Probabilistic Communication Structured Acyclic Nets

Nadiyah Almutairi

Supervisor: Prof. Maciej Koutny

School of Computing
Newcastle University

This dissertation is submitted for the degree of
Doctor of Philosophy

Monday 1st April, 2024

I would like to dedicate this thesis to My World:
loving Mom
Beloved Husband and Daughters
Sisters and Brothers
and my Supervisor.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Nadiyah Almutairi
Monday 1st April, 2024

Acknowledgements

First and foremost, praise to Allah the Almighty for the blessings that have been given to me. His support has armed me with patience and determination throughout the challenging moments of completing this thesis.

Without intensive discussions, this work would never have been achieved. Firstly, I would like to acknowledge my sincere gratitude to my supervisor Prof. MACIEJ KOUTNY for his guidance, expertise, and unlimited support during my study. His constructive feedback, suggestions, and encouragement have been influential in shaping this thesis. I keep remembering your eagerness and attention to making weekly meetings that have helped me work on my research smoothly. Thank you so much for your continuous exhortation and advice, which have improved me professionally and personally.

Also, I would like to thank our SON group research members. In particular, I am grateful to Prof. Brian Randell for his valuable discussions. Moreover, special thanks to Dr. Anirban Bhattacharyya for his great help and excellent comments. In addition, I would like to thank Dr. Bowen Li for his help. Massive thanks go to Dr. Talal Alharbi for his support and fruitful discussions. Also, I cannot forget to thank Mohammed Alahmadi, Salma Alharbi, and Tuwailaa Alshammari for their abundance of help and support which have inspired and motivated me throughout my research. Their encouragement was not only related to the research, but also in my daily life, which builds up the strong foundation of our friendship.

To all my colleagues in the AMBER group at the School of Computing at Newcastle University, thanks for your friendly support and our amusing times together during my PhD. In particular, I would like to thank Dr. Marta Koutny for her kindness, support, and advice, especially during the 43–44 international conferences of Petri Nets. Tremendous thanks go to Dr. Aisha Ahmed, Dr. Hanadi Alkhudhayr and Maram Alshahrani for their encouragement and strong support.

Many thanks to my best friends who have always been a source of happiness and for their support during all the difficult times: Dr. Salha Alqahtani, Mona Almutairi, and Wesaif Alrewali.

I would like to thank my examiners, Prof. Hanna Klaudel and Dr. Jason Steggles for their expertise and suggestions for improving my thesis.

I extend my heartfelt thanks to my family who have played a significant role in completing this thesis. Great thanks go to my Mom, sisters, and brother for their support. More importantly, I would like to express my deepest gratitude to my love, my heroic husband, Adel, who always saves me in times of need, especially during this research. I would like to thank the fuel for my success, my two daughters, Norah and Laiyla.

Finally, I would like to express my gratitude to the University of Hafr Albatin and the Saudi Arabia Cultural Bureau for their financial support. This thesis would never have become a reality without their financial assistance.

In conclusion, I can go on and on about thanking the people who have helped me over the years of completing this thesis and I am truly grateful for your presence in my academic journey.

Abstract

As the real world is full of uncertainty, we often estimate, or even guess to quantify uncertainty. Probabilistic models are the key to cope with uncertainty. One of the most prominent concurrent probabilistic models are *probabilistic Petri nets*. Petri nets are one of the mathematical modeling languages for the representation of distributed systems. They have been characterised as one of the expressive models to capture the notion of concurrency.

Developing probabilistic concurrent system has been proved to be a difficult and ongoing problem. This is because in a concurrent systems different executions of a concurrent computation should have the same probability. However, this is not always guaranteed as concurrent systems may exhibit *confusion*.

Confusion is an overlapping between conflict and concurrency – two fundamental concepts used in the area of concurrent systems modelling – which interferes with probability analysis. In this thesis, we set out to develop a probabilistic framework and outline approaches leading to a model where distributed choices are resolved in a way which allows one to carry out probabilistic estimation. In particular, the concept of *cluster-acyclic net* is introduced to transform a net with confusion into another net whose structure is free-choice which facilitates probabilistic estimation.

Moreover, we formally extend calculating probabilities and the definition of conflict and confusion to Communication Structured Acyclic Nets (CSA-nets). Intuitively, in CSA-nets, acyclic nets are integrated into one structure that allows them to interact by the means of *asynchronous* and *synchronous* communications.

CSA-nets are sets of interacting acyclic nets derived from Structured Occurrence Nets (SO-nets), which are a Petri net based formalism for representing the behaviour of complex evolving systems.

We show that a CSA-net with confusion can be translated into another, confusion-free net, whose behaviour is closely linked to the behaviour of the original CSA-net. Also, a boolean satisfiability model is introduced to formally verify behavioural properties of CSA-nets.

Table of contents

List of figures

Chapter 1

Introduction

1.1 Background

In everyday life, the presence of uncertainty can never be eliminated completely. Predicting tomorrow's weather, where we left our keys, or even the outcomes of a simple task of flipping a coin cannot be predicted with certainty. Generally, this type of unpredictability is a result of our ignorance and limited knowledge. Usually, we find ourselves obligated to guess or estimate our beliefs. As the real world is immensely complicated, mathematical models are used to cope with the real world complexity. In particular, *probability theory* plays a key role in coping with uncertainty. In fact, the probability theory is the science of uncertainty. It is based on statistical details that estimates the likelihood of an event occurrence. Hence, probability models are frameworks that mitigate our limited knowledge in the sense that we are capable of making reasonable decisions or predictions.

There are ubiquitous uses of probability models in real life applications. For example, in crime or accident investigation systems – which are examples of Complex Evolving Systems (CES) – a crime or accident is portrayed by numerous scenarios due to the lack of enough information concerning the crime or accident. This is because a full information about a specific activity is, in general, not available [106]. Hence (often numerous) alternate scenarios are pursued by investigators in order to clarify the status of a crime or accident. Such systems are characterised as being inherently non-deterministic which results from the lack of the ability to observe all the events and their causes. To cope with this difficulty, investigators need to consider all the possible scenarios. However, considering all possible scenarios might result in meaningless conclusion. Thus, reasoning about what is probable along with what is possible is essential to obtain meaningful conclusion [77]. Equipping such system with probability estimations can enhance the investigators decisions regarding

crimes or accidents. In this case, investigators can narrow down their hypothesis and reason about what was most likely to occur. As a result, they are able to prioritise the investigation of alternative scenarios. Therefore, providing effective probabilistic estimation of different scenarios would greatly help the investigators to find answers to crucial questions concerning the incident. This would enable one to interpret relative likelihood of events involved and then combine them into a meaningful information, which would increase the degree of certainty of crime or accident representation.

Not limited to analysing crime or accident investigations systems, there is a rapidly growing interest in probabilistic models for a variety of applications, such as machine learning and medical researches. In this thesis we are interested in *concurrent probabilistic models*.

Developing concurrent probabilistic systems has bee a long standing challenge. One of the most prominent concurrent probabilistic models are *probabilistic Petri nets*. Probabilistic Petri nets were introduced in [26, 69, 3, 128, 68, 19, 87, 9, 4]. In probabilistic Petri nets, the ordering of concurrent events is irrelevant and behaviours differ only in such orderings should be assigned the same probability. Also, the conflict between the transitions is resolved probabilistically, and there is a probability distribution over a *cell/cluster*, which is a set of transitions that are in direct conflict. Then, the probability of a step sequence is the product of probabilities of its constituent transitions. If all the conflict transitions have the same set of causal predecessors, then they are always enabled together. Hence, we can say that such a *cell/cluster* is either enabled or not enabled. However, conflict transitions do not always have identical causal predecessors.

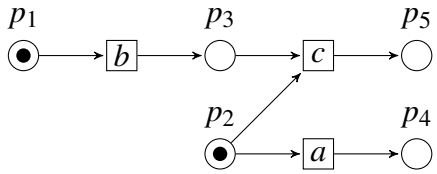


Fig. 1.1 An acyclic Petri net.

Figure 1.1 shows an acylic net with three transitions: a, b, c. Note that a and c are in direct conflict, but they have different input places. Resolving the conflict between a and c depends on firing b first (before firing a). In this case, the probability of choosing a or choosing c is 0.5. However, if a is fired before b, then the probability of a is 1 and so it is different than in the first case. Hence, it is not obvious what the probability of a should be. That is due to the fact that a and c have different causal predecessors. The situation where

the transitions have different probabilities based on the ordering in which they are executed is called *confusion*.

Confusion arises when conflict and concurrency overlap. Assigning probabilities to transitions in the presence of confusion is problematic and interferes with probability analysis. That is because transitions are not probabilistically independent despite the fact that they are concurrent. For instance, even though a and b in Figure 1.1 above are concurrent, firing b has influence on the probability of a. Remedying confusion has generated great interest over the past years. Several studies explored the effect of confusion and proposed approaches to remove it [26, 1, 69, 3, 68, 19, 4, 31, 32].

In this thesis, we develop a theoretical concurrent probabilistic framework based on Communication Structured Acyclic Nets (CSA-nets) and Behavioural Abstraction Communication Structured Acyclic Nets (BSA-nets). The concept of CSA-nets is derived from Structured Occurrence Nets (SO-nets), which were first introduced in [105] and elaborated in [78]. CSA-nets consists of multiple acyclic nets that are associated with varying kinds of formal relationships, with the objective of recording information about the actual or expected behavior of CESs. [84] characterises a CESs as a system being composed of wide array of (sub)systems that interact with each other and with its surrounding environment, resulting in a highly complex structure. The expressive structure of CSA-nets is suitable to represent the activities of such systems since the cognitive complexity can be reduced and managed by choosing the adequate notation for describing the behavior of complex evolving systems. [78] introduced the basic formalisation of SO-nets to represent complex fault-error-failure chains. Also, [84] showed that SO-nets can be used to visualise and analyse behaviour of CESs, and [11] demonstrated SO-nets capabilities for modelling cybercrime investigation. Moreover, the previous work on SO-nets provided a framework for provenance [92], timed behaviours [23, 10], and events extraction [15, 16]. A SAT-based model checking for CSA-nets was introduced in [14], and improving the visualisation aspects was discussed in [5–7]. Equipping CSA-nets with probabilities was introduced in [12, 13].

BSA-nets can be defined as a way of capturing the evolution of a set of related acyclic nets. They provide a mechanism to abstract parts of a complex activity by another system. More precisely, the behaviour can be embodied at two levels of abstraction, namely the upper-level and lower-level. The upper-level provides a simple view and hides unimportant details of the behaviour. The lower-level, on the other hand, shows the full details of behaviour during different evolution stages.

This thesis will address theoretical aspect of probabilistic CSA-nets and probabilistic BSA-nets. The main focus of the thesis is to propose approaches for handling confusion.

Also, we explore how this probabilistic analysis can be facilitated by a SAT-based model checking verification of behavioural properties of CSA-nets and BSA-nets.

1.2 Aim and Objectives of the Thesis

Aim:

To develop a formal concurrent probabilistic framework for CSA-nets and their behavioural abstractions and provide techniques to handle the case of *confusion*.

Objectives:

1. **Surveying the existing probabilistic models**

   Different mathematical probability models have been introduced over the past decades. We review a number of the existing probability models. These probability models serve different purposes. For instance, the formalism of automata based models provide probability analysis for sequential behaviour only. Also, the next state probability is based on the present state only excluding the history of the process. Providing probability analysis for complex evolving systems cannot be achieved with these restrictions. Concurrent probabilistic models, in particular token-based models such as Perti nets, are compatible with the features of such systems. Reviewing several probability models provides us the basic concepts for understanding their features and how they can be used in practical applications.

2. **Surveying the existing approaches for removing confusion in Petri nets**

   There exists an extensive literature on the topic of confusion and approaches for handling it. The aim of reviewing a range of the existing techniques for removing confusion is to gain understanding of its effects and propose acceptable approaches of removing it.

3. **Developing a theoretical probabilistic framework for the analysis of CSA-nets**

   To develop principles for probabilistic analysis of composed acyclic nets captured by CSA-nets, we will develop (in Chapter 3) the basic theory of probabilistic analysis for a single acyclic net. In particular, a formula calculating the probability of a step sequence so that conflicts between transitions are resolved probabilistically will be defined. Confusion is defined, with some illustrative examples, as a situation where

the probabilistic analysis is not feasible. In Chapter 4, we will extend the probabilities calculation and the definition of confusion to the level of interacting acyclic nets (CSA-nets).

4. **Developing different approaches for removing confusion in CSA-nets**

   Identifying the cases where confusion arises are needed in order to propose techniques for remedying it. This leads to defining a novel class of acyclic nets, namely *cluster-acyclic nets*. Chapter 3 introduces two approaches of removing confusion for acyclic nets in this class. Extending the definition of confusion and the approaches of removing it to be applicable to sets of interacting acyclic nets is the core contribution of Chapter 4. Moreover, *cascading* CSA-nets are defined as a new class of CSA-nets, and a novel technique of handling confusion for this class of CSA-nets is proposed.

5. **Developing a theoretical probabilistic framework for the analysis of BSA-nets and proposing an approach for handling confusion**

   In Chapter 5, we extend our probabilistic framework so that the probability analysis is applicable for two-level BSA-nets. In particular, the probabilities calculation that is reflected at the upper-level acyclic nets is based on the details provided at the lower-level acyclic nets. In this case, the abstracted view captured by upper-level acyclic nets is seen not only as a means of simplifying the representation, but also as a tool of analysing the lower-level behaviour. Moreover, an initial approach is proposed to handle the confusion through utilizing the behavioural abstraction relation.

6. **Developing a SAT-based verification method for CSA-nets**

   To facilitate the development of our probabilistic framework in practical applications, we investigate verifying CSA-nets and BSA-nets properties using SAT-based model checking. In particular, we focus on developing formulas for detecting conflict and confusion. Also, formulas are introduced to ensure that a set of transitions represents a valid behaviour.

1.3 Contributions

Our main contributions developed to satisfy the aim and objectives are as follows:

1. **A survey of the existing approaches of removing confusion in probabilistic Petri nets**

Probabilistic Petri nets has been of interest for a considerable period. Consequently, there is also a large body of work in this area considering handling confusion. To meet *Objective 2*, we would like to review a collection of studies that proposed techniques for removing confusion. For example, the work introduced by [1] and [2] provided an approach of handling confusion by means of *branching cells* in such way that a net is decomposed dynamically. The most recent theory proposed by [26] offers a new approach for removing confusion by static decomposition of branching cells. Additional investigation about other approaches for removing confusion are explored in Chapter 2.

2. **A theoretical probabilistic framework for the analysis of cluster-acyclic nets and CSA-nets, and different approaches for removing confusion**

   Cluster-acyclic nets are defined as a new class of acyclic nets, where conflict transitions are grouped into a *cluster*. The acyclicity over all the clusters is an important restriction, and the clusters are probabilistically independent. Probabilities are calculated based on the weights assigned to the transitions in conflict. Overlapping conflict and concurrency causes confusion which means that the probabilities are not calculated accurately. Two approaches are proposed to remedy this situation for cluster-acyclic nets.

   We then extend the definition of confusion and techniques for removing it to sets of interacting acyclic nets represented by CSA-nets. It turns out that the acyclicity constraint can be lifted to the level of CSA-nets. We also define *cascading* CSA-nets, and propose a simple approach for removing confusion from cascading CSA-nets. These outcomes are essential to meet *Objective 4*.

3. **A theoretical probabilistic framework for the analysis of BSA-nets, and an approach for handling confusion in BSA-nets**

   To satisfy Objective 5, we first extend the model of BSA-nets discussed in [78, 106, 105, 84, 11, 92]. In particular, the upper-level nets are defined as *free-choice* acyclic nets instead of being *line-like* in order to represent alternatives in a given lower-level behaviour. Being able to represent alternative behaviours motivated us to extend the probabilistic framework so that probabilities are represented at the upper-level nets. This is a novel representation of probabilities in the area of net theory. The abstracted view captured by upper-level nets can be seen as a tool for not only providing a simpler representation but also contributing in the analysis. In this case, the probability of a upper-level transition is derived from the weights associated with the lower-

level transitions that are ascribed to it. Moreover, we propose a preliminary idea of controlling confusion in BSA-nets. More precisely, exploiting the structural constraints of upper-level nets, behavioural relation is used to filter out undesirable representation of a given behaviour.

4. **A SAT-based verification for CSA-nets and BSA-nets**

   We investigate the development of a SAT-based model checking technque for our framework. Regarding *Objective 6*, we formalise important behavioural specifications of CSA-nets and BSA-nets as satisfiability formulas. For instance, we show how the presence of confusion can be detected using a suitable SAT formula. Checking of other crucial behavioural properties is discussed in Chapter 6.

1.4 Thesis Outline

The thesis is organised as follows:
Chapter 2 provides an overview of the background details and related work of probabilistic Petri nets.
Chapter 3 presents the notions and properties concerning acyclic nets. Also, it discusses the notion of *confusion* and the ways it can affect the probabilistic analysis of acyclic nets. Moreover, different approaches of removing confusion are presented.
Chapter 4 discusses CSA-nets and their properties. In particular, the definition of confusion is extended to CSA-nets as well as the methods of removing it.
Chapter 5 discusses BSA-nets and their properties.
Chapter 6 introduces SAT-based model checking for CSA-nets and BSA-nets.
Chapter 7 summarises and concludes the work, and proposes directions for further work.

1.5 List of Publications

Parts of this thesis have been documented in the following publications:

1. Almutairi, N. and Koutny, M. (2021). Verification of communication structured acyclic nets using SAT. In PNSE@Petri Nets, volume 2907 of CEUR Workshop Proceedings, pages 175–194. CEUR-WS.org.

2. Almutairi, N. (2022). Probabilistic communication structured acyclic nets. In PNSE@Petri Nets, volume 3170 of CEUR Workshop Proceedings, pages 168–187. CEUR-WS.org.

3. Almutairi, N. (2023). Probabilistic Behavioural Acyclic Nets. In PNSE@Petri Nets, volume 3430 of CEUR Workshop Proceedings. CEUR-WS.org.

4. Alahmadi, M., Alharbi, S., Alharbi, T., Almutairi, N., Alshammari, T., Bhattacharyya,A., Koutny, M., Li, B. and Randell, B. (2024) Structured Acyclic Nets. arXiv preprint arXiv:2401.07308

Chapter 2

Background

This chapter reviews the main graph-based formalisms used to model probabilistic computing systems. First, we give an introduction of the automata-based models. These automata-based models include: *Markov Automata* (MA), *Probabilistic Automata* (PA), and *Interactive Markov Chains (*IMC*s)*. After that, *Bayesian Networks* (BNs) and *Factor Graphs* (FGs) are presented as other classes of probabilistic graphs.

We then focus on token-based models of Petri nets, where probabilities are associated with conflicting transitions that are related to our work, specifically considering approaches for handling *confusion*.

2.1 Probabilistic automata-based models

2.1.1 Probabilistic Automata

The seminal work by Rabin [101] provided the initial proposal of Probabilistic Automata (PA). The initial idea was re-formulated by Segala and Lynch [117, 118, 116]. The model is a mathematical framework for probabilistic analysis and it can be seen as a generalisation of non-deterministic finite state automata with probabilistic distributions. A PA consists of states represented by circles which are connected by arcs that represent transitions. Choices can be resolved probabilistically or non-deterministically. That is, a transition from a state s with an action a occurs with the probability distribution over several next states. Also, a transition can be taken non-deterministically when there are several transitions with the same action but with different distributions [121]. In general, PA are an extension of the *labelled transitions systems* such that transitions are non-deterministic, but governed by probability distribution (the labelled transitions systems lack similar quantitative analysis). PA have

been used extensively in both theory and practice-oriented research [43, 68, 19, 42]. In particular, PA have been used for modelling asynchronous systems with discrete probabilistic analysis. Such systems include distributed algorithms, fault tolerant systems, and probabilistic communication protocols. The probabilities that can be associated with these systems are related to correctness and performance evaluation [121].

Example 1. A probabilistic automaton representing a vending machine (taken from [19]) is shown in Figure 2.1. The initial state is s_0 which is also the final state. The probability of accepting an inserted coin is 0.9 and in this case a transition from s_0 to s_1 occurs. On the other hand, the probability of rejecting the coin is 0.1 where the system does not change its state s_0. In s_1 there are two probabilistic transitions captured by choosing *coke* and moving to s_2 with probability 0.5 or choosing *chock* and moving to s_3 with probability 0.5. Then with a deterministic transition the system returns to its initial state. ◇

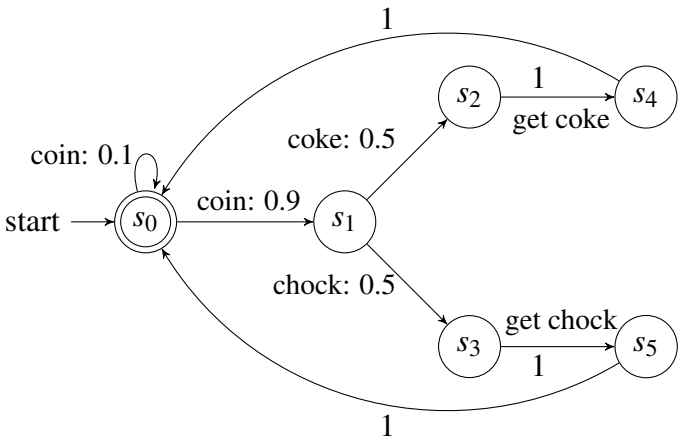


Fig. 2.1 Probabilistic automaton of a vending machine.

2.1.2 Interactive Markov Chains

Interactive Markov Chains (IMCs) are also one of the automata-based models. They distinguish between non-deterministic choices, which are captured by *interactive transitions*, and exponentially distributed delays captured by *Markovian transitions*. An interactive transition is represented by $s \xrightarrow{\alpha} s'$ where the action α is executed in zero time with non-deterministic choices, denoted as (s, α, s'). The set of interactive transitions is $IT(s) = \{(s, \alpha, s') \in IT\}$. A Markovian transition denoted by (s, λ, s'), written as $s - - \xrightarrow{\lambda} s'$, which means that the system moves from state s to state s' after a delay exponentially distributed with rate λ

[61]. The set of Markovian transitions is $MT(s) = \{(s, \lambda, s') \in MT\}$. In a state where both kinds of transitions are enabled, interactive transitions are taken immediately. The outgoing transitions identify the type of the states. For instance, we say that a state s is an interactive state if there is $s \xrightarrow{\alpha} s'$, and s is a Markovian state if there is $s - - \xrightarrow{\lambda} s'$. For Markovian states, $\mathbf{R}(s,s') = \sum\{\lambda \mid (s,\lambda,s') \in MT(s)\}$ is the *rate* for sate s to move to state s'. Also, $E(s) = \sum_{s' \in S} \mathbf{R}(s,s')$ is the *exit rate* of state s. Then the probability to move from state s to s' is given by $\mathbf{P}(s,s') = \frac{\mathbf{R}(s,s')}{E(s)}$ [124].

Examples of using IMCs include dynamic fault trees [29] and architectural description language [25].

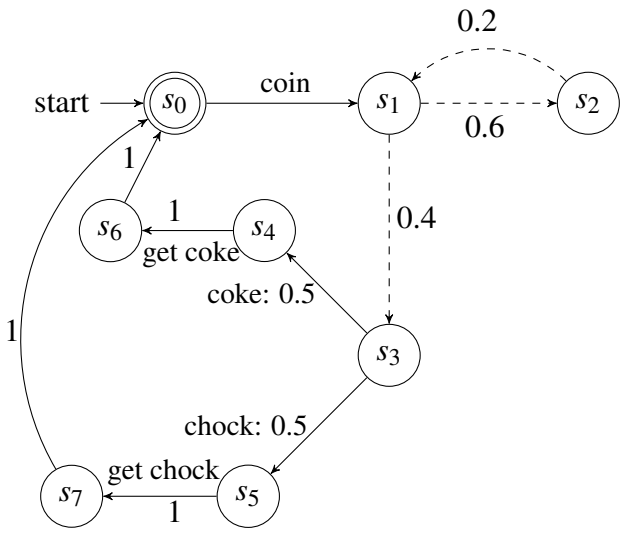


Fig. 2.2 Interactive Markov Chain of vending machine (from [19]).

Example 2. A vending machine modelled by IMC is presented in Figure 2.2 (taken from [19]). After a coin is inserted, the machine enters state s_1 which is a *Markovian state* as all its outgoing transitions are Markovian transitions. Then, states s_2 and s_3 are racing each other as they are both available at s_1, the one whose rate expires first is taken. s_2 is an error state which represents the situation when the machine is not working. The probability of moving from state s_1 to state s_2 is $\mathbf{P}(s_1,s_2) = 0.6$ and the probability of moving to state s_3 is $\mathbf{P}(s_1,s_3) = 0.4$. As the delay of the transition to s_3 expires first, s_3 is chosen where either *coke* or *chock* is taken non-deterministically. ◇

2.1.3 Markov Automata

Markov Automata (MA) is one of the general automata-based models. A powerful aspect of MA is that they can describe systems in terms of non-deterministic, probabilistic, and timed behaviour [40]. The semantics of MA is captured by actions or events which are responsible of moving a system from one state to another. Transferring to the next state is governed by a probabilistic distribution and exponentially distributed delays. In other words, transitions in MA are classified into *immediate probabilistic transitions* and *Markov timed transitions* [52]. Hence, MA is a combination of *Probabilistic Automata (*PA*)* and *Interactive Markov Chains (*IMC*s)*. That means MA are similar to PA in terms of resolving the choices probabilistically for probabilistic transition, and choices are resolved stochastically for the Markovian transitions as in IMC [19].

MAs are expressive and suitable to model systems in both theoretical and practical research [40, 42, 52, 56, 57, 124, 123, 43, 19]. For instance, in [55], MA is used to establish quantitative analysis for cost and risk related to Bitcoin, as well as to evaluate performance of resource-sharing systems.

Example 3. Figure 2.3 shows the same vending machine in Figure 2.1 as a Markov automaton. After the inserted coin is accepted with probability 0.9, two transitions are enabled at s_1 as in Example 2. States s_1 and s_2 are *Markovian states*. If the error state s_2 is chosen, the machine can be in state s_1 again after some time represented by rate 0.2. This shows how this MA is different form the PA in Figure 2.1 as not only probabilistic behaviour is captured, but also the stochastic one. ◇

2.2 Bayesian Nets

The theory of Bayesian Nets (BNs) is based on the work introduced by Bayes in 1764 [96]. The term "Bayesian network" was coined by Judea Pearl in 1985 [97]. Bayesian nets use an underlying graphical structure and the probability to provide a theoretical foundation to handle uncertainty. Basically, BNs are directed acyclic graphs representing a set of variables and their conditional relationships. More precisely, they consist of nodes that represent variables and direct edges that capture causal dependencies. BNs are used to capture the possibility that one of a range of potential known causes contributed to an event that already happened. For instance, BNs could depict the probability connections between diseases and symptoms. In other words, BNs can be used to calculate the likelihood that a certain set of diseases will be present given a set of symptoms.

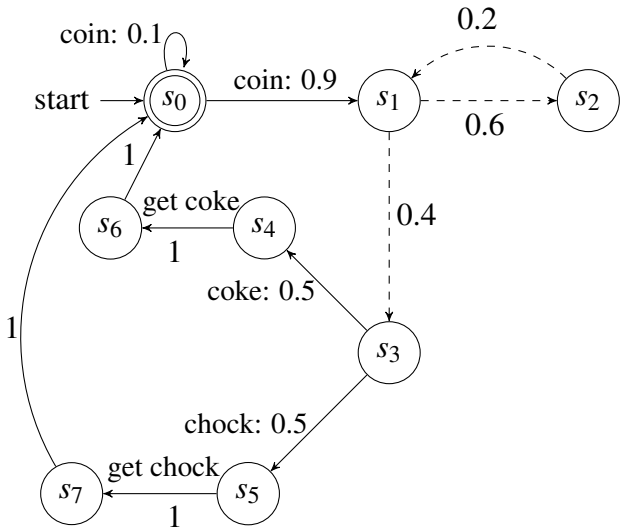


Fig. 2.3 Markov automaton of a vending machine (from [19]).

The probability distribution in BN of an event is conditional on the set of its parents for each possible outcome. Intuitively, if $Q = \{x_1, \ldots, x_n\}$ is the set of variables represented by the nodes in BN and Pa_i denotes the parents for each node x_i, then the joint probability is $\mathbf{P}(Q) = \prod_{i=1}^{n} \mathbf{P}(x_i \mid Pa_i)$ [120], which is the probability of x_i given the fact that its parents Pa_i are observed. Hence, one can say that in BN the probability is based on the background knowledge.

There exists a large body of works on BNs, both theory-based and practical ones [60, 96, 94, 65, 63].

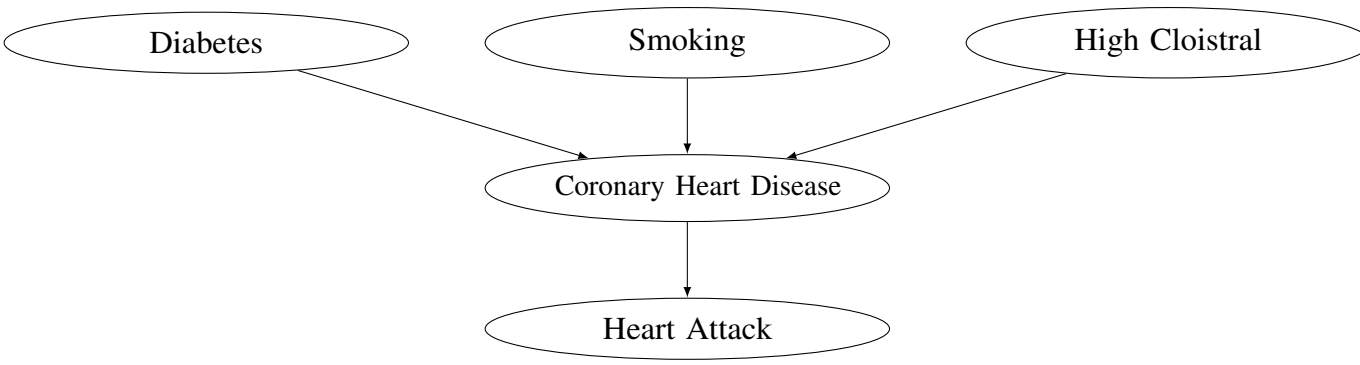


Fig. 2.4 Bayesian network representing the likelihood of a heart attack.

Example 4. Figure 2.4 shows an example of Bayesian network representing the causes of *heart attack*. The causal dependencies of *Coronary Heart Disease* are *Smoking*, *Diabetes*, and

High cloistral. Hence, the probability distribution of coronary heart disease is conditionally on all its parents. Similarly, the probability distribution of *Heart Attack* is based on observing *Coronary Heart Disease*. Hence, the joint probability distribution of this Bayesian network represents is

$$\mathbf{P}(D,S,H,CHD,A) = \mathbf{P}(D) \cdot \mathbf{P}(S) \cdot \mathbf{P}(H) \cdot \mathbf{P}(CHD \mid D,S,H) \cdot \mathbf{P}(A \mid CHD)$$

where D stands for *Diabetes*, S for *Smoking*, H for *High Cloistral*, CHD for *Coronary Heart Disease*, and A for *Heart Attack* (note that the conditional probability tables for each node is not presented). ◇

2.3 Factor Graphs

A factor graph (FG) is an undirected bipartite graph that shows how a function is factorised. In fact, factor graphs are a class of probabilistic graphical models that are used to depict the factorization of probability distributions. The main components of an FG are *variables* represented by circles and *factors* represented by boxes, which represent the relationship and probabilistic information of the graph variables. The independence relation between the variables is visualised as an edge added between a variable and a factor if the variable appears in the factor. In more details, each factor is associated with a factor function that explains the relation between the various variables it might be attached to. Each factor function has a weight that indicates the factor's influence on its variables. Basically, the weight of the factor function represents our confidence in the correlations reflected by the factor function [88]. The overall function of all the variables is the product of all independent factor functions. Hence, FG represents joint probability distribution.

As BNs are suitable for modelling, FGs are recommended for performing inference [39]. For example, FGs are used effectively in simultaneous localisation and mapping (SLAM) as one of the central large scale inference problems in robotics [39] due to the fact that in FGs one can utilize certain factors to obtain fast inference. FGs have become increasingly popular in recent years as a tool for modelling complex systems and inference in robotics [115, 88, 80, 39, 38].

Example 5. A FG is shown in Figure 2.5. It has three variables x_1, x_2, x_3 and four factors f_1, f_2, f_3, f_4. The joint probability distribution is the product of the factors over the subset of

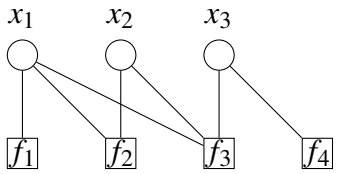


Fig. 2.5 A Factor graph

variables. Hence, the function $g(x_1,x_2,x_3)$ factorizes as

$$g(x_1,x_2,x_3) = f_1(x_1)f_2(x_1,x_2)f_3(x_1,x_2,x_3)f_4(x_3).$$

◇

All the previous probabilistic models including automata-based formalisms, Bayesian nets, and factor graphs represent the sequential behaviour of systems. In particular, interleaving semantics only, whereas concurrency is not captured. In addition, they rely on the current states only to find the probability distribution of the next states. This property is known as *memoryless* which means that the probability of occurring of some future events is not influenced by the occurrence of some past events.

2.4 Related Work

As this thesis is focused on formalism and developing a probabilistic framework to facilitate the analysis of acyclic nets and CSA-nets, we review in this section a collection of related studies considering resolving conflicts probabilistically, and specifically handling confusion. This is widely reported and extensively explored in the literature.

