can be seen from Fig. 3.1 that in all markings either $rn = sn$ or $rn = sn \oplus_n 1$ so we define for $0 \leq i \leq n$:

$$V_{(i,i)}^n = \{M \in V_n \mid M(\text{recv_seq_no}) = M(\text{send_seq_no}) = i\} \quad (5.1)$$

and

$$V_{(i, i \oplus_n 1)}^n = \{M \in V_n \mid (M(\text{recv_seq_no}) = M(\text{send_seq_no}) \oplus_n 1 = i \oplus_n 1)\} \quad (5.2)$$

The markings are given in Tables 5.1 and 5.2. The first column shows the marking name. The subscript of a marking (sn, rn) is the sender and receiver sequence number for that marking and the

Table 5.1: $V_{(i,i)}^n$ for $0 \leq i \leq n$

Node	sender_state	send_seq_no	mess_ch.	ack_ch.	receiver_state	recv_seq_no
$M_{(i,i)}^{(n,1)}$	s_ready	i	$[\]$	$[\]$	r_ready	i
$M_{(i,i)}^{(n,2)}$	wait_ack	i	$[i]$	$[\]$	r_ready	i
$M_{(i,i)}^{(n,3)}$	wait_ack	i	$[\]$	$[\]$	r_ready	i

 Table 5.2: $V_{(i,i \oplus_n 1)}^n$ for $0 \leq i \leq n$

Node	sender_state	send_seq_no	mess_ch.	ack_ch.	receiver_state	recv_seq_no
$M_{(i,i \oplus_n 1)}^{(n,1)}$	wait_ack	i	$[\]$	$[\]$	process	$i \oplus_n 1$
$M_{(i,i \oplus_n 1)}^{(n,2)}$	wait_ack	i	$[\]$	$[i \oplus_n 1]$	r_ready	$i \oplus_n 1$
$M_{(i,i \oplus_n 1)}^{(n,3)}$	wait_ack	i	$[\]$	$[\]$	r_ready	$i \oplus_n 1$

superscripts correspond to the maximum sequence number (n) and the ‘type’ of marking described in Chapter 4. The remaining columns of these tables show the markings of each place in the SWP CPN of Fig. 3.1. Note that `retrans_counter` always has a value of 0 (one ‘0’ token) and is thus omitted from the tables. Correspondingly, the transition `timeout_retrans` is never enabled.

To match the partitioning of nodes, we define a similar partitioning of the arcs into groups of three. Let $A_{(sn,rn)}^n \subset A_n$ denote the subset of arcs in OG_n in which the *source* node of the arc has the sender sequence number `sn` and receiver sequence number `rn`. As before, $rn = sn$ or $sn \oplus_n 1$. For $0 \leq i \leq n$ we define:

$$A_{(i,i)}^n = \{(M, b, M') \in A_n \mid M(\text{recv_seq_no}) = M(\text{send_seq_no}) = i\} \quad (5.3)$$

and

$$A_{(i,i \oplus_n 1)}^n = \{(M, b, M') \in A_n \mid (M(\text{recv_seq_no}) = M(\text{send_seq_no}) \oplus_n 1 = i \oplus_n 1)\} \quad (5.4)$$

Partitioning the arcs in this way means that $A_{(sn,rn)}^n$ contains all the outgoing arcs of the nodes in the corresponding set $V_{(sn,rn)}^n$, for a given `sn` and `rn`.

The arcs in $A_{(i,i)}^n$ represent: 1. sending a message with sequence number i ; 2. losing this message; and 3. receiving this message. The arcs in $A_{(i,i \oplus_n 1)}^n$ represent: 1. sending an acknowledgement $i \oplus_n 1$; 2. loss of this acknowledgement; and 3. receiving this acknowledgement. These arcs are given in Tables 5.3 and 5.4. The subscript indicates the sender and receiver sequence number of the *source* node of each arc. The superscript corresponds to the maximum sequence number (n) and the ‘type’ of arc as just defined.

5.1 A Recursive Formula for the OG

Chapter 4 shows that OG_2 can be easily constructed from OG_1 through subtraction of 3 nodes and 4 arcs, and addition of 9 nodes and 10 arcs. Furthermore, we are able to obtain OG_3 from OG_2 through the subtraction of 3 nodes and 4 arcs, and the addition of 9 nodes and 10 arcs. This pattern continues for OG_4, OG_5 , etc. and suggests that the OG can be recursively defined for a given value of `MaxSeqNo` using a first-order recursive formula.

Table 5.3: $A_{(i,i)}^n$ for $0 \leq i \leq n$

Name	Source node	Binding Element	Dest. Node
$a_{(i,i)}^{(n,1)}$	$M_{(i,i)}^{(n,1)}$	send_mess<sn=i, queue=[]>	$M_{(i,i)}^{(n,2)}$
$a_{(i,i)}^{(n,2)}$	$M_{(i,i)}^{(n,2)}$	mess_loss<sn=i, queue=[]>	$M_{(i,i)}^{(n,3)}$
$a_{(i,i)}^{(n,3)}$	$M_{(i,i)}^{(n,2)}$	receive_mess<sn=i, rn=i, queue=[]>	$M_{(i,i \oplus n 1)}^{(n,1)}$

 Table 5.4: $A_{(i,i \oplus n 1)}^n$ for $0 \leq i \leq n$

Name	Source node	Binding Element	Dest. Node
$a_{(i,i \oplus n 1)}^{(n,1)}$	$M_{(i,i \oplus n 1)}^{(n,1)}$	send_ack<rn=i \oplus n 1, queue=[]>	$M_{(i,i \oplus n 1)}^{(n,2)}$
$a_{(i,i \oplus n 1)}^{(n,2)}$	$M_{(i,i \oplus n 1)}^{(n,2)}$	ack_loss<rn=i \oplus n 1, queue=[]>	$M_{(i,i \oplus n 1)}^{(n,3)}$
$a_{(i,i \oplus n 1)}^{(n,3)}$	$M_{(i,i \oplus n 1)}^{(n,2)}$	receive_ack<sn=i, rn=i \oplus n 1, rc=0, queue=[]>	$M_{(i \oplus n 1, i \oplus n 1)}^{(n,1)}$

5.1.1 Base case

Let us now formally define the base case of $\text{MaxSeqNo}=1$ corresponding to $OG_1 = (V_1, A_1)$ as shown in Fig. 4.1 (a). The set of nodes corresponds to $V_1 = V_{(0,0)}^1 \cup V_{(0,1)}^1 \cup V_{(1,1)}^1 \cup V_{(1,0)}^1$, i.e. the set of nodes defined by Tables 5.1 and 5.2 evaluated for $0 \leq i \leq 1$. The set of arcs is similarly expressed by $A_1 = A_{(0,0)}^1 \cup A_{(0,1)}^1 \cup A_{(1,1)}^1 \cup A_{(1,0)}^1$, i.e. the set of arcs defined by Tables 5.3 and 5.4 evaluated for $0 \leq i \leq 1$. V_1 and A_1 are explicitly represented in Tables 5.5 and 5.6 respectively.

5.1.2 Recursive Expression

Now that we have an explicit representation of the base case of $\text{MaxSeqNo}=1$, let us formalise the intuition from Chapter 4 and define the set of nodes and arcs to remove and the set of nodes and arcs to add, to obtain OG_k from OG_{k-1} .

The set of nodes to add to V_{n-1} to obtain V_n corresponds to the set

$$V_n^{add} = V_{(n-1,n)}^n \cup V_{(n,n)}^n \cup V_{(n,0)}^n \quad (5.5)$$

i.e. the set of nodes specified by evaluating Table 5.2 for $i = n - 1$ and $i = n$, and Table 5.1 for $i = n$. The set V_n^{add} is explicitly defined in Table 5.7. The set of nodes to delete from V_{n-1} to obtain V_n corresponds to the set

$$V_n^{del} = V_{(n-1,0)}^{n-1} \quad (5.6)$$

i.e. the set of nodes specified by evaluating Table 5.2 for $i = n - 1$. The set V_n^{del} is explicitly defined in Table 5.8.