2.4.1 Probabilistic Petri nets

An increased interest in merging quantitative analysis and Petri nets has emerged in recent years. Providing a probabilistic framework for a concurrent model like Petri nets relies on the presence of conflict transitions. Then the positive numerical weights are either assigned to those conflict transitions or to the arcs in such way that the conflict is resolved by taking the weights into consideration. This seems trivial when the structure of nets is restricted to *free-choice*, a structure that ensures resolving the conflict locally. [9] is an example of practical application. Probabilities and free-choice Petri nets are used as a basic model

to represent human activity based on bank surveillance videos. The local probabilistic distribution is defined over the output arcs for each place. Then the probabilistic model is used to reason about two queries. First, finding the minimal clips of video that captured a specific event with a probability larger than a given threshold, that was known as Threshold Activity Queries. Second, what is the most likely activity to occur in a given video, which is Activity Recognition Queries. Also, reasoning about cyber-physical attack was investigated using coloured probabilistic Petri nets in [87]. The aim was to provide probabilistic analysis to the threat propagation among and between the cyber and physical components which then can support practitioners to predict possible attacks and provide defence techniques. In [133] the fuzzy-timing Petri nets with probabilistic choices were developed to provide probabilistic analysis for the response time of web service systems. The authors extended the time Petri nets so that not only time delay is considered, but also the probability of choosing the event 'read' over the event 'write' in the read-write systems.

Establishing a general probabilistic framework requires taking a special case into consideration. A *confusion* arises when conflict and concurrency are overlapped which causes problematic probabilistic estimation. In [128], the main result was to show that in *confusion-free* probabilistic net system the runs are Mazurkiewicz equivalent. The choices are resolved probabilistically as the transitions in conflict are associated with weights. Basically, the authors extended Mazurkiewicz equivalence to probabilistic words, which were defined as probability distribution, to describe the probabilistic net systems. Also, [21] investigated the approach of equipping Petri nets with probabilities by proposing what is called *Markov Nets* in such way that interacting components that compose Markov net are probabilistically independent. The proposed model was motivated through observation and fault diagnosis of distributed systems, specifically telecommunication networks. However, *free-choice* nets were only considered which is quite limiting in practice. The probabilistic framework presented in [54] was introduced to overcome the limitations of [21]. Similarly, the motivation was to provide probabilistic model for fault diagnosis in telecommunication networks. Essentially, finding the most likely causes of faults was the core of the diagnosis task. The *cluster net* was defined so that the net is partitioned into nodes that are closed under conflict and then the clusters are scheduled in which they make their local choices. The unfolding was obtained based on the cluster net considering the steps, instead of single transitions, enabled at each cluster. More precisely, the branching processes of the unfolding were obtained based on the cluster processes. However, the construction is quite complicated as some places and transitions were unnecessarily added. For example, the empty steps were generated in the construction derived from each cluster. More importantly, the non-determinism between the

clusters leads to different probabilities of the same run. In terms of probabilities calculation, no details were given of how the probabilities would be calculated after adding the empty step. The attempt in [53] of investigating the confusion in the unfolding semantics by introducing the clusters and the prosaic development in [119] to analyse it did not actually remedy the confusion. In [89] and [24] only the formalisation and the algorithms of detecting confusion in both safe and unsafe nets were investigated. However, no approaches were proposed to handle confusion. The exclusion of handling the confusion is a critical constraint on all of the work discussed above. Next, we provide an overview of the approaches of removing confusion related to our probabilistic theoretical framework.

2.4.2 Approaches of handling confusion

Localising decisions made as the executions proceed is a key challenge in creating true-concurrency probabilistic models [4]. There exists an extensive literature on this topic. For example, we can find the following contributions. The work introduced by [1] and [2] studied the construction of probabilistic event structure arising from the unfolding of a finite safe Petri net. Their construction is based on dynamic decomposition of configurations by the means of *branching cells*. The confusion was handled as the branching cells decompose the *maximal* configurations recursively and dynamically. Hence, the available choices in each branching cell should be maximal (the maximal configurations are seen as stack of choices inside each branching cell). More precisely, their definition of dynamic branching cells allows: (i) associating a local transition probability with each branching cell, (ii) using the chain rule to combine the local probabilities such that concurrent choices are taken independently and the probabilities are given to the maximal configurations. Due to the confusion, the branching cells may overlap, which means that some events may appear in different branching cells and hence they are defined dynamically. However, this can increase the complexity as the number of branching cells can be exponential. The authors in [4] extended the results in [1, 3, 2] so that the infinite event structures with confusion are investigated. In fact, the probabilistic framework in [129], which only considered the free-choice structure, is the foundation for [1] and [2].

The *Covered Petri Nets* (CPNs) are nets covered by *agents* proposed by [69] to remedy confusion. The agents are responsible for resolving conflicts between their enabled transitions probabilistically and locally based on their local states. The priority of selecting an agent to resolve the conflict is given to the agent that has a complete view of the non-determinism of the net. Basically, their distributed algorithm, which is based on *semaphores*, of choosing

the agents provides a strategy or scheduler that selects which agent should be picked to resolve the next probabilistic choice. The confusion is handled by semaphores. Intuitively, certain places of the net are associated with a semaphore. The agents can *'lock'* or *'free'* the semaphore such that the confusion disappears. When an agent locks a semaphore associated with a place pivotal to the confusion, it may delay the execution of some transitions involved in the confusion or it may need to change the place as being marked by firing some transitions. The goal behind the semaphores is to ensure that the agents' perception of their probabilistic decisions is constant. In fact, this yields two-level scheduling technique: schedulers select the agents based on the global states and the agents make their probabilistic choices based on their own local states.

Petri nets with *priorities* are defined in [18]. There, transitions are given different priorities levels to address confusion. As the confusion arises due to the different ordering of the transitions, adding priority imposes a deterministic ordering of conflict resolution and hence the confusion is avoided. In General Stochastic Petri Nets (GSPNs), where the transitions are classified as timed and immediate, the confusion was *partially* handled using the weights and priorities. However, the authors in [43, 68, 19] emphasized that confusion is a dynamic phenomenon so it is not always solved by weights and priorities. Their approach for removing confusion in GSPNs is to translate confused GSPNs into Markov Automata where non-determinism and probabilities are presented. In fact, the analysis of GSPNs is generally limited to confusion-free nets [19]. It is worth to mention that the difference between Stochastic Petri Nets (SPNs) and GSPN is that in the former all the transitions are associated with a rate, concurrently enabled timed transitions are taken probabilistically, and the confusion is absent. However, in the latter the presence of immediate transitions causes the emergence of confusion [69].

Algorithms to detect confusion and an approach for avoiding it were considered in [31]. Basically, enforcing a constraint of supervisory control ensures that conflict transitions are enabled together so that confusion cannot occur in marking evolution. After detecting the subnet involved in a confusion, the *Places* and *Enabling* degree of some transitions belong to the confused subnet are identified. Then, *P/E* constraint is enforced via a monitor in order to avoid firing some transitions when certain places are marked to ensure that all the conflict transitions belong to the confused subnet are enabled at the same time. However, no approach is given of how the original behaviour is maintained. Confusion was controlled in [32] by attaching an external event with each transition involved in confusion. Then a control sequence is chosen so that the execution of these transitions is controlled.

The most recent and advanced theory proposed by [26] offers a new approach for removing confusion. Basically, the structural branching cells were utilized to statically identify the locus of alternatives, allowing a confusion-free net to be created from a given confused net by recursive static decomposition. Constructing a confusion-free net required generating a special class of places; persistence negative places were employed to create additional causality between the freshly generated transitions. The new non-deterministic net was then associated with probabilities to account for the conflict between transitions. However, only backward deterministic nets were studied in this paper. A further barrier is the complexity of the encoding procedure as the number of nested cells might grow exponentially.

A general comment which applies to the above past approaches is that they are all concerned with standard Petri nets, whereas the proposed work needs to find solutions for nets supporting different kinds of interprocess communication and abstraction. Also, all the previous work related to probabilities in Petri nets did not consider systems like crime investigation systems. In this thesis, we develop probabilistic framework for such systems.

Chapter 3

Acyclic Petri nets

3.1 Introduction

One of the mathematical modelling languages for the representation of distributed systems are Petri nets. They are directed graphs with two fundamental components, namely places and transitions. Their concept was developed by Carl Adam Petri at the age of 13 to illustrate chemical processes and introduced formally in his PhD Thesis in 1964 [99]. Petri Nets' execution semantics is precisely defined in mathematical terms, and for years they have been regarded as one of most suitable models to capture the notion of *concurrency*.

Acyclic Petri Nets, or just acyclic nets, are a class of Petri nets where iterated execution is excluded. However, they provide an explicit representation of *causality*, *conflict*, and *confusion*, three essential characteristics for concurrent systems. Causality is captured when a transition's enabledness is being dependent on the occurring of another transition. Conflict, on the other hand, means that the execution of one transition disables other transitions. Confusion in a mix between conflict and concurrency.

Acyclic nets are *probabilistic* when conflicts are resolved with probability estimation, and *non-deterministic* when conflicts are resolved randomly and without details of how it is resolved.

Developing probabilistic concurrent systems has been a challenge and a long standing problem. It is even more complicated when the *choices* are distributed. One reason is that in concurrent systems the computations are undistinguishable if they only differ by the execution order of independent transitions. The probability of a concurrent computation should be the same for all its different executions.

In this chapter, after introducing the relevant behavioural properties of acyclic nets, we develop a probabilistic acyclic nets framework such that transitions in conflict are associated

with positive numerical weights which represents their likelihood. Then the *scenarios*, which represents execution histories of an acyclic net, are derived with probabilities estimation. In addition, the notion of *confusion* is discussed as an undesirable situation that causes inaccurate probabilities calculation for same scenarios.

Handling confusion within concurrent semantics is a complicated problem. We propose two approaches to remove it based on a new class of acyclic nets, called *cluster-acyclic nets*. This class of acyclic nets uses the notion of *clusters* which partition the transitions into equivalence classes of conflict relation. Then, a confused net is encoded into another net which is *confusion-free*.

The chapter is organised as follows. In Section 3.2, the basic definitions and examples concerning acyclic nets are present. In Section 3.3, we introduce the notion of *probabilistic* acyclic nets, and we show with illustrated examples how the different executions of a concurrent scenario are assigned the same probability when confusion in not present. In Section 3.4, confusion is formally defined. Additionally, some examples are provided to demonstrate how confusion interferes with probabilistic calculation. Cluster-acyclic nets are introduced in Section 3.5 as a new class of acyclic nets. Section 3.6 provides two approaches of removing confusion from a fixed cluster acyclic net together with related algorithms. Extending these approaches of removing confusion to the unfolding semantics is presented in Section 3.7. Section 3.8 concludes the chapter.

3.2 Basic definitions

Definition 3.2.1 (acyclic net [8])**.** An *acyclic net* is a triple $acnet = (P, T, F)$, where P and T are disjoint finite sets of *places* and *transitions* respectively, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation* such that:

1. P is nonempty and F is acyclic.
2. For every $t \in T$, there are $p, q \in P$ such that pFt and tFq.

The set of all acyclic nets is denoted by AN. ◇

Graphically, places are represented by circles, transitions by boxes, and arcs between the nodes represent the flow relation.

We denote P, T, and F by P_{acnet}, T_{acnet}, and F_{acnet}, respectively, when it is required to indicate explicitly the net. To indicate relationships between different nodes, for all $x \in P \cup T$

and $X \subseteq P \cup T$, we denote the *directly preceding* and *directly following* nodes as follows:

$$\begin{array}{rclcrclcl} {}^\bullet x & = & \mathrm{pre}_{acnet}(x) & \triangleq & \{z \mid zFx\} & {}^\bullet X & = & \mathrm{pre}_{acnet}(X) & \triangleq & \bigcup\{{}^\bullet z \mid z \in X\} \\ x^\bullet & = & \mathrm{post}_{acnet}(x) & \triangleq & \{z \mid xFz\} & X^\bullet & = & \mathrm{post}_{acnet}(X) & \triangleq & \bigcup\{z^\bullet \mid z \in X\}\,. \end{array}$$

The *initial* and *final* places of *acnet* are respectively given by:

$$P^{init}_{acnet} \triangleq \{p \in P \mid {}^\bullet p = \varnothing\} \text{ and } P^{fin}_{acnet} \triangleq \{p \in P \mid p^\bullet = \varnothing\}\,.$$

Definition 3.2.2 (free-choice net [22])**.** An acyclic net $acnet = (P,T,F)$ is free-choice if, for every two transitions $t \neq u \in T$, if ${}^\bullet t \cap {}^\bullet u \neq \varnothing$ then $|{}^\bullet t| = |{}^\bullet u| = 1$. ◇

Definition 3.2.3 (extended free-choice net [41])**.** An acyclic net $acnet = (P,T,F)$ is extended free-choice if the following hold :

1. for every two places p and r, either $p^\bullet \cap r^\bullet = \varnothing$ or $p^\bullet = r^\bullet$.
2. for every two transitions t and u, either ${}^\bullet t \cap {}^\bullet u = \varnothing$ or ${}^\bullet t = {}^\bullet u$. ◇

3.2.1 Step sequence semantics

In this section, a set of behavioural notions are introduced. Given an acyclic net, its execution proceeds by the occurrence (or firing) of sets of transitions. The definitions below specify the conditions under which a marking enables a set of transitions (called a step), and how the execution of the transitions changes the current marking.

Definition 3.2.4 (marking and step [8])**.** Let *acnet* be an acyclic net.

1. $\mathrm{markings}(acnet) \triangleq \mathbb{P}(P_{acnet})$ are the *markings* and $M^{init}_{acnet} \triangleq P^{init}_{acnet}$ is the default *initial* marking.
2. $\mathrm{steps}(acnet) \triangleq \{U \in \mathbb{P}(T_{acnet}) \setminus \{\varnothing\} \mid \forall t \neq u \in U : {}^\bullet t \cap {}^\bullet u = \varnothing\}$ are the *steps*. ◇

Graphically, markings are shown by placing tokens within the circles. In an acyclic net *acnet*, a step U is a set of the transitions T_{acnet} such that all transitions in U are conflict-free.

Definition 3.2.5 (enabled and executed step [8])**.** Let M be a marking of an acyclic net *acnet*.

1. $\mathrm{enabled}_{acnet}(M) \triangleq \{U \in \mathrm{steps}(acnet) \mid {}^\bullet U \subseteq M\}$ are the steps *enabled* at M.

2. A step U enabled at M can be *executed* yielding a new marking $M' \triangleq (M \cup U^\bullet) \setminus {}^\bullet U$. This is denoted by $M[U\rangle_{acnet} M'$ or $M \xrightarrow{U} M'$. ◇

Markings of acyclic nets are safe by definition, i.e., a place can 'hold' only one token. Note that if $M[U\rangle_{acnet} M'$ and $t \neq u \in U$ are such that $p \in t^\bullet \cap u^\bullet$, then p will 'hold' only one token in M'. Such a feature will be useful when dealing with *OR causality* and negative tokens later on. For a step U to be enabled at marking M, all its input places are required to be included in M. Executing the enabled steps is responsible of transferring the tokens from one global state into another. That is, executing a step U removes all the tokens in its input places and then produces tokens in all its output places.

Definition 3.2.6 (mixed step sequence and step sequence [8])**.** Let $M_0, \ldots, M_k$ $(k \geq 0)$ be markings and $U_1, \ldots, U_k$ be steps of an acyclic net *acnet* such that $M_{i-1}[U_i\rangle_{acnet} M_i$, for every $1 \leq i \leq k$.

1. $\mu = M_0 U_1 M_1 \ldots M_{k-1} U_k M_k$ is a *mixed step sequence from M_0 to M_k*.
2. $\sigma = U_1 \ldots U_k$ is a *step sequence from M_0 to M_k*.

The above is denoted by $M_0[\mu\rangle\rangle_{acnet} M_k$ and $M_0[\sigma\rangle_{acnet} M_k$, respectively. Moreover, $M_0[\rangle_{acnet} M_k$ denotes that M_k is *reachable from M_0*. ◇

If $k = 0$ then $\mu = M_0$ and the corresponding step sequence σ is the *empty* sequence denoted by λ.

Definition 3.2.7 (behavioural notions [8])**.** The following sets capture various behavioural notions related to step sequences and reachable markings of an acyclic net *acnet*.

1. $\text{sseq}(acnet) \triangleq \{\sigma \mid M^{init}_{acnet}[\sigma\rangle_{acnet} M\}$ are the *step sequences*.
2. $\text{mixsseq}(acnet) \triangleq \{\mu \mid M^{init}_{acnet}[\mu\rangle\rangle_{acnet} M\}$ are the *mixed step sequences*.
3. $\text{maxsseq}(acnet) \triangleq \{\sigma \in \text{sseq}(acnet) \mid \nexists U : \sigma U \in \text{sseq}(acnet)\}$ are the *maximal step sequences*.
4. $\text{maxmixsseq}(acnet) \triangleq \{\mu \in \text{mixsseq}(acnet) \mid \nexists U, M : \mu U M \in \text{mixsseq}(acnet)\}$ are the *maximal mixed step sequences*.
5. $\text{reachable}(acnet) \triangleq \{M \mid M^{init}_{acnet}[\rangle_{acnet} M\}$ are the *reachable markings*.
6. $\text{finreachable}(acnet) \triangleq \{M \mid \exists \sigma \in \text{maxsseq}(acnet) : M^{init}_{acnet}[\sigma\rangle_{acnet} M\}$ are the *final reachable markings*

7. $\mathrm{fseq}(acnet) = \{U_1 \ldots U_k \in \mathrm{sseq}(acnet) \mid k \geq 1 \implies |U_1| = \cdots = |U_k| = 1\}$ are the *firing sequences*.

8. $\mathrm{conflset}_{acnet}(M,t) \triangleq \{t\} \cup \{u \in \mathrm{enabled}_{acnet}(M) \mid t \#_0 u\}$ is the *conflict set* of a transition $t \in T$ enabled at a marking M. ◇

We can treat individual transitions as singleton steps; for instance, a step sequence $\{t\}\{u\}\{w,v\}\{z\}$ can be denoted by $tu\{w,v\}z$.

Note that an occurrence net is a *deterministic process* capturing an equivalence class of maximal step sequences that are distinguishable only in the order of concurrent transitions [26].

3.2.2 Conflict, causality and concurrency

This section introduces structural properties of acyclic nets.

Definition 3.2.8 (structural notions [8])**.** Let $acnet = (P,T,F)$ be an acyclic net.

1. Two transitions $t \neq u \in T$ are in *direct (forward) conflict*, denoted $t \#_0 u$, if they have a common place in their presets, i.e., ${}^\bullet t \cap {}^\bullet u \neq \varnothing$.

2. Two transitions $t \neq u \in T$ are in *direct backward conflict* if they have a common place in their postsets, i.e., $t^\bullet \cap u^\bullet \neq \varnothing$.

3. Two nodes $x,y \in P \cup T$ are in *conflict*, denoted $x \# y$, if there are transitions t and u such that $t \#_0 u$ and $(t,x),(u,y) \in F^*$.

4. A *self-conflict* transition is any $t \in T$ such that $t \# t$.

5. $\preceq$ given by $F^+|_{T \times T}$ is the *causality* relation on transitions.

6. Two nodes $x \neq y \in P \cup T$ are *concurrent*, denoted $x \,\mathrm{co}\, y$, if neither $x \# y$ nor $x F^+ y$ nor $y F^+ x$.

7. $\mathrm{caused}_{acnet}(x) \triangleq \{y \in P \cup T \mid x F^+ y\}$ are the elements *caused* by $x \in P \cup T$. Moreover, $\mathrm{caused}_{acnet}(X) = \bigcup_{x \in X} \mathrm{caused}_{acnet}(x)$, for $X \subseteq P \cup T$.

8. $\mathrm{subnet}_{acnet}(V) \triangleq ({}^\bullet V \cup V^\bullet, V, F|_{({}^\bullet V \times V) \cup (V \times V^\bullet)})$ is the *subnet* induced by a set of transitions $V \subseteq T$. ◇

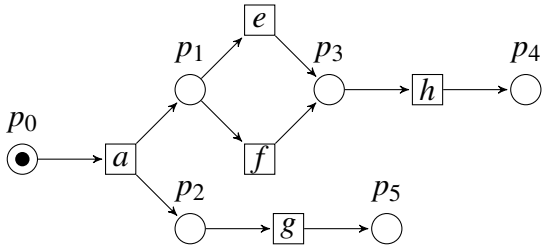


Fig. 3.1 Acyclic net with initial marking.

Example 6. Figure 3.1 shows a free-choice acyclic net *acnet*, where

$$\begin{aligned} P &= \{p_0, p_1, p_2, p_3, p_4, p_5\} \\ T &= \{a, e, f, g, h\} \\ F &= \{(p_0, a), (a, p_1), (a, p_2), \ldots, (h, p_4)\}. \end{aligned}$$

The sets of initial and final places are $\{p_0\}$ and $\{p_4, p_5\}$, respectively. This acyclic net exhibits backward and forward non-determinism, namely transitions e and f are involved both in direct forward conflict and direct backward conflict. The two possible maximal steps sequences are $\sigma_1 = \{a\}\{e,g\}\{h\}$ and $\sigma_2 = \{a\}\{f,g\}\{h\}$. ◇

Intuitively, conflicts between transitions emerge from having a common pre-place (in the case of forward non-determinism) or a common post-place (in the case of backward non-determinism), while concurrency results from multiple post-places emerging from a transition [106].

An acyclic net can exhibit *backward non-determinism* (when $|{}^\bullet p| > 1$, for some place p) and *forward non-determinism* (when $|p^\bullet| > 1$, for some place p).

Definition 3.2.9 (backward deterministic acyclic net [8])**.** A *backward deterministic acyclic net* is an acyclic net such that $|{}^\bullet p| \leq 1$, for every place p. The set of all backward deterministic acyclic nets is denoted by *BDAN*. ◇

Example 7. Figure 3.2 shows an example of backward deterministic acyclic net. c_1 is the common pre-place for the transitions e and f (which are in direct conflict), and each place except c_0 has at most one incoming arc. ◇

Definition 3.2.10 (occurrence net [8])**.** An *occurrence net* is an acyclic net such that $|{}^\bullet p| \leq 1$ and $|p^\bullet| \leq 1$, for every place p. The set of all occurrence nets is denoted by *ON*. ◇

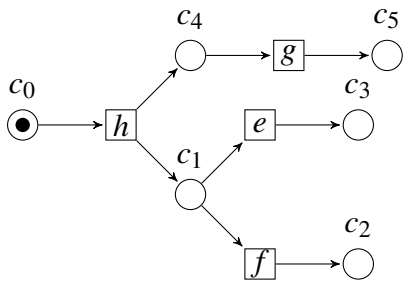


Fig. 3.2 Backward deterministic acyclic net with initial marking.

An occurrence net describes a single execution of some concurrent system in such a way that only the details of causality and concurrency between transitions are captured [78]. It can represent an execution history of a system. In other words, one can build an occurrence net to model a system's history [104]. Moreover, the transitions with causal dependencies relations are ordered whereas those that are casually independent are unordered [84].

Next, we define *scenarios* as structurally unique execution histories of an acyclic net that can be used later to analyse the acyclic net *probabilistically*.

Definition 3.2.11 (scenario and maximal scenario [8])**.** Let *acnet* be an acyclic net.

1. A *scenario* of *acnet* is an occurrence net *ocnet* such that:
   (a) $T_{ocnet} \subseteq T_{acnet}$ and $P_{ocnet} = M^{init}_{acnet} \cup \text{post}_{acnet}(T_{ocnet})$.
   (b) $\text{pre}_{ocnet}(t) = \text{pre}_{acnet}(t)$ and $\text{post}_{ocnet}(t) = \text{post}_{acnet}(t)$, for every $t \in T_{ocnet}$.
2. A *maximal scenario* of *acnet* is a scenario *ocnet* such that there is no scenario *ocnet*$'$ satisfying $T_{ocnet} \subset T_{ocnet'}$.

The sets of all scenarios and maximal scenarios are denoted by scenarios(*acnet*) and maxscenarios(*acnet*), respectively. ◇

Basically, scenarios are subnets that adhere to forward and backward determinism and start with the same initial marking. Also, they provide an abstract view of the behaviour without details of the actual executions. As maximal scenarios are complete and cannot be extended, an occurrence net is itself its only maximal scenario.

Each scenario of an acyclic net $acnet = (P,T,F)$ is identified by the set of its transitions. The scenario with the transition set $V \subseteq T$ is denoted by $\text{scenario}_{acnet}(V)$ and given by:

$$\text{scenario}_{acnet}(V) \triangleq (P,V,F|_{(P\times V)\cup(V\times P)}), \quad \text{where} \quad P = P^{init}_{acnet} \cup V^{\bullet}.$$

Example 8. Figure 3.3 shows the only two maximal scenarios of the acyclic net in Figure 3.1. The net in (a) represents the first maximal scenario which is associated with the transitions a, e, g, h, i.e., $ocnet_1 = \text{scenario}_{acnet}(\{a, e, g, h\})$. The second maximal scenario $ocnet_2 = \text{scenario}_{acnet}(\{a, f, g, h\})$ is shown in (b). ◇

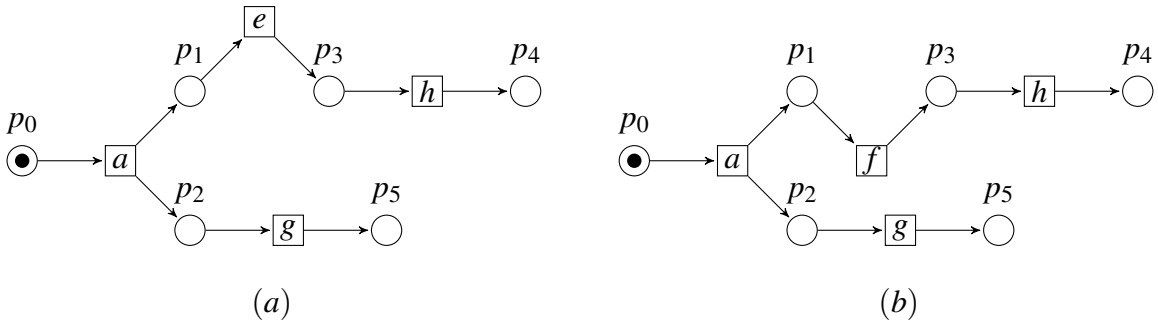


Fig. 3.3 Maximal scenarios (with initial markings) of the acyclic net in Figure 3.1.

3.2.3 Well-formed acyclic nets

In this section, well-formedness is defined as a fundamental consistency property of acyclic nets. Its main purpose is to guarantee an unambiguous and unique set of causal dependencies in behaviours of acyclic nets, where, in terms of graph theory, the causality relation between e and f is represented by a directed path from e to f. Understanding causality in acyclic nets can be related to their step sequences [8]. For instance, let $e, f \in T_{acnet}$ be such that e is a cause of f (this is captured by the flow relation $eF^{+}_{acnet}f$). Then, for every step sequence $\sigma U \in \text{sseq}(acnet)$ such that $f \in U$, e appears in σ.

Definition 3.2.12 (well-formedness [8])**.** An acyclic net *acnet* is *well-formed* if each transition occurs in at least one step sequence and the following hold, for every step sequence $U_1 \ldots U_k \in \text{sseq}(acnet)$:

1. $t^\bullet \cap u^\bullet = \varnothing$, for every $1 \leq i \leq k$ and all $t \neq u \in U_i$.
2. $U_i^\bullet \cap U_j^\bullet = \varnothing$, for all $1 \leq i < j \leq k$. ◇

Intuitively, a well-formed acyclic net does not have 'useless' transitions and no place is filled more than once in any given step sequence.

In the rest of this chapter, an acyclic net which is not a result of encoding is always assumed to be well-formed.

Proposition 3.2.1 ([8])**.** A backward deterministic acyclic net is well-formed iff it has no self-conflict transitions.

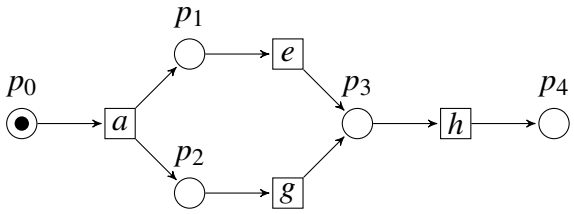


Fig. 3.4 An acyclic net which is not a well-formed

Example 9. Figure 3.4 illustrates how the well-formedness property is violated. For example, in the step sequence $\sigma = a\{e,g\}$, we have $e^\bullet \cap g^\bullet = \{p_3\} \neq \varnothing$.
We further observe that the only maximal 'scenario', which includes all the transitions, does not respect Definitions 3.2.10 and Definition 3.2.11 as the number of incoming arcs for p_3 is greater than one. Hence, in terms of causal history for transition h, it is not obvious whether it is caused by e or by g. More precisely, h is fireable whenever a token generated by e or g arrives at p_3. Intuitively, the 'OR causality' between e and g violates the property of well-formedness. ◇

Algorithm 1 Verifying well-formedness.

```
1:  function BOOLEAN WELLFORMED(σ)
2:      Input: step sequence σ = U_1 ... U_k
3:      Output: true if σ is well-formed; otherwise false
4:      for each 1 ≤ i < j ≤ k do
5:          if (U_i• ∩ U_j• ≠ ∅ ) then
6:              return false
7:          end if
8:      end for
9:      for each step U_i do
10:         for each two transitions t ≠ t' ∈ U_i do
11:             if (t• ∩ t'• ≠ ∅ ) then
12:                 return false
13:             end if
14:         end for
15:     end for
16:     return true
17: end function
```

Algorithm 1 checks whether a given step sequence σ is 'well-formed', i.e., whether the conditions formulated as Definition 3.2.12(1,2) are satisfied. The input is a step sequence $\sigma = U_1 \dots U_k$. The conditions in the if-statements are based on Definition 3.2.12(1,2). If neither steps U_i and U_j nor their transitions t and t' break these conditions, *true* is returned.

Further details of checking for well-formedness are discussed in Chapter 6.

Proposition 3.2.2 ([8])**.** Each occurrence net is well-formed.

Proposition 3.2.3 ([8])**.** The following statements are equivalent, for every acyclic net *acnet*.

1. *acnet* is well-formed.
2. Each transition occurs in at least one scenario, and each step sequence is a step sequence of at least one scenario.
3. Each transition occurs in at least one scenario, and each firing sequence is a firing sequence of at least one scenario.
4. Each transition occurs in at least one firing sequence and, for every firing sequence $t_1 \dots t_k \in \text{fseq}(acnet)$ and for all $1 \leq i < j \leq k$, $t_i^\bullet \cap t_j^\bullet = \varnothing$.

Proposition 3.2.4 ([8])**.** Step sequences of a well-formed acyclic net do not contain multiple occurrences of transitions.

Each step sequence σ of a well-formed acyclic net *acnet* induces a scenario

$$\text{scenario}_{acnet}(\sigma) \triangleq \text{scenario}_{acnet}(\bigcup \sigma)$$

such that $\sigma \in \text{maxsseq}(\text{scenario}_{acnet}(\sigma))$. Thus, in a well-formed acyclic net, different step sequences may generate the same scenario, and conversely, each scenario is generated by at least one step sequence. Moreover, two maximal step sequences generate the same scenario iff their executed transitions are identical.

Example 10. Figure 3.3 can be used to illustrate the last point by showing the maximal scenario $ocnet_1 = \text{scenario}_{acnet}(\{a,e,g,h\})$ has five executions:
$\{p_0\} \xrightarrow{a} \{p_1,p_2\} \xrightarrow{\{e,g\}} \{p_3,p_5\} \xrightarrow{h} \{p_4,p_5\}$,
$\{p_0\} \xrightarrow{a} \{p_1,p_2\} \xrightarrow{g} \{p_1,p_5\} \xrightarrow{e} \{p_3,p_5\} \xrightarrow{h} \{p_4,p_5\}$,
$\{p_0\} \xrightarrow{a} \{p_1,p_2\} \xrightarrow{e} \{p_2,p_3\} \xrightarrow{\{g,h\}} \{p_4,p_5\}$,
$\{p_0\} \xrightarrow{a} \{p_1,p_2\} \xrightarrow{e} \{p_2,p_3\} \xrightarrow{g} \{p_3,p_5\} \xrightarrow{h} \{p_4,p_5\}$, and
$\{p_0\} \xrightarrow{a} \{p_1,p_2\} \xrightarrow{e} \{p_2,p_3\} \xrightarrow{h} \{p_2,p_4\} \xrightarrow{g} \{p_4,p_5\}$.

The above five maximal step sequences generate the same scenario involving transitions $\{a,e,g,h\}$. ◇

3.3 Calculating probabilities in acyclic nets

Probabilistic models are the key to cope with uncertainty. They provide mathematical frameworks to analyse behaviours with a limited knowledge so that we are capable of making reasonable decisions.

As the real world is full of uncertainty, we often estimate, or even guess, when quantifying uncertainty. For instance, in CESs full information about a specific activity is, in general, not available [106]. Moreover, available evidence in crime or accident investigation is usually unspecific and/or contradictory. Hence (often numerous) alternate scenarios are pursued by investigators in order to clarify the status of a crime or accident [84]. Therefore, adding probability estimates to transitions that represent crime events would crucially help to increase the degree of certainty of different scenarios and to prioritise investigation of alternate scenarios. The investigators could also interpret relative likelihood of crime transitions and then combine them into meaningful reports.

In acyclic nets, in order to find the probabilities of alternate scenarios and identify the probabilistic behaviour, conflicting transitions can be assigned *positive numerical weights*. Weights represent the likelihood of transitions and a *zero* weight for a transition is not allowed. *In our discussion, we do not care where a probability estimates comes from, or how are they derived.*

Associating conflict transitions with weights to resolve conflict probabilistically has been investigated in [43, 68, 128]. However, in [26, 87, 9] the conflict is resolved by probabilistic arcs. The examples in [19] have shown that assigning weights to the transitions or to the arcs would produce the same result.

From now on, we assume that each acyclic net *acnet* has an additional (last) component ω defined as a mapping from the set of transitions to positive integers. For each transition t, $\omega(t)$ is its *weight* which in diagrams annotates the corresponding node. Also, we assume that the weights are assigned to the transitions rather than the arcs and are represented inside the boxes. In such cases the names of transitions are represented outside.

Probabilities of concurrent transitions are given by the products of their weights over the sum of the weights of transitions in their conflict sets. More precisely, we define the

probability of executing a transition t and step U enabled at a reachable marking M as:

$$\mathbf{P}_{acnet}(M,t) \triangleq \frac{\omega(t)}{\sum_{u \in \mathrm{conflset}_{acnet}(M,t)} \omega(u)} \quad \textit{and} \quad \mathbf{P}_{acnet}(M,U) \triangleq \prod_{t \in U} \mathbf{P}_{acnet}(M,t) \ .$$

We then define the probability of an execution $\sigma = U_1 \ldots U_k$ as

$$\mathbf{P}_{acnet}(\sigma) \triangleq \mathbf{P}_{acnet}(M_0,U_1) \cdot \ldots \cdot \mathbf{P}_{acnet}(M_{k-1},U_k) \ ,$$

where $M_0, \ldots, M_{k-1}$ are such that $M_0 \xrightarrow{U_1} \ldots M_{k-1} \xrightarrow{U_k} M_k$.

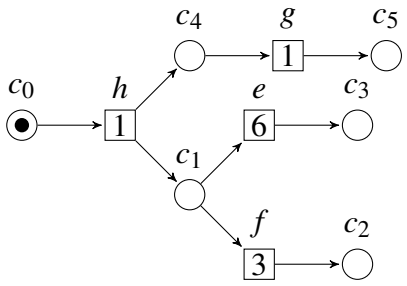


Fig. 3.5 Backward deterministic acyclic net with weights.

Example 11. Consider the acyclic net *acnet* in Figure 3.5. Transitions e and f are in direct conflict and their weights are $\omega(e) = 6$ and $\omega(f) = 3$. Such a conflict can be resolved probabilistically at reachable markings $M = \{c_1, c_4\}$ and $M' = \{c_1, c_5\}$ by calculating probabilities as follows:

$$\begin{array}{rclcrcl} \mathbf{P}_{acnet}(M,e) & = & \mathbf{P}_{acnet}(M',e) & = & \frac{\omega(e)}{\omega(e)+\omega(f)} & = & \frac{6}{9} \\ \mathbf{P}_{acnet}(M,f) & = & \mathbf{P}_{acnet}(M',f) & = & \frac{\omega(f)}{\omega(e)+\omega(f)} & = & \frac{3}{9} \ . \end{array}$$

Note that h and g are *certain transitions* since no transition competes with them for a token, and so their actual weights are irrelevant.

There are two maximal scenarios for the acyclic net in Figure 3.5, namely $ocnet_1 = \mathrm{scenario}(\{h,g,e\})$ and $ocnet_2 = \mathrm{scenario}(\{h,g,f\})$. Each of the scenarios can be executed in several ways by following different maximal step sequences:

- $ocnet_1$ has three executions: $\sigma_1 = heg$, $\sigma_2 = hge$, and $\sigma_3 = h\{e,g\}$. Their probabilities are the same as we have:

$$\begin{array}{lcll} \mathbf{P}_{acnet}(\sigma_1) & = & \frac{\omega(h)}{\omega(h)} \cdot \frac{\omega(e)}{\omega(e)+\omega(f)} \cdot \frac{\omega(g)}{\omega(g)} & = 1 \cdot \frac{6}{9} \cdot 1 \quad = \quad \frac{6}{9} \\ \mathbf{P}_{acnet}(\sigma_2) & = & \frac{\omega(h)}{\omega(h)} \cdot \frac{\omega(g)}{\omega(g)} \cdot \frac{\omega(e)}{\omega(e)+\omega(f)} & = 1 \cdot 1 \cdot \frac{6}{9} \quad = \quad \frac{6}{9} \\ \mathbf{P}_{acnet}(\sigma_3) & = & \frac{\omega(h)}{\omega(h)} \cdot \left(\frac{\omega(e)}{\omega(e)+\omega(f)} \cdot \frac{\omega(g)}{\omega(g)} \right) & = 1 \cdot (\frac{6}{9} \cdot 1) \quad = \quad \frac{6}{9} \end{array}$$

- $ocnet_2$ has also three executions: $\sigma_4 = hfg$, $\sigma_5 = hgf$, and $\sigma_6 = h\{f,g\}$. Their probabilities are equal as well:

$$\mathbf{P}_{acnet}(\sigma_4) = \mathbf{P}_{acnet}(\sigma_5) = \mathbf{P}_{acnet}(\sigma_6) = \frac{3}{9}\,.$$

As a result, we can assign probabilities to the two maximal scenarios:

$$\mathbf{P}_{acnet}(ocnet_1) = \frac{6}{9} \;\; \textit{and} \;\; \mathbf{P}_{acnet}(ocnet_2) = \frac{3}{9}\,.$$

Note that $\mathbf{P}_{acnet}(ocnet_1) + \mathbf{P}_{acnet}(ocnet_2) = 1$. ◇

The last example illustrates the point that a key issue for our investigation is to

> *"ensure in each case that no matter which of the executions of a scenario is followed in the calculation of probabilities, the same value is obtained (in other words, all maximal step sequences generated from a scenario should have the same probability)".*

That is, for every $ocnet \in \text{scenarios}(acnet)$ and all $\sigma, \sigma' \in \text{maxsseq}(ocnet)$, it should be the case that $\mathbf{P}_{acnet}(\sigma) = \mathbf{P}_{acnet}(\sigma')$. One can then define the probability of the scenario as $\mathbf{P}_{acnet}(ocnet) = \mathbf{P}_{acnet}(\sigma)$, where σ is any execution of *ocnet*.

Moreover, to avoid *probability normalisation*, one might also expect that the sum of the probabilities of all maximal scenarios is one, i.e.,

$$\sum_{ocnet \in \text{maxscenarios}(acnet)} \mathbf{P}_{acnet}(ocnet) = 1.$$

However, the above is not always the case as acyclic nets can exhibit *confusion* described next.

3.4 Confusion

Confusion is an interplay between conflict and concurrency which interferes with the calculation of probabilities when the conflict is resolved probabilistically. It basically occurs when the execution of one of two concurrent (independent) enabled transitions *influences*, in terms of expanding or shrinking, the conflict set of the other transition. Confusion has been of interest for a considerable period since the early interest in Petri net research [31, 26, 43, 68, 90, 19, 111, 32, 69].

Confusion exists in the real world in different forms, depending on the context of describing concurrent systems [111]. For instance, we have the following examples of *confusion*:

- The paper [26] describes the situation where you can choose either to come to the town or go to the sea, and the permanent choice of your partner to go to the theatre clarifies that the availability of alternatives for you.
- In [69], two scientists confront a tough decision as they either have the choice to write a paper separately for a conference, where each can participate in only one paper, or join together to produce a paper.
- The concurrency between *"Update"* event and *"Fault alarm"* event in the fault recovery system in [32], where *"Recovery"* event can never be fired when the system executes *"Update"*. In the same vein, the model of a control system in [89] the emergency system never be executed when the system is automatically shout down.

The two types of confusion are introduced below:
In a *symmetric confusion*, executing transition f concurrent with e removes at least one transition from the conflict set of e. In an *asymmetric confusion*, executing transition f concurrent with e adds a new transition to the conflict set of e [31]. Hence as the conflict set is modified according to the order in which one chooses to execute e and f, the same scenario would obtain different probabilities.

For each well-defined confusion-free acyclic net one can calculate probabilities of individual scenarios. This may no longer be the case if confusion is present. As a result, systems with confusion are typically challenging to analyse.

Definition 3.4.1 (confusion [26])**.** A well-formed acyclic net *acnet* has a *confusion* at a reachable marking M if there are distinct transitions e, f, h such that $\{e, f\} \in \text{enabled}_{acnet}(M)$, ${}^\bullet e \cap {}^\bullet f = \varnothing$ and one of the following holds:

- $e\#_0 h\#_0 f$ and $h \in \text{enabled}_{acnet}(M)$.
- $e\#_0 h$ and $h \in \text{enabled}_{acnet}(M') \setminus \text{enabled}_{acnet}(M)$, where $M[f\rangle_{acnet} M'$.

We then denote

- $\text{symconfused}_{acnet}(M,e,f,h)$ in the first (*symmetric*) case, and
- $\text{asymconfused}_{acnet}(M,e,f,h)$ in the second (*asymmetric*) case.

We also say that the acyclic net *acnet* is *confusion-free* if it has no confusion in all its reachable markings. ◇

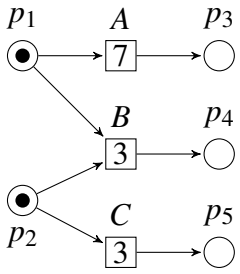


Fig. 3.6 An acyclic net with a symmetric confusion.

Example 12. Figure 3.6 depicts a well-formed acyclic net *acnet* with symmetric confusion as we have $\text{symconfused}_{acnet}(M,A,C,B)$, where $M = \{p_1, p_2\}$. There are two maximal scenarios:

- $ocnet_1 = \text{scenario}_{acnet}(\{A,C\})$ with three executions ($\sigma_1 = AC$, $\sigma_2 = CA$, and $\sigma_3 = \{A,C\}$), and
- $ocnet_2 = \text{scenario}_{acnet}(\{B\})$ with one execution only.

Transitions A and C are in direct conflict with B, and they can be independently enabled at M. In σ_1, where A is executed first then C yields to disable B which reduces the size of the conflict set of C. More precisely, $\text{conflset}_{acnet}(\{p_1,p_2\},C) = \{B\} \neq \text{conflset}_{acnet}(\{p_3,p_2\},C) = \varnothing$. Similarly, executing C then A as in σ_2 modifies the conflict set of A. That means $\text{conflset}_{acnet}(\{p_1,p_2\},A) = \{B\} \neq \text{conflset}_{acnet}(\{p_1,p_5\},A) = \varnothing$. Accordingly, the same scenario $ocnet_1 = \text{scenario}_{acnet}(\{A,C\})$ obtains different probabilities as shown below:

If σ_1 is executed, we follow $\{p_1,p_2\} \xrightarrow{A} \{p_3,p_2\} \xrightarrow{C} \{p_3,p_5\}$. Hence the probability of C is 1 (C is in this case a certain transition), and so the probability of σ_1 is

$$\mathbf{P}_{acnet}(\sigma_1) = \frac{\omega(A)}{\omega(A)+\omega(B)} \cdot \frac{\omega(C)}{\omega(C)} = \frac{7}{10} \cdot 1 = 0.7 \; .$$

However, if σ_2 is executed, then the resulting probability is

$$\mathbf{P}_{acnet}(\sigma_2) = \frac{\omega(C)}{\omega(B)+\omega(C)} \cdot \frac{\omega(A)}{\omega(A)} = \frac{3}{6} \cdot 1 = 0.5 \neq \mathbf{P}_{acnet}(\sigma_1) \; .$$

Hence one cannot assign a probability to $ocnet_1 = \text{scenario}_{acnet}(\{A,C\})$.

A similar conclusion can be reached for the acyclic net in Figure 3.7 which exhibits asymmetric confusion at a reachable marking $M = \{p_1,p_2\}$ captured by $\text{asymconfused}_{acnet}(M^{init}_{acnet},a,b,c)$. The three maximal scenarios are:

$$\begin{array}{rcl} ocnet_1 & = & \text{scenario}_{acnet}(\{d,a\}) \\ ocnet_2 & = & \text{scenario}_{acnet}(\{b,c\}) \\ ocnet_3 & = & \text{scenario}_{acnet}(\{b,a\}). \end{array}$$

$\text{scenario}_{acnet}(\{b,a\})$ has three executions : $\sigma_1 = ba$, $\sigma_2 = ab$, and $\sigma_3 = \{b,a\}$, with different probabilities. For example, if σ_1 is executed, then firing b first adds c to the conflict set of a; $\text{conflset}_{acnet}(\{p_1,p_2\},a) = \varnothing \neq \text{conflset}_{acnet}(\{p_2,p_3\},a) = \{c\}$. Then the obtained probability is

$$\mathbf{P}_{acnet}(\sigma_1) = \frac{\omega(b)}{\omega(b)+\omega(d)} \cdot \frac{\omega(a)}{\omega(a)+\omega(c)} = \frac{3}{10} \cdot \frac{4}{10} = \frac{12}{100} \; .$$

However, in σ_2 and σ_3, a is taken with probability one as no transitions are enabled competing with a for the token in p_2. Hence we obtain

$$\mathbf{P}_{acnet}(\sigma_2) = \mathbf{P}_{acnet}(\sigma_3) = \frac{\omega(a)}{\omega(a)} \cdot \frac{\omega(b)}{\omega(b)+\omega(d)} = 1 \cdot \frac{3}{10} = \frac{3}{10} \neq \mathbf{P}_{acnet}(\sigma_1) \; .$$

Due to the asymmetric confusion involving the transitions a,b,c, calculating probabilities for $ocnet_3 = \text{scenario}_{acnet}(\{b,a\})$ in Figure 3.7 is not accurate. ◇

The result below shows that due to the presence of confusion, conflict set of a given transition is modified by the execution of another transition with which it is not in conflict.

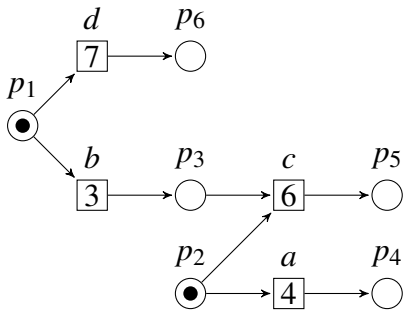


Fig. 3.7 An acyclic net with an asymmetric confusion.

Proposition 3.4.1. Let *acnet* be a well-formed acyclic net and M be its reachable marking. If $\text{symconfused}_{acnet}(M,e,f,h)$ or $\text{asymconfused}_{acnet}(M,e,f,h)$, then $\text{conflset}_{acnet}(M,e) \neq \text{conflset}_{acnet}(M',e)$ and $e \in \text{enabled}_{acnet}(M')$, where $M[f\rangle_{acnet} M'$.

Proof. We will consider the two cases in Definition 3.4.1.

- First case: Let us assume that there is a symmetric confusion at the reachable marking M. So, by the first case of Definition 3.4.1, we have $\text{symconfused}_{acnet}(M,e,f,h)$ implying ${}^\bullet e \cap {}^\bullet f = \varnothing$, $e \#_0 h \#_0 f$, and $\{e,f\} \in \text{enabled}_{acnet}(M)$.

  Since $e\#_0 h$ and $f\#_0 h$, we have $h \in \text{conflset}_{acnet}(M,e) \cap \text{conflset}_{acnet}(M,f)$. Executing f disables h and leads to a marking M' such that $e \in \text{enabled}_{acnet}(M')$ and $h \notin \text{enabled}_{acnet}(M')$. Hence,

$$h \in \text{conflset}_{acnet}(M,e) \setminus \text{conflset}_{acnet}(M',e)$$

  and $e \in \text{enabled}_{acnet}(M) \cap \text{enabled}_{acnet}(M')$.

- Second case: Let assume that there is an asymmetric confusion at the reachable marking M. So, by the second case of Definition 3.4.1, we have $\text{asymconfused}_{acnet}(M,e,f,h)$ implying ${}^\bullet e \cap {}^\bullet f = \varnothing$, $e\#_0 h$, and $\{e,f\} \in \text{enabled}_{acnet}(M)$, whereas $h \in \text{enabled}_{acnet}(M') \setminus \text{enabled}_{acnet}(M)$, where $M[f\rangle_{acnet} M'$. Since $e\#_0 h$ and $h \in \text{enabled}_{acnet}(M')$, we have

$$h \in \text{conflset}_{acnet}(M',e) \setminus \text{conflset}_{acnet}(M,e)$$

  and $e \in \text{enabled}_{acnet}(M) \cap \text{enabled}_{acnet}(M')$.

□

A crucial property is that in a confusion-free acyclic net, conflict sets of transitions are constant for all the executions of each scenario.

Proposition 3.4.2. Let *acnet* be a confusion-free well-formed acyclic net.

1. If M and M' are two reachable markings of *acnet* such that $M[\rangle_{acnet} M'$, then

$$\text{conflset}_{acnet}(M,t) = \text{conflset}_{acnet}(M',t) \ ,$$

   for every transition $t \in \text{enabled}_{acnet}(M) \cap \text{enabled}_{acnet}(M')$.

2. If M and M' are two reachable markings of $ocnet \in \text{scenarios}(acnet)$, then

$$\text{conflset}_{acnet}(M,t) = \text{conflset}_{acnet}(M',t) \ ,$$

   for every transition $t \in \text{enabled}_{ocnet}(M) \cap \text{enabled}_{ocnet}(M')$.

Proof.

1. We will prove this by showing that if M' is obtained by executing only one transition and that $\text{conflset}_{acnet}(M,t) \neq \text{conflset}_{acnet}(M',t)$, then *acnet* is not confusion-free acyclic net producing a contradiction. So, there are two cases:

   - First case: Let us assume that there is a transition h such that

$$h \in \text{conflset}_{acnet}(M,t) \setminus \text{conflset}_{acnet}(M',t)$$

     where M' is obtained by executing only one transition $u \neq t$, so $M[u\rangle_{acnet} M'$. We have $\{t,u\} \in \text{enabled}_{acnet}(M)$, $t \#_0 h \#_0 u$, and $h \in \text{enabled}_{acnet}(M) \setminus \text{enabled}_{acnet}(M')$. Then, according to the first part of Definition 3.4.1, there is a symmetric confusion at reachable marking M such that $\text{symconfused}_{acnet}(M,t,u,h)$ holds.

   - Second case: Let us assume that there is a transition h such that

$$h \in \text{conflset}_{acnet}(M',t) \setminus \text{conflset}_{acnet}(M,t)$$

     where M' is obtained by executing only one transition $u \neq t$, so $M[u\rangle_{acnet} M'$. We have $\{t,u\} \in \text{enabled}_{acnet}(M)$, $t \#_0 h$, and $h \in \text{enabled}_{acnet}(M') \setminus \text{enabled}_{acnet}(M)$. Then, according to the second part of Definition 3.4.1, there is an asymmetric confusion at reachable marking M such that $\text{asymconfused}_{acnet}(M,t,u,h)$ holds.

The above proof of two cases considers the situation where generating M' requires executing only one transition u. However, this is not always the case. Thus, we might have $M^{init}_{acnet}[t_1 \dots t_m\rangle M[u_1 \dots u_k\rangle M'$, which means that

$$M = M_0[u_1\rangle M_1 \dots M_{k-1}[u_k\rangle M_k = M'$$

where t is enabled at every marking M_i (knowing that *acnet* is well-formed and using the result of Proposition 9(4) from [8] implies that $t \in \text{enabled}_{acnet}(M_i)$ for $i = 0, \dots, k$). Hence, by what we already demonstrated in this proof, we have:

$$\text{conflset}_{acnet}(M_0, t) = \text{conflset}_{acnet}(M_1, t) = \dots = \text{conflset}_{acnet}(M_k, t).$$

2. Since it is not guaranteed that M and M' are reachable from each other, we first use *twice* the result of Proposition 8(4) from [8] to conclude that there is a reachable marking M'' such that $M''[\rangle M$ and $M''[\rangle M'$ and $t \in \text{enabled}_{acnet}(M'')$. As both markings M and M' are reachable from M'', then the first part of this result can be applied. So, it follows that

$$\text{conflset}_{acnet}(M, t) = \text{conflset}_{acnet}(M'', t) = \text{conflset}_{acnet}(M', t).$$

Hence, $\text{conflset}_{acnet}(M, t) = \text{conflset}_{acnet}(M', t)$

□

Next we show that cofusion-freeness holds for any occurrence net.

Proposition 3.4.3. Each occurrence net is confusion-free.

Proof. It follows from the structure and Definition 3.2.10. □

As it was already mentioned, the probability of concurrent transitions should not be changed by the order in which they were executed; otherwise, assigning correct probability to a scenario is not possible. As we have seen above, this can fail due to confusion. Therefore, excluding confusion is crucial. Verification of confusion-freeness dynamically has high complexity. However, there are classes of nets which exclude confusion by imposing structural restrictions. One of such classes are *free-choice nets* [41]. For instance, [129] introduced confusion-free probabilistic models where free-choice nets are only considered. This, however, results in an excessively limited framework.

Several contributions in the literature have been published on handling confusion. We provide a review of some techniques of removing confusion in standard Petri nets in Section 2.4.2.

In the remainder of this chapter, we develop two novel approaches of removing confusion in well-formed acyclic nets following the work in [26]. More precisely, a confused acyclic net is encoded into another net that is confusion-free. For localising decisions made, the conflict transitions are grouped into *maximal* clusters, which are equivalence classes of conflict relation. The encoding involves generating negative places. However, as the models that we consider in this chapter are acyclic nets that are well-formed, such places can hold only one token. The approaches of removing confusion are explained in more details in the next section.

3.5 Cluster-acyclic nets

The approach of removing confusion is based on identifying the choice points by applying the concept of *clusters*.

Definition 3.5.1 (cluster)**.** Let $acnet = (P, T, F)$ be a well-formed acyclic net.

- A *cluster* is a non-empty set $\kappa \subseteq T$ such that $\kappa \times \kappa \subseteq (\#_0)^*$ and if ${}^\bullet t \subseteq {}^\bullet\kappa$, then $t \in \kappa$, for every $t \in T$.
- A *maximal cluster* is a cluster $\kappa \subseteq T$ such that there is no cluster κ' such that $\kappa \subset \kappa'$.
- $\sqsubset$ is a relation on the clusters such that $\kappa \sqsubset \kappa'$ if $\text{caused}(\kappa) \cap {}^\bullet\kappa' \neq \varnothing$, for all clusters κ and κ'.

The set of all clusters is denoted by $\text{clusters}(acnet)$, and the set of all maximal clusters is denoted by $\text{maxclusters}(acnet)$. ◇

A maximal cluster is an equivalence class of the relation $(\#_0)^*$, and so the set of maximal clusters partitions the set of all transitions.

Below we will consider a subclass of acyclic nets where the ordering $\sqsubset$ is a *strict partial order* on the set maximal of clusters.

Definition 3.5.2 (cluster-acyclic net)**.** A well-formed backward deterministic acyclic net is *cluster-acyclic* if the relation $\sqsubset$ on its maximal clusters is a strict partial order. ◇

Note that cluster-acyclic nets include all free-choice well-formed backward deterministic acyclic nets as well as all extended free-choice well-formed acyclic nets.

From now on, we generally assume that every cluster-acyclic net is well-formed and backward deterministic.

Definition 3.5.3 (transaction)**.** Let κ be a cluster of an acyclic net, and $M \subseteq {}^\bullet\kappa$. A *transaction* of κ with M is a maximal step θ included in κ such that ${}^\bullet\theta \subseteq M$, and there is no step θ' included in κ such that ${}^\bullet\theta' \subseteq M$ and $\theta \subset \theta'$.

The set of all transactions of κ with M is denoted by $\mathrm{trans}(\kappa, M)$. ◇

Intuitively, a transaction is one of possible ways of executing the subnet induced by the cluster. Note that we have:

$$\begin{aligned}\mathrm{trans}(\kappa, M) \;=\; \{\theta \in \mathrm{steps}(\mathrm{subnet}_{acnet}(\kappa)) \mid {}^\bullet\theta \subseteq M \wedge \\ (\nexists \theta' \in \mathrm{steps}(\mathrm{subnet}_{acnet}(\kappa)) : {}^\bullet\theta' \subseteq M \wedge \theta \subset \theta'\}.\end{aligned}$$

Proposition 3.5.1. Let κ be a cluster of a cluster-acyclic net $acnet = (P, T, F)$. Then

$$\bigcup \mathrm{trans}(\kappa, {}^\bullet\kappa) = \kappa.$$

Proof. It follows directly from the definitions. □

Proposition 3.5.2. Let κ be a cluster of a cluster-acyclic net $acnet = (P, T, F)$.

1. $(\kappa \times \kappa) \cap F^+ = \varnothing$.
2. $\mathrm{caused}(\kappa) = \bigcup\{\mathrm{caused}(\theta) \mid \theta \in \mathrm{trans}(\kappa, {}^\bullet\kappa)\}$.
3. $\mathrm{trans}(\kappa, {}^\bullet\kappa) \subseteq \mathrm{maxsseq}(\mathrm{subnet}_{acnet}(\kappa))$.
4. $\mathrm{scenario}_{\mathrm{subnet}_{acnet}(\kappa)}(\mathrm{trans}(\kappa, {}^\bullet\kappa)) = \mathrm{maxscenarios}(\mathrm{subnet}_{acnet}(\kappa))$.

Proof.

1. It follows from Definitions 3.5.1 and Definition 3.5.2 and the fact that there is a cluster $\kappa' \in \mathrm{maxclusters}(acnet)$ such that $\kappa \subseteq \kappa'$.
2. It follows from Proposition 3.5.1.
3. From Definition 3.5.3, we have

$$\begin{aligned}\mathrm{trans}(\kappa, {}^\bullet\kappa) \;=\; \{\theta \in \mathrm{steps}(\mathrm{subnet}_{acnet}(\kappa)) \mid \\ \nexists \theta' \in \mathrm{steps}(\mathrm{subnet}_{acnet}(\kappa))) : \theta \subset \theta'\},\end{aligned}$$

which means that each transaction θ is a maximal step for the subnet induced by κ. Also, from Definition 3.2.7(3), we have

$$\begin{aligned}\text{maxsseq}(\text{subnet}_{acnet}(\kappa)) \;=\; \{\sigma \in \text{sseq}(\text{subnet}_{acnet}(\kappa)) \mid \\ \nexists U : \sigma U \in \text{sseq}(\text{subnet}_{acnet}(\kappa))\}.\end{aligned}$$

Then, each transaction $\theta \in \text{trans}(\kappa, {}^\bullet\kappa)$ is one way of executing the subnet induced by κ and included also in $\text{maxsseq}(\text{subnet}_{acnet}(\kappa))$. Hence, the following holds:

$$\text{trans}(\kappa, {}^\bullet\kappa) \subseteq \text{maxsseq}(\text{subnet}_{acnet}(\kappa)).$$

4. It follows from the result of Proposition 14 (2) in [8] and the above result since $\text{trans}(\kappa, {}^\bullet\kappa) \subseteq \text{maxsseq}(\text{subnet}_{acnet}(\kappa))$. Hence, the following holds:

$$\text{scenario}_{\text{subnet}_{acnet}(\kappa)}(\text{trans}(\kappa, {}^\bullet\kappa)) = \text{maxscenarios}(\text{subnet}_{acnet}(\kappa)).$$

□

Since the transactions of a cluster are maximal step sequences generating all maximal scenarios of the subnet induced by the cluster, one can say that the *behaviour* of a cluster is described by its transactions.

Example 13. Consider again the acyclic net in Figure 3.7. Figure 3.8 (using dotted boxes) shows its two maximal clusters, $\kappa_1 = \{b, d\}$ and $\kappa_2 = \{a, c\}$, together with their presets. The transactions of these clusters and markings are:

$$\begin{aligned}\text{trans}(\kappa_1, \{p_1\}) &= \{\{b\}, \{d\}\} \\ \text{trans}(\kappa_2, \{p_2, p_3\}) &= \{\{a\}, \{c\}\} \\ \text{trans}(\kappa_2, \{p_2\}) &= \{\{a\}\} \\ \text{trans}(\kappa_2, \{p_3\}) &= \varnothing.\end{aligned}$$

We have $\kappa_1 \sqsubset \kappa_2$ and $\kappa_2 \not\sqsubset \kappa_1$ as well as $\kappa_1 \not\sqsubset \kappa_1$ and $\kappa_2 \not\sqsubset \kappa_2$. Hence this acyclic net is clearly cluster-acyclic. ◇

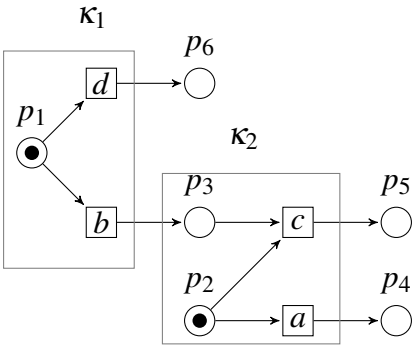


Fig. 3.8 Cluster-acyclic for the acyclic net in Figure 3.7.

3.6 Removing confusion from cluster-acyclic net

3.6.1 Approach A: based on maximal clusters and markings

In this section, $acnet = (P,T,F)$ is a fixed cluster-acyclic net.

The encoding of a confused cluster-acyclic net *acnet* into an confusion-free acyclic net necessitates generating *negative* places to capture the additional causality between the concurrent transitions belonging to the partially ordered clusters in the original *acnet*. For a place $p \in P$, its negative image is denoted by $\overline{p}$. In the executions of *acnet*, once $\overline{p}$ is marked the regular place p will never be marked. Moreover, the original transitions are replaced by transitions representing transactions associated with maximal clusters κ and markings $M \subseteq {}^\bullet\kappa$. We will proceed as follows:

- All places of *acnet* are retained. In addition, for each place $p \in P$, a negative place $\overline{p}$ is created.
- For each maximal cluster κ with marking $M \subseteq {}^\bullet\kappa$, and all the transactions θ in $\mathrm{trans}(\kappa,M)$, a new transition $t_{\kappa,\theta,M}$ is created. Its preset is M together with the negative versions of all places in ${}^\bullet\kappa \setminus M$. Its postset are all the places in the postset of θ and the negative versions of places caused by κ which are not caused by θ.
- Negative places with empty post-sets, which have no influence on the behaviour, are removed.

The following definition provides full details of the encoding.

Definition 3.6.1 (encoding cluster-acyclic net)**.** The *confusion-free encoding* of a cluster-acyclic net $acnet = (P,T,F)$ is an acyclic $\mathrm{confreeA}(acnet) = (P',T',F')$ constructed in the following stages:

- $P'' \triangleq P \cup \overline{P}$.
- $T' \triangleq \{t_{\kappa,\theta,M} \mid \kappa \in \text{maxclusters}(acnet) \wedge M \subseteq {}^\bullet\kappa \wedge \theta \in \text{trans}(\kappa, M)\}$.
- ${}^\bullet t \triangleq M \cup \overline{{}^\bullet\kappa \setminus M}$ and $t^\bullet \triangleq \theta^\bullet \cup \overline{\text{caused}(\kappa \setminus \theta) \cap P}$, for every $t = t_{\kappa,\theta,M} \in T'$.
- All negative places $\overline{p} \in P'$ such that $\overline{p}^\bullet = \varnothing$ are deleted. Then, $P' \triangleq P'' \setminus \{\overline{p} \in P'' \mid \overline{p}^\bullet = \varnothing\}$. ◇

We first verify that the result of the above encoding is indeed an acyclic net.

Proposition 3.6.1. Let *acnet*′ be as in Definition 3.6.1. Then *acnet*′ is an acyclic net.

Proof. It follows directly from Definition 3.6.1, and the fact that *acnet* is cluster-acyclic. □

The executions of the acyclic net constructed in Definition 3.6.1 (and other nets with negative places constructed later) are as follows:

> *The initial marking of acnet*′ *in Definition 3.6.1 is* $M^{init}_{acnet'} = M^{init}_{acnet} = P$ *with all the execution rules and notations being the same as for acnet.*

We will now formulate and prove a number of important properties of the encoding described in Definition 3.6.1. First, however, we show that if *acnet* is an extended free-choice net, then *acnet*′ is basically the same as *acnet*.

Theorem 3.6.1. Let *acnet* and *acnet*′ be as in Definition 3.6.1. If *acnet* is an extended free-choice net, then *acnet*′ and *acnet* are isomorphic nets.

Proof. If *acnet* is extended free-choice and $t_{\kappa,\theta,M} \in T'$, then ${}^\bullet\kappa = {}^\bullet\theta = M$ and $\theta = \{u\}$, for some $u \in T$. This also means that before the last part of Definition 3.6.1, we have $\overline{p}^\bullet = \varnothing$, for every $\overline{p} \in \overline{P}$. It therefore follows that all the negative places are removed in the last stage of Definition 3.6.1. Hence, in turn, the presets of $t_{\kappa,\theta,M}$ and u are the same, and the postsets of $t_{\kappa,\theta,M}$ and u are the same. As a result, *acnet* and *acnet*′ are isomorphic acyclic nets. □

The next result gathers together a number of structural properties of the net constructed in the last definition.

In this, and subsequent results, we link the steps executed by *acnet*′ with the steps executed by *acnet*, by using the following notation, for every set of transitions U of confreeA$(acnet)$:

$$T_U \triangleq \bigcup_{t_{\kappa,\theta,M} \in U} \theta\,.$$

Proposition 3.6.2. Let *acnet* and *acnet*$'$ be as in Definition 3.6.1. Moreover, let p be a place of *acnet*, and $w = t_{\kappa,\theta,M}$ and $u = t_{\kappa',\theta',M'}$ be transitions of *acnet*$'$.