The set of arcs to add to A_{n-1} to obtain A_n corresponds to the set

$$A_n^{add} = A_{(n-1,n)}^n \cup A_{(n,n)}^n \cup A_{(n,0)}^n \cup \{a_{(n-1,n-1)}^{(n,3)}\} \quad (5.7)$$

i.e. the set of arcs specified by evaluating Table 5.4 for $i = n - 1$ and $i = n$, Table 5.3 for $i = n$, and the third arc from the set defined by evaluating Table 5.3 for $i = n - 1$. The set A_n^{add} is defined explicitly in Table 5.9. The set of arcs to delete from A_{n-1} to obtain A_n corresponds to the set

$$A_n^{del} = A_{(n-1,0)}^{n-1} \cup \{a_{(n-1,n-1)}^{(n-1,3)}\} \quad (5.8)$$

Table 5.5: V_1 - Nodes for the base case of $\text{MaxSeqNo}=1$.

Node	sender_state	send_seq_no	mess_ch.	ack_ch.	receiver_state	recv_seq_no
$M_{(0,0)}^{(1,1)}$	s_ready	0	[]	[]	r_ready	0
$M_{(0,0)}^{(1,2)}$	wait_ack	0	[0]	[]	r_ready	0
$M_{(0,0)}^{(1,3)}$	wait_ack	0	[]	[]	r_ready	0
$M_{(0,1)}^{(1,1)}$	wait_ack	0	[]	[]	process	1
$M_{(0,1)}^{(1,2)}$	wait_ack	0	[]	[1]	r_ready	1
$M_{(0,1)}^{(1,3)}$	wait_ack	0	[]	[]	r_ready	1
$M_{(1,1)}^{(1,1)}$	s_ready	1	[]	[]	r_ready	1
$M_{(1,1)}^{(1,2)}$	wait_ack	1	[1]	[]	r_ready	1
$M_{(1,1)}^{(1,3)}$	wait_ack	1	[]	[]	r_ready	1
$M_{(1,0)}^{(1,1)}$	wait_ack	1	[]	[]	process	0
$M_{(1,0)}^{(1,2)}$	wait_ack	1	[]	[0]	r_ready	0
$M_{(1,0)}^{(1,3)}$	wait_ack	1	[]	[]	r_ready	0

 Table 5.6: A_1 - Arcs for the base case of $\text{MaxSeqNo}=0$.

Name	Source node	Binding Element	Dest. Node
$a_{(0,0)}^{(1,1)}$	$M_{(0,0)}^{(1,1)}$	send_mess<sn=0,queue=[]>	$M_{(0,0)}^{(1,2)}$
$a_{(0,0)}^{(1,2)}$	$M_{(0,0)}^{(1,2)}$	mess_loss<sn=0,queue=[]>	$M_{(0,0)}^{(1,3)}$
$a_{(0,0)}^{(1,3)}$	$M_{(0,0)}^{(1,2)}$	receive_mess<sn=0,rn=0,queue=[]>	$M_{(0,1)}^{(1,1)}$
$a_{(0,1)}^{(1,1)}$	$M_{(0,1)}^{(1,1)}$	send_ack<rn=1,queue=[]>	$M_{(0,1)}^{(1,2)}$
$a_{(0,1)}^{(1,2)}$	$M_{(0,1)}^{(1,2)}$	ack_loss<rn=1, queue=[]>	$M_{(0,1)}^{(1,3)}$
$a_{(0,1)}^{(1,3)}$	$M_{(0,1)}^{(1,2)}$	receive_ack<sn=0, rn=1, rc=0, queue=[]>	$M_{(1,1)}^{(1,1)}$
$a_{(1,1)}^{(1,1)}$	$M_{(1,1)}^{(1,1)}$	send_mess<sn=1,queue=[]>	$M_{(1,1)}^{(1,2)}$
$a_{(1,1)}^{(1,2)}$	$M_{(1,1)}^{(1,2)}$	mess_loss<sn=1,queue=[]>	$M_{(1,1)}^{(1,3)}$
$a_{(1,1)}^{(1,3)}$	$M_{(1,1)}^{(1,2)}$	receive_mess<sn=1,rn=1,queue=[]>	$M_{(1,0)}^{(1,1)}$
$a_{(1,0)}^{(1,1)}$	$M_{(1,0)}^{(1,1)}$	send_ack<rn=0,queue=[]>	$M_{(1,0)}^{(1,2)}$
$a_{(1,0)}^{(1,2)}$	$M_{(1,0)}^{(1,2)}$	ack_loss<rn=0, queue=[]>	$M_{(1,0)}^{(1,3)}$
$a_{(1,0)}^{(1,3)}$	$M_{(1,0)}^{(1,2)}$	receive_ack<sn=1, rn=0, rc=0, queue=[]>	$M_{(0,0)}^{(1,1)}$

i.e. the arcs specified by evaluating Table 5.4 for $i = n - 1$ and the third arc from the set defined by evaluating Table 5.3 for $i = n - 1$. The set A_n^{del} is defined explicitly in Table 5.10.

We are now ready to state a recursive expression for the OG of CPN_n for $n > 0$.

Theorem 1. For $n \in \mathbb{N}^+$, $OG_n = (V_n, A_n)$ where

1. $OG_1 = (V_1, A_1)$ is the OG as given in Fig. 4.1 (a) and specified in Tables 5.5 and 5.6; and

Table 5.7: V_n^{add} - Nodes to add to obtain V_n from V_{n-1} .

Node	sender_state	send_seq_no	mess_ch.	ack_ch.	receiver_state	recv_seq_no
$M_{(n-1,n)}^{(n,1)}$	wait_ack	n-1	[]	[]	process	n
$M_{(n-1,n)}^{(n,2)}$	wait_ack	n-1	[]	[n]	r_ready	n
$M_{(n-1,n)}^{(n,3)}$	wait_ack	n-1	[]	[]	r_ready	n
$M_{(n,n)}^{(n,1)}$	s_ready	n	[]	[]	r_ready	n
$M_{(n,n)}^{(n,2)}$	wait_ack	n	[n]	[]	r_ready	n
$M_{(n,n)}^{(n,3)}$	wait_ack	n	[]	[]	r_ready	n
$M_{(n,0)}^{(n,1)}$	wait_ack	n	[]	[]	process	0
$M_{(n,0)}^{(n,2)}$	wait_ack	n	[]	[0]	r_ready	0
$M_{(n,0)}^{(n,3)}$	wait_ack	n	[]	[]	r_ready	0

Table 5.8: V_n^{del} - Nodes to delete to obtain V_n from V_{n-1} .

Node	sender_state	send_seq_no	mess_ch.	ack_ch.	receiver_state	recv_seq_no
$M_{(n-1,0)}^{(n-1,1)}$	wait_ack	n-1	[]	[]	process	0
$M_{(n-1,0)}^{(n-1,2)}$	wait_ack	n-1	[]	[0]	r_ready	0
$M_{(n-1,0)}^{(n-1,3)}$	wait_ack	n-1	[]	[]	r_ready	0

2. for $n \geq 2$, $OG_n = (V_{n-1} \cup V_n^{add} \setminus V_n^{del}, A_{n-1} \cup A_n^{add} \setminus A_n^{del})$, where

- $V_n^{add} = V_{(n-1,n)}^n \cup V_{(n,n)}^n \cup V_{(n,0)}^n$ (Equation 5.5) are the new nodes, the nodes in OG_n that are not in OG_{n-1} , as given in Table 5.7;
- $V_n^{del} = V_{(n-1,0)}^{n-1}$ (Equation 5.6) are the nodes to be deleted, the nodes in OG_{n-1} that are not in OG_n , as given in Table 5.8;
- $A_n^{add} = A_{(n-1,n)}^n \cup A_{(n,n)}^n \cup A_{(n,0)}^n \cup \{a_{(n-1,n-1)}^{(n,3)}\}$ (Equation 5.7) are the new arcs, the arcs in OG_n that are not in OG_{n-1} , as given in Table 5.9; and
- $A_n^{del} = A_{(n-1,0)}^{n-1} \cup \{a_{(n-1,n-1)}^{(n-1,3)}\}$ (Equation 5.8) are the arcs to be deleted, the arcs in OG_{n-1} that are not in OG_n , as given in Table 5.10.

The proof of this theorem is given in Section 5.3.

5.2 An Explicit Algebraic Formula for the OG

The recursive construction given above follows directly from the observations of the difference between two OGs for consecutive values of the `MaxSeqNo` parameter. However, it is possible to solve this first-order difference equation by inspection and state an explicit algebraic formula for the OG directly, without recursion.

We have already partitioned the nodes into sets of size 3, specified by $V_{(i,i)}^n$ (Table 5.1) and $V_{(i,i \oplus_n 1)}^n$ (Table 5.2) for a given i such that $0 \leq i \leq n$. Similarly, the arcs have also been partitioned into sets of size 3, namely $A_{(i,i)}^n$ and $A_{(i,i \oplus_n 1)}^n$. We know that the set of nodes of OG_1 is

Table 5.9: A_n^{add} - Arcs to add to obtain A_n from A_{n-1} .

Name	Source node	Binding Element	Dest. Node
$a_{(n-1,n-1)}^{(n,3)}$	$M_{(n-1,n-1)}^{(n,2)}$	receive_mess<sn=n-1,rn=n-1,queue=[]>	$M_{(n-1,n)}^{(n,1)}$
$a_{(n-1,n)}^{(n,1)}$	$M_{(n-1,n)}^{(n,1)}$	send_ack<rn=n,queue=[]>	$M_{(n-1,n)}^{(n,2)}$
$a_{(n-1,n)}^{(n,2)}$	$M_{(n-1,n)}^{(n,2)}$	ack_loss<rn=n, queue=[]>	$M_{(n-1,n)}^{(n,3)}$
$a_{(n-1,n)}^{(n,3)}$	$M_{(n-1,n)}^{(n,2)}$	receive_ack<sn=n-1, rn=n, rc=0, queue=[]>	$M_{(n,n)}^{(n,1)}$
$a_{(n,n)}^{(n,1)}$	$M_{(n,n)}^{(n,1)}$	send_mess<sn=n,queue=[]>	$M_{(n,n)}^{(n,2)}$
$a_{(n,n)}^{(n,2)}$	$M_{(n,n)}^{(n,2)}$	mess_loss<sn=n,queue=[]>	$M_{(n,n)}^{(n,3)}$
$a_{(n,n)}^{(n,3)}$	$M_{(n,n)}^{(n,2)}$	receive_mess<sn=n,rn=n,queue=[]>	$M_{(n,0)}^{(n,1)}$
$a_{(n,0)}^{(n,1)}$	$M_{(n,0)}^{(n,1)}$	send_ack<rn=0,queue=[]>	$M_{(n,0)}^{(n,2)}$
$a_{(n,0)}^{(n,2)}$	$M_{(n,0)}^{(n,2)}$	ack_loss<rn=0, queue=[]>	$M_{(n,0)}^{(n,3)}$
$a_{(n,0)}^{(n,3)}$	$M_{(n,0)}^{(n,2)}$	receive_ack<sn=n, rn=0, rc=0, queue=[]>	$M_{(0,0)}^{(n,1)}$

 Table 5.10: A_n^{del} - Arcs to delete to obtain A_n from A_{n-1} .

Name	Source node	Binding Element	Dest. Node
$a_{(n-1,n-1)}^{(n-1,3)}$	$M_{(n-1,n-1)}^{(n-1,2)}$	receive_mess<sn=n-1,rn=n-1,queue=[]>	$M_{(n-1,0)}^{(n-1,1)}$
$a_{(n-1,0)}^{(n-1,1)}$	$M_{(n-1,0)}^{(n-1,1)}$	send_ack<rn=0,queue=[]>	$M_{(n-1,0)}^{(n-1,2)}$
$a_{(n-1,0)}^{(n-1,2)}$	$M_{(n-1,0)}^{(n-1,2)}$	ack_loss<rn=0, queue=[]>	$M_{(n-1,0)}^{(n-1,3)}$
$a_{(n-1,0)}^{(n-1,3)}$	$M_{(n-1,0)}^{(n-1,2)}$	receive_ack<sn=n-1, rn=0, rc=0, queue=[]>	$M_{(0,0)}^{(n-1,1)}$

exactly specified by the union of the evaluation of the sets $V_{(i,i)}^n$ and $V_{(i,i\oplus n1)}^n$ for each value of i from 0 to n , which can be confirmed by direct inspection of OG_1 . Similarly, the set of arcs of OG_1 is exactly specified by the union of the evaluation of the sets $A_{(i,i)}^n$ and $A_{(i,i\oplus n1)}^n$ for all values of i from 0 to n . The same is true of OG_2 , and so on. We are therefore able to specify the OG directly, for any value of n (MaxSeqNo), as a union of sets of nodes and a union of sets of arcs. The sets of nodes and arcs are determined by evaluation of the four sets as previously described, with the maximum value of i determined by the value of the MaxSeqNo parameter.

We state the theorem for this parametric OG below.

Theorem 2. For $n \in \mathbb{N}^+$, $OG_n = (V_n, A_n)$ where

$$V_n = \bigcup_{0 \leq i \leq n} (V_{(i,i)}^n \cup V_{(i,i\oplus n1)}^n), \quad A_n = \bigcup_{0 \leq i \leq n} (A_{(i,i)}^n \cup A_{(i,i\oplus n1)}^n)$$

The proof is given in Section 5.3.

5.3 Proof of Recursive and Algebraic Expressions

Both Theorem 1 and 2 can be proved in a near-identical way. We present a combined proof below.

Proof. We prove Theorem 1 and 2 by induction over the maximum sequence number MaxSeqNo (n). The proof draws inspiration from [27] for recursively defining a service OG. Firstly we prove the basis case for $n=1$ using Design/CPN. Then we prove the induction step in two parts. The first part shows that all but the last 3 nodes in OG_k ($V_{(k,0)}^k$) are identical to the nodes in OG_{k+1} , with a similar situation holding for the arcs. Then, using reachability analysis, we generate the last 9 nodes and 10 arcs for OG_{k+1} .

Basis. We assume that Design/CPN can correctly generate an OG of a CPN model with a finite occurrence graph. Thus OG_1 as shown in Fig. 4.1(a) matches the nodes and arcs defined in Tables 5.5 and 5.6 (Theorem 1) and also matches the nodes and arcs defined in Tables 5.1 to 5.4 for $n=1$ (Theorem 2).

Induction. We assume both Theorem 1 and 2 to be true for $n=k$. This gives us:

$$OG_k = (V_{k-1} \cup V_k^{add} \setminus V_k^{del}, A_{k-1} \cup A_k^{add} \setminus A_k^{del})$$

for the recursive expression; and

$$OG_k = \left(\bigcup_{0 \leq i \leq n} (V_{(i,i)}^n \cup V_{(i,i \oplus n)}^n), \bigcup_{0 \leq i \leq n} (A_{(i,i)}^n \cup A_{(i,i \oplus n)}^n) \right) \quad (5.9)$$

for the explicit algebraic expression. We now prove that both hold for $n = k + 1$.

The following lemma states that the behaviour of the CPN is unchanged for $n = k$ and $n = k + 1$ up to the point where sequence numbers wrap.

Lemma 1. *For CPN_k and OG_k as given in Definition 2, the subgraph of OG_k corresponding to $(V_k \setminus V_{(k,0)}^k, A_k \setminus (A_{(k,0)}^k \cup \{a_{(k,k)}^{(k,3)}\}))$ is invariant on increasing MaxSeqNo from k to $k + 1$, where $V_{(k,0)}^k$ and $A_{(k,0)}^k$ are defined in Tables 5.2 and 5.4 respectively by substituting $i=k$, and $a_{(k,k)}^{(k,3)}$ is the third arc in Table 5.3 with $i=k$.*

Proof. By examining the CPN model, it can be seen that the only way that the parameter MaxSeqNo affects the model is through the NextSeq function. Further, an increase in MaxSeqNo only affects the behaviour of the model when sequence numbers wrap, i.e. in CPN_k , $\text{NextSeq}(k) = 0$, but in CPN_{k+1} , $\text{NextSeq}(k) = k + 1$.