1. If $p \in {}^\bullet\kappa$, then either $p \in {}^\bullet w$ or $\overline{p} \in {}^\bullet w$.
2. If $w\#_0 u$ then $\kappa = \kappa'$.
3. If $\kappa = \kappa'$, then either $w\#_0 u$, or $M \cap M' = \varnothing$, $M \cup M' = {}^\bullet\kappa$, ${}^\bullet w \cap \overline{P} = \overline{M'}$ and ${}^\bullet u \cap \overline{P} = \overline{M}$.
4. ${}^\bullet\theta \subseteq {}^\bullet w$ and $\theta^\bullet = w^\bullet \cap P$.
5. If U is step of *acnet*$'$, then T_U is a step of *acnet* such that ${}^\bullet T_U \subseteq {}^\bullet U$ and $T_U^\bullet = U^\bullet \cap P$.

Proof. It follows directly from Definition 3.6.1. □

The first part of the next result demonstrates that the executions of *acnet*$'$ have direct representations in the executions of *acnet*. The second part explains why $\overline{p}$ can be seen as a 'negative' version of p.

Proposition 3.6.3. Let *acnet* and *acnet*$'$ be as in Definition 3.6.1. Moreover, let

$$(M^{init}_{acnet'}) = M_0 U_1 M_1 \ldots M_{k-1} U_k M_k$$

be a mixed step sequence of *acnet*$'$. Then:

1. There is a marking M of *acnet* such that $M^{init}_{acnet}[T_{U_1} \ldots T_{U_k}\rangle_{acnet} M$ and $M_k \cap P \subseteq M$.
2. For every place p of *acnet*, $\overline{p} \in M_0 \cup \cdots \cup M_k$ implies $p \notin M_0 \cup \cdots \cup M_k$.

Proof.

1. By induction on the length of the mixed step sequence, as follows:

   - Base case ($k = 0$): It holds as *acnet* generates the empty step sequence and $M^{init}_{acnet'} \subseteq M^{init}_{acnet}$.
   - Inductive case ($k+1$): Let

     $$M^{init}_{acnet'}[U_1 \ldots U_k\rangle_{acnet'} M_k[U_{k+1}\rangle_{acnet'} M_{k+1}.$$

     By inductive hypothesis, there is a making M of *acnet* such that

     $$M^{init}_{acnet}[T_{U_1} \ldots T_{U_k}\rangle_{acnet} M$$

and $M_k \cap P \subseteq M$. Moreover, by Proposition 3.6.2(5), $T_{U_{k+1}}$ is a step of *acnet* such that ${}^\bullet T_{U_{k+1}} \subseteq {}^\bullet U_{k+1}$. Hence $T_{U_{k+1}}$ is enabled at M, and so there is a marking M' of *acnet* such that

$$M^{init}_{acnet}[T_{U_1} \ldots T_{U_k}\rangle_{acnet} M[T_{U_{k+1}}\rangle_{acnet} M'.$$

In addition, by Proposition 3.6.2(5), $T^\bullet_{U_{k+1}} = U^\bullet_{k+1} \cap P$. As we also have $M_k \cap P \subseteq M$ and ${}^\bullet T_{U_{k+1}} \subseteq {}^\bullet U_{k+1}$, it follows that $M_{k+1} \cap P \subseteq M'$.

2. It follows from the construction in Definition 3.6.1, and the fact that the acyclic net *acnet* is backward-deterministic.

□

As an immediate corollary of Proposition 3.6.3(1), we obtain that all executions of *acnet'* can be regarded as the executions of *acnet*.

Theorem 3.6.2. Let *acnet* and *acnet'* be as in Definition 3.6.1. Then

$$\{T_{U_1} \ldots T_{U_k} \mid U_1 \ldots U_k \in \mathrm{sseq}(acnet')\} \subseteq \mathrm{sseq}(acnet)\ .$$

The reverse inclusion does not hold in general. The reason is that transitions of *acnet'* represent maximal steps (transactions). As a consequence, a result which can be seen as a converse of Theorem 3.6.2, will be formulated in a different way (and will apply to maximal step sequences).

The essence of the next result is that the net constructed in Definition 3.6.1 exhibits the extended free-choice property in its behaviours.

Proposition 3.6.4. Let *acnet* and *acnet'* be as in Definition 3.6.1. Moreover, let M_0 be a reachable marking of *acnet'* and

$$w = t_{\kappa,\theta,M} \in \mathrm{enabled}_{acnet'}(M_0) \quad \text{and} \quad u = t_{\kappa',\theta',M'} \in \mathrm{enabled}_{acnet'}(M_0).$$

Then $w \#_0 u$ implies ${}^\bullet w = {}^\bullet u$.

Proof. We consider two cases:

- First case: $\kappa \neq \kappa'$. Then, the transactions θ and θ' are derived from two different clusters. Hence, their corresponding transitions as well as presets are disjoint. Hence, ${}^\bullet w \cap {}^\bullet u = \varnothing$, which produces a contradiction with $w \#_0 u$.

- Second case: $\kappa = \kappa'$. Then, the transactions are derived from the same cluster. Suppose that $M \neq M'$ and (without loss of generality) $p \in M \setminus M'$. Then, by ${}^\bullet u \cap P = M'$, $p \notin {}^\bullet u$. Hence, by Definition 3.6.1, $\overline{p} \in {}^\bullet u$. Moreover, by $u \in \text{enabled}_{acnet'}(M_0)$ and Proposition 3.6.3, $p \notin M_0$. This produces a contradiction with ${}^\bullet w \cap P = M$ and $w \in \text{enabled}_{acnet'}(M_0)$. Hence $M = M'$, and so ${}^\bullet w = {}^\bullet u$.

□

An immediate corollary of the last result is that in the net constructed in Definition 3.6.1, a transition has always the same conflict set in all the markings in which it is enabled.

Proposition 3.6.5. Let *acnet'* be as in Definition 3.6.1. Moreover, let M and M' be reachable markings of *acnet'*, and $t \in \text{enabled}_{acnet'}(M) \cap \text{enabled}_{acnet'}(M')$. Then

$$\text{conflset}_{acnet'}(M,t) = \text{conflset}_{acnet'}(M',t).$$

Proof. It follows directly from Proposition 3.4.2. □

The above result has far reaching implications as it essentially means that, in the net constructed in Definition 3.6.1, one can calculate transitions probabilities statically.

Next, we show that the constructed acyclic nets are confusion-free, and their behaviour is closely related to the original cluster-acyclic nets.

Theorem 3.6.3. Let *acnet'* be as in Definition 3.6.1. Then *acnet'* is a confusion-free acyclic net.

Proof. We observe that *acnet'* is acyclic since *acnet* is cluster-acyclic. Moreover, it follows directly from Propositions 3.4.1 and 3.6.5 that *acnet'* is confusion-free. □

A cluster-acyclic net *acnet* and its confusion-free version $acnet' = \text{confreeA}(acnet)$ have closely related behaviours. The paper [112] explored several ways (structurally and dynamically) to define how two systems are similar to each other. Dynamically, one technique of showing that two systems have equivalent behaviours concerns with transitions not with the places. Our approach of proving that behaviours are closely related also concentrates on transitions that induce scenarios. Note that some executions of scenario in the original *acnet* are not preserved in *acnet'* (see Examples 14 and Example 17 as well as Figures 3.8, Figure 3.9 and Figure 3.11 Pages 48 - 51).

In our case, we have already shown in Theorem 3.6.2 that the executions of *acnet'* correspond to executions of *acnet*. The next result can be seen as a converse of this as it

shows that each maximal scenario of *acnet* corresponds to at least one maximal execution of *acnet*$'$. Hence, the executions of a confusion-free acyclic net *acnet*$'$ can be used to assign probabilities to the maximal scenarios of the acyclic net *acnet*.

Theorem 3.6.4. Let *acnet* and *acnet*$'$ be as in Definition 3.6.1. Then, for every maximal scenario $ocnet \in \text{maxscenarios}(acnet)$, there is a maximal step sequence $U_1 \dots U_m$ of *acnet*$'$ such that $T_{U_1} \dots T_{U_m}$ is a maximal step sequence of *ocnet*.

Proof. For every $\kappa \in \text{maxclusters}(acnet)$, let $T_\kappa = \kappa \cap T_{ocnet}$. Moreover, let

$$T_K = \{\kappa \in \text{maxclusters}(acnet) \mid T_\kappa \neq \varnothing\}.$$

Due to the cluster-acyclicity of *acnet*, there is an enumeration $\kappa_1, \dots, \kappa_m$ of the clusters in T_K such that if $\kappa_i \sqsubset \kappa_j$ then $i < j$. This and the definition of $\sqsubset$ means that $T_{\kappa_1} \dots T_{\kappa_m}$ is a step sequence of *ocnet*. Let $M_0 T_{\kappa_1} M_1 \dots M_{m-1} T_{\kappa_m} M_m$ be the corresponding mixed step sequence of *ocnet*. For every $1 \leq i \leq m$, we define:

$$M'_i = M_{i-1} \cap {}^\bullet\kappa_i, \quad t_i = t_{\kappa_i, T_{\kappa_i}, M'_i}, \text{ and } U_i = \{t_i\}.$$

We observe that $T_{\kappa_i} \in \text{trans}(\kappa_i, M'_i)$ since *ocnet* is maximal and $(T_{\kappa_i} \cup \dots \cup T_{\kappa_m})^\bullet \cap {}^\bullet T_{\kappa_i} = \varnothing$. Hence t_i is a transition of *acnet*$'$, and U_i is a step of *acnet*$'$. We then observe that, for every $p \in {}^\bullet\kappa_i \setminus M'_i$, there must be $u \in T_{\kappa_1} \cup \dots T_{\kappa_{i-1}}$ such that $p \# u$, since *ocnet* is maximal and backward deterministic and, moreover, we have $p \notin {}^\bullet(T_{\kappa_1} \cup \dots \cup T_{\kappa_{i-1}}) \cup u \in T_{\kappa_1} \cup \dots T_{\kappa_{i-1}}$.

It then follows that $U_1 \dots U_m$ is a step sequence of *acnet*$'$ such that $T_{U_1} \dots T_{U_m} = T_{\kappa_1} \dots T_{\kappa_m}$ is a maximal step sequence of *ocnet*. This and Theorem 3.6.2 also means that $U_1 \dots U_m$ is maximal. □

Example 14. Figure 3.9 shows the result of the encoding described in Definition 3.6.1 for the confused cluster-acyclic net in Figure 3.8. The transitions have been created before the last stage of the construction (i.e., the deletion of negative places with empty postsets) in the following way (see also the transactions in Example 13).

- For the transactions associated with $\kappa_1 = \{b, d\}$ and marking $M = \{p_1\}$, transitions $t_{\kappa_1, \{b\}, \{p_1\}}$ and $t_{\kappa_1, \{d\}, \{p_1\}}$ are created. Their common preset is

$$\{p_1\} \cup \overline{\{p_1\} \setminus \{p_1\}} = \{p_1\}.$$

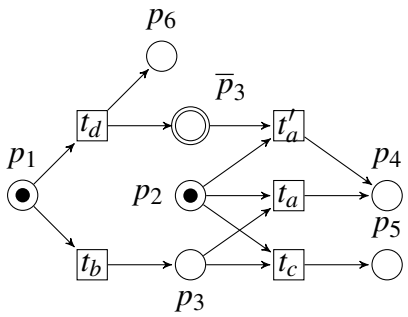


Fig. 3.9 Encoding the cluster-acyclic net in Figure 3.8 into a confusion-free acyclic net with negative place, where: $t_b = t_{\kappa_1,\{b\},\{p_1\}}$, $t_d = t_{\kappa_1,\{d\},\{p_1\}}$, $t_a = t_{\kappa_2,\{a\},\{p_2,p_3\}}$, $t_c = t_{\kappa_2,\{c\},\{p_2,p_3\}}$, and $t'_a = t_{\kappa_2,\{a\},\{p_2\}}$.

Their postsets are, respectively,

$$b^\bullet \cup \overline{\text{caused}(\kappa_1 \setminus \{b\}) \cap P} = \{p_3, \overline{p}_6\} \text{ and } d^\bullet \cup \overline{\text{caused}(\kappa_1 \setminus \{d\}) \cap P} = \{p_6, \overline{p}_3, \overline{p}_5\}.$$

- For the transactions associated with $\kappa_2 = \{a,c\}$ and marking $M = \{p_2, p_3\}$, transitions $t_{\kappa_2,\{a\},\{p_2,p_3\}}$ and $t_{\kappa_2,\{c\},\{p_2,p_3\}}$ are created. Their common preset is

$$\{p_2, p_3\} \cup \overline{\{p_2, p_3\} \setminus \{p_2, p_3\}} = \{p_2, p_3\}.$$

  Their postsets are, respectively, $\{p_4, \overline{p}_5\}$ and $\{p_5, \overline{p}_4\}$.

- For the transaction associated with $\kappa_2 = \{a,c\}$ and marking $M = \{p_2\}$, transition $t_{\kappa_2,\{a\},\{p_2\}}$ is created. Its preset is

$$\{p_2\} \cup \overline{\{p_2, p_3\} \setminus \{p_2\}} = \{p_2, \overline{p}_3\}$$

  and its postset is $\{p_4, \overline{p}_5\}$.

- No transitions are created for κ_1 with $M = \varnothing$, κ_2 with $M = \varnothing$, and κ_2 with $M = \{p_3\}$, as in each case there are no transactions.

In the last stage of encoding all negative places except for $\overline{p}_3$ are deleted.

Note that in the original *acnet* in Figure 3.8, a is concurrent with d and b; however, their corresponding transitions in Figure 3.9 are causally dependent. This emphasises that the decision about conflict resolution should be made first at κ_1, and then all conflict transitions corresponding to the transactions in κ_2 are enabled together. In other words, a in the original

acnet can be executed before c becomes enabled, which causes confusion in calculating probabilities. In the confusion-free version in Figure 3.9, the execution of the corresponding transition t_a is postponed until the choice between t_b and t_d has been resolved. Despite the fact that there are two transactions t_a and t'_a based on transition a, their presets are different as they are associated, respectively, with

$$\text{trans}(\kappa_2, \{p_2, p_3\}) = \{\{a\}, \{c\}\} \text{ and } \text{trans}(\kappa_2, \{p_2\}) = \{\{a\}\},$$

and they are never executed in the same step sequence as p_3 and $\overline{p}_3$ cannot receive tokens in the same execution (see Proposition 3.6.3(2)).

The behaviour of the constructed confusion-free acyclic net is *closely* linked to the original one. At first, both t_b and t_d are enabled. If t_b is executed, t_a and t_c become enabled together, to avoid the situation in the original net where a can be executed before c being enabled. Executing t_d, on the other hand, produces tokens in places p_6 and $\overline{p}_3$, which enables t'_a. ◇

From the probabilistic analysis point of view, the weights of transitions should not be affected by the encoding result. In fact, the confusion-free version is used for the purpose of probabilities calculation. To illustrate this, let us assume that the transitions of the cluster-acyclic net in Figure 3.8 are assigned weights as in Figure 3.7. Then, the weight for each transition represented by the transaction is mapped to the corresponding transition in the encoding depicted in Figure 3.9. The example below illustrates how probabilities could be derived from the constructed confusion-free acyclic net.

Example 15. Figure 3.7 shows asymmetric confusion where the transitions' weights are as follows: $\omega(b) = 3$, $\omega(d) = 7$, $\omega(a) = 4$ and $\omega(c) = 6$.

According to Example 12, $ocnet_3 = \text{scenario}_{acnet}(\{b, a\})$ has different executions and each has different probability. However, in the confusion-free net in Figure 3.9 a similar scenario is induced by the corresponding transitions. This scenario can be executed only in one way, and its probability is calculated based on the weights associated with the original *acnet*. This means that the corresponding scenario for $\text{scenario}_{acnet}(\{b, a\})$ is $\text{scenario}_{acnet'}(\{t_b, t_a\})$ with one execution $\sigma' = \{t_b\}\{t_a\}$ and its probability is

$$\mathbf{P}_{acnet'}(\sigma') = \frac{\omega(t_b)}{\omega(t_b) + \omega(t_d)} \cdot \frac{\omega(t_a)}{\omega(t_a) + \omega(t_c)} = \frac{3}{10} \cdot \frac{4}{10} = \frac{12}{100}.$$

where $acnet' = \text{confreeA}(acnet)$. Note that the rest of the scenarios and their probabilities are treated similarly. ◇

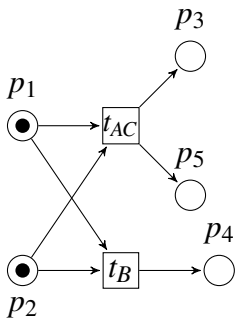


Fig. 3.10 Encoding the acyclic net in Figure 3.6 into a confusion-free acyclic net.

Example 16. Figure 3.10 shows the transformation of the acyclic net with symmetric confusion in Figure 3.6 into a confusion-free one. Let the weights associated with the transitions be the same as in Example 12 on Page 35. Then, the corresponding of scenario$_{acnet}(\{A,C\})$ in the original net is scenario$_{\text{confreeA}(acnet)}(\{t_{AC}\})$ with one execution $\sigma = \{t_{AC}\}$ and its probability calculated as below:

$$\mathbf{P}_{\text{confreeA}(acnet)}(\sigma) = \frac{\omega(t_{AC})}{\omega(t_{AC})+\omega(t_B)} = \frac{7+3}{7+3+3} = \frac{10}{13} = 0.8.$$

Similarly, scenario$_{\text{confreeA}(acnet)}(\{t_B\})$ is the corresponding scenario involving transition $\{B\}$ in the original one and its probability is 0.2. ◇

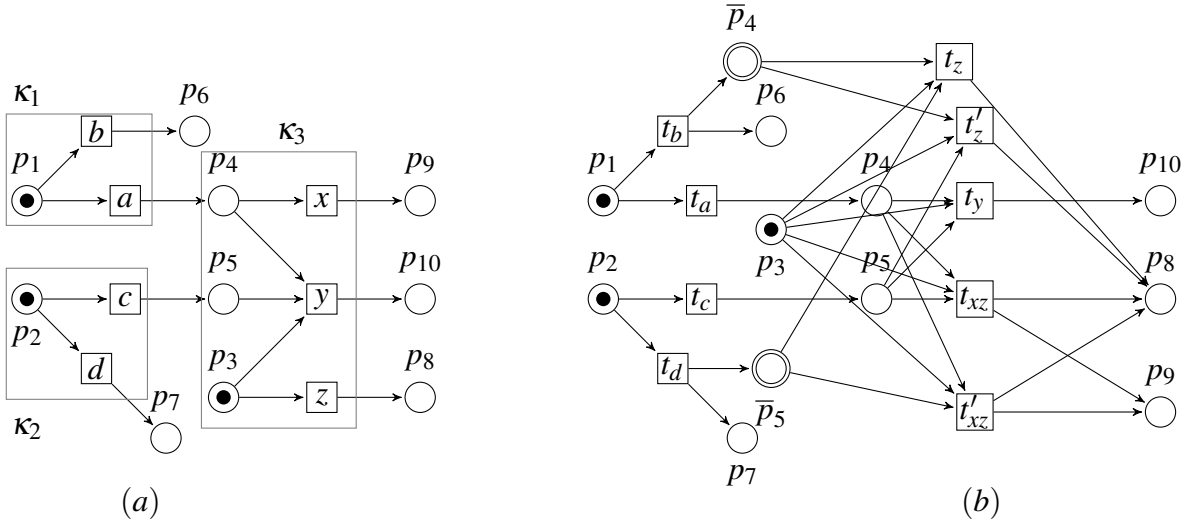


Fig. 3.11 Cluster-acyclic net with confusions (a) and its confusion-free version in (b).

Example 17. Consider the confused cluster-acyclic net in Figure 3.11(a). There are three maximal clusters: $\kappa_1 = \{a,b\}$, $\kappa_2 = \{c,d\}$, and $\kappa_3 = \{x,y,z\}$. Their transactions are

$$\begin{aligned}
&\text{trans}(\kappa_1,\{p_1\}) = \{\{a\},\{b\}\},\\
&\text{trans}(\kappa_2,\{p_2\}) = \{\{c\},\{d\}\},\\
&\text{trans}(\kappa_3,\{p_3,p_4,p_5\}) = \{\{x,z\},\{y\}\},\\
&\text{trans}(\kappa_3,\{p_3,p_4\}) = \{\{x,z\}\},\\
&\text{trans}(\kappa_3,\{p_3,p_5\}) = \{\{z\}\},\\
&\text{trans}(\kappa_3,\{p_4,p_5\}) = \varnothing,\\
&\text{trans}(\kappa_3,\{p_3\}) = \{\{z\}\},\\
&\text{trans}(\kappa_3,\{p_4\}) = \varnothing,\\
&\text{trans}(\kappa_3,\{p_5\}) = \varnothing,
\end{aligned}$$

Its confusion-free version confreeA$(acnet)$ is depicted in Figure 3.11(b).

◇

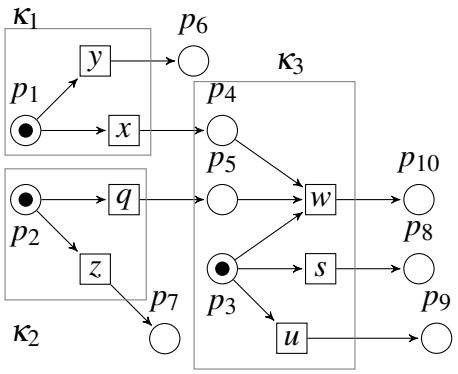


Fig. 3.12 A cluster-acyclic net with confusion.

Example 18. Figure 3.12 shows a cluster-acyclic net with three clusters: $\kappa_1 = \{x,y\}$, $\kappa_2 = \{q,z\}$, and $\kappa_3 = \{s,u,w\}$. Figure 3.13 shows all transactions of κ_3 with all different marking $M \subseteq {}^\bullet\kappa_3$. Also, Figure 3.14 shows all the maximal scenarios that can be produced from the confusion-free encoding of cluster-acyclic net in Figure 3.12 ◇

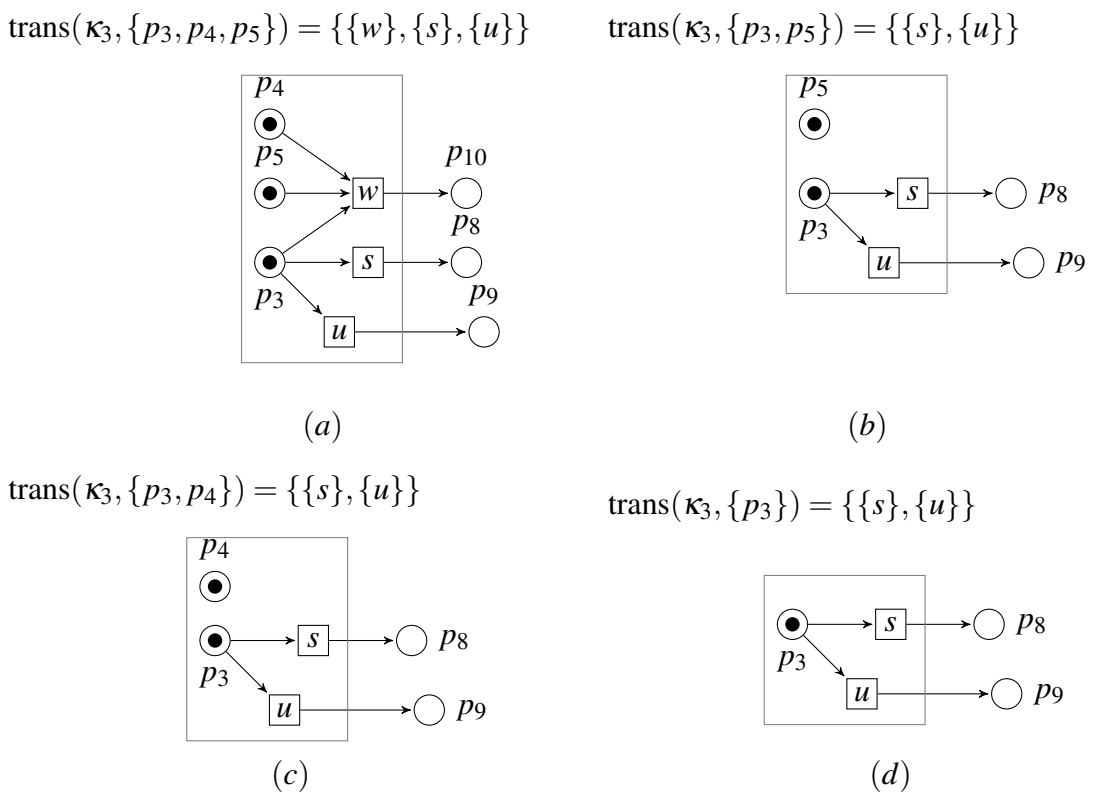


Fig. 3.13 All the transactions of cluster κ_3 with marking $M \subseteq {}^\bullet\kappa_3$ for the cluster-acyclic net in Figure 3.12.

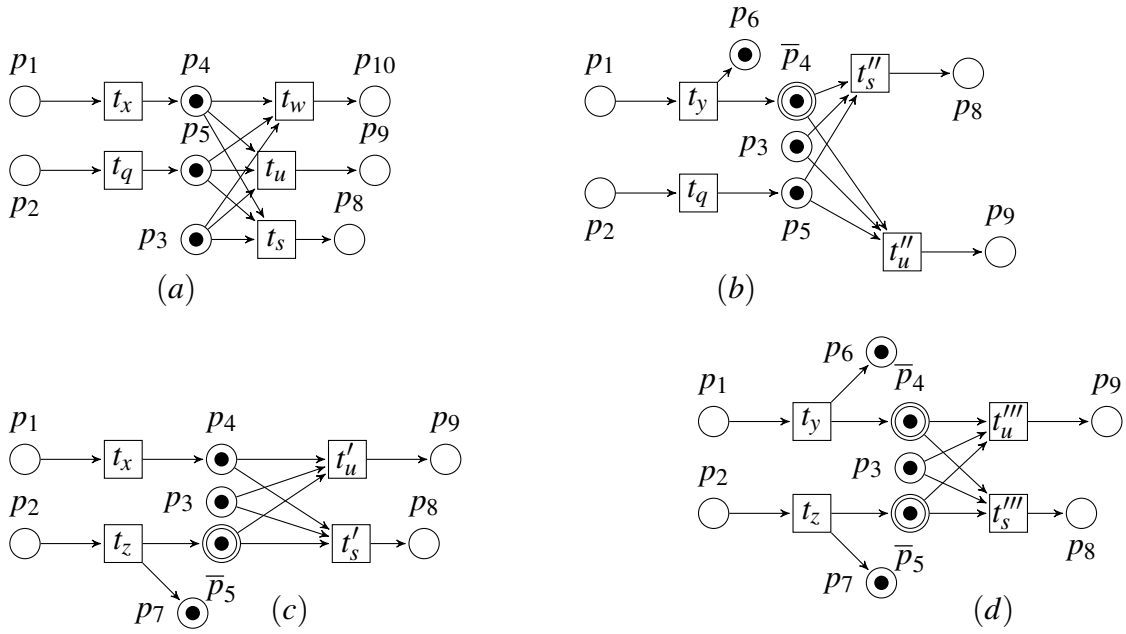


Fig. 3.14 All the maximal scenarios for the encoding of a cluster-acyclic net in Figure 3.12.

3.6.2 Constructing clusters of conflict transitions

The encoding procedure is based on partitioning transitions of a cluster-acyclic net so that a set of conflict transitions is grouped in one cluster. This implies that we are interested in the case where places have multiple output transitions. To group conflict transitions in one cluster, one can first specify all the output transitions $\{t_1, t_2, \ldots, t_n\}$ of a place p. If at least one of such transitions has another place r in its preset, then the local behaviour depends on presence of tokens in both p and r [109]. Hence, it is essential to check iteratively, for each transition $t \in p^\bullet$, whether it has another place r with multiple post-transitions.

Algorithm 2 Calculating maximal clusters of acyclic net

```
1:  function MAXCLUSTER(acnet)
2:      Input: acyclic net acnet.
3:      Output: maxclusters(acnet)
4:      Visited ← ∅, Cluster ← ∅, AllPreset ← ∅, Maxclusters ← ∅
5:      for each place p ∈ P do
6:          if |p•| > 0 and p ∉ Visited then
7:              Cluster = p•
8:              Visited = Visited ∪ {p}
9:              for each t ∈ p• do
10:                 AllPreset = •(p•) \ {p}
11:                 while AllPreset ≠ ∅ do
12:                     for each r ∈ AllPreset do
13:                         Visited = Visited ∪ {r}
14:                         Cluster = Cluster ∪ r•
15:                         AllPreset = (AllPreset ∪ •(r•)) \ Visited
16:                     end for
17:                 end while
18:             end for
19:             Maxclusters = Maxclusters ∪ {Cluster}
20:             Cluster = ∅, AllPreset = ∅
21:         end if
22:     end for
23:     return Maxclusters
24: end function
```

Algorithm 2 partitions the transitions of the acyclic net into maximal clusters. The input is a given acyclic net *acnet*. The first *for* loop iterates non-visited places $p \in P$. Each examined place is added to *Visited* list to keep track of the places that have already been explored. All the output transitions of p are added to the accumulative list *Cluster*. Then, for each transition $t \in p^\bullet$, all its input places are processed so that their output transitions

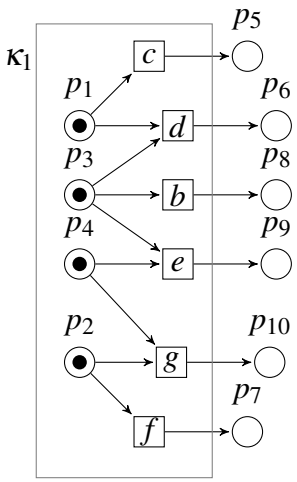


Fig. 3.15 Constructing a cluster for conflict transitions.

are considered as well and added to *Cluster*. The *while* loop stops when all the places of the preset of a maximal cluster are already visited.

Example 19. Consider the acyclic net in Figure 3.15. Each two conflict transitions have different presets. That is, for example, $c\#_0 d$ and ${}^\bullet c \neq {}^\bullet d$. Similarly, $e\#_0 g$ and ${}^\bullet e \neq {}^\bullet g$. If Algorithm 2 is performed, then the set of transitions $\{b,c,d,e,f,g\}$ forms a maximal cluster κ_1. ◇

3.6.3 Constructing all transactions of a cluster

After identifying all maximal clusters, transactions are computed for each one and all potential initial markings. According to Definition 3.5.3 on Page 41, for a given cluster κ and marking $M \subseteq {}^\bullet\kappa$, trans$(\kappa, M)$ is the set of maximal steps for κ with marking M.

Chapter 6 introduces a satisfiability formula that computes the maximal scenarios of an acyclic net. As SAT-solver provides a general solution, considering the net induced by a cluster κ and the set of transitions in κ such that ${}^\bullet t \subseteq M$ for each $t \in \kappa$, all the transactions can be computed using the satisfiability formula.

Another technique to calculate the set of all the transactions of a cluster κ and marking M is to use algorithms for a well-known *All Maximal Independent Sets (AMIS)* graph problem. In this case, κ and M are converted into an undirected graph $G_{\kappa,M} = (V,E)$, defined as follows:

- V is the set of transitions in κ such that ${}^\bullet t \subseteq M$.

- $\{t,v\} \in E$ iff t and v are transitions in direct conflict.

A *MIS* algorithm finds a maximal set of vertices such that each two are not adjacent. That is, $S \subseteq V$ is an *independent set* if its vertices are not connected by an edge. Moreover, S is a *maximal independent set* (or mi-set) if there is no other independent set S' such that $S \subset S'$ [107].

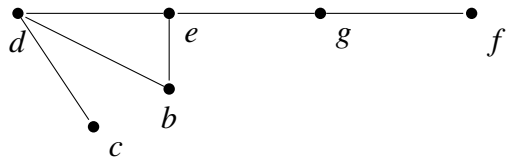


Fig. 3.16 Undirected graph $G_{\kappa_1,\{p_1,p_2,p_3,p_4\}}$ for $\kappa_1 = \{b,c,d,e,f,g\}$ and $M = \{p_1,p_2,p_3,p_4\}$ in Figure 3.15.

Example 20. Figure 3.16 shows the undirected graph representation for $\kappa_1 = \{b,c,d,e,f,g\}$ in Figure 3.15. The nodes are the transitions in κ_1 such ${}^\bullet t \subseteq M = {}^\bullet\kappa_1$, and each two transitions in direct conflict are connected via an edge. ◇

The problem of finding all mi-sets received significant attention, e.g., in [125, 30, 67, 83]. It is also related to the problem of finding all maximal cliques [125]. In [30], the problem of generating all mi-sets in *lexicographical* order is investigated in the case of trees. Note that, in an ordered set,

> *two subsets S and Q are sorted* lexicographically *(or lex-sorted) with S being before Q, if the first element on which they disagree belongs to S [67].*

The basic idea behind the algorithm proposed in [30] is as follows:

- Let $G = (V,E)$ be a tree, where $V = \{1,2,\ldots,n\}$. The vertices are ordered numerically. Let MIS_1 be the first mi-set which starts with the smallest vertex $v = 1$. Then, the next mi-set is generated which starts with vertex $v = 1$, if such a mi-set exists.

- Let assume that $MIS_1 = \{v_1,v_2,\ldots,v_k\}$ starts with vertex v_1. Generating the first mi-set MIS_1 is done by searching vertices V in ordered and never excluding a vertex unless it is connected to a vertex already in MIS_1.