Because NextSeq behaves identically up to the point where wrapping occurs in OG_k , we conclude that the behaviour of the CPN, (thus the OG) is unchanged for all values of sender and receiver sequence number up to the point where wrapping occurs, i.e. for the subset of nodes $V_{(0,0)}^k \cup V_{(0,1)}^k \cup V_{(1,1)}^k \cup \dots \cup V_{(k,k)}^k \subseteq V_k$. The arc $a_{(k,k)}^{(k,3)}$ where receive_mess occurs from $M_{(k,k)}^{(k,2)} \in V_{(k,k)}^k$ causes the receiver sequence number to wrap to 0, so the markings in $V_{(k,0)}^k$ are no longer reachable when n is increased from k to $k + 1$. The subset of arcs that remains unaffected is given by $A_{(0,0)}^k \cup A_{(0,1)}^k \cup A_{(1,1)}^k \cup \dots \cup A_{(k,k)}^k \setminus \{a_{(k,k)}^{(k,3)}\} \subseteq A_k$. The remaining arcs $(A_{(k,0)}^k \cup \{a_{(k,k)}^{(k,3)}\})$ can no longer occur as either their source or destination nodes (or both) are no longer reachable when MaxSeqNo equals $k + 1$. \square

Corollary 1. *Following directly from Lemma 1:*

$$\forall 0 \leq i \leq k, \quad V_{(i,i)}^{k+1} = V_{(i,i)}^k \quad (5.10)$$

$$\forall 0 \leq i \leq k - 1, \quad V_{(i,i+1)}^{k+1} = V_{(i,i+1)}^k \quad (5.11)$$

$$\forall 0 \leq i \leq k - 1, \quad A_{(i,i)}^{k+1} = A_{(i,i)}^k \quad (5.12)$$

$$\forall 0 \leq i \leq k - 1, \quad A_{(i,i+1)}^{k+1} = A_{(i,i+1)}^k \quad (5.13)$$

$$a_{(k,k)}^{(k,1)} = a_{(k,k)}^{(k+1,1)} \quad \text{and} \quad a_{(k,k)}^{(k,2)} = a_{(k,k)}^{(k+1,2)} \quad (5.14)$$

Invoking the induction step, because we assume that our expressions for V_k and A_k are correct, we can assume that the nodes and arcs remaining invariant in V_{k+1} and A_{k+1} are also correct. Having determined the subset of nodes and arcs from OG_k that are invariant in the occurrence graph of CPN_{k+1} we must now determine the set of additional reachable markings and arcs:

Lemma 2. For CPN_{k+1} and the set of invariant nodes from Lemma 1, the set of additional reachable markings of CPN_{k+1} is given by $V_{(k,k+1)}^{k+1} \cup V_{(k+1,k+1)}^{k+1} \cup V_{(k+1,0)}^{k+1}$

Lemma 3. For CPN_{k+1} and the set of invariant arcs from Lemma 1, the set of additional arcs in OG_{k+1} is given by $A_{(k,k+1)}^{k+1} \cup A_{(k+1,k+1)}^{k+1} \cup A_{(k+1,0)}^{k+1} \cup \{a_{(k,k)}^{(k+1,3)}\}$.

Proof. From Lemma 1, the behaviour of CPN_{k+1} is the same as CPN_k up until the point where sequence numbers wrap. The occurrence of `receive_mess`<sn=k, rn=k, queue=[]> in $M_{(k,k)}^{(k+1,2)}$ leads to a new marking, $M_{(k,k+1)}^{(k+1,1)}$ in which the receiver sequence number is $k+1$ (not 0). The only transition enabled in $M_{(k,k+1)}^{(k+1,1)}$ is `send_ack` with binding <rn=k+1, queue=[]>. Occurrence of this leads to the marking $M_{(k,k+1)}^{(k+1,2)}$. From here, either the acknowledgement just sent may be lost through occurrence of `ack_loss`<rn=k+1, queue=[]> leading to the dead marking $M_{(k,k+1)}^{(k+1,3)}$, or the acknowledgement will be received and the sender sequence number incremented through occurrence of `receive_ack`<sn=k, rn=k+1, rc=0, queue=[]>, leading to the marking $M_{(k+1,k+1)}^{(k+1,1)}$.

From $M_{(k+1,k+1)}^{(k+1,1)}$ the only possible action is `send_mess`<sn=k+1, queue=[]> leading to the marking $M_{(k+1,k+1)}^{(k+1,2)}$. Two transitions are now enabled, namely `mess_loss` and `receive_mess`. Occurrence of `mess_loss`<sn=k+1, queue=[]> leads to the dead marking $M_{(k+1,k+1)}^{(k+1,3)}$. Occurrence of `receive_mess`<sn=k+1, rn=k+1, queue=[]> leads to marking $M_{(k+1,0)}^{(k+1,1)}$.

From this marking the only possible action is `send_ack`<rn=0, queue=[]> leading to the marking $M_{(k+1,0)}^{(k+1,2)}$. The acknowledgement just sent may be lost (`ack_loss`<rn=0, queue=[]>) leading to the dead marking $M_{(k+1,0)}^{(k+1,3)}$, or the acknowledgement may be received successfully by the sender (`receive_ack`<sn=k+1, rn=0, rc=0, queue=[]>) leading back to the initial marking, $M_{(0,0)}^{(k+1,1)}$.

From Equations 5.1 and 5.2 defining the sets of nodes, the 9 nodes generated above correspond to $V_{(k,k+1)}^{k+1} \cup V_{(k+1,k+1)}^{k+1} \cup V_{(k+1,0)}^{k+1}$, obtained by substituting $n = k+1$ into Tables 5.1 and 5.2 and then evaluating Table 5.2 for $i = k$ and both Tables 5.1 and 5.2 for $i = k+1$.

From Equations 5.3 and 5.4 the 10 new arcs correspond to $A_{(k,k+1)}^{k+1} \cup A_{(k+1,k+1)}^{k+1} \cup A_{(k+1,0)}^{k+1} \cup \{a_{(k,k)}^{(k+1,3)}\}$, obtained in a similar way. \square

It follows that the set of nodes V_{k+1} is equal to the union of the set of invariant nodes $V_k \setminus V_{(k,0)}^k$ from Lemma 1 and the set of additional reachable nodes from Lemma 2. Thus:

$$V_{k+1} = V_k \setminus V_{(k,0)}^k \cup V_{(k,k+1)}^{k+1} \cup V_{(k+1,k+1)}^{k+1} \cup V_{(k+1,0)}^{k+1} \quad (5.15)$$

Similarly, it follows that the set of arcs A_{k+1} is equal to the union of the set of invariant arcs from Lemma 1 and the set of additional arcs from Lemma 3, i.e.

$$A_{k+1} = A_k \setminus (A_{(k,0)}^k \cup \{a_{(k,k)}^{(k,3)}\}) \cup A_{(k,k+1)}^{k+1} \cup A_{(k+1,k+1)}^{k+1} \cup A_{(k+1,0)}^{k+1} \cup \{a_{(k,k)}^{(k+1,3)}\} \quad (5.16)$$

To complete the proof of Theorem 1 we invoke on Equation 5.15 the definitions of V_n^{add} and V_n^{del} from Equations 5.5 and 5.6 for $n = k+1$ to obtain

$$\begin{aligned} V_{k+1} &= V_k \cup V_{(k,k+1)}^{k+1} \cup V_{(k+1,k+1)}^{k+1} \cup V_{(k+1,0)}^{k+1} \setminus V_{(k,0)}^k \\ &= V_k \cup V_{k+1}^{add} \setminus V_{k+1}^{del} \end{aligned}$$

Similarly, the definitions of A_n^{add} and A_n^{del} from Equations 5.7 and 5.8 for $n = k + 1$ are substituted into Equation 5.16 to obtain

$$\begin{aligned} A_{k+1} &= A_k \cup A_{(k,k+1)}^{k+1} \cup A_{(k+1,k+1)}^{k+1} \cup A_{(k+1,0)}^{k+1} \cup \{a_{(k,k)}^{(k+1,3)}\} \setminus (A_{(k,0)}^k \cup \{a_{(k,k)}^{(k,3)}\}) \\ &= A_k \cup A_{k+1}^{add} \setminus A_{k+1}^{del} \end{aligned}$$

Thus $OG_{k+1} = (V_k \cup V_{k+1}^{add} \setminus V_{k+1}^{del}, A_k \cup A_{k+1}^{add} \setminus A_{k+1}^{del})$, the induction holds, and Theorem 1 is proved.

To complete the proof of Theorem 2 we use alternate substitutions into Equations 5.15 and 5.16. We begin with the set of nodes:

$$\begin{aligned} V_{k+1} &= V_k \cup V_{(k,k+1)}^{k+1} \cup V_{(k+1,k+1)}^{k+1} \cup V_{(k+1,0)}^{k+1} \setminus V_{(k,0)}^k && \text{(Equation 5.15)} \\ &= \bigcup_{0 \leq i \leq k} (V_{(i,i)}^k \cup V_{(i,i \oplus_n 1)}^k) \cup V_{(k,k+1)}^{k+1} \cup V_{(k+1,k+1)}^{k+1} \cup V_{(k+1,0)}^{k+1} \setminus V_{(k,0)}^k && \text{(Substitution of } V_k \\ & && \text{from Eqn. 5.9)} \\ &= \left(\bigcup_{0 \leq i \leq k} V_{(i,i)}^k \right) \cup \left(\bigcup_{0 \leq i \leq k-1} V_{(i,i \oplus_n 1)}^k \right) \cup V_{(k,k+1)}^{k+1} \cup V_{(k+1,k+1)}^{k+1} \cup V_{(k+1,0)}^{k+1} && \text{(Cancellation} \\ & && \text{of terms)} \\ &= \left(\bigcup_{0 \leq i \leq k} V_{(i,i)}^{k+1} \right) \cup \left(\bigcup_{0 \leq i \leq k-1} V_{(i,i \oplus_n 1)}^{k+1} \right) \cup V_{(k,k+1)}^{k+1} \cup V_{(k+1,k+1)}^{k+1} \cup V_{(k+1,0)}^{k+1} && \text{(From Equations.} \\ & && \text{5.10 and 5.11)} \\ &= \bigcup_{0 \leq i \leq k+1} (V_{(i,i)}^{k+1} \cup V_{(i,i \oplus_n 1)}^{k+1}) && \text{(Simplification)} \end{aligned}$$

Similarly for the set of arcs:

$$\begin{aligned} A_{k+1} &= A_k \cup A_{(k,k+1)}^{k+1} \cup A_{(k+1,k+1)}^{k+1} \cup A_{(k+1,0)}^{k+1} \cup \{a_{(k,k)}^{(k+1,3)}\} \\ & \quad \setminus (A_{(k,0)}^k \cup \{a_{(k,k)}^{(k,3)}\}) && \text{(Equation 5.16)} \\ &= \bigcup_{0 \leq i \leq k} (A_{(i,i)}^k \cup A_{(i,i \oplus_n 1)}^k) \cup A_{(k,k+1)}^{k+1} \cup A_{(k+1,k+1)}^{k+1} \cup A_{(k+1,0)}^{k+1} \\ & \quad \cup \{a_{(k,k)}^{(k+1,3)}\} \setminus (A_{(k,0)}^k \cup \{a_{(k,k)}^{(k,3)}\}) && \text{(Substitution of } A_k \\ & && \text{from Eqn. 5.9)} \\ &= \left(\bigcup_{0 \leq i \leq k-1} A_{(i,i)}^k \right) \cup \left(\bigcup_{0 \leq i \leq k-1} A_{(i,i \oplus_n 1)}^k \right) \cup A_{(k,k+1)}^{k+1} \cup A_{(k+1,k+1)}^{k+1} \\ & \quad \cup A_{(k+1,0)}^{k+1} \cup \{a_{(k,k)}^{(k+1,3)}\} \cup \{a_{(k,k)}^{(k,1)}, a_{(k,k)}^{(k,2)}\} && \text{(Cancellation of terms)} \\ &= \left(\bigcup_{0 \leq i \leq k-1} A_{(i,i)}^{k+1} \right) \cup \left(\bigcup_{0 \leq i \leq k-1} A_{(i,i \oplus_n 1)}^{k+1} \right) \cup A_{(k,k+1)}^{k+1} \cup A_{(k+1,k+1)}^{k+1} \\ & \quad \cup A_{(k+1,0)}^{k+1} \cup \{a_{(k,k)}^{(k+1,1)}, a_{(k,k)}^{(k+1,2)}, a_{(k,k)}^{(k+1,3)}\} && \text{(From Equations 5.12,} \\ & && \text{5.13 and 5.14)} \\ &= \left(\bigcup_{0 \leq i \leq k-1} A_{(i,i)}^{k+1} \right) \cup \left(\bigcup_{0 \leq i \leq k-1} A_{(i,i \oplus_n 1)}^{k+1} \right) \cup A_{(k,k+1)}^{k+1} \cup A_{(k+1,k+1)}^{k+1} \\ & \quad \cup A_{(k+1,0)}^{k+1} \cup A_{(k,k)}^{k+1} && \text{(Definition of } A_{(k,k)}^{k+1}) \\ &= \bigcup_{0 \leq i \leq k+1} (A_{(i,i)}^{k+1} \cup A_{(i,i \oplus_n 1)}^{k+1}) && \text{(Simplification)} \end{aligned}$$

Thus $V_{k+1} = \bigcup_{0 \leq i \leq k+1} (V_{(i,i)}^{k+1} \cup V_{(i,i \oplus_n 1)}^{k+1})$ and $A_{k+1} = \bigcup_{0 \leq i \leq k+1} (A_{(i,i)}^{k+1} \cup A_{(i,i \oplus_n 1)}^{k+1})$, the induction holds and Theorem 2 is proved. \square

Chapter 6

The Algebraic Protocol Language

Of the many properties of interest to be investigated (bounds, deadlocks, livelocks) we firstly consider conformance to the Stop-and-Wait property of alternating send and receive events. (We hesitate to call this the Stop-and-Wait *service* because the service should encompass all properties of interest, not just allowable sequences of events.) The protocol language comprises all sequences of user-observable events (*service primitives*) exhibited by the protocol. The Stop-and-Wait property is concerned with sending and receiving data only and the acknowledgement and retransmission mechanisms (including sequence numbers) are not visible to users of the protocol. We therefore define the service primitives of the Stop-and-Wait property as $SP = \{\text{send}, \text{receive}\}$.

Obtaining an expression for the protocol language from the parametric OG will enable us to verify the conformance of the SWP to the Stop-and-Wait property for *all* values of MaxSeqNo . The Stop-and-Wait property over a lossy medium is defined as $(\text{send}, \text{receive})^* \text{send}^\dagger$ where send^\dagger represents 0 or 1 occurrences of send . This represents that the last send event may not be followed by a corresponding receive event if the last message that was sent is lost, i.e. the system will halt.

We state this in the following theorem, which is proved in the following sections.

Theorem 3. *The Stop-and-Wait protocol, as defined by the CPN in Figs. 3.1 and 3.2 with the MaxRe-trans parameter set to 0, conforms to the Stop-and-Wait property of alternating send and receive events, i.e. $(\text{send}, \text{receive})^* \text{send}^\dagger$ for all values of MaxSeqNo , where send^\dagger represents 0 or 1 occurrences of send .*

6.1 Mapping from the Algebraic OG to an Algebraic FSA

By interpreting the parametric OG of Theorem 2 as a Finite State Automaton (FSA) and relabelling binding elements as either service primitives or epsilon (empty) moves, standard algorithms [4] can be used to obtain the minimal deterministic FSA of the protocol language. The complication is that these standard algorithms must be applied to a parametric representation of a FSA.