- Let MIS_2 be the next mi-set starting with the same vertex v_1. Before constructing MIS_2, we find *potential* vertices $P_s \subseteq V \setminus MIS_1$ such that each $p \in P_s$ can be used to generate MIS_2 from MIS_1.

- Since MIS_2 starts with v_1, each vertex $p \in P_s$ is not connected with v_1. Also, $p > v_1$, and there is a vertex $v_m \in MIS_1$ such that $v_m < p$ is the smallest vertex in MIS_1 connected with p. Hence, once MIS_1 is constructed, the set of the potential vertices and their corresponding vertices that are already in MIS_1 are derived to produce MIS_2 which is in the form $\{v_1, \ldots, v_{m-1}, p, \ldots\}$ [30].

We follow the algorithms proposed in [30] with some modifications. As the graph $G_{\kappa,M}$ may include cycles, undirected graphs are considered instead of just trees. Moreover, the transitions in the cluster κ are assumed to be integers $V = \{1, 2, \ldots, n\}$.

Algorithm 3 outlines the proposed algorithm of generating all mi-sets in [30] with modifications mentioned above. The input is the graph $G_{\kappa,M}$.

The first mi-set MIS_1 is generated by Algorithm 4 starting with the smallest vertex $v_1 = 1$. At each iteration of the while loop, the smallest vertex $w \in V'$ is chosen and it is excluded along with its neighbours from V'. The set I stores the selected independent vertices.

Based on MIS_1, candidate vertices are computed by the CANDIDATEVER function which takes $G_{\kappa,M}$ and MIS_1 as input. The set *CandidateVer* includes all vertices in V excluding those vertices that have been added to MIS_1 along with the neighbours of v_1. It returns the set of pairs of the candidate vertex p and the corresponding vertex $v_m \in MIS_1$. Then, for each p and v_m in *PairVer*, the set of *potential* mi-set *PotenMIS* is generated after removing v_m and all vertices that are larger than v_m from MIS_1 and adding p instead. Algorithm 4 is used to find the mi-set for the reduced graph $G_{reduced}$ resulting removing all vertices of *PotenMIS* and their connected vertices in $G_{\kappa,M}$ from the graph $G_{\kappa,M}$. The mi-set for the reduced graph $G_{reduced}$ is stored in *NewMIS*. The union of *PotenMIS* and *NewMIS* is the next mi-set *NLMIS*. The set of all *NLMIS* is stored in *AllNextMIS*. The returned result is the next mi-set *NLMIS* of MIS_1.

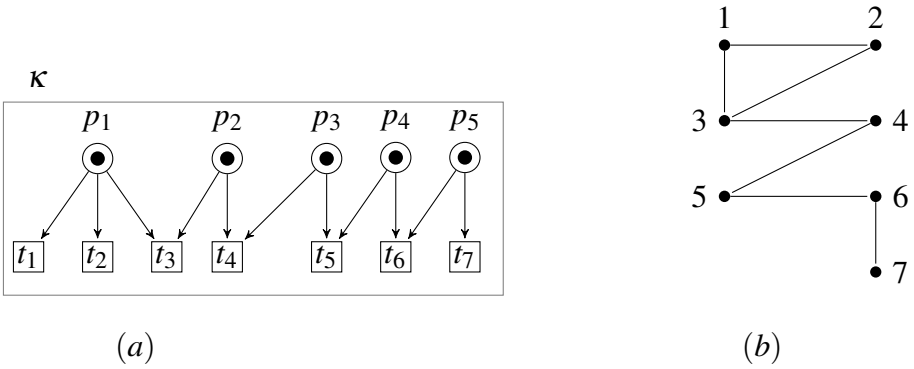
Algorithms 3, 4, and 5 are illustrated in the following examples.

Algorithm 3 Computing all transactions

function TRANSACTIONS(κ, M)
  Input: cluster κ and marking M such that $M \subseteq {}^\bullet\kappa$
  Output: trans(κ, M)

  $v_1 = 1, G_{temp} = G_{\kappa,M} = (V,E)$ where $V = \{t \in \kappa \mid {}^\bullet t \subseteq M\}$ and $E = \{(t,u) \mid t \#_0 u\}$

  while $v_1 \leq v_n$ **do** /* v_n is the largest vertex in G */
    $G = G_{temp}$
    find the first mi-set MIS_1 starting with v_1 using Algorithm 4
    while $MIS_1 \neq \varnothing$ **do**
      $AllMIS = AllMIS \cup \{MIS_1\}$
      find the next mi-set MIS_2 starting with vertex v_1 using Algorithm 5
      $MIS_1 = MIS_2$
    end while
      $v_1 = v_1 + 1$.
    end while
    return $AllMIS$
  end function



(a) (b)

Fig. 3.17 A cluster $\kappa = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ with $M = {}^\bullet\kappa$ (a) and its undirected graph $G_{\kappa,M}$ in Example 21 (b). Note: each i stands for t_i.

Example 21. The graph $G_{\kappa,M} = (V,E)$ in Figure 3.17 represents some cluster and marking as in Algorithm 3. The smallest vertex is 1, so the first mi-set is $MIS_1 = \{1,4,6\}$. The candidate vertices that can be added to some vertices in MIS_1 to generate the next mi-set are:

$$CandidateVer = V \setminus (MIS_1 \cup \{2,3\}) = \{5,7\} \quad \text{(note that } \{2,3\} \text{ are the neighbours of 1).}$$

The smallest vertices in MIS_1 that are connected to 5 and 7 are 4 and 6, respectively. Hence, $PairVer(5,4)$ and $PairVer(7,6)$ are obtained. The potential mi-set $PotenMIS$ is generated for each pair as follows:
for $PairVer(5,4)$, 4 and all the vertices larger than 4 are removed from MIS_1 and 5 is added. So, $PotenMIS = \{1,5\}$.

The reduced graph $G_{reduced}$ is obtained after removing all vertices in $PotenMIS$ and their connected vertices in $G_{\kappa,M}$ from the graph $G_{\kappa,M}$. This step is depicted in Figure 3.18(a), where all removed vertices are in gray circles. That means, 7 is the only remaining vertex in $G_{reduced}$. The new mi-set $NewMIS$ for the graph $G_{reduced}$ is $\{7\}$. Hence the next mi-set added to $AllNextMIS$ is:

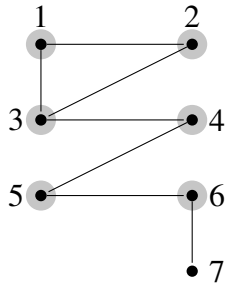
$$NLMIS = PotenMIS \cup NewMIS = \{1,5\} \cup \{7\} = \{1,5,7\}.$$

Similarly, for $PairVer(7,6)$, 6 and all the vertices larger than 6 are removed from MIS_1 and 7 is added. Hence, $PotenMIS = \{1,4,7\}$. In this case, $G_{reduced}$ has no vertices as shown in Figure 3.18(b), which means $NLMIS = PotenMIS = \{1,4,7\}$. The two next mi-sets $NLMISs$ of $MIS_1 = \{1,4,6\}$ are $\{1,5,7\}$ and $\{1,4,7\}$. The latter is the next mi-set of MIS_1. The remaining vertices in $G_{\kappa,M}$ are processed similarly, and all mi-sets are:

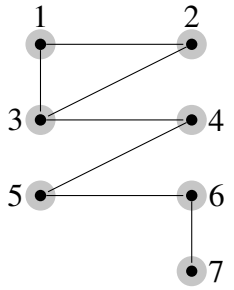
$$\{1,4,6\},\ \{1,4,7\},\ \{1,5,7\},\ \{2,4,6\},\ \{2,4,7\},\ \{2,5,7\},\ \{3,5,7\},\ \{3,6\}.$$

$\diamond$

removing all vertices in $PotenMIS = \{1,5\}$ and all their connected vertices in graph $G_{\kappa,M}$ from $G_{\kappa,M}$



(a)



(b)

removing all vertices in $PotenMIS = \{1,4,7\}$ and all their connected vertices in graph $G_{\kappa,M}$ from $G_{\kappa,M}$

Fig. 3.18 An illustrative example of Algorithm 5 for the graph in Figure 3.17.

Algorithm 4 Computing the first mi-set (MIS)

function MIS($G_{\kappa,M}$)
    Input: $G_{\kappa,M} = (V,E)$
    Output: the first mi-set (MIS) of $G_{\kappa,M}$

    $I = \varnothing,\ V' = V$
    while $V' \neq \varnothing$ **do**
        choose the smallest vertex $w \in V'$
        $I = I \cup \{w\}$
        $V' = V' \setminus (\{w\} \cup N(w))$ /* $N(w)$ is the set of the neighbours of w */
    end while
    return I
end function

Algorithm 5 Computing the next mi-set for MIS_1 (NLMIS)

function NLMIS($G_{\kappa,M}, MIS_1$)
    Input: graph $G_{\kappa,M} = (V,E)$ and the first mi-set $MIS_1 = \{v_1, v_2, \ldots, v_n\}$
    Output: the next mi-set of MIS_1 starting with v_1 if such set exists

    $G_{temp} = G_{\kappa,M}$, $PotenMIS = \varnothing$, $NewMIS = \varnothing$, $AllNextMIS = \varnothing$

    Call Function CANDIDATEVER($G_{\kappa,M}, MIS_1$)
    for each vertex p and the corresponding vertex v_m in $PairVer$ **do**
        $G_{\kappa,M} = G_{temp}$
        $PotenMIS = \{p\} \cup (MIS_1 \setminus (v_m \cup v_q))$ for each $v_q \in MIS_1$ with $q > m$
        remove all vertices in $PotenMIS$ and all their connected vertices in $G_{\kappa,M}$ from $G_{\kappa,M}$
        let $G_{reduced}$ be the new graph.
        compute the mi-set for $G_{reduced}$ using Algorithm 4
        let $NewMIS$ be the returned result
        $NLMIS = NewMIS \cup PotenMIS$
        $AllNextMIS = AllNextMIS \cup \{NLMIS\}$
    end for
    return the next mi-set $NLMIS$ of MIS_1 in $AllNextMIS$
end function

Algorithm 6 Computing the set of candidate vertices and their corresponding vertices in MIS_1

```
1: function CANDIDATEVER( G_{κ,M}, MIS_1 )
2:     Input: G_{κ,M} = (V,E) and the first mi-set MIS_1 = {v_1, v_2, ..., v_n}
3:     Output: the set of candidate vertices and the corresponding vertices in MIS_1
4:
5:     CandidateVer = ∅, PairVer = ∅
6:     CandidateVer = V \ (MIS_1 ∪ N(v_1))      /* N(v_1) is the set of the neighbors of v_1 */
7:
8:     for each vertex p ∈ CandidateVer do
9:         Let v_m be the smallest vertex in MIS_1 connected to p
10:        if p < v_m then
11:            CandidateVer = CandidateVer \ {p}
12:        else
13:            add the pair vertices p and v_m to PairVer(p, v_m)
14:        end if
15:    end for
16:    return PairVer(p, v_m)
17: end function
```

Example 22. If Algorithm 4 is applied to the graph representation of κ_1 in Figure 3.16, then the first mi-set would be $MIS_1 = \{b,c,f\}$ as b is smallest vertex in the graph $G_{\kappa_1,\{p_1,p_2,p_3,p_4\}}$ and all the nodes in MIS_1 are independent. ◇

In Algorithm 3, if the first mi-set which starts with the smallest vertex v_1 does not exist, $MIS_1 = \varnothing$, then the next vertex is chosen which is depicted by $v_1 = v_1 + 1$ in line 18.

In general, generating a mi-set is not complicated as generating all the mi-sets as their number might be exponential.

3.6.4 Cluster-acyclicity and places caused by clusters

Before constructing the confusion-free net confreeA($acnet$), one must ensure that the relation $\sqsubset$ on the maximal clusters of *acnet* is a strict partial order. Algorithm 8 examines whether this property holds, calling the CAUSED function which is listed in Algorithm 7. This method uses the depth-first search technique to traverse an acyclic net starting from the subnet induced by κ to return not only all the places caused by a transition $t \in \kappa$, but also all its post-places. This is illustrated in the example below.

Algorithm 7 Computing places caused by a cluster

```
1:  function CAUSED( Q)
2:      Input: a subset Q of cluster κ.
3:      Output: the set caused(Q) of places caused by Q.
4:
5:      caused = ∅,  stack S = ∅,  visited = ∅
6:
7:      for each  t ∈ Q do
8:          S.push(t•)                    /* the output places of t are added one-by-one */
9:          visited = visited ∪ t•.
10:         while S ≠ ∅ do
11:             temp = S.top()
12:             if (temp is a place) then
13:                 caused = caused ∪ {temp}
14:             end if
15:             S.pop()
16:             for each  v ∈ temp• do
17:                 if v ∉ visited then
18:                     S.push(v)
19:                     visited = visited ∪ {v}
20:                 end if
21:             end for
22:         end while
23:         end for
          return caused
24:     end function
```

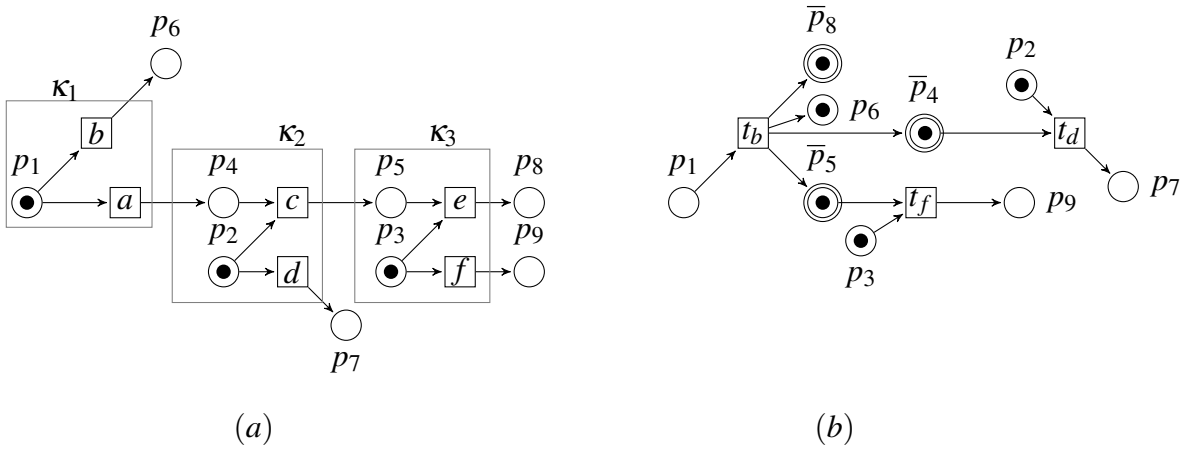


Fig. 3.19 Cluster-acyclic net (a), and generating tokens in marked negative places in caused$(\kappa_1 \setminus \{b\}) \cap P$ after t_b is executed.

Example 23. The cluster-acyclic net in Figure 3.19 has three maximal clusters:

$$\kappa_1 = \{a,b\}, \quad \kappa_2 = \{c,d\} \text{ and } \kappa_3 = \{e,f\}.$$

According to Definition 3.2.8 (7) on Page 25, caused$(\kappa) \cap P$ for each cluster is as follows:

$$\begin{aligned} \text{caused}(\kappa_1) \cap P &= \{p_4, p_5, p_6, p_8\} \\ \text{caused}(\kappa_2) \cap P &= \{p_5, p_7, p_8\} \\ \text{caused}(\kappa_3) \cap P &= \{p_8, p_9\}. \end{aligned}$$

Executing b leads to the absence of token in p_4, which implies eventually that p_5 and p_8 will never be marked. In the construction of confusion-free net, that means $\overline{p}_4, \overline{p}_5$ and $\overline{p}_8$ are marked as soon as the corresponded transaction t_b is executed, which in return enables t_d and t_f immediately as they are the only available ones. Figure 3.19 (b) shows that disabling the transition t_a due to executing t_b is propagated across the net by inserting tokens into negative versions of the places belonging to caused(κ_1). Note that the whole confusion-free net is not presented in Figure 3.19(b), only the relevant part. ◇

Once all the places caused by the clusters are calculated, then one can verify that an acyclic net *acnet* is a cluster-acyclic, as shown in Algorithm 8.

Example 24. The acyclic net depicted in Figure 3.20 is not cluster-acyclic as we have:

$$\text{caused}(\kappa) \cap P = \{p_2, p_3, p_5\} \cap {}^\bullet\kappa' \neq \varnothing \text{ and } \text{caused}(\kappa') \cap P = \{p_2, p_5, p_6\} \cap {}^\bullet\kappa \neq \varnothing.$$

◇

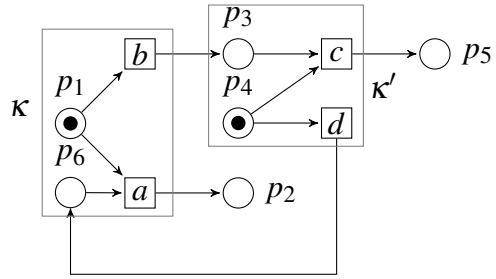


Fig. 3.20 An acyclic net which is not cluster-acyclic.

Algorithm 8 Verifying cluster-acyclicity

```
1:  function BOOLEAN CLUSTERACYCLIC(clusters(acnet))
2:      Input: maxclusters(acnet)
3:      Output: true if acnet is a cluster-acyclic; otherwise false
4:      for each two maximal clusters κ, κ' ∈ maxclusters(acnet) do
5:          if CAUSED(κ) ∩ •κ' ≠ ∅ and
6:              CAUSED(κ') ∩ •κ ≠ ∅ then
7:              return false
8:          end if
9:      end for
10:     return true
11: end function
12:
13:
```

3.6.5 Constructing confusion-free net

The encoding steps are presented in Algorithm 9. The input is a cluster-acyclic *acnet*. For each iteration, a new transition is generated for each transaction θ belongs to trans(κ, M) for each cluster κ. Then, the presets and postsets are added according to Definition 3.6.1.

Compared to the encoding in [26], the proposed encoding is simpler because of the following:

- The contact-free acyclic net resulting from encoding uses the same net model as all acyclic nets (i.e., persistent places and extended markings are not needed).

Algorithm 9 Encoding cluster-acyclic net (Approach A)

1: **function** REMOVECONFUSION(*acnet*)
2:
3: **Input:** cluster-acyclic net $acnet = (P,T,F)$
4: **Output:** confusion-free acyclic net confreeA($acnet$)
5:
6: $P' = P' \cup \overline{p}$, $T' = \varnothing$
7:
8: **for each** $\kappa \in$ maxclusters($acnet$) and $M \subseteq {}^\bullet\kappa$ **do**
9: **for each** $\theta \in$TRANSACTIONS(κ, M) **do**
10: caused =CAUSED$(\kappa \setminus \theta)$
11: $T' = T' \cup \{t_{\kappa,\theta,M}\}$
12: ${}^\bullet t_{\kappa,\theta,M} = M \cup \overline{{}^\bullet\kappa \setminus M}$
13: $t^\bullet_{\kappa,\theta,M} = \theta^\bullet \cup \overline{caused}$
14: **end for**
15: **end for**
16: remove all $\overline{p} \in P'$ such that $\overline{p}^\bullet = \varnothing$
17: **return** (P', T', F')
18: **end function**

- The number of negative places is smaller than in [26]. That because in [26], auxiliary places are generated as their cells are static. Also, the dynamic net is used as an intermediate step of the construction. Hence, a persistent place is added to the preset of a transition t to enable it dynamically. Figure 3.21 shows the confusion-free version for the net in Figure 3.7 according to the construction in [26]. Note that there are five new places created whereas our encoding shown in Figure 3.9 only adds one negative place.

  In general, if we have a cluster-acyclic net with n places, then there can be at most n clusters κ. For each cluster κ, the number of negative places to be added is determined by the set of unmarked places belonging to the preset of κ. More precisely, $|{}^\bullet\kappa \setminus M_{init}|$ negative places are generated. The worst case is when none of the pre-places of κ is initially marked. In this case, a negative place $\overline{p}$ is generated for each place $p \in {}^\bullet\kappa$. On the other hand, the best case is when all the pre-places of κ are initially marked. In this case, no negative paces are generated.

- There is no need to use *dynamic nets* as an intermediate step of the construction.

- We expect that our encoding will be much easier to comprehend and use by practitioners with relatively limited formal methods skills.

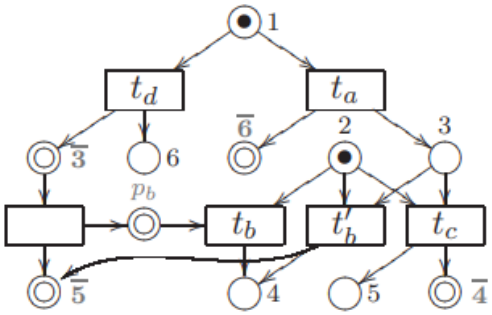


Fig. 3.21 The confusion-free version for the acyclic net in Figure 3.7 according to the encoding steps defined in [26]

Additionally, one of the limitations of [26] is that only the case of backward deterministic nets has been addressed. We will extend the encoding to the unfolding semantics as it is discussed in Section 3.7, where more general cases can be considered, and to communication structured acyclic nets in Chapter 4.

We do not expect the fact that our encoding works for a subclass of cluster-acyclic nets to be limiting in practical applications. This is based on the examples modelling investigations we evaluated, and also on the fact that in case of non-compliance it is always to require an investigator to provide additional information to 'repair' the not, or using other source of information, e.g., timing information associated with places and/or transitions.

3.6.6 Approach B: based on all clusters and borders

Approach *A* introduced in the previous sections to eliminate confusion imposed no restrictions on the structure of cluster-acyclic nets. In Approach *A*, the number of newly generated transitions depends on the marked pre-places of the cluster κ. In other words, all the considered cases of generating new transitions are according to the different combinations of the marking of ${}^\bullet\kappa$. Hence, for instance, if n is the number of pre-places of κ, then the number of all considered cases of creating new transitions is 2^n as each place $p \in {}^\bullet\kappa$ can either be marked or unmarked. As a result, the potential number of all possible new transitions is exponential. In this section, we assume that each cluster-acyclic net is binary synchronised. This restriction reduces the number of instances of the same transition. As a result, the size of the newly generated net is never worse compared to Approach *A*.

In this section, another approach of removing confusion is introduced. It is based on determining sets of places, called *borders*, of clusters. The intention behind borders is to generate negative places in the construction, to serve a similar purpose as in Approach *A*.

Definition 3.6.2 (binary synchronised net)**.** A *binary synchronised net* is a cluster-acyclic net such that each transition has at most two pre-places. ◇

Definition 3.6.3 (borders)**.** Let $acnet = (P,T,F)$ be a cluster-acyclic net and $\kappa \in \text{clusters}(acnet)$. A *border* of κ is a minimal set of places δ such that $({}^\bullet\kappa)^\bullet \setminus \kappa \subseteq \delta^\bullet$.
The set of all the borders of κ is denoted by $\text{borders}(\kappa)$. ◇

A maximal cluster has only the empty border.

Below, we will use the notation $\delta_{p_1\ldots p_k}$ for a border $\{p_1,\ldots,p_k\}$, the notation $\kappa_{t_1\ldots t_m}$ for a cluster $\{t_1,\ldots,t_m\}$, and the notation $\tau_{u_1\ldots u_n}$ for a transaction $\{u_1,\ldots,u_n\}$.

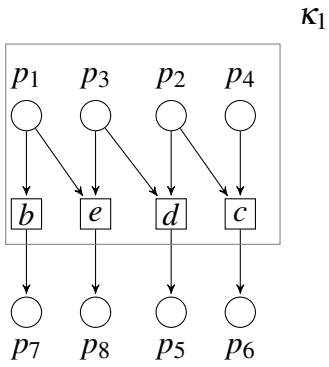


Fig. 3.22 A maximal cluster κ_1 of a binary synchronised acyclic net.

Example 25. For the binary synchronised cluster-acyclic net in Figure 3.22, transitions b,c,d,e form the maximal cluster κ_1, for which $\text{borders}(\kappa_1) = \{\varnothing\}$. A non-maximal cluster $\kappa_b = \{b\}$ also has one border, $\text{borders}(\kappa_b) = \{\delta_{p_3}\}$. ◇

Proposition 3.6.6. $|\text{borders}(\kappa)| = 1$, for every cluster κ of a binary synchronised cluster-acyclic net.

Proof. Clearly, if κ is maximal, then $\text{borders}(\kappa) = \{\varnothing\}$. In general, since we are dealing with a binary synchronised cluster-acyclic net, $\text{borders}(\kappa) = \{\delta\}$, where ${}^\bullet(({}^\bullet\kappa)^\bullet) \setminus {}^\bullet\kappa \subseteq \delta$. □

Definition 3.6.4 (transaction)**.** A *transaction* of a cluster κ of a cluster-acyclic net is a maximal step θ included in κ. The set of all transactions of a cluster κ is denoted by $\text{trans}(\kappa)$. ◇

Example 26. The binary synchronisation acyclic net *acnet* in Figure 3.22 exhibits symmetric confusion. The maximal cluster $\kappa_1 = \kappa_{bdec} = \{b,d,e,c\}$ has the empty border, $\text{borders}(\kappa_1) =$

$\{\varnothing\}$. The transactions included in κ_1 are captured by $\text{trans}(\kappa_1) = \{\tau_{bd}, \tau_{bc}, \tau_{ec}\}$. For other clusters, there are at most one place in their borders. Their borders and transactions are as follows:

$$\begin{array}{lcll}
\kappa_{bed} & : & \text{borders}(\kappa_{bed}) = \{\delta_{p_4}\} & \text{trans}(\kappa_{bed}) = \{\tau_{bd}, \tau_e\} \\
\kappa_{be} & : & \text{borders}(\kappa_{be}) = \{\delta_{p_2}\} & \text{trans}(\kappa_{be}) = \{\tau_b, \tau_e\} \\
\kappa_{cd} & : & \text{borders}(\kappa_{cd}) = \{\delta_{p_1}\} & \text{trans}(\kappa_{cd}) = \{\tau_d, \tau_c\} \\
\kappa_b & : & \text{borders}(\kappa_b) = \{\delta_{p_3}\} & \text{trans}(\kappa_b) = \{\tau_b\} \\
\kappa_c & : & \text{borders}(\kappa_c) = \{\delta_{p_3}\} & \text{trans}(\kappa_c) = \{\tau_c\} \\
\kappa_d & : & \text{borders}(\kappa_d) = \{\delta_{p_1 p_4}\} & \text{trans}(\kappa_d) = \{\tau_d\}
\end{array}$$

$\diamond$

After finding the transactions of a maximal cluster and all its non-maximal clusters with their transactions, a confused binary synchronised cluster-acyclic net *acnet* is encoded into a confusion-free net *acnet*$'$. The steps are as below:

- All places of *acnet* are retained and for each place $p \in \text{borders}(\kappa)$, a negative place $\overline{p}$ is added.
- All transitions in *acnet* are deleted, and for each cluster κ with its maximal step $\theta \in \text{trans}(\kappa)$ a new transition $t_{\kappa,\theta,\delta}$ is generated. Its presets are all the pre-places of κ and the negative places in $\delta \in \text{borders}(\kappa)$. Its postsets are the output places of θ and the places caused by κ excluding those caused by θ.
- All places with empty postsets are deleted.
- All places with empty presets together with their postsets are deleted. This is repeated until no more places with empty presets remain.
- Each negative places with non-singleton postset is split, and individual copies are provided for all transitions in its postset.

Below is the formal definition of producing a confusion-free acyclic net of a confused binary synchronised one.

Definition 3.6.5 (encoding binary synchronisation cluster-acyclic net)**.** The *confusion-free encoding* of a binary synchronisation cluster-acyclic net $acnet = (P, T, F)$ is an acyclic $\text{confreeB}(acnet) = (P', T', F')$ constructed as follows:

- $P'' \triangleq P \cup \overline{P}$.

- $T' \triangleq \{t_{\kappa,\theta,\delta} \mid \kappa \in \text{clusters}(acnet) \wedge \theta \in \text{trans}(\kappa) \wedge \delta \in \text{borders}(\kappa)\}$.
- ${}^\bullet t \triangleq {}^\bullet\kappa \cup \overline{\delta}$ and $t^\bullet \triangleq \theta^\bullet \cup \overline{\text{caused}(\kappa \setminus \theta) \cap P}$, for every $t = t_{\kappa,\theta,\delta} \in T'$.
- Each negative place $\overline{p}$ such that $\overline{p}^\bullet = \{v_1, \ldots, v_k\}$ $(k \geq 2)$ is replaced by negative places $\overline{p}_1, \ldots, \overline{p}_k$ satisfying ${}^\bullet\overline{p}_i = {}^\bullet\overline{p}$ and $\overline{p}_i^\bullet = \{v_i\}$, for every $1 \leq i \leq k$. Let $\overline{P}_{new}$ be the set of the new generated negative places.
- All places with empty postsets are deleted, and all places with empty presets together with their postsets are deleted (this is repeated until no more places with empty presets remain).

  Then, $P' \triangleq P'' \cup \overline{P}_{new} \setminus \{p' \in P' \mid p'^\bullet = \varnothing \text{ or } {}^\bullet p' = \varnothing\}$. ◇

Example 27. Figure 3.23 shows on the left the maximal cluster κ_1 and its sub-clusters. The transitions derived from their transactions according to Definition 3.6.5 are shown on the right.

After deriving all the clusters together with their borders and transactions, a confusion-free acyclic net is generated which is depicted in Figure 3.24. For instance, for the transactions $\text{trans}(\kappa_1) = \{\tau_{bd}, \tau_{bc}, \tau_{ec}\}$ of κ_1, transitions

$$t_{\kappa_1,\tau_{bd},\varnothing} \qquad t_{\kappa_1,\tau_{bc},\varnothing} \qquad t_{\kappa_1,\tau_{ec},\varnothing}$$

are generated. Similarly, for the cluster κ_{bed}, transitions

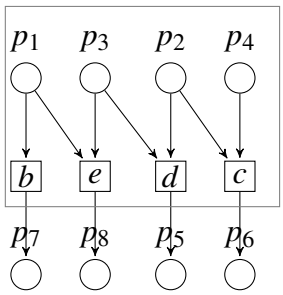
$$t_{\kappa_{bed},\tau_{bd},\delta_{p_4}} \qquad t_{\kappa_{bed},\tau e,\delta p_4}$$

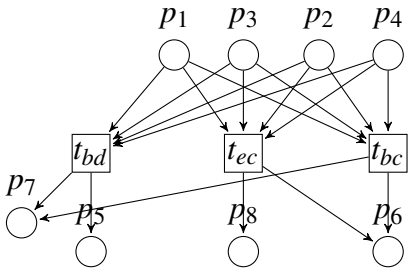
are created. The remaining transitions are created in a similar same way.

Note that $\text{borders}(\kappa_b) = \text{borders}(\kappa_c) = \{\delta_{p_3}\}$ and b, c are not in conflict hence, in the encoding $\overline{p}_3$ is split into $\overline{p}_3$ and $\overline{p}'_3$. Also, it is worth to mention that the non-maximal clusters are always affected by the minimal outside places. For instance, for κ_{be}, κ_c, and κ_b their outside places respectively are $\{p_2, p_4\}$, $\{p_1, p_3\}$, and $\{p_2, p_3, p_4\}$. However, in the construction, the negative versions of the places in their borders are considered only. Except for κ_d where all the outside places are in its border as well. ◇

$\text{trans}(\kappa_{bdec}) = \{\tau_{bd}, \tau_{bc}, \tau_{ec}\}$ and $\text{borders}(\kappa_{bdec}) = \{\varnothing\}$

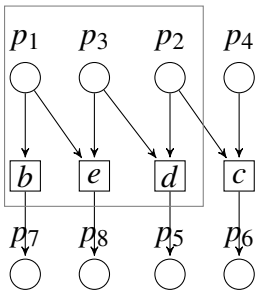
κ_{bdec}

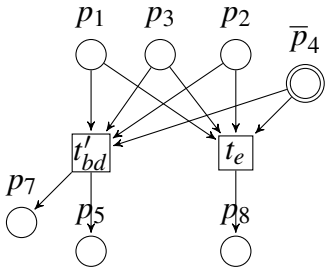




$\text{trans}(\kappa_{bed}) = \{\tau_{bd}, \tau_{e}\}$ and $\text{borders}(\kappa_{bed}) = \{\delta_{p_4}\}$

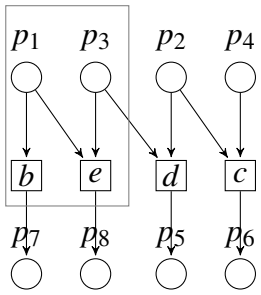
κ_{bed}

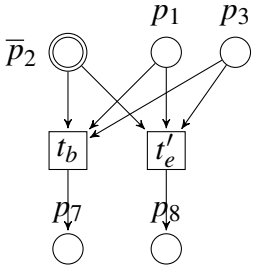




$\text{trans}(\kappa_{be}) = \{\tau_{b}, \tau_{e}\}$ and $\text{borders}(\kappa_{be}) = \{\delta_{p_2}\}$

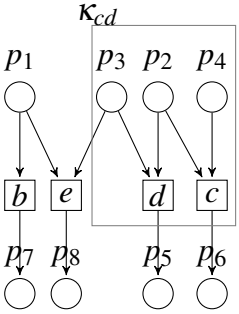
κ_{be}

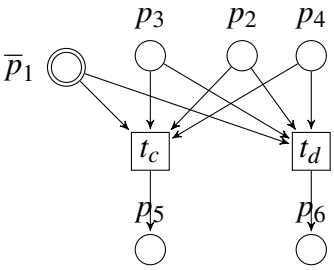




$\text{trans}(\kappa_{cd}) = \{\tau_{c}, \tau_{d}\}$ and $\text{borders}(\kappa_{cd}) = \{\delta_{p_1}\}$

κ_{cd}





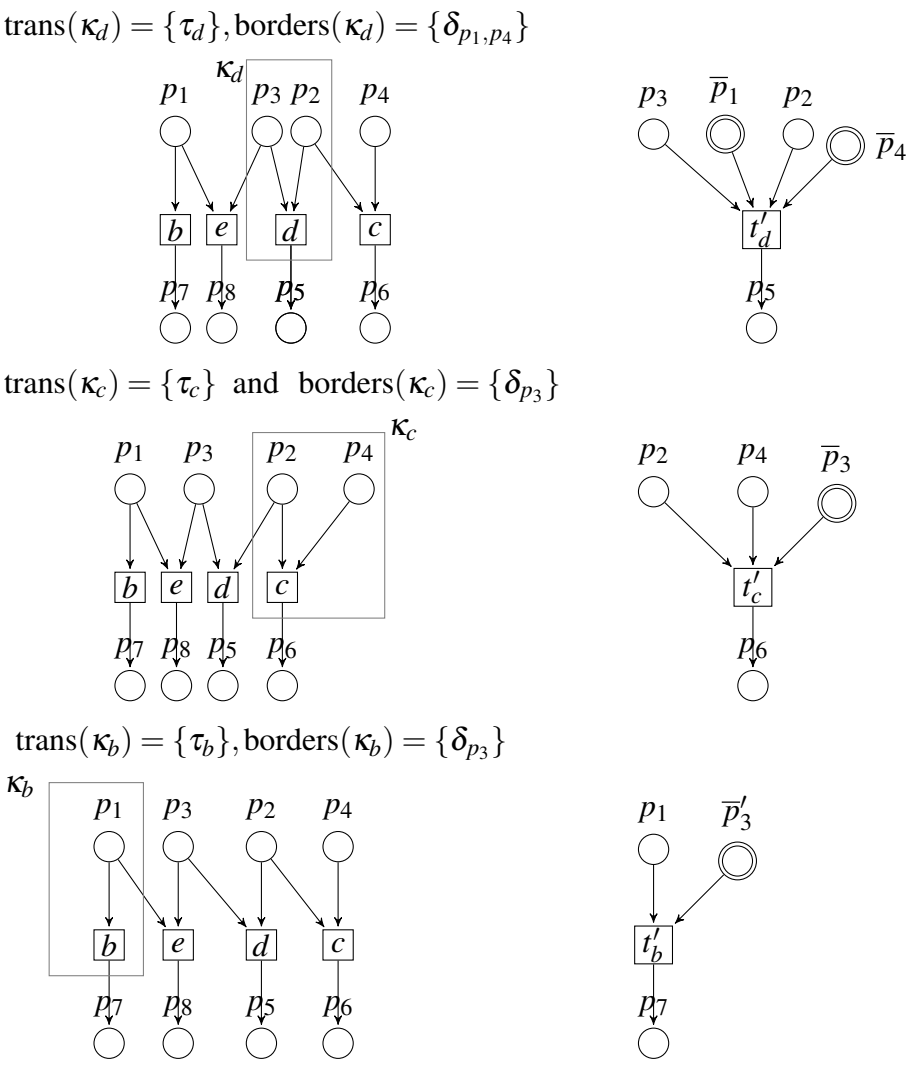


Fig. 3.23 All the sub-clusters and their associated transactions of the binary synchronised acyclic net in Figure 3.22.

Example 28.

Figure 3.25 shows the same maximal cluster κ_1 in Figure 3.22 however with the initial marking $M = \{p_1, r_1, r_2, r_3\}$. Figure 3.26 shows its encoding according to Approach *A* in steps where the transactions of κ_1 are computed based on the marking M (note that the clusters κ, κ', and κ'' are free-choice so in the construction they are not modified, hence we did not show this part). Based on the number of the new generated transitions for κ_1 in Figure 3.24 and Figure 3.26 respectively, it is shown that the size of confusion-free net using the two approaches is almost the same. We hypothesise that Approach *B* introduces less-dependence compared to Approach *A*. That is for example in Figure 3.24 for the transitions t_b, t'_e to be

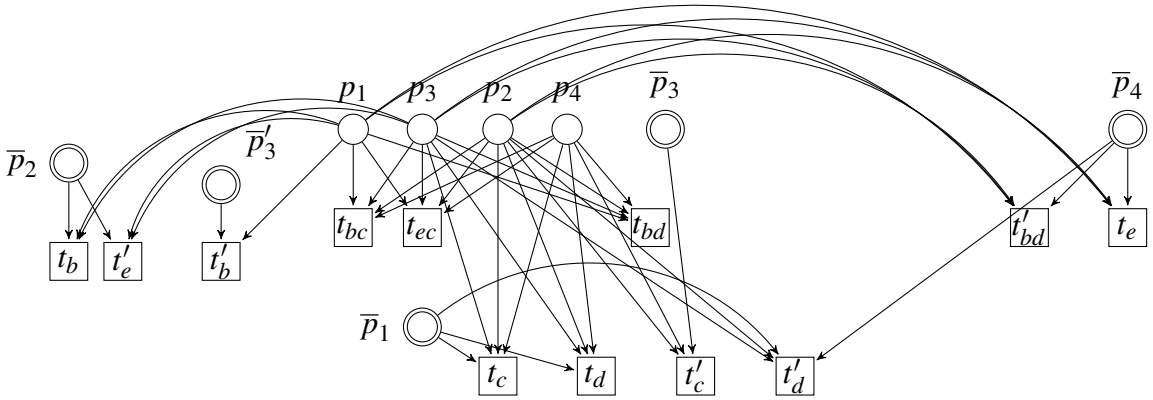


Fig. 3.24 Encoding of confused binary synchronised cluster-acyclic net in Figure 3.22 according to Approach *B*.

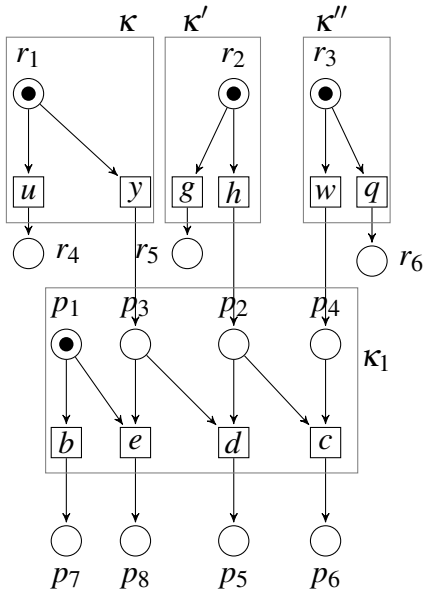


Fig. 3.25 The maximal clusters of a binary synchronised acyclic net with initial marking.

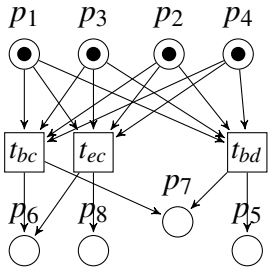
enabled the absence of the token in p_2 is required as their presets are $\{p_1, p_3, \overline{p}_2\}$. On the other hand, for the transitions t''_b, t''_e in Figure 3.26, the absence of the tokens in both p_2 and p_4 are required as their presets are $\{p_1, p_3, \overline{p}_2, \overline{p}_4\}$. The observation applies for t'_b and t''''_b in Figure 3.24 and Figure 3.26 respectively. That is because in Approach *A all* the places belonging into ${}^\bullet\kappa \setminus M$ and the marked pre-places of κ are considered whereas in Approach *B* only the places belonging into the cluster's border as well as the pre-places of κ are used

in the construction. Also, there are only two instances of transition t_b created when using Approach *B* as shown in Figure 3.24. However, five instances of t_b are generated using Approach *A* as it is depicted in Figure 3.26. ◇

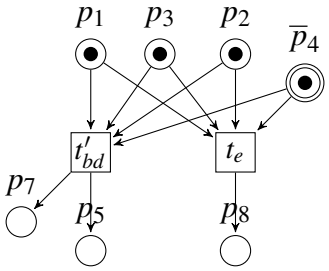
The main difference between Approaches *A* and *B* is that in the former there are no imposed structural restrictions on the cluster-acyclic. The latter, on the other hand, requires that a cluster-acyclic net to be binary synchronised, which in returns limited cases are considered compared to Approach *A*. In terms of the size of the resulting confusion-free net, it turns out that the size of nets produced by both approaches are quite similar. However, Approach *B* retains much more concurrency of the original behaviour, which my be exploited both for further modelling and analyses.

The worse case for Approach *A* is when all the pre-places of a cluster κ are not marked. In this case, if there are p places where $p \in {}^\bullet\kappa$ and $p \notin M$, then there are 2^p different possibilities for p places to be marked. That means, the number of generating new transitions in the constructed net might be exponential. A similar comment is applicable for Approach *B* as the number of the sub-clusters would be large as well.

$t_{bc} = t_{\kappa_1,\{b,c\},\{p_1,p_2,p_3,p_4\}}$
$t_{bd} = t_{\kappa_1,\{b,d\},\{p_1,p_2,p_3,p_4\}}$
$t_{ec} = t_{\kappa_1,\{e,c\},\{p_1,p_2,p_3,p_4\}}$

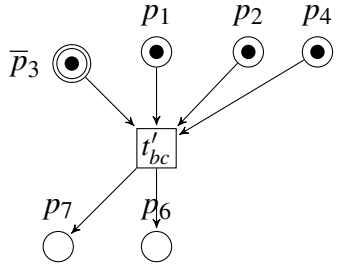


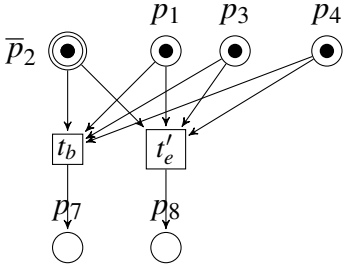
$t_e = t_{\kappa_1,\{e\},\{p_1,p_2,p_3\}}$
$t_{bd} = t_{\kappa_1,\{b,d\},\{p_1,p_2,p_3\}}$



$t_b = t_{\kappa_1,\{b\},\{p_1,p_3,p_4\}}$
$t'_e = t_{\kappa_1,\{e\},\{p_1,p_3,p_4\}}$

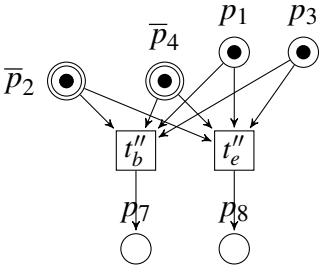
$t_{bc} = t_{\kappa_1,\{b,d\},\{p_1,p_2,p_4\}}$

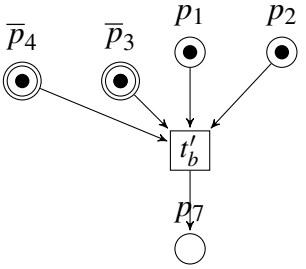




$t''_b = t_{\kappa_1,\{b\},\{p_1,p_3\}}$
$t''_e = t_{\kappa_1,\{e\},\{p_1,p_3\}}$

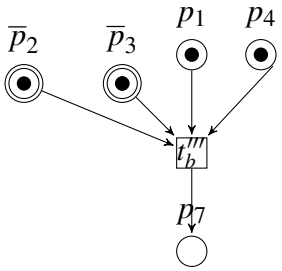
$t'_b = t_{\kappa_1,\{b\},\{p_1,p_2\}}$





$t'''_b = t_{\kappa_1,\{b\},\{p_1,p_4\}}$

$t''''_b = t_{\kappa_1,\{b\},\{p_1\}}$



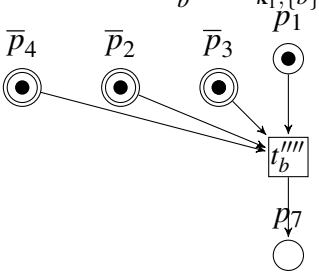


Fig. 3.26 All the transaction associated with cluster κ_1 in Figure 3.25 according to Approach *A*.

The key properties of the second kind of encoding are similar to those obtained in Approach *A*, as formulated below. We omit proofs which are similar to those of the corresponding results presented erlier.

Theorem 3.6.5. Let *acnet* and *acnet*$'$ be as in Definition 3.6.5. Then

$$\{T_{U_1} \dots T_{U_k} \mid U_1 \dots U_k \in \text{sseq}(acnet')\} \subseteq \text{sseq}(acnet) \ .$$

Theorem 3.6.6. Let *acnet*$'$ be as in Definition 3.6.5. Then *acnet*$'$ is a confusion-free acyclic net.

Theorem 3.6.7. Let *acnet* and *acnet*$'$ be as in Definition 3.6.5. Then, for every maximal scenario $ocnet \in \text{maxscenarios}(acnet)$, there is a maximal step sequence $U_1 \dots U_k$ of *acnet*$'$ such that $T_{U_1} \dots T_{U_k}$ is a maximal step sequence of *ocnet*.

Theorem 3.6.8. Let *acnet* and *acnet*$'$ be as in Definition 3.6.5. If *acnet* is an extended free-choice net, then *acnet*$'$ and *acnet* are isomorphic nets.

3.6.7 Non-cluster-acyclic nets

We now revisit Example 24 and Figure 3.20, where cluster-acyclicity constraint is not satisfied. The aim behind imposing this restriction over the clusters is to ensure that the decisions are taken first *locally* at the precedent clusters, and then the posterior clusters' resolutions are made accordingly. That was captured by tokens produced by the former clusters which were consumed by some transitions belonging to the latter clusters. For clusters κ and κ' in Figure 3.20 Page 64, it is not obvious which cluster should resolve the conflict first. In fact, even if we assume that the choice must be resolved first at κ, then some missing information is not taken into account. This information is represented by some transitions being causally dependent on transitions in κ' and leads to incorrect conflict resolving. Hence, Approaches *A* and *B* introduced earlier are not applicable. However, there is an intuitive idea that encoding the net in Figure 3.20 differently could produce a confusion-free net.

The key to possible solution here is to create new *auxiliary* transitions to play the role of transitions attached to the output arcs of places in the interface between the clusters. In Figure 3.20, the transitions *a* and *c* are such transitions that can be replaced by the auxiliary transitions as they belong to the postsets of the interface places p_6 and p_3, respectively. Replacing such transitions should be concerned only with their *weights*. The example below illustrates how to handle the situation where the clusters acyclicity restriction is broken in such way that the confusion has disappeared.

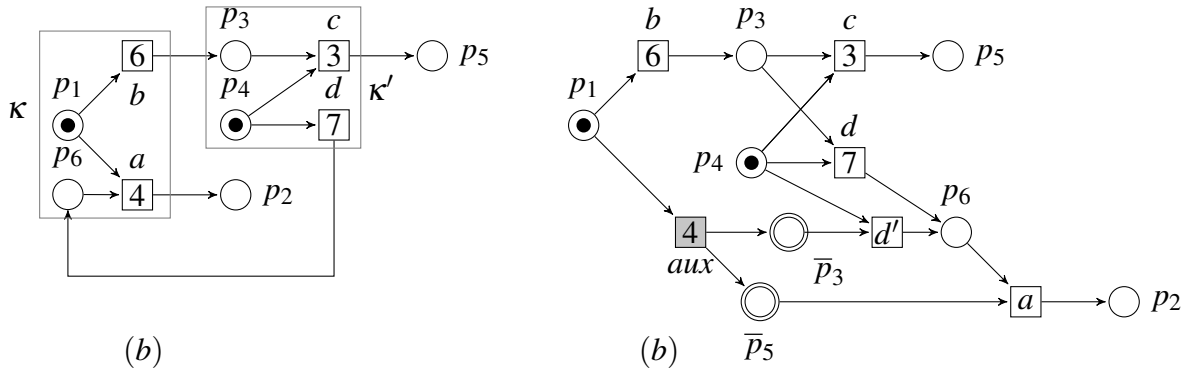


Fig. 3.27 An acyclic net which is not a cluster-acyclic (a), and its encoding into confusion-free (b).

Example 29. Figure 3.27(a) shows the same net in Figure 3.20 but with weights associated with conflict transitions. Its encoding into a confusion-free net is depicted in Figure 3.27(b). Assuming that the conflict should be resolved at κ first, then the *auxiliary* transition *aux* is added instead of the non-enabled transition a. This transition uses the weight of a to preserve resolving the conflict against b probabilistically. The concept of encoding is still the same as in Approach A. Also, an important observation about the encoding in Figure 3.27(b) is that transition a is only available when both $\overline{p}_3$ and $\overline{p}_5$ are marked. Finally, it is worth to mentioned that the encoding can be obtained the other way around, by assuming that the conflict at κ' should be resolved first and replacing transition c and its weight to be associated with the *auxiliary* transition *aux* conflict with d, which would produce the same result. ◇

The next section extends the two approaches of removing confusion to the unfolding semantics.

3.7 Unfolding backward conflicts

In the previous section, we discussed approaches of removing confusion by constructing a new confusion-free acyclic net. The constructions rely on generating negative places for each original place. All the examples discussed previously are *backward deterministic*, and so each transition and place has a unique set of causes. Hence, the encodings proposed in Section 3.6 are not applicable for general acyclic nets, where a place can have multiple incoming arcs. This section extends the proposed approach of removing confusion to the

unfolding semantics. Note that in this section we assumed that the reader is familiar with the basic concepts net unfolding (for more information see [70]).

3.7.1 Branching Process of acyclic nets

The net unfolding can be seen as the partial order behaviour of a concurrent system and relies on acyclic net to represent the system states. The unfolding has been one of the central topics in the net theory. Essentially, unfolding a Petri net results in creating a labelled acyclic net *Unf* which is *complete* and *finite* in our case. Since we assume in this thesis that Petri net is acyclic, its unfolding generates a *finite backward deterministic acyclic net*. The mapping procedure from the unfolding to the original net is called *branching process*, defined next.

Definition 3.7.1 (Branching process)**.** A *homomorphism* from a backward deterministic acyclic net $acnet' = (P', T', F')$ to an acyclic net $acnet' = (P, T, F)$ is a mapping $h : P' \cup T' \to P \cup T$ such that:

- $h(P') \subseteq P$ and $h(T') \subseteq T$.
- For each $e \in T'$, the restriction of h to ${}^\bullet e$ is a bijection between ${}^\bullet e$ and ${}^\bullet h(e)$, and similarly for $e^\bullet$ and $h(e)^\bullet$.
- The restriction of h to $M^{init}_{acnet'}$ is a bijection between $M^{init}_{acnet'}$ and M^{init}_{acnet}.
- For all $e, f \in T'$, if ${}^\bullet e = {}^\bullet f$ and $h(e) = h(f)$, then $e = f$.

Then the pair $\pi = (acnet', h)$ is called a *branching process* of *acnet*. ◇

Two branching processes π and π' are *isomorphic* if there is a bijective homomorphism h from π' to π such that $\pi' \circ h = \pi$. As stated in [70], there is a unique (up to label-preserving isomorphism) *maximal* (w.r.t. prefix relation) branching process of *acnet* called *unfolding*.

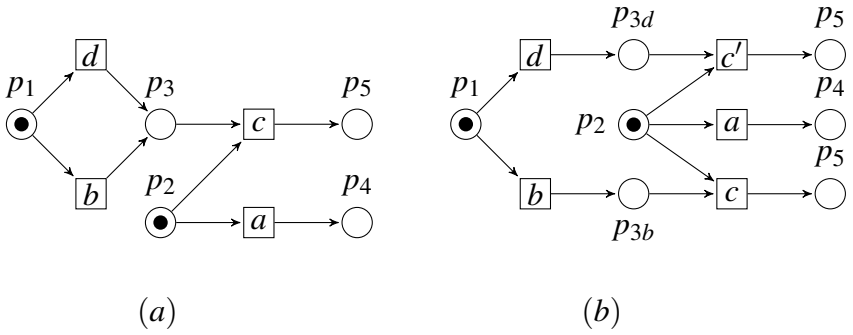


Fig. 3.28 An acyclic net (*a*) and its unfolding into backward deterministic acyclic net (*b*) .

Figure 3.28 shows an unfolding of *acnet* into its maximal branching process, where each place has at most one incoming arc.

The reachable markings of the unfolding correspond to reachable markings of the original net. More precisely, $h(M)$ is a reachable markings of the original net, for every reachable marking M of π. Moreover, for a reachable marking M of π, $M \xrightarrow{e} M'$ implies $h(M) \xrightarrow{h(e)} h(M')$. And, conversely, for a reachable marking M of π, $h(M) \xrightarrow{t} M'$ implies that there are e and M'' such that $h(M'') = M'$, $h(e) = t$, and $h(M) \xrightarrow{t} M''$.

Since we interested in building a complete and finite *unfolding*, generating a *complete prefix* is not considered. Therefore, there is no need to take the *cut-off* events in the consideration.

In this thesis, we will consider an existing unfolding algorithm presented in [70]. However, such an algorithm is focused on generating a complete prefix of the unfolding. In order to derive a maximal unfolding, suitable modifications need to added to this algorithm.

Example 30. If the algorithm presented in [70] is performed on the the acyclic net in Figure 3.28(a), then the net in Figure 3.29 will be the result. It is a complete prefix but not the full unfolding. Note that, d is considered as a *cut-off* event. Thus, the construction stopped at this point. ◇

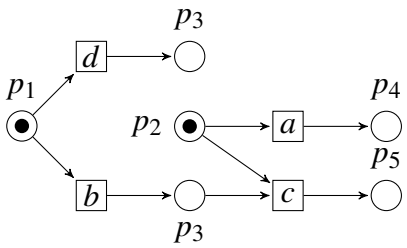


Fig. 3.29 Complete prefix for the acyclic net in Figure 3.28(a).

Since the previous example shows that the algorithm in [70] generates just a complete prefix, our modifications should produce a full unfolding of any *acnet*. The modified version of the algorithm in [70] is represented in Algorithm 10. The difference between the two algorithms is that the former takes into account the *cut-off* events so that the construction of the prefix is *pruned* without losing information such as transition executability, whereas the latter continues the construction and appends all the *potential extensions* (the set of transitions that can be added to the unfolding to extend it), even if they have been already added to the prefix being constructed. As a result, many instances of the same transition may

be added several times as the full unfolding depicted in Figure 3.28(b) where the transition c is added for each branch.

Algorithm 10 Unfolding algorithm for an acyclic net

Input: $acnet = (P,T,F,M_0)$
Output: a finite Unfolding $Unf_{acnet'}$ of $acnet$
$Unf_{acnet'} \leftarrow$ the empty branching process
$pe = \varnothing$
add instances of the places in the initial marking of $acnet$ to $Unf_{acnet'}$
add all possible extension of $Unf_{acnet'}$ to pe
while $pe \neq \varnothing$ **do**
    $ProcessEvent(e)$
    add to $Unf_{acnet'}$ a transition $t' = (t,p)$ of pe and a place (p'_t,t') for every output place p of t
end while

    procedure PROCESSEVENT(e)
        for all $g \in UpdatedPotentialExt(Unf_{acnet'},e)$ **do**
            $pe \leftarrow pe \cup \{g\}$
        end for
    end procedure

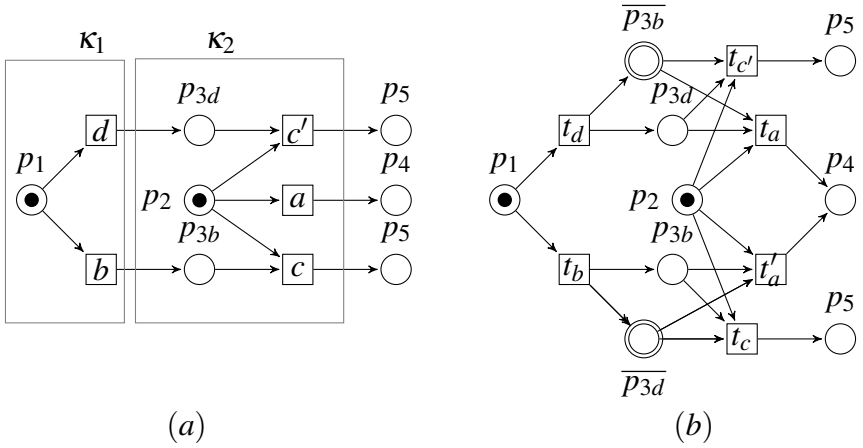


Fig. 3.30 (a) A cluster-acyclic net for the unfolding in Figure 3.28 (b) and its encoding according to Approach A (b).

Example 31. Consider the net in Figure 3.30. In Figure 3.30(a) the cluster-acyclic net for the unfolding in Figure 3.28 is portrayed. There are two maximal clusters: $\kappa_1 = \{b,d\}$,

$\kappa_2 = \{a, c, c'\}$. It is worth to notice that the weight of transitions c and c' should not be split as they never be enabled at the same time. According to Approach A, the transactions associated with clusters are as follows:

$$\begin{aligned} \text{trans}(\kappa_1, \{p_1\}) &= \{\{b\}, \{d\}\} \\ \text{trans}(\kappa_2, \{p_2, p_{3d}\}) &= \{\{a\}, \{c'\}\} \\ \text{trans}(\kappa_2, \{p_2, p_{3b}\}) &= \{\{a\}, \{c\}\}. \end{aligned}$$

Since $\text{caused}(\kappa_2) \cap {}^\bullet\kappa_1 = \varnothing$, we have $\kappa_2 \not\sqsubset \kappa_1$ and the strict partial order relation is satisfied. Therefore, Approach A in Definition 3.6.1 is applicable, and the result is shown in Figure 3.30(b). $\diamond$

3.8 Conclusion

This chapter developed a theoretical framework that concerns the probabilistic analysis of acyclic nets. The fundamental concept of the probabilistic acyclic nets is the presence of *conflict* between transitions. We formally define how conflicts can be resolved probabilistically based on the positive integer *weights* associated with transitions. The *scenarios* defined to represent execution histories of acyclic nets are used to analyse probabilistically their behaviour. An important requirement in concurrent probabilistic models is that the execution order of a scenario should be independent from the probability calculation. More precisely, all the executions of a single scenario should obtain the same probability. Then, *confusion* was discussed as an undesirable situation, where the conflict is not resolved correctly leading to inaccurate results of probability calculation.

We have formally defined the class of *cluster-acyclic net* that appear to be a new kind of acyclic nets which was not previously discussed in the literature. The purpose of introducing this class is to partition the acyclic net in such way that the conflict transitions are grouped into clusters, and all the clusters are strictly partially ordered. To localize nondeterministic choices made, conflicts are resolved locally at each cluster. The class of cluster-acyclic net is used as a basis to remove confusion by translating confused cluster-acyclic nets into confusion-free acyclic nets.

Two approaches were proposed to eliminate confusion. Approach A (motivated by the work in [26]) concentrates on markings M included in the pre-places of a cluster. In Approach B, only the binary synchronised acyclic nets are considered.

Both approaches use *negative* places to introduce additional causality between transitions, and the encoding produced relies on postponing the executions of some transitions (see Example 14 on Page 48, Example 17 on Page 52, and Example 27 on Page 69). A key result is that the constructions according to Definitions 3.6.1 and Definition 3.6.5 produce *confusion-free* nets. Proving such a property is based on ensuring that the executions of resulting nets satisfy the structure of *free-choice or extended free-choice* nets, where the conflict transitions are always enabled together at the same marking. Then, we formalised the correspondence of behaviour in the original confused net and its confusion-free version. Basically, our approach of maintaining the original behaviour is obtained through the *maximal scenarios* that can be simulated in both nets.

The probabilistic framework conducted in this chapter has limited applicability as we considered only acyclic nets. We intend to extend the definitions and theoretical results to improve the applicability of the framework. The extended version of the framework concerns the set of communicating acyclic nets as done in the subsequent chapter (Chapter 4). Moreover, our probabilistic framework can be extended to the abstraction relation as discussed in Chapter 5. Satisfiability checking to detect the case of confusion is discussed in Chapter 6.

The probabilistic framework developed in this chapter can provide a basis for possible directions of future work. For example, *time* can be considered to handle the confusion. In such a case, transitions involved in confusion would exhibit some delay duration to ensure they are enabled at the same time. Also, another direction is a combination of weighted and *unweighted* transitions to allow non-deterministic conflict resolution. However, this is outside the scope of this thesis.

Chapter 4

Communication structured acyclic nets

4.1 Introduction

In previous chapter (Chapter 3), a probabilistic framework was introduced for a single acyclic net. Also, we investigated the *confusion* phenomenon, which is a situation where conflict between transitions is resolved incorrectly and results in an inaccurate probabilistic analysis. In Section 3.5, a new class of acyclic nets, *cluster-acyclic nets*, was defined to remove confusion. Then, we proposed two approaches to convert a confused cluster-acyclic net into another acyclic net which is behaviourally equivalent and confusion-free.

In this chapter, we investigate how to extend the probabilistic framework to Communication Structured Acyclic Nets (CSA-nets), which are sets of communicating acyclic nets. Intuitively, in CSA-nets, acyclic nets are integrated in one structure that allows them to interact by the means of *asynchronous* and *synchronous* communication using extra nodes called *buffer places*.

The definition of conflict and confusion is also extended to CSA-nets. The proposed solution of removing confusion from a CSA-net requires translating it into a single acyclic net such that the underlying transitions involved in asynchronous communications are expanded and the buffer places are replaced by regular places. The transitions forming synchronised cycles via synchronous communication are glued together and the buffer places are deleted. Respecting the cluster-acyclicity constraint is essential to reuse the approaches introduced in Section 3.5. It turns out that the the cluster-acyclicity constraint can sometimes be checked at the level of CSA-nets. This constraint is satisfied when asynchronous communication between the component acyclic nets is *unidirectional*. More precisely, as an extension of the work in [12], we formally define another approach concerned with eliminating confusion in *cascading* CSA-nets. In terms of maintaining the behaviour of the original net, the

constructed confusion-free version simulates the original behaviour if all possible scenarios are reproduced.

This chapter is organised as follows: Formal definitions of CSA-nets and their behavioural and structural properties are introduced in Section 4.2–Section 4.6. Probabilistic CSA-nets are introduced in Section 4.7. In Section 4.8, we show how adding communications at the level of alternative scenarios can cause a confusion, and then we formally provide the definition of confusion in CSA-nets. Section 4.9 proposes an approach for removing confusion in CSA-nets, where a confused CSA-net is translated into a single acyclic net. A new class of cascading CSA-nets is introduced in Section 4.10. Then a new approach of removing confusion in such class is presented in Section 4.10.1. Unfolding CSA-nets is discussed in Section 4.11 as a step needed to remove confusion in non-backward-deterministic CSA-nets. The chapter is concluded in Section 4.12.

4.2 Communication Structured Acyclic Nets (CSA-nets)

Communication Structured Acyclic Nets (CSA-nets) are derived from Structured Occurrence Nets (SO-nets) which were introduced in [103, 105] and elaborated in [78]. [103] mentioned that gaining a deep understanding of the dependability between *failures*, *errors*, and *faults* in complex evolving system composed of distributed sub-systems was the motivation behind introducing SO-nets, where fault-error-failure chain can be represented by event-cause-state. In [106], SO-nets were used to support the analysis of evidence related to the activities of complex systems, including cybercrime and major accidents. The goal was to indicate those responsible for cybercrime or identify the causes of accident. [84] showed that SO-nets can be used as a framework for visualising and analysing behaviour of complex evolving systems. SO-nets were used in [11] as a framework for modelling cybercrime investigation. Other works on SO-nets were related to provenance [92] and timed behaviours [23]. A recently designed and implemented tool SONCraft [85] (based on the WORKCraft platform [132, 100] which is a flexible common underpinning for graph based models) provides an extensive and powerful support for dealing with models of complex evolving systems based on SO-nets, including visualisation and verification.

Generalising SO-nets, CSA-nets are defined as sets of related acyclic nets, including various types of relationships and supporting abstraction in such a way that it is possible to record the past, current, and future behaviour of CESs [78]. This makes CSA-nets a suitable candidate for the modelling of sophisticated systems, where the complexity of behaviour can be captured by the structure of CSA-nets.

CSA-nets add communication to represent the interaction among several separate subsystems. The idea is to build component acyclic nets with different communications between them instead of only one acyclic net that reflects the combined activity of numerous interacting systems [78, 84, 11].

Each CSA-net is a set of acyclic nets with synchronous or asynchronous communication links between their transitions implemented using extra nodes called *buffer places* (which provided a motivation for a/syn connections discussed, e.g., in [74]). These buffer places can be used to represent the causality between component acyclic nets [73]. When two transitions are subject to synchronous communication, they are always executed together, but under asynchronous communication they may be executed simultaneously or sequentially.

Below are formal definitions of CSA-nets and their behavioural and structural properties.

Definition 4.2.1 (CSA-net [8])**.** A *communication structured acyclic net (or* CSA*-net)* is a tuple $csan = (acnet_1, \ldots, acnet_n, Q, W)$ $(n \geq 1)$ such that:

1. $acnet_1, \ldots, acnet_n$ are acyclic nets with disjoint sets of nodes. We denote:

$$\begin{array}{lcllcl} P_{csan} & \triangleq & P_{acnet_1} \cup \cdots \cup P_{acnet_n} & P^{init}_{csan} & \triangleq & P^{init}_{acnet_1} \cup \cdots \cup P^{init}_{acnet_n} \\ T_{csan} & \triangleq & T_{acnet_1} \cup \cdots \cup T_{acnet_n} & P^{fin}_{csan} & \triangleq & P^{fin}_{acnet_1} \cup \cdots \cup P^{fin}_{acnet_n} \\ F_{csan} & \triangleq & F_{acnet_1} \cup \cdots \cup F_{acnet_n} \,. & & & \end{array}$$

2. Q is a finite set of *buffer places* disjoint from $P_{csan} \cup T_{csan}$.

3. $W \subseteq (Q \times T_{csan}) \cup (T_{csan} \times Q)$ is a set of arcs. We denote: $Q_{csan} \triangleq Q$ and $W_{csan} \triangleq W$.