The explicit algebraic formula of Theorem 2 is chosen for this task in preference to the recursive expression of Theorem 1 because the standard FSA manipulation algorithms are designed for explicit FSAs and thus intuition suggests it will be easier to apply such algorithms to an explicit parametric FSA rather than a recursively defined FSA.

Following the methodology from [10] would require us to define an *abstract occurrence graph* with respect to *markings* for our Stop-and-Wait CPN using a mapping from markings to integers, before defining a mapping from the abstract OG to its FSA. In our case, however, the intermediate step of defining an abstract OG with respect to markings is not necessary. Our situation is simple enough to allow us to map directly from OG_n to its FSA. We begin by defining the necessary mappings to relabel the markings with integers and relabel the arcs with service primitives (or ϵ):

Definition 3. Let $I : [M_0] \rightarrow \mathbb{N}^+$ be an injection, mapping from the reachable markings of CPN_n with initial marking M_0 into the set of positive integers.

Let $Prim : BE'_n \rightarrow SP \cup \{\epsilon\}$ be a mapping from the set of binding elements of CPN_n to either a service primitive name or to ϵ , where

- $BE'_n \subseteq BE_n$ is the set of binding elements that occur in CPN_n ; and
- $SP = \{\text{send}, \text{receive}\}$.

To relabel markings $M_{(i,j)}^{(n,t)}$ in OG_n with an integer we define the injection I as $I(M_{(i,j)}^{(n,t)}) = 6i + t$ when $j = i$ and $3(2i + 1) + t$ when $j = i \oplus_n 1$, where $1 \leq t \leq 3$ corresponds to the three ‘types’ of markings described in Chapter 4 and $0 \leq i \leq n$. This numbers the markings as in Fig. 4.1. We do not simplify the node label $3(2i + 1) + t$ to $6i + (t + 3)$ as in doing so, the useful information about the type t of the marking is not readily evident from the node label.

The mapping $Prim$ will map all occurrences of `send_mess` to the primitive `send` and only those occurrences of `receive_mess` corresponding to acceptance of a new message (`sn = rn`) to the primitive `receive`. All other transition occurrences (including occurrences of `receive_mess` corresponding to detection and discarding of a duplicate) are mapped to ϵ .

All that remains to interpret OG_n as a FSA is to define the initial and halt states. We define the initial state of the FSA as the equivalent initial marking of the CPN, i.e. $I(M_0) = I(M_{(0,0)}^{(n,1)}) = 1$. We have an arbitrary number of messages to send from the sender to the receiver, and so we define a legitimate halt state as any state in which $l \in \mathbb{N}$ messages have been transmitted and successfully acknowledged, so that both the sender and receiver are in their ready states and there are no messages or acknowledgements in the channel. This corresponds to the markings $I(M_{(i,i)}^{(n,1)})$ for all $0 \leq i \leq n$. We also include the dead markings of OG_n in the set of halt states, i.e. $I(M_{(i,i)}^{(n,3)})$ and $I(M_{(i,i \oplus_n 1)}^{(n,3)})$ for all $0 \leq i \leq n$. Because we have no retransmissions, the loss of a message or acknowledgement will lead directly to a dead marking, as the system cannot recover. In line with our definition of the Stop-and-Wait property, these dead markings represent expected halt states of the protocol when operating over a lossy medium.

We are now ready to define the FSA associated with OG_n :

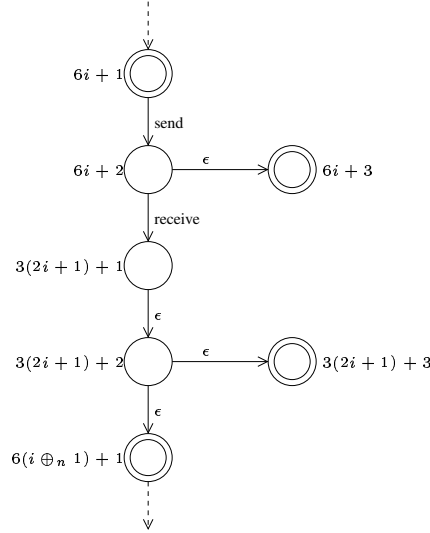
Definition 4. The FSA associated with OG_n of CPN_n , with initial marking M_0 , is $FSA_{OG_n} = (V_{FSA}^n, SP, A_{FSA}^n, v_0, F_{FSA}^n)$ where

- $V_{FSA}^n = \{I(M) | M \in V_n\}$ is the set of nodes;
- $SP = \{\text{send}, \text{receive}\}$ is the set of service primitive names of interest (the alphabet of the FSA);
- $A_{FSA}^n = \{I(M), Prim(b), I(M') | (M, b, M') \in A_n\}$ is the set of transitions labelled by service primitives or epsilons for internal events (the transition relation of the FSA);
- $v_0 = I(M_0) = I(M_{(0,0)}^{(n,1)}) = 1$ is the initial state of the FSA as defined above, corresponding to the initial marking of CPN_n ; and
- $F_{FSA}^n \subseteq V_{FSA}^n$ is the set of final states defined above as $\bigcup_{0 \leq i \leq n} \{I(M_{(i,i)}^{(n,1)}), I(M_{(i,i)}^{(n,3)}), I(M_{(i,i \oplus_n 1)}^{(n,3)})\}$.

Table 6.1 summarises FSA_{OG_n} and Fig. 6.1 gives the graphical structure of Table 6.1. To illustrate the mapping from the OG to the FSA, we continue with our `MaxSeqNo = 1` example by relabelling the markings, applying the mapping $Prim$ to the binding elements and marking the initial

Table 6.1: FSA_{OG_n} where $0 \leq i \leq n$

Source node	Arc Label	Dest. node	Dest. = Halt?
$6i + 1$	send	$6i + 2$	false
$6i + 2$	ϵ	$6i + 3$	true
$6i + 2$	receive	$3(2i + 1) + 1$	false
$3(2i + 1) + 1$	ϵ	$3(2i + 1) + 2$	false
$3(2i + 1) + 2$	ϵ	$3(2i + 1) + 3$	true
$3(2i + 1) + 2$	ϵ	$6(i \oplus_n 1) + 1$	true

Figure 6.1: The fragment of FSA_{OG_n} corresponding to any given i , $0 \leq i \leq n$

and halt states of OG_1 . OG_1 becomes FSA_{OG_1} as shown in Fig. 6.3 (a). Halt states are drawn as pairs of concentric circles and the initial state is drawn in bold. Note that the relabelling of arcs has led to nondeterminism.

6.2 Removing Empty Cycles

The first step in determining a FSA using standard algorithms [4] is the removal of empty cycles. In our case, there are no empty cycles:

Proposition 1. FSA_{OG_n} , given by Table 6.1, contains no empty cycles.

Proof. There is only one cycle in our FSA and it is not an empty cycle. This can be verified if we evaluate the source and destination nodes of all arcs for all values $0 \leq i \leq n$. For any given value of i , the graphical structure given by Fig. 6.1 contains no cycles. The last node in the i^{th} segment becomes the first node in the $(i \oplus_n 1)^{\text{th}}$ segment. For all values of i up to $n - 1$, no arcs lead back to previously seen nodes. However, the last node in the n^{th} segment (i.e. when $i = n$) is node $(6(n \oplus_n 1) + 1) = 1$, the initial node. Therefore the only arc to return to an already seen node (and thus close a cycle) is the last arc in the n^{th} segment, thus there is only one cycle. This cycle contains non- ϵ primitives (one send and receive for each segment) and thus it is not an empty cycle. \square

Table 6.2: FSA_{OG_n} after removal of ϵ moves and deletion of inaccessible states, where rows 2 and 3 are evaluated for $0 \leq i \leq n$.

Source node	Arc Label	Dest. node	Dest. = Halt?
1	send	2	true
$6i + 2$	receive	$3(2i + 1) + 1$	true
$3(2i + 1) + 1$	send	$6(i \oplus_n 1) + 2$	true

6.3 Removing Empty Moves

The next step is removal of empty moves [4]. For an empty move from state p to state q the successors of q are added to the successors of p and the empty move is deleted. If q was a halt state, then p becomes a halt state. Once all empty moves are deleted, any inaccessible states (those not reachable from the initial node) are deleted.

Removing the empty moves to deadlocked nodes of type $t = 3$ (nodes $6i + 3$ and $3(2i + 1) + 3$) involves deleting the arcs leading to these nodes and flagging the source nodes of these arcs (nodes $6i + 2$ and $3(2i + 1) + 2$) as halt states. Note that this causes the nodes $6i + 3$ and $3(2i + 1) + 3$ to become inaccessible. Removal of the remaining empty moves between nodes $3(2i + 1) + 1$, $3(2i + 1) + 2$ and $6(i \oplus_n 1) + 1$ is done in a similar way, resulting in the nodes $3(2i + 1) + 1$, $3(2i + 1) + 2$ and $6(i \oplus_n 1) + 1$ (except for the initial state itself, $6(n \oplus_n 1) + 1$) becoming inaccessible from the initial state. Table 6.2 shows the result of removing all ϵ -moves and deleting all inaccessible states. Continuing with our example, the result of removing empty moves from FSA_{OG_1} is shown in Fig. 6.3 (b).

6.4 Determinisation and Minimisation

After the removal of empty moves, the FSA given by Table 6.2 is once again deterministic, which can be proved by inspection. All nodes accept exactly one service primitive and have exactly one successor. The FSA is, however, not minimal. This is readily evident from our example in Fig. 6.3 (b), which represents the language generated by the regular expression $(\text{send}, \text{receive})^*(\text{send} \mid \epsilon)$, but which could be represented by a FSA with fewer states.

To obtain a minimal representation in the general case [4] we begin by partitioning the states into two subsets, based on halt-state status. Both subsets are further divided if states within each subset are *distinguishable*. From [4], two states are distinguishable if they accept different input symbols or if they can be differentiated by some single input symbol in the following way. Suppose two states p and q from the same subset both accept symbol a , leading to states p' and q' respectively. If p' and q' are in different subsets, p and q are distinguishable and should also be placed in different subsets. To illustrate, Fig. 6.2 (a) shows two states that are distinguishable and Fig. 6.2 (b) shows two states that are not. The process of refining the partitioning of states continues until no more refinements can be made and all states within each subset are indistinguishable from each other but distinguishable from all other states in all other subsets. The minimum FSA is a new FSA constructed by selecting a representative from each subset of states and defining arcs corresponding to the actions between the subsets. Remember that either all states or no states in a subset are halt states. The representative from the subset containing the initial state is marked as the initial state of the minimal FSA.

From Table 6.2 it can be seen that all states are halt states, so we begin with all states placed in the same subset, i.e. $\{1, 6i + 2, 3(2i + 1) + 1 \mid 0 \leq i \leq n\}$. States are now divided based on the input symbols they accept, either **send** or **receive**, giving us the two subsets $\{1, 3(2i + 1) + 1 \mid 0 \leq i \leq n\}$ and $\{6i + 2 \mid 0 \leq i \leq n\}$. These subsets cannot be further refined, as all states in the first subset

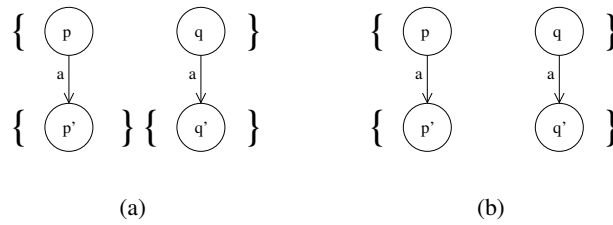


Figure 6.2: Refining the partition of states. (a) p and q are distinguishable on a , as p' and q' are in different subsets. (b) p and q are not distinguishable.

accept **send** leading to a state from the second subset, and vice versa for the states in the second subset accepting **receive**. We choose the representative ‘1’ to represent the first subset in the minimal FSA and the representative ‘2’ (when $i = 0$) to represent the second subset. Both are halt states and ‘1’ is the initial state. **send** and **receive** arcs are defined accordingly. The resulting minimised deterministic FSA is given in Table 6.3. This final step is illustrated by our example in Fig. 6.3 (c).

This minimised deterministic FSA represents the protocol language for all positive values of **MaxSeqNo**. This FSA is identical to the Stop-and-Wait property of $(\text{send}, \text{receive})^*(\text{send} \mid \epsilon)$.

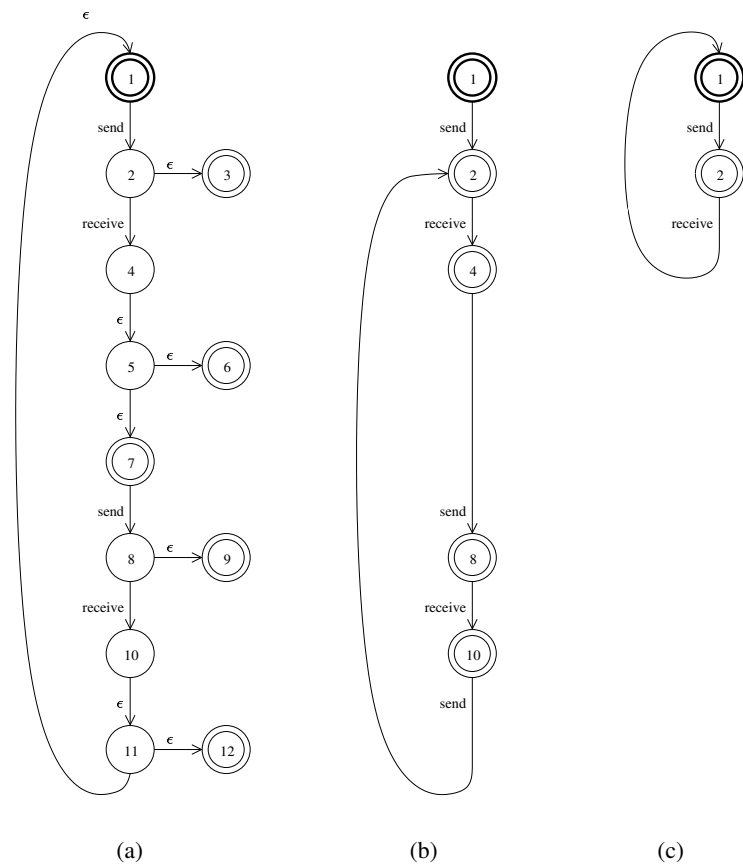


Figure 6.3: Illustration of ϵ removal and minimisation of FSA_{OG_1} .

Table 6.3: The minimised deterministic $FS A_{OG_n}$ representing the protocol language of CPN_n .

Source node	Arc Label	Dest. node	Dest. = Halt?
1	send	2	true
2	receive	1	true

This verifies that the SWP does indeed satisfy the Stop-and-Wait property for all positive values of MaxSeqNo , and thus Theorem 3 is proved.

Chapter 7

Conclusions and Future Work

We have represented an infinite number of OGs for our parameterised CPN, representing a class of Stop-and-Wait protocols, as a recursive and an explicit algebraic expression for the simplest case when there are no retransmissions. Additionally, we derived for the first time formulas for the size of the OG, as a product of a function of MaxRetrans and a function of $(\text{MaxSeqNo} + 1)$. Both the number of nodes and the number of arcs are linear in MaxSeqNo and quartic in MaxRetrans .

From the explicit algebraic expression we have obtained an expression for the protocol language which is identical to the Stop-and-Wait service. We believe this is the first time such a result has been obtained. This result is important as it eliminates successive reachability analyses for each value of the parameter and is a general result for all values. The next stage is to generalise this result for any value of MaxRetrans .

Our ultimate goal is to use these symbolic expressions representing an infinite set of occurrence graphs for more complete protocol verification. For example, we are able to detect deadlocks by inspection of Tables 5.3 and 5.4. The states superscripted by the type '3' never appear as source nodes, thus are deadlocked states representing that the message or the acknowledgement has been lost. Further, from Tables 5.1 and 5.2 we can see that there is a maximum of one item in the channel at a time (the message and acknowledgement channels are bounded by 1), a result that is expected, but is now verified for all values of MaxSeqNo from the expressions for the markings. We had proved in [10] that the protocol satisfies its Stop-and-Wait property of alternating sends and receives for a range of MaxSeqNo and MaxRetrans parameter values using automatic automata reduction techniques. This result is generalised in this report by proving it for arbitrary MaxSeqNo . This was done by using automata reduction techniques on the symbolic OG to obtain a simple automaton for the protocol language.