4. For every $q \in Q$, there is $t \in T_{csan}$ such that tWq, and qWu then t and u belong to different $acnet_i$'s.

The set of all CSA-nets is denoted by *CSAN*. ◇

To indicate relationships between different nodes, for all $x \in P_{csan} \cup T_{csan} \cup Q_{csan}$ and $X \subseteq P_{csan} \cup T_{csan} \cup Q_{csan}$, we denote the *directly preceding* and *directly following* nodes as follows:

$$\begin{array}{lcllcl} \mathrm{pre}_{csan}(x) & \triangleq & \{z \mid (z,x) \in F_{csan} \cup W_{csan}\} & \mathrm{pre}_{acnet}(X) & \triangleq & \bigcup\{\mathrm{pre}_{csan}(z) \mid z \in X\} \\ \mathrm{post}_{csan}(x) & \triangleq & \{z \mid (x,z) \in F_{csan} \cup W_{csan}\} & \mathrm{post}_{csan}(X) & \triangleq & \bigcup\{\mathrm{post}_{csan}(z) \mid z \in X\}\,. \end{array}$$

That is, in a CSA-net the the preset and postset of a node are extended to include buffer places [84] – responsible of transferring tokens among component acyclic nets – in order to capture the relation of *weak causality* between transitions of different acyclic nets [64].

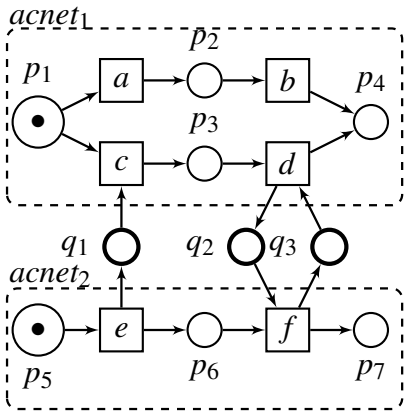


Fig. 4.1 CSA-net with initial marking.

Example 32. Figure 4.1 shows a CSA-net composed of two acyclic nets. Transitions e and c are communicating *asynchronously*, so they can be executed together, or e then c (but not c before e). Therefore, we can say that there is a *weak causality* relationship between e and c captured by the presence of $q_1 \in \text{post}_{csan}(e) \cap \text{pre}_{csan}(c)$. On the other hand, d and f must be executed simultaneously as they are involved in *synchronous* communication. The set of buffer places $\{q_1, q_2, q_3\}$ is responsible for passing the tokens among the component acyclic nets. Also, it is worth to notice that $\text{post}_{acnet}(e) = \{p_6, q_1\}$ since the firing of e will produce tokens in those two places. ◇

Just as acyclic nets, CSA-nets may exhibit backward and forward non-determinism. Moreover, they can contain synchronous cycles involving only buffer places.

Definition 4.2.2 (BDCSA-net [8])**.** A *backward deterministic* CSA*-net (or* BDCSA*-net)* is $bdcsan \in CSAN$ such that:

1. The component acyclic nets belong to *BDAN*.
2. $|\text{pre}_{bdcsan}(q)| = 1$, for every $q \in Q_{bdcsan}$.

The set of all BDCSA-nets is denoted by *BDCSAN*. ◇

In BDCSA-nets backward determinism is required, whereas forward determinism is not necessary. The next class of CSA-nets can provide full and unique records of causal histories.

Definition 4.2.3 (CSO-net [8])**.** A *communication structured occurrence net (or* CSO*-net)* is $cson \in CSAN$ such that:

1. The component acyclic nets belong to *ON*.

2. $|\mathrm{pre}_{cson}(q)| = 1$ and $|\mathrm{post}_{cson}(q)| \leq 1$, for every $q \in Q_{cson}$.

3. No place in P_{cson} belongs to a cycle in the graph of *cson*.

The set of all CSO-nets is denoted by *CSON*. ◇

A CSO-net exhibits backward determinism and forward determinism providing full and unambiguous information about a single causal history of all transitions it involves [8].

Scenarios for CSA-nets can be defined similarly as for acyclic nets.

Definition 4.2.4 (scenario and maximal scenario [8])**.** A *scenario* of a CSA-net $csan = (acnet_1, \ldots, acnet_n, Q, W)$ is a CSO-net $cson = (ocnet_1, \ldots, ocnet_n, Q', W')$ such that:

1. $ocnet_i \in \mathrm{scenarios}(acnet_i)$, for every $1 \leq i \leq n$.

2. $Q' \subseteq Q$ and $W' \subseteq W$.

3. $\mathrm{pre}_{cson}(t) = \mathrm{pre}_{csan}(t)$ and $\mathrm{post}_{cson}(t) = \mathrm{post}_{csan}(t)$, for every $t \in T_{cson}$.

Moreover, *cson* is *maximal* if there is no scenario $cson'$ satisfying $T_{cson} \subset T_{cson'}$.
The set of all scenarios of *csan* is denoted by $\mathrm{scenarios}(csan)$, and the set of all maximal scenarios of *csan* by $\mathrm{maxscenarios}(csan)$. ◇

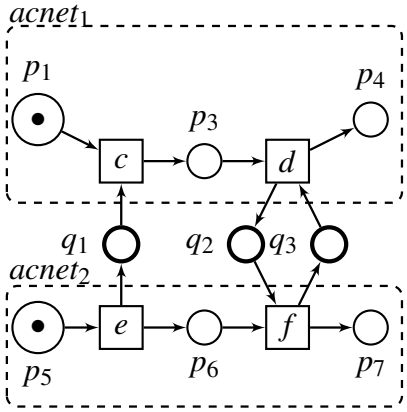


Fig. 4.2 CSO-net with initial marking.

One can say that scenarios represent possible *deterministic executions* (concurrent histories), and maximal scenarios are complete in the sense that they cannot be extended any further.

Each scenario of a CSA-net $csan = (acnet, \ldots, acnet_k, Q, W)$ is identified by the set of its transitions. The scenario with the transition set $V \subseteq T_{csan}$ is denoted by $\mathrm{scenario}_{csan}(V)$.

Example 33. Figure 4.2 shows a CSO-nets which is also one of the maximal scenarios of the CSA-net in Figure 4.1. ◇

4.3 Step sequence semantics

Definition 4.3.1 (step and marking [8])**.** Let *csan* be a CSA-net.

1. markings$(csan) \triangleq \mathbb{P}(P_{csan} \cup Q_{csan})$ are the *markings.*
2. $M^{init}_{csan} \triangleq P^{init}_{csan}$ is the default *initial* marking.
3. steps$(csan) \triangleq \{U \in \mathbb{P}(T_{csan}) \setminus \{\varnothing\} \mid \forall t \neq u \in U : \text{pre}_{csan}(t) \cap \text{pre}_{csan}(u) = \varnothing\}$ are the *steps*. ◇

The initial marking of a CSA-net is the union of the initial markings of all the component acyclic nets. More precisely, $M^{init}_{csan} = M^{init}_{acnet_1} \cup M^{init}_{acnet_2} \cdots \cup M^{init}_{acnet_n}$ with an assumption that the buffer places are always excluded from the initial marking M^{init}_{csan}. However, in contrast to acyclic nets, in general the marking in CSA-net can be a set of places and buffer places. The transitions belonging to a step of a CSA-net come from one or more component acyclic nets.

Example 34. For the CSA-net in Figure 4.1, the initial marking is $M_0^{csan} = \{p_1, p_5\}$. The steps are steps$(csan) = \{U \in \mathbb{P}(\{a,b,c,d,e,f\}) \setminus \{\varnothing\} \mid a \in U \implies c \notin U\}$. ◇

Definition 4.3.2 (enabled and executed step [8])**.** Let M be a marking of *csan*.

1. enabled$_{csan}(M) \triangleq \{U \in \text{steps}(csan) \mid \text{pre}_{csan}(U) \subseteq M \cup (\text{post}_{csan}(U) \cap Q)\}$ are the steps *enabled* at M.
2. A step U enabled at M can be *executed* yielding a new marking $M' \triangleq (M \cup \text{post}_{csan}(U)) \setminus \text{pre}_{csan}(U)$. This is denoted by $M[U\rangle_{csan} M'$. ◇

In CSA-net, an enabled step U requires not only all the input places of the set of acyclic nets to be included in the marking, but also the the buffer places connected to the incoming arcs of U must be in the marking. However, if a buffer place is not included in the marking, then it must belong to the postset of a transition in U [8]. In other words, the step semantic defined above means that a step U can not only use the tokens available locally at each acyclic net, but also the tokens from other component acyclic nets which are passed by the buffer places. Hence, an enabled step U in CSA-net can include sets of transitions fired simultaneously due to the presence of communication between interacting acyclic nets. This execution semantics is more expressive than that in acyclic nets [84].

Definition 4.3.3 (mixed step sequence and step sequence [8])**.** Let $M_0,\ldots,M_k$ ($k \geq 0$) be markings and $U_1,\ldots,U_k$ be steps of a CSA-net *csan* such that $M_{i-1}[U_i\rangle_{csan} M_i$, for every $1 \leq i \leq k$.

1. $\mu = M_0U_1M_1\ldots M_{k-1}U_kM_k$ is a *mixed step sequence from M_0 to M_k*.
2. $\sigma = U_1\ldots U_k$ is a *step sequence from M_0 to M_k*.

This is denoted by $M_0[\mu\rangle\rangle_{csan} M_k$ and $M_0[\sigma\rangle_{csan} M_k$, respectively. Moreover, $M_0[\rangle_{csan} M_k$ denotes that M_k is *reachable from M_0*. ◇

If $k = 0$ then $\mu = M_0$ and the corresponding step sequence σ is the *empty* sequence denoted by λ.

Definition 4.3.4 (behavioural notions [8])**.** The following sets capture various behavioural notions related to step sequences and reachable markings of a CSA-net *csan*, assuming that the the initial marking is the starting point of system execution.

1. $\text{sseq}(csan) \triangleq \{\sigma \mid M^{init}_{acnet}[\sigma\rangle_{csan} M\}$ are the *step sequences*.
2. $\text{mixsseq}(csan) \triangleq \{\mu \mid M^{init}_{acnet}[\mu\rangle\rangle_{csan} M\}$ are the *mixed step sequences*.
3. $\text{maxsseq}(csan) \triangleq \{\sigma \in \text{sseq}(csan) \mid \neg\exists U : \sigma U \in \text{sseq}(csan)\}$ are the *maximal step sequences*.
4. $\text{maxmixsseq}(csan) \triangleq \{\mu \in \text{mixsseq}(csan) \mid \neg\exists U, M : \mu UM \in \text{mixsseq}(csan)\}$ are the *maximal mixed step sequences*.
5. $\text{reachable}(csan) \triangleq \{M \mid M^{init}_{csan}[\rangle_{csan} M\}$ are the *reachable markings*.
6. $\text{finreachable}(csan) \triangleq \{M \mid \exists\sigma \in \text{maxsseq}(csan) : M^{init}_{csan}[\sigma\rangle_{csan} M\}$ are the *final reachable markings*.

Example 35. For the CSA-net in Figure 4.1, all possible maximal step squences in $\text{maxsseq}(csan)$ are:

$$\begin{array}{lllllllll} \sigma_1 & = & \{a,e\}\{b\} & \sigma_2 & = & \{a\}\{e\}\{b\} & \sigma_3 & = & \{a\}\{b\}\{e\} \\ \sigma_4 & = & \{a\}\{e,b\} & \sigma_5 & = & \{e\}\{a\}\{b\} & \sigma_6 & = & \{c,e\}\{d,f\} \\ \sigma_7 & = & \{e\}\{c\}\{d,f\} & & & & & & \end{array}$$

and all possible maximal mixed step sequences in maxmixsseq($csan$) are:

$$\begin{aligned}
\mu_1 &= \{p_1,p_5\}\{a,e\}\{p_2,p_6,q_1\}\{b\}\{p_4,p_6,q_1\}\\
\mu_2 &= \{p_1,p_5\}\{a\}\{p_2,p_5\}\{e\}\{p_2,p_6,q_1\}\{b\}\{p_4,p_6,q_1\}\\
\mu_3 &= \{p_1,p_5\}\{a\}\{p_2,p_5\}\{b\}\{p_4,p_5\}\{e\}\{p_4,p_6,q_1\}\\
\mu_4 &= \{p_1,p_5\}\{a\}\{p_2,p_5\}\{e,b\}\{p_4,p_6,q_1\}\\
\mu_5 &= \{p_1,p_5\}\{e\}\{p_1,p_6,q_1\}\{a\}\{p_2,p_6,q_1\}\{b\}\{p_4,p_6,q_1\}\\
\mu_6 &= \{p_1,p_5\}\{c,e\}\{p_3,p_6\}\{d,f\}\{p_4,p_7\}\\
\mu_7 &= \{p_1,p_5\}\{e\}\{p_1,p_6,q_1\}\{c\}\{p_3,p_6\}\{d,f\}\{p_4,p_7\}.
\end{aligned}$$

Moreover, all the reachable markings are:

$$\text{reachable}(csan) = \{\{p_1,p_5\},\{p_2,p_6,q_1\},\{p_4,p_6,q_1\},\{p_3,p_6\},\{p_4,p_7\},\dots\}$$

and the final reachable markings are finreachable($csan$) $= \{\{p_4,p_6,q_1\},\{p_4,p_7\}\}$. ◇

As discussed in [73], the standard Petri nets can usually be seen as asynchronous concurrency model with a firing sequence semantics or a step semantics based on sets of transitions that may occur simultaneously when enough resources are available. Hence, whenever a step occurs each of its transitions (or subsets) could also be fired. There is no (structural) possibility to express that an enabled transition has to (wait in order to) synchronise with another one. On the other hand, it is not difficult to make an otherwise enabled transition wait for the occurrence of a second one by using a message (in the form of a token left by the second one in a special input place of the first transition). These considerations motivated the introduction of *buffer places* used by the CSA-nets. Suppose that transition t has an output buffer place q and transition v has q as an input buffer place. Then if t occurs it adds a 'message' (a token) to the buffer place q; this message may either remain there to serve later as input to v (the usual asynchronous communication of Petri nets), or be directly picked up by v in the *same* step. The communication connection provided by the buffer place q (an a/synchronous communication channel) can be compared to a telephone connection with an answering machine: either the caller waits for the callee to answer the phone (and then they communicate synchronously), or the caller leaves a message on the answering machine to be listened to later by the callee.

Concepts similar to the buffer places used in CSA-nets can be found in the existing literature. [33] observed that such a mechanism is not a concept in the standard Petri net models, and to model synchronous communication one needs additional places and transitions which may result in complicated structures. Hence, [33] proposed to extend the Coloured

Petri Net model to support communication through channels inspired by the synchronisation operators of CCS [91] and CSP [62] (as well as communication constructs expressed in high level programming languages). Such an extension is seen as a powerful description primitive (see also [81]), and a way to structure Petri net models.

As another example, [27] introduced Petri nets with so-called zero places, allowing one to consider firing sequences leading from one 'stable' marking (a marking in which all zero places are empty) to the next stable marking without affecting ordinary places on the way. For example, [76] used such a feature to model protocols in which one partner can be ahead of another one in the middle of communication.

At the level of abstract system modelling, REO [17] used channels operating like buffer places to model coordination of (software) components. [114] discussed how systems that communicate through REO channels can be modeled as Petri nets. Again, this leads to relatively complex net structures.

As a simple example of a modelling advantage of using buffer place, let us consider two acyclic nets, *acnet* and *acnet*$'$, such that *acnet* wants to communicate synchronously with *acnet*$'$ using exactly one of n transitions $t_1,\ldots,t_n$, and *acnet*$'$ wants to communicate synchronously with *acnet* using exactly one of n transitions $v_1,\ldots,v_n$. Assume also, for the sake of argument, that each of these transitions has m adjacent arcs. One could model this quite easily by 'gluing' together each t_i with each v_j, creating n^2 new transitions and $2mn^2$ new arcs. On the other hand, it is possible to achieve the same effect by constructing just 2 new buffer places, q and q', and $4n$ new arcs (i.e., $(t_1,q),\ldots,(t_n,q),(v_1,q'),\ldots,(v_n,q')$ and $(q',t_1),\ldots,(q',t_n),(q,v_1),\ldots,(q,v_n)$). One could take this example further and consider k acyclic nets and a synchronisation requirement such that exactly one of n transitions in these k acyclic nets has to be used. Then the explicit construction of all possible synchronised transition would create n^k new transitions and $2mn^k$ new arcs, whereas the same could be achieved using only k buffer places and $2kn$ new arcs.

4.4 Well-formed CSA-nets

A basic property of CSA-nets is well-formedness. The motivation of this property is to obtain unambiguous records of causal histories [8] and to interpret the causality and concurrency explicitly so one can ensure that each transition is executed only once in each step sequence.

Definition 4.4.1 (well-formedness [8])**.** A CSA-net *csan* is *well-formed* if each transition occurs in at least one step sequence and the following hold, for every step sequence $U_1 \ldots U_k \in$ sseq(*csan*):

1. $\text{post}_{csan}(t) \cap \text{post}_{csan}(u) = \varnothing$, for every $1 \leq i \leq k$ and all $t \neq u \in U_i$.

2. $\text{post}_{csan}(U_i) \cap \text{post}_{csan}(U_j) = \varnothing$, for all $1 \leq i < j \leq k$. ◇

Intuitively, a well-formed CSA-net does not have 'useless' transitions and no place or buffer place is filled by a token more than once in any given step sequence. In other words, no token is used more than once and no transition is executed more than once in step sequences of a well-formed CSA-net. As a result, one can guarantee that the resulting marking is not effected by the order of the execution of transitions [8].

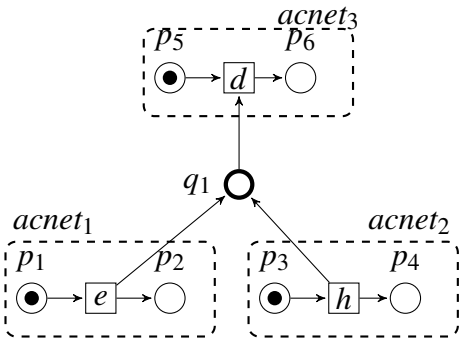


Fig. 4.3 CSA-net which is not a well-formed.

Example 36. Figure 4.3 shows a non-well-formed CSA-net. The asynchronous communication between e and d and between h and d through the same buffer place $q_1 \in \text{post}_{csan}(e) \cap \text{post}_{csan}(h)$ produces unclear causality relation. More precisely, it is not obvious 'who' causes d. Even though this CSA-net is *safe* so each regular and buffer place can hold only one token, transition d is fireable once a token generated by either e or h arrives at q_1. The 'OR-causality' captured by the incoming arcs to q_1 violates Definition 4.2.3 and Definition 4.2.4 where the regular and buffer places must have at most one input and output arcs. Hence, the step sequence $\sigma = \{e,h\}\{d\}$ leads to an invalid scenario. Well-formed definition is introduced to exclude such cases. ◇

By satisfying the backward and forward determinism, SO-nets enjoy some essential behavioural properties 'for free'.

Proposition 4.4.1 ([8])**.** Each CSO-net is well-formed.

Proposition 4.4.2 ([8])**.** A CSA-net is well-formed iff each transition occurs in at least one scenario, and each step sequence is a step sequence of at least one scenario.

Proposition 4.4.3 ([8])**.** Step sequences of a well-formed CSA-net do not contain multiple occurrences of transitions.

Each step sequence σ of a well-formed CSA-net *csan* induces a scenario $\text{scenario}_{csan}(\sigma) \triangleq \text{scenario}_{csan}(\bigcup\sigma)$ such that $\sigma \in \text{maxsseq}(\text{scenario}_{csan}(\sigma))$. Thus, in a well-formed CSA-net, different step sequences may generate the same scenario, and conversely, each scenario is generated by at least one step sequence. Moreover, two maximal step sequences generate the same scenario iff their executed transitions are identical.

4.5 Syn-cycles

In the case of CSO-nets each executed step can be unambiguously represented as a disjoint union of sub-steps which cannot be further decomposed.

Definition 4.5.1 (syn-cycle [8])**.** A *syn-cycle* of a CSO-net *cson* is a maximal non-empty set of transitions $S \subseteq T_{csan}$ such that, for all $t \neq u \in S$, $(t,u) \in W^{+}_{cson}$. The set of all syn-cycles is denoted by $\text{syncycles}(cson)$. ◇

The idea behind the notion of syn-cycles is to represent maximum number of synchronous communications [84].

Example 37. In Figure 4.1, there is one non-singleton syn-cycle $S_1 = \{d, f\}$. Moreover, $S_2 = \{a\}$, $S_3 = \{b\}$, $S_4 = \{c\}$, and $S_5 = \{e\}$ are singleton syn-cycles. ◇

The set of all syn-cycles is a partition of the transition set of a CSO-net (in a unique way).

Proposition 4.5.1 ([8])**.** $\text{syncycles}(cson)$ forms a partition of T_{cson}, for every CSO-net *cson*.

As stated next, each step occurring in step sequences of a CSO-net can be partitioned into syn-cycles which can then be fired sequentially.

Proposition 4.5.2 ([8])**.** Let M be a reachable marking of a CSO-net *cson* and $M[U\rangle_{cson} M'$. Then there are $U_1, \ldots, U_k \in \text{syncycles}(cson)$ such that $U = U_1 \uplus \cdots \uplus U_k$ and $M[U_1 \ldots U_k\rangle_{cson} M'$.

The above result means that syn-cycles is responsible for generating all reachable markings of a CSON *cson* rather than all the potential steps. Moreover, the same holds for every well-formed CSA-net *csan* and the syn-cycles of its scenarios given by $\text{syncycles}(csan) \triangleq \bigcup\{\text{syncycles}(cson) \mid cson \in \text{scenarios}(csan)\}$ [8].

Proposition 4.5.3 ([8])**.** Let M be a reachable marking of a well-formed CSO-net *csan* and $M[U\rangle_{csan} M'$. Then there are $S_1, \ldots, S_k \in \text{syncycles}(csan)$ such that $U = S_1 \uplus \cdots \uplus S_k$ and $M[S_1 \ldots S_k\rangle_{cson} M'$.

The introduction of probabilistic analysis to the model of CSA-nets leads to a slight re-formulation of the execution semantics CSA-nets (motivated by general modelling considerations). More precisely, suppose that M is a reachable marking of a well-formed CSA-net *csan* and $M[U\rangle_{csan}M'$. By Proposition 4.5.3, we know there are $S_1,\ldots,S_k \in \text{syncycles}(csan)$ such that $U = S_1 \uplus \cdots \uplus S_k$. In such a case, it may happen some of the syn-cycles S_i are not enabled at M and so their inclusion in the probabilistic analysis is not justified.

We will therefore require until the end of the thesis that the following holds for the enabledness condition of a well-formed CSA*-net.*

$$\begin{array}{rcl} U \in \text{enabled}_{csan}(M) & \Longrightarrow & \exists S_1,\ldots,S_k \in \text{syncycles}(csan) : \\ & & U = S_1 \uplus \cdots \uplus S_k \ \wedge S_1,\ldots,S_k \in \text{enabled}_{csan}(M). \end{array} \tag{4.1}$$

No other changes are made to the notations and definitions.

The above additional condition is harmless from the executability point of view since $M[U\rangle_{csan}M'$ in the original formulation implies that for the syn-cycles $S_1,\ldots,S_k$ as in Eq.(4.1), we have $M[S_1,\ldots,S_k\rangle_{csan}M'$ in the modified formulation (moreover, the syn-cycles $S_1,\ldots,S_k$ can be executed in any order).

4.6 Conflict, causality, and concurrency

This section extends the structural properties of acyclic nets presented in Section 3.2.2 to the level of CSA-nets, but they are now based on syn-cycles rather than transitions.

Definition 4.6.1 (structural notions)**.** Let *csan* be a well-formed CSA-net.

1. Two syn-cycles $S \neq S' \in \text{syncycles}(csan)$ are in *direct forward conflict*, denoted $S\#_0S'$, if $\text{pre}_{csan}(S) \cap \text{pre}_{csan}(S') \neq \varnothing$.

2. Two syn-cycles $S \neq S' \in \text{syncycles}(csan)$ are in *direct backward conflict* if $\text{post}_{csan}(S) \cap \text{post}_{csan}(S') \neq \varnothing$.

3. Two nodes $x,y \in P_{csan} \cup Q_{csan} \cup T_{csan}$ are in *conflict*, denoted $x\#y$, if there are $S \neq S' \in \text{syncycles}(csan)$ such that $S\#_0S'$ and $(S \times \{x\}) \cap (F_{csan} \cup W_{csan})^* \neq \varnothing$ and $(S' \times \{y\}) \cap (F_{csan} \cup W_{csan})^* \neq \varnothing$.

4. The *conflict set* of $S \in \text{syncycles}(csan)$ enabled at a marking M of *csan* is

$$\text{conflset}_{csan}(M,S) \triangleq \{S\} \cup \{S' \in \text{enabled}_{csan}(M) \mid S\#_0S'\}.$$

◇

Example 38. For the CSA-net in Figure 4.1, the syn-cycles $S_1 = \{d, f\}$ and $S_2 = \{a\}$ are in conflict, $S_1 \# S_2$, at marking $M = \{p_1, p_6, q_1\}$. ◇

4.7 Calculating probabilities in CSA-nets

In this section, we extend the calculation of probabilities in acyclic nets presented in Section 3.3 to well-formed CSA-nets. In this case, CSA-nets are extended by adding the 'weight' argument ω assigning positive numerical weights to transitions. Then each syn-cycle S is assigned its weight $\omega(S) = \sum_{t \in S} \omega(t)$.

After that, we define the probability of a syn-cycle S enabled at a reachable marking M as the weight of S divided the combined weights of all the syn-cycles that are in conflict with S and are enabled at M, and then calculate the probability of a step U composed of syn-cycles $S_1, \ldots, S_k$ as follows:

$$\begin{array}{lcl} \mathbf{P}_{csan}(M, S) & \triangleq & \frac{\omega(S)}{\sum_{S' \in \text{conflset}_{csan}(M,S)} \omega(S')} \\ \mathbf{P}_{csan}(M, S_1 \uplus \cdots \uplus S_k) & \triangleq & \prod_{i=1}^{k} \mathbf{P}_{csan}(M, S_i) \ . \end{array}$$

We then define the probability of a step sequence $\sigma = U_1 \ldots U_k$ as

$$\mathbf{P}_{csan}(\sigma) \triangleq \mathbf{P}_{csan}(M_0, U_1) \cdot \ldots \cdot \mathbf{P}_{csan}(M_{k-1}, U_k) \ ,$$

where $M_0, \ldots, M_{k-1}$ are such that $M_0 \xrightarrow{U_1} \ldots M_{k-1} \xrightarrow{U_k} M_k$.

Example 39. Figure 4.4 shows a CSA-net with weights (shown inside transitions in conflict). There are two maximal scenarios:

$$cson_1 = \text{scenario}_{csan}(\{e, f, A, D\}) \quad \textit{and} \quad cson_2 = \text{scenario}_{csan}(\{e, f, B, C\}) \ .$$

There are also four syn-cycles: $S_1 = \{e\}$, $S_2 = \{f\}$, $S_3 = \{A, D\}$, and $S_4 = \{B, C\}$. The probabilities of two of the maximal step sequences are as follows:

$$\begin{array}{lcll} \mathbf{P}_{csan}(\{e\}\{f\}\{A,D\}) & = & \frac{\omega(S_1)}{\omega(S_1)} \cdot \frac{\omega(S_2)}{\omega(S_2)} \cdot \frac{\omega(S_3)}{\omega(S_3)+\omega(S_4)} & = \frac{\omega(S_1)}{\omega(S_1)} \cdot \frac{\omega(S_2)}{\omega(S_2)} \cdot \frac{\omega(A)+\omega(D)}{\omega(A)+\omega(D)+\omega(B)+\omega(C)} \\ & = & 1 \cdot 1 \cdot \frac{9}{14} = \frac{9}{14} = 0.6 & \\ \mathbf{P}_{csan}(\{e\}\{f\}\{B,C\}) & = & \frac{\omega(S_1)}{\omega(S_1)} \cdot \frac{\omega(S_2)}{\omega(S_2)} \cdot \frac{\omega(S_4)}{\omega(S_3)+\omega(S_4)} & = \frac{\omega(S_1)}{\omega(S_1)} \cdot \frac{\omega(S_2)}{\omega(S_2)} \cdot \frac{\omega(B)+\omega(C)}{\omega(A)+\omega(D)+\omega(B)+\omega(C)} \\ & = & 1 \cdot 1 \cdot \frac{5}{14} = \frac{5}{14} = 0.4 & \end{array}$$

As mentioned earlier, different step sequences may generate the same scenario. For example, the three step sequences: $\sigma_1 = \{e,f\}\{A,D\}$, $\sigma_2 = \{e\}\{f\}\{A,D\}$, and $\sigma_3 = \{f\}\{e\}\{A,D\}$ generate scenario $cson_1$ with the same probability $\frac{9}{14}$, and so $\mathbf{P}_{csan}(cson_1) = \frac{9}{14}$.
Also, $cson_2$ has three different executions with the same probability. As a result, one can assign probabilities to the two maximal scenarios. It is worth to notice that the sum of the probabilities of the scenarios in this case is as follows: $\mathbf{P}_{csan}(cson_1) + \mathbf{P}_{csan}(cson_2) = 1$. $\diamond$

For the above example, the calculation of probabilities was fully successful. However, this is not always the case as CSA-nets can exhibit *confusion* described next.

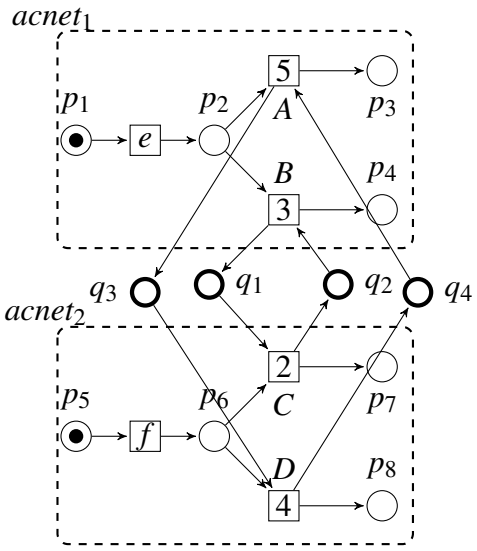


Fig. 4.4 CSA-net with weights.

4.8 Confusion in probabilistic CSA-nets

In this section, the definition of confusion in acyclic nets presented in Section 3.4 is extended to CSA-nets. An acyclic net has a confusion at a reachable marking M whenever a transition is fired and its firing expands or reduces the conflict set of another concurrent transition. Extending this situation to a CSA-net *csan* requires to consider the buffer places as a step in *csan* is enabled if all its input places belonging to the component acyclic nets and all its 'external' input buffer places are included in the marking. Hence, assuming that each acyclic net of *csan* is confusion-free, *csan* can exhibit confusion when communications are added at the level of alternative scenarios. In this case, there is a possibility that a syn-cycle $S \in \text{syncycles}(csan)$ enabled at reachable marking M is fired and its firing influences the

conflict set of another concurrent syn-cycle S' enabled at M. Such a situation is illustrated in the examples below.

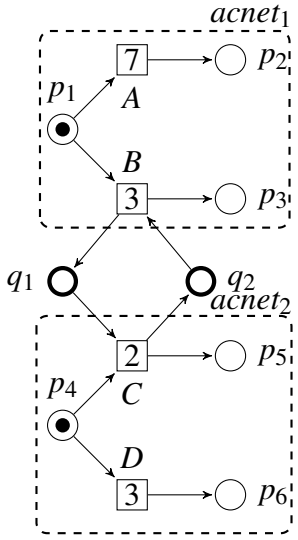


Fig. 4.5 CSA-net with a symmetric confusion.

Example 40. Consider the CSA-net *csan* in Figure 4.5 where the communications captured by the buffer places q_1 and q_2 are added at the level of alternative scenarios. The two possible maximal scenarios are $cson_1 = \text{scenario}_{csan}(\{A,D\})$ and $cson_2 = \text{scenario}_{csan}(\{B,C\})$. Note that in $cson_1$ each transition belongs to a separate syn-cycle: $S_1 = \{A\}$ and $S_2 = \{D\}$. Also it has three maximal step sequences: $\sigma_1 = S_1S_2$, $\sigma_2 = S_2S_1$, and $\sigma_3 = S_1 \cup S_2$. The second scenario has a single maximal step sequence $\sigma_4 = \{B,C\}$ due to the *synchronous communication* that forms the syn-cycle $S_3 = \{B,C\}$.

If σ_1 is executed, the conflict set of S_2 is reduced. More precisely, $\text{conflset}_{csan}(\{p_1,p_4\},S_2) = \{S_3\} \neq \text{conflset}_{csan}(\{p_2,p_4\},S_2) = \varnothing$. Then in σ_1, the probability of S_2 is 1 (D is a certain transition), and so:

$$\mathbf{P}_{csan}(\sigma_1) = \frac{\omega(S_1)}{\omega(S_1)+\omega(S_3)} \cdot \frac{\omega(S_2)}{\omega(S_2)} = \frac{7}{12} \cdot 1 = \frac{7}{12}.$$

However, if σ_2 is executed, then the resulting probability is

$$\mathbf{P}_{csan}(\sigma_2) = \frac{\omega(S_2)}{\omega(S_2)+\omega(S_3)} \cdot \frac{\omega(S_1)}{\omega(S_1)} = \frac{3}{8} \cdot 1 = \frac{3}{8}$$

that is because in σ_2, executing S_2 first modifies the conflict set of S_1. That is, $\text{conflset}_{csan}(\{p_1,p_4\},S_1) = \{S_3\} \neq \text{conflset}_{csan}(\{p_1,p_6\},S_1) = \varnothing$.

As a result, $\mathbf{P}(\sigma_1) \neq \mathbf{P}(\sigma_2)$. Hence one cannot assign probability to $cson_1$.
The behaviour of this CSA-net is similar to the *symmetric confusion* in Figure 3.6 on Page 35. Here, S_1 and S_2 are independent, and firing any of them has influence on the conflict set of the other one, i.e., firing S_1 decreases the conflict set of S_2, which is where the confusion arises. ◇

Below, we provide the formal definition of *confusion* in CSA-nets.

Definition 4.8.1 (confusion in CSA-nets)**.** A well-formed CSA-net *csan* has a *confusion* at a reachable marking M if there are distinct syn-cycles $S_1,S_2,S_3 \in \text{syncycles}(csan)$ such that $S_1,S_2,S_1 \uplus S_2 \in \text{enabled}_{csan}(M)$, $\text{pre}_{csan}(S_1) \cap \text{pre}_{csan}(S_2) = \varnothing$, and one of the following holds:

- $S_1 \#_0 S_3 \#_0 S_2$ and $S_3 \in \text{enabled}_{csan}(M)$.
- $S_2 \#_0 S_3$ and $S_3 \in \text{enabled}_{csan}(M') \setminus \text{enabled}_{csan}(M)$, where $M[S_1\rangle_{csan} M'$.

We then denote

- $\text{symconfused}_{csan}(M,S_1,S_2,S_3)$ in the first *symmetric* case and
- $\text{asymconfused}_{csan}(M,S_1,S_2,S_3)$ in the second *asymmetric* case.

We also say that CSA-net *csan* is *confusion-free* if there is no confusion in all its reachable markings. ◇

Example 41. The CSA-net *csan* in Figure 4.6 exhibits asymmetric confusion represented by $\text{asymconfused}_{csan}(\{p_1,p_4\},S_2,S_4,S_3)$ where the syn-cycles are: $S_1 = \{a\}$, $S_2 = \{b\}$, $S_3 = \{c\}$, and $S_4 = \{d\}$. All possible maximal scenarios are: $cson_1 = \text{scenario}_{csan}(\{a,d\})$, $cson_2 = \text{scenario}_{csan}(\{b,c\})$, and $cson_3 = \text{scenario}_{csan}(\{b,d\})$. The last scenario can be generated by step sequences $\sigma_1 = S_2S_4$ and $\sigma_2 = S_4S_2$, which have different probabilities. That is because in σ_1, executing S_2 first enables S_3, which is in conflict with S_4. That implies that the conflict set of S_4 is increased. $\text{conflset}_{csan}(\{p_1,p_4\},S_4) = \varnothing \neq \text{conflset}_{csan}(\{p_3,p_4,q_1\},S_4) = \{S_3\}$. The resulting probability is:

$$\mathbf{P}_{csan}(\sigma_1) = \frac{\omega(S_2)}{\omega(S_1)+\omega(S_2)} \cdot \frac{\omega(S_4)}{\omega(S_3)+\omega(S_4)} = \frac{4}{10} \cdot \frac{8}{10} = \frac{32}{100}.$$

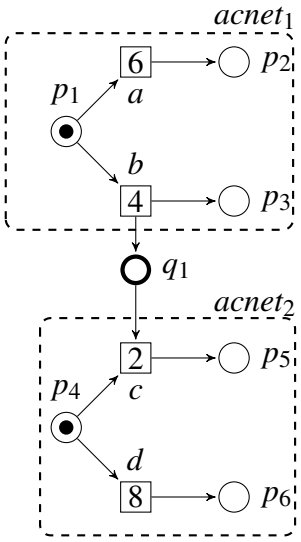


Fig. 4.6 CSA-net with an asymmetric confusion.

However for σ_2, S_4 is executed first with probability 1 as there are no enabled syn-cycles to compete with it for the tokens in the initial marking ($\text{conflset}_{csan}(\{p_1,p_4\},S_4)=\varnothing$). Hence, we have:

$$\mathbf{P}_{csan}(\sigma_2)=\frac{\omega(S_4)}{\omega(S_4)}\cdot\frac{\omega(S_2)}{\omega(S_1)+\omega(S_2)}=1\cdot\frac{4}{10}=\frac{4}{10}.$$

Obviously, $\mathbf{P}_{csan}(\sigma_1)\neq\mathbf{P}_{csan}(\sigma_2)$ which implies that the probability of $\text{scenario}_{csan}(\{b,d\})$ depends on the order of transition executions.

◇

One can infer from the above discussion that the presence of communication in CSA-net can cause confusion which modifies the conflict sets of enabled syn-cycles.

The result below shows that due to the presence of confusion, conflict sets of an enabled syn-cycle can change after executing a concurrent syn-cycle.

Proposition 4.8.1. Let *csan* be a well-formed CSA-net and M be its reachable marking. If $\text{symconfused}_{csan}(M,S_1,S_2,S_3)$ or $\text{asymconfused}_{csan}(M,S_1,S_2,S_3)$, then $\text{conflset}_{csan}(M,S_1)\neq\text{conflset}_{csan}(M',S_1)$ and $S_1\in\text{enabled}_{csan}(M')$, where $M[S_2\rangle_{csan}M'$.

Proof. We will consider the two cases in Definition 4.8.1.

- Let us assume that $\text{symconfused}_{csan}(M,S_1,S_2,S_3)$ holds.
  Then, by the first case of Definition 4.8.1, we have that $\text{pre}_{csan}(S_1)\cap\text{pre}_{csan}(S_2)=\varnothing$, $S_1\#_0S_3\#_0S_2$, and $S_1\uplus S_2\in\text{enabled}_{csan}(M)$.

Since $S_1 \#_0 S_3$ and $S_2 \#_0 S_3$, then $S_3 \in \text{conflset}_{csan}(M, S_1) \cap \text{conflset}_{csan}(M, S_2)$. Executing S_2 disables S_3. So, $S_1 \in \text{enabled}_{csan}(M')$ whereas $S_3 \notin \text{enabled}_{csan}(M')$. Hence,

$$S_3 \in \text{conflset}_{csan}(M, S_1) \setminus \text{conflset}_{csan}(M', S_1)$$

where $S_1 \in \text{enabled}_{csan}(M) \cap \text{enabled}_{csan}(M')$.

- Let us assume that $\text{asymconfused}_{csan}(M, S_1, S_2, S_3)$ holds.
  Then, by the second case of Definition 4.8.1, we have that $\text{pre}_{csan}(S_1) \cap \text{pre}_{csan}(S_2) = \varnothing$, $S_1 \#_0 S_3$, and $S_1 \uplus S_2 \in \text{enabled}_{csan}(M)$, whereas $S_3 \in \text{enabled}_{csan}(M') \setminus \text{enabled}_{csan}(M)$, where $M[S_2\rangle_{csan} M'$. Since $S_1 \#_0 S_3$ and $S_3 \in \text{enabled}_{csan}(M')$, then

$$S_3 \in \text{conflset}_{csan}(M', S_1) \setminus \text{conflset}_{csan}(M, S_1)$$

where $S_1 \in \text{enabled}_{csan}(M) \cap \text{enabled}_{csan}(M')$.

□

In the result below, we show that if a CSA-net has no confusion, then the conflict sets of an enabled syn-cycle are constant for all the executions of a given scenario.

Proposition 4.8.2. Let *csan* be a confusion-free well-formed CSA-net.

1. If M and M' are two reachable markings of *csan* such that $M[\rangle_{csan} M'$, then

$$\text{conflset}_{csan}(M, S) = \text{conflset}_{csan}(M', S) \ ,$$

for every syn-cycle $S \in \text{enabled}_{csan}(M) \cap \text{enabled}_{csan}(M')$.

2. If M and M' are two reachable markings of $cson \in \text{scenarios}(csan)$, then

$$\text{conflset}_{csan}(M, S) = \text{conflset}_{csan}(M', S) \ ,$$

for every syn-cycle $S \in \text{enabled}_{cson}(M) \cap \text{enabled}_{cson}(M')$.

Proof. (1) We first prove this for M' obtained by executing only one syn-cycle and that if $\text{conflset}_{csan}(M, S) \neq \text{conflset}_{acnet}(M', S)$, then *csan* is not confusion-free acyclic net producing a contradiction. There are two cases:

- Let us assume that there is a syn-cycle S' such that

$$S' \in \text{conflset}_{csan}(M, S) \setminus \text{conflset}_{csan}(M', S)$$

where M' is obtained by executing only one syn-cycle S'', so $M[S''\rangle_{csan} M'$. As $S \uplus S'' \in \text{enabled}_{csan}(M)$, $S\#_0 S'\#_0 S''$, and $S' \in \text{enabled}_{csan}(M) \setminus \text{enabled}_{csan}(M')$ then, according to the first part of Definition 4.8.1, $\text{symconfused}_{acnet}(M,S,S',S'')$ holds.

- Let us assume that there is a syn-cycle S' such that

$$S' \in \text{conflset}_{csan}(M',S) \setminus \text{conflset}_{csan}(M,S)$$

where M' is obtained by executing only one syn-cycle S'', so $M[S''\rangle_{csan} M'$. As $S \uplus S'' \in \text{enabled}_{csan}(M)$, $S\#_0 S'$, and $S' \in \text{enabled}_{csan}(M') \setminus \text{enabled}_{csan}(M)$ then, according to the second part of Definition 4.8.1, $\text{asymconfused}_{csan}(M,S,S',S'')$ holds.

The above proof of two cases considers the situation where generating M' requires executing only one syn-cycle. However, this is not always the case. Thus, we might have $M^{init}_{csan}[U_1 \dots U_m\rangle_{csan} M[S_1 \dots S_k\rangle_{csan} M'$, for some syncycles $S_1 \dots S_k$, which means that

$$M = M_0[S_1\rangle_{csan} M_1 \dots M_{k-1}[S_k\rangle_{csan} M_k = M'$$

where S is enabled at every marking M_i (knowing that *csan* is well-formed and using the result of Proposition 4.5.3 implies that $S \in \text{enabled}_{csan}(M_i)$ for $i = 0,\dots,k$). Hence, by what we already demonstrated in this proof, we have:

$$\text{conflset}_{csan}(M_0,S) = \text{conflset}_{csan}(M_1,S) = \dots = \text{conflset}_{csan}(M_k,S).$$

(2) Since it is not guaranteed that M and M' are reachable from each other, then using twice the result of Proposition 8(4) in [8], we conclude that there is a reachable marking M'' such that $M''[\rangle_{cson} M$ and $M''[\rangle_{cson} M'$ and $S \in \text{enabled}_{csan}(M'')$. As both markings M and M' are reachable from M'', then the first part can be applied. So, it follows that

$$\text{conflset}_{csan}(M,S) = \text{conflset}_{csan}(M'',S) = \text{conflset}_{csan}(M',S).$$

Hence, $\text{conflset}_{csan}(M,S) = \text{conflset}_{csan}(M',S)$. □

4.9 Removing confusion from CSA-nets

4.9.1 From CSA-net to acyclic net

In this section, we assume that a well-formed CSA-net has a confusion due to the presence of communication between the syn-cycles involved in the alternative scenarios (i.e., each component acyclic net is confusion-free). Our approach of removing confusion from the CSA-net is to encode it into a behaviourally closely equivalent single acyclic net. This means the whole CSA-net, which is a group of acyclic nets, becomes one large acyclic net such that the original buffer places are regular places. The encoding is based on expanding the underlying transitions involved in *asynchronous* communications. Buffer places of asynchronous communications are considered as regular places.

Transitions communicating *synchronously* are combined into one synchronised transition, and the buffer places involved are removed. The original transitions are also removed. Instead, for each syn-cycle S, a transition τ_S is created (graphically, we may put the transitions of S inside the box representing τ_S). Its preset and postset are those of S except those involved in synchronous communication. If the result is a well-formed *cluster-acyclic* net, then the approaches proposed in Chapter 3 can be applied.

The encoding is done following these steps:

- All the places of *csan* together with their markings are retained.
- Each buffer place becomes a regular place.
- For each syn-cycle $S \in \text{syncycles}(csan)$, a transition τ_S is created. Its presets are the pre-places of S except the buffer places in $\text{pre}_{csan}(S) \cap \text{post}_{csan}(S)$. Its post-sets are the post-places of S except the buffer places in $\text{pre}_{csan}(S) \cap \text{post}_{csan}(S)$.
- The original buffer places with empty pre-sets are removed.

The following definition formally captures the details of the encoding.

Definition 4.9.1. The encoding of a well-formed CSA-net *csan* is an acyclic net $\text{acyclicnet}(csan) \triangleq (P, T, F)$ constructed in the following steps:

- $P \triangleq P_{csan} \cup Q_{csan}$.
- $T \triangleq \{\tau_S \mid S \in \text{syncycles}(csan)\}$.

- F is such that, for every $S \in \text{syncycles}(csan)$:

$$\begin{aligned} \text{pre}_{acnet}(\tau_S) &\triangleq \text{pre}_{csan}(S) \setminus \{q \in Q_{csan} \mid q \in \text{pre}_{csan}(S) \cap \text{post}_{csan}(S)\} \\ \text{post}_{acnet}(\tau_S) &\triangleq \text{post}_{csan}(S) \setminus \{q \in Q_{csan} \mid q \in \text{pre}_{csan}(S) \cap \text{post}_{csan}(S)\} \,. \end{aligned}$$

- Finally, the original buffer places with empty pre-sets are removed from P. ◇

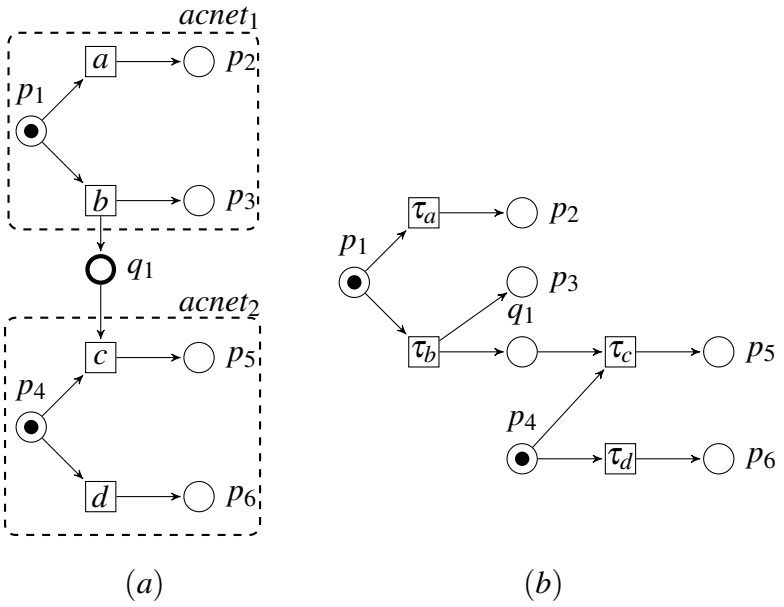


Fig. 4.7 CSA-net *csan* with asymmetric confusion (a) and its encoding to an acyclic net acyclicnet($csan$) (b).

Example 42. The CSA-net in Figure 4.7(a) is the same *csan* in Figure 4.6 and Example 41, which exhibits asymmetric confusion, i.e., we have $\text{asymconfused}_{csan}(\{p_1, p_4\}, S_2, S_4, S_3)$. Recall the four syn-cycles from Example 41:

$$S_1 = \{a\},\ \ S_2 = \{b\},\ \ S_3 = \{c\},\ \ S_4 = \{d\}.$$

To remove confusion from CSA-net, it is translated into a single acyclic net $acnet =$ acyclicnet($csan$), which is depicted in Figure 4.7(b). All the syn-cycles in Figure 4.7(a) are removed and instead a transition τ_S is created for each syn-cycle S. Hence, $S_1 = \tau_a$, $S_2 = \tau_b$,

$S_3 = \tau_c$, and $S_4 = \tau_d$. Moreover,

$$\begin{array}{lcl} \mathrm{pre}_{acnet}(\tau_a) & = & \mathrm{pre}_{csan}(S_1) = \{p_1\} \\ \mathrm{pre}_{acnet}(\tau_b) & = & \mathrm{pre}_{csan}(S_2) = \{p_1\} \\ \mathrm{pre}_{acnet}(\tau_c) & = & \mathrm{pre}_{csan}(S_3) = \{p_4, q_1\} \\ \mathrm{pre}_{acnet}(\tau_d) & = & \mathrm{pre}_{csan}(S_4) = \{p_4\}. \end{array}$$

Note that despite the fact that there is an asynchronous communication between S_2 and S_3 and they might be executed simultaneously, they are encoded as singleton transitions (not combined) and the buffer place q_1 becomes a regular place. The significant fact is that the behaviour of the constructed acyclic net is closely linked to the original *csan*. First, τ_a, τ_b, τ_d are enabled and the transitions of $\mathrm{scenario}_{acnet}(\{\tau_a, \tau_d\})$ and $\mathrm{scenario}_{acnet}(\{\tau_b, \tau_d\})$ can be executed in any order. The global causality between S_2 and S_3 captured by the buffer place q_1 is simulated by their corresponding transitions τ_b and τ_c in Figure 4.7(b) where q_1 is a regular place. ◇

The step sequences executed by acyclicnet$(csan)$ and the step sequences executed by *csan* are related to each other by using the following notation. For every set of transitions U of acyclicnet$(csan)$, T_U is the set of transitions of *csan* given by:

$$T_U \triangleq \bigcup \{S \mid \tau_S \in U\}.$$

Below we show a structural property of the net acyclicnet$(csan)$ obtained according to Definition 4.9.1.

Proposition 4.9.1. Let *csan* and acyclicnet$(csan)$ be as in Definition 4.9.1. If U is a step of the acyclic net acyclicnet$(csan)$, then T_U is a step of *csan* such that ${}^\bullet U \subseteq \mathrm{pre}_{csan}(T_U)$ and $U^\bullet = \mathrm{post}_{csan}(T_U)$.

Proof. It follows directly from Definition 4.9.1. □

Hence, the result below shows that the executions of acyclicnet$(csan)$ have direct representationss in the executions of *csan*.

Proposition 4.9.2. Let *csan* and acyclicnet$(csan)$ be as in Definition 4.9.1. Moreover, let

$$(M^{init}_{\mathrm{acyclicnet}(csan)}) = M_0 U_1 M_1 \ldots M_{k-1} U_k M_k$$

be a mixed step sequence of acyclicnet$(csan)$. Then there is a marking M of *csan* such that $M^{init}_{csan}[T_{U_1} \ldots T_{U_k}\rangle_{csan} M$ and $M_k \subseteq M$.

Proof. By induction on the length of the mixed step sequence, as follows:

- Base case ($k = 0$): The result holds as acyclicnet($csan$) generates the empty step sequence and $M^{init}_{\text{acyclicnet}(csan)} \subseteq M^{init}_{csan}$.
- Inductive case ($k+1$): Let

$$M^{init}_{\text{acyclicnet}(csan)}[U_1 \dots U_k\rangle_{\text{acyclicnet}(csan)} M_k[U_{k+1}\rangle_{\text{acyclicnet}(csan)} M_{k+1}.$$

  By the inductive hypothesis, there is a making M of $csan$ such that

$$M^{init}_{csan}[T_{U_1} \dots T_{U_k}\rangle_{csan} M$$

  and $M_k \subseteq M$. Moreover, by Proposition 4.9.1, $T_{U_{k+1}}$ is a step of $csan$ such that ${}^\bullet U_{k+1} \subseteq \text{pre}_{csan}(T_{U_{k+1}})$. Hence $T_{U_{k+1}}$ is enabled at M, and so there is a marking M' of $csan$ such that

$$M^{init}_{csan}[T_{U_1} \dots T_{U_k}\rangle_{csan} M[T_{U_{k+1}}\rangle_{acnet} M'.$$

  In addition, by Proposition 4.9.1, $\text{post}_{csan}(T_{U_{k+1}}) = U^\bullet_{k+1}$. As we also have $M_k \subseteq M$ and ${}^\bullet U_{k+1} \subseteq \text{pre}_{csan}(T_{U_{k+1}})$, it follows that $M_{k+1} \subseteq M'$.

□

Proposition 4.9.2 implies that all the executions of acyclicnet($csan$) can be simulated by the executions of $csan$.

Theorem 4.9.1. Let $csan$ and acyclicnet($csan$) be as Definition 4.9.1. Then:

$$\{T_{U_1} \dots T_{U_k} \mid U_1 \dots U_k \in \text{sseq}(\text{acyclicnet}(csan))\} \subseteq \text{sseq}(csan).$$

Together with the following result it demonstrates that the behaviour of the constructed acyclic net acyclicnet($csan$) is closely related to the behaviour of original $csan$.

Theorem 4.9.2. Let $csan$ and $acnet = \text{acyclicnet}(csan)$ be as in Definition 4.9.1. Then for every maximal scenario $cson \in \text{maxscenarios}(csan)$, there is a maximal step sequence $U_1 \dots U_k$ of $acnet$ such that $T_{U_1} \dots T_{U_k}$ is a maximal step sequence of $cson$.

Proof. We know from results proven in [8] that there exists a maximal step sequence $S_1 \dots S_k$ of $cson$ such that each S_i is a syn-cycle of $cson$, and so also a syn-cycle of $csan$. Let $U_i = \{\tau_{S_i}\}$, for every $1 \leq i \leq k$. We show by induction on $m \leq k$ that $U_1 \dots U_m$ is a step

sequence of *acnet* leading to the same marking M_m as $S_1 \dots S_m$, for each $m \leq k$. Note that, as required, $T_{U_i} = S_i$ for every $1 \leq i \leq k$.

- Base case: The result holds for $m = 0$ as there is an empty step sequence of *acnet*.
- Inductive Case: We assume that $m > 1$ and the result holds for $m-1$. By the induction hypothesis, $U_1 \dots U_{m-1}$ is a step sequence of *acnet* leading to the same marking M_{m-1} as $S_1 \dots S_{m-1}$. It then follows from Definition 4.9.1 that U_m is enabled at M_{m-1} and its firing leads to the same marking M_m as the firing of $S_1 \dots S_m$.

We further observe that, by Theorem 4.9.1, $U_1 \dots U_k$ is a maximal step sequence of *acnet*. □

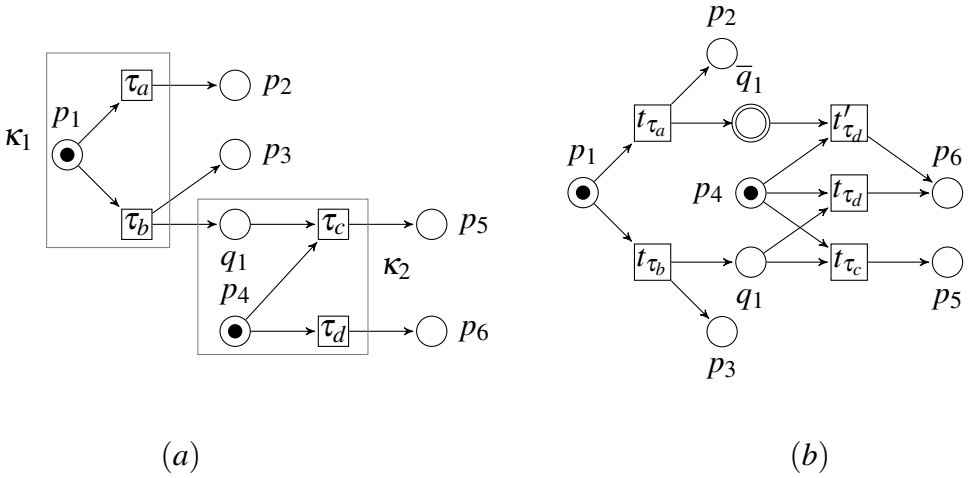


Fig. 4.8 A cluster-acyclic net for acyclicnet($csan$) in Figure 4.7 (a) and its encoding according to Approach A (b).

Next, if acyclicnet($csan$) is a well-formed acyclic net whose clusters are acyclic, then we apply the construction according to Approach A in Section 3.6.1.

Example 43. Let *csan* be CSA-net. The first stage of the approach of removing confusion in *csan* (i.e., translating into an acyclic net) is presented in Example 42. In this example we introduce the second and the third stage of the approach as it is portrayed in Figure 4.8. In Figure 4.8(a), acyclicnet($csan$) is partitioned into its maximal clusters: $\kappa_1 = \{\tau_a, \tau_b\}$, and $\kappa_2 = \{\tau_c, \tau_d\}$. It is worth to notice that the clusters acyclicity in Definition 3.5 is satisfied, i.e., $\kappa_1 \sqsubset \kappa_2$ since $\sqsubset$ is a strict partial order on the maximal clusters. The associated maximal

transactions for each maximal cluster are:

$$\begin{aligned} \text{trans}(\kappa_1, \{p_1\}) &= \{\{\tau_a\}, \{\tau_b\}\} \\ \text{trans}(\kappa_2, \{p_4, q_1\}) &= \{\{\tau_c\}, \{\tau_d\}\} \\ \text{trans}(\kappa_2, \{p_4\}) &= \{\{\tau_d\}\}. \end{aligned}$$

The net in Figure 4.8(b) represents the third stage, which is confreeA(acyclicnet($csan$)), the encoding using Approach *A* proposed in Section 3.6.1. The construction is similar to the net in Figure 3.9 in Example 14. ◇

Similarly, removing the symmetric confusion with the synchronous communication discussed in Example 40 in Figure 4.5 is illustrated in the following example.

Example 44. The encoding acyclicnet($csan$) of CSA-net *csan* in Figure 4.5 where the syn-cycles are as follow: $S_1 = \{A\}, S_2 = \{D\}$, and $S_3 = \{B,C\}$ is represented in Figure 4.9(a). All the syn-cycles are deleted, and a new transition is created for each transition included in the syn-cycles S. Hence, τ_A, τ_D, and τ_{BC} are generated. Note that the transitions of $S_3 = \{B,C\}$ involve in a synchronous communication are combined together in a synchronised transition τ_{BC}, hence the buffer places are removed. There is only one cluster $\kappa = \{\tau_A, \tau_{BC}, \tau_D\}$ and its maximal transactions are $\text{trans}(\kappa, \{p_1, p_4\}) = \{\{\tau_A, \tau_D\}, \{\tau_{BC}\}\}$. The confusion-free version is depicted in Figure 4.9(b). Hence, for each maximal transaction in κ, these transitions are created: $t_{\kappa,\{\tau_A\tau_D\},\{p_1,p_4\}}$ and $t_{\kappa,\{\tau_{BC}\},\{p_1,p_4\}}$, which are $t_{\tau_{AD}}$ and $t_{\tau_{BC}}$ respectively as shown in Figure 4.9(b).

Note that the weights associated with the syn-cycles in the original CSA-net are not affected by the translation. That is, the probabilities still can be calculated for the confusion-free version. Hence, the corresponding scenario for the first scenario $cson_1 = \text{scenario}_{csan}(\{A,D\})$ in the original CSA-net in Figure 4.5 is $\text{scenario}_{\text{confreeA(acyclicnet}(csan))}(\{t_{\tau_{AD}}\})$, which is executed by only one step sequence $\sigma_1 = \{t_{\tau_{AD}}\}$ with probability:

$$\mathbf{P}_{\text{confreeA(acyclicnet}(csan))}(\sigma_1) = \frac{\omega(t_{\tau_{AD}})}{\omega(t_{\tau_{AD}}) + \omega(t_{\tau_{BC}})} = \frac{7+3}{7+3+3+2} = \frac{10}{15} = 0.7$$

Similarly, the corresponding scenario for the second scenario $cson_2 = \text{scenario}_{csan}(\{B,C\})$ in the original CSA-net in Figure 4.5 is $\text{scenario}_{\text{confreeA(acyclicnet}(csan))}(\{t_{\tau_{BC}}\})$ which is executed by only one step sequence $\sigma_2 = \{t_{\tau_{BC}}\}$ with probability:

$$\mathbf{P}_{\text{confreeA(acyclicnet}(csan))}(\sigma_2) = \frac{\omega(t_{\tau_{BC}})}{\omega(t_{\tau_{AD}}) + \omega(t_{\tau_{BC}})} = \frac{3+2}{7+3+3+2} = \frac{5}{15} = 0.3$$

It is worth to notice that $\mathbf{P}_{\text{confreeA(acyclicnet}(csan))}(\sigma_1) + \mathbf{P}_{\text{confreeA(acyclicnet}(csan))}(\sigma_2) = 1$. $\diamond$

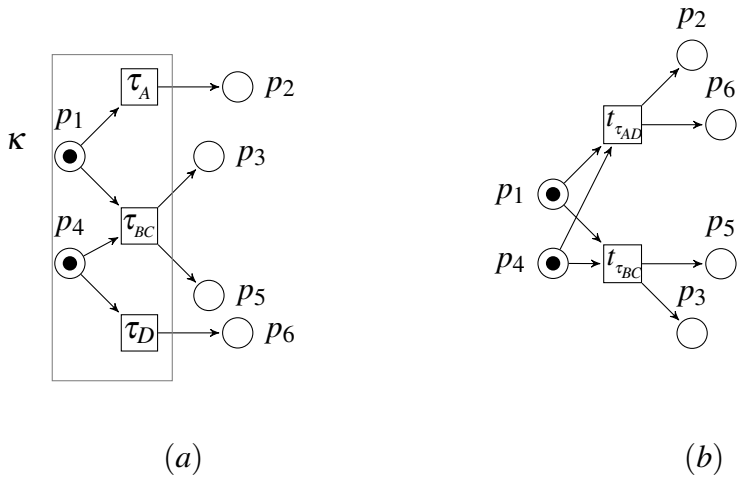


Fig. 4.9 A cluster-acyclic net for acyclicnet($csan$) in Figure 4.5 (a) and its encoding according to Approach A (b).

The proposed approaches for removing confusion in Section 3.6 are based on transforming a confused acyclic nets into another one that is confusion-free. This section also applies the same technique to eliminate confusion in CSA-nets. Since CSA-nets consist of sets of acyclic nets, encoding large confused CSA-nets into acyclic nets is more complex than the encoding of confused acyclic nets. In the next section, another approach of handling confusion in CSA-nets is introduced for the *cascading* CSA-net, a sub-class of CSA-nets where the acyclicity constraint is lifted to the level of CSA-nets.

4.10 Cascading CSA-nets

Consider again the CSA-net $csan$ in Figure 4.7 (a) and its encoding acyclicnet($csan$) in Figure 4.8 (a), where the acyclicity constraint over the clusters holds. Therefore, it is possible to reuse the proposed approaches discussed in Chapter 3 after transforming $csan$ into acyclicnet($csan$). There is also an intuition that the clusters acyclicity constraint can be established at the level of CSA-net. This happens, in particular, when the *asynchronous* communication between the transitions of component cluster-acyclic nets is *unidirectional*. That is, for example, if we have $acnet_i$ and $acnet_j$ in $csan$ such that there are no $t \in T_{acnet_j}$ and $u \in T_{acnet_i}$ satisfying $(t,u) \in W^2_{csan}$, for all $i < j$. The nets in Figure 4.6 and Figure 4.7(a)

adhere to this condition. Such a structure can be seen as an *information cascade* where decisions are made and propagated sequentially from one acyclic net to another.

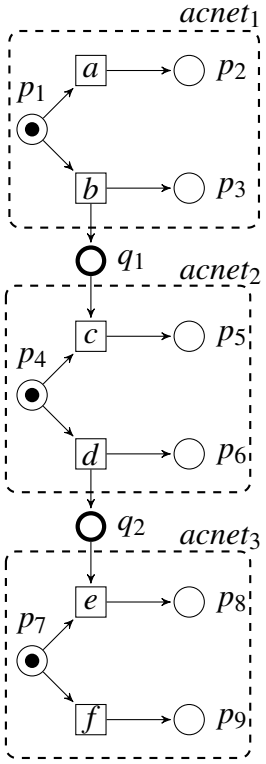


Fig. 4.10 Cascading CSA-net.

Definition 4.10.1 (cascading CSA-net)**.** Let $csan = (acnet_1, \ldots, acnet_n, Q, W)$ $(n \geq 1)$ be a well-formed CSA-net composed of confusion-free acyclic nets such that the following hold:

1. For all $1 \leq i < j \leq n$, there are no $t \in T_{acnet_j}$ and $u \in T_{acnet_i}$ such that $(t,u) \in W^2_{csan}$.
2. $|\operatorname{pre}_{csan}(q)| = |\operatorname{post}_{csan}(q)| = 1$, for every $q \in Q$.
3. If $\operatorname{asymconfused}_{csan}(M, S_1, S_2, S_3)$ is an asymmetric confusion in $csan$, then we have $\operatorname{pre}_{csan}(S_2) \setminus Q = \operatorname{pre}_{csan}(S) \setminus Q$, for every syn-cycle S such that $S\#_0 S_2$. ◇

Example 45. The CSA-net in Figure 4.10 is cascading. The communication between the two component acyclic nets is unidirectional