In the future, we would like to prove properties of the Stop-and-Wait Protocol for arbitrary values of both the MaxSeqNo and MaxRetrans parameters, over FIFO and reordering channels. Establishing an algebraic expression for the occurrence graph over both parameters and following the same procedure used here for the MaxSeqNo parameter seems promising and is the subject of current investigations. We would then like to use this result to verify such properties as expected terminal states, absence of livelock, boundedness and the Stop-and-Wait property for arbitrary parameter values.

Acknowledgement

The authors are grateful to their colleague, Lin Liu, for her inspiration for the proof of Theorem 2 and technical discussions regarding the minimisation procedure.

References

- [1] P. Abdulla, A. Annichini, and A. Bouajjani. Symbolic Verification of Lossy Channel Systems: Application to the Bounded Retransmission Protocol. In *Proceedings of TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 208–222. Springer-Verlag, 1999.
- [2] P. Aziz Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using Forward Reachability Analysis for Verification of Lossy Channel Systems. *Formal Methods in System Design*, 25(1):39–65, 2004.
- [3] S. Bardin, A. Finkel, and J. Leroux. FASTer Acceleration of Counter Automata in Practice. In *Proceedings of TACAS'2004*, volume 2988 of *Lecture Notes in Computer Science*, pages 576–590. Springer, 2004.
- [4] W.A. Barrett and J.D. Couch. *Compiler Construction: Theory and Practice*. Science Research Associates, 1979.
- [5] K.A. Bartlett, R.A. Scantlebury, and P.T. Wilkinson. A Note on Reliable Full-Duplex Transmission over Half-Duplex Links. *Communications of the ACM*, 12(5):260–261, May 1969.
- [6] J. Billington. Formal specification of protocols: Protocol Engineering. In *Encyclopedia of Microcomputers*, volume 7, pages 299–314. Marcel Dekker, New York, 1991.
- [7] J. Billington, M. Diaz, and G. Rozenberg, editors. *Application of Petri Nets to Communication Networks*, volume 1605 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.
- [8] J. Billington and G. E. Gallasch. How Stop and Wait Protocols Can Fail Over The Internet. In *Proceedings of FORTE'03*, volume 2767 of *Lecture Notes in Computer Science*, pages 209–223. Springer-Verlag, 2003. (invited paper).
- [9] J. Billington and G. E. Gallasch. An Investigation of the Properties of Stop-and-Wait Protocols over Channels which can Re-order messages. Technical Report CSEC-15, Computer Systems Engineering Centre Report Series, University of South Australia, May 2004.
- [10] J. Billington, G. E. Gallasch, and B. Han. A Coloured Petri Net Approach to Protocol Verification. In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 210–290. Springer-Verlag, 2004.
- [11] J. Billington, G.E. Gallasch, and L. Petrucci. Transforming Coloured Petri Nets to Counter Systems for Parametric Verification: A Stop-and-Wait Protocol Case Study. In *Proceedings of 2nd International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES 2005*. Turku Centre for Computer Science (TUCS), 2005. (to appear).
- [12] J. Billington, G.R. Wheeler, and M.C. Wilbur-Ham. PROTEAN: A High-level Petri Net Tool for the Specification and Verification of Communication Protocols. *IEEE Transactions on Software Engineering*, 14(3):301–316, March 1988.

- [13] CADP homepage.
<http://www.inrialpes.fr/vasy/cadp/>.
- [14] CCITT. ISDN user-network interface Data link layer specification. Technical report, Draft Recommendation Q.921, Working Party XI/6, Issue 7, Jan. 1984.
- [15] Design/CPN Online. <http://www.daimi.au.dk/designCPN/>.
- [16] M. Diaz. Modelling and Analysis of Communication and Co-operation Protocols Using Petri Net Based Models. In *Protocol Specification, Testing and Verification*, pages 465–510. North-Holland, 1982.
- [17] FAST - Fast Acceleration of Symbolic Transition systems.
<http://www.lsv.ens-cachan.fr/fast/>.
- [18] A. Finkel and J. Leroux. How to Compose Presburger-Accelerations: Applications to Broadcast Protocols. In *Proceedings of FST&TCS'2002*, volume 2556 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2002.
- [19] S. Gordon. *Verification of the WAP Transaction Layer using Coloured Petri Nets*. PhD thesis, School of Electrical and Information Engineering, University of South Australia, 2001.
- [20] ISO/IEC. *Software and Systems Engineering – High-level Petri Nets – Part 1: Concepts, Definitions and Graphical Notation*. ISO/IEC 15909-1, 1 December 2004.
- [21] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 1. Springer, second edition, 1997.
- [22] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts*. Springer-Verlag, 2nd edition, 1997.
- [23] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 2, Analysis Methods*. Springer-Verlag, 2nd edition, 1997.
- [24] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 3, Practical Use*. Springer-Verlag, 1997.
- [25] L.M. Kristensen, S. Christensen, and K. Jensen. The Practitioner’s Guide to Coloured Petri Nets. *International Journal on Software Tools for Technology Transfer*, 2(2):98–132, 1998.
- [26] L. Liu and J. Billington. Modelling and Analysis of the CES Protocol of H.245. In *Proc. of the Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 95–114, Aarhus, Denmark, 29-31 August 2001.
- [27] L. Liu and J. Billington. Tackling the Infinite State Space of a Multimedia Control Protocol Service Specification. In *Proceedings of ICATPN'02*, volume 2360 of *Lecture Notes in Computer Science*, pages 273–293. Springer-Verlag, 2002.
- [28] M. A. Marsan, A. Bianco, L. Ciminiera, R. Sisto, and A. Valenzano. A LOTOS Extension for the Performance Analysis of Distributed Systems. *IEEE Transactions on Networking*, 2(2):151–165, 1994.
- [29] R. Milner, R. Harper, and M. Tofte. *The Definition of Standard ML*. MIT Press, 1990.

- [30] C. Ouyang. *Formal Specification and Verification of the Internet Open Trading Protocol using Coloured Petri Nets*. PhD thesis, Computer Systems Engineering Centre, School of Electrical and Information Engineering, University of South Australia, Adelaide, Australia, June 2004.
- [31] J. Postel. Transmission Control Protocol. RFC 793, September 1981.
- [32] W. Reisig. *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets*. Springer-Verlag, 1998.
- [33] W. Stallings. *Data and Computer Communications*. Prentice Hall, 6th edition, 2000.
- [34] L.J. Steggles and P. Kosiuczenko. A Timed Rewriting Logic Semantics for SDL: a case study of the Alternating Bit Protocol. *Electronic Notes in Theoretical Computer Science*, 15, 1998.
- [35] I. Suzuki. Formal Analysis of the Alternating Bit Protocol by Temporal Petri Nets. *IEEE Transactions on Software Engineering*, 16(11):1273–1281, 1990.
- [36] A. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.
- [37] The TReX Tool. <http://www.liafa.jussieu.fr/~sighirea/trex/>.
- [38] K. J. Turner (Ed.). *Using Formal Description Techniques: An Introduction to Estelle, Lotos and SDL*. Wiley Series in Communication and Distributed Systems. John Wiley & Sons, 1993.
- [39] A. Valmari. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer-Verlag, 1998.
- [40] A. Valmari and I. Kokkarinen. Unbounded Verification Results by Finite-State Compositional Techniques: 10^{any} States and Beyond. In *Proceedings of International Conference on Application of Concurrency to System Design*, pages 75–85. IEEE Computer Society, March 1998.
- [41] M. E. Villapol. *Modelling and Analysis of the Resource Reservation Protocol*. PhD thesis, Electrical and Information Engineering, University of South Australia, Australia, November 2003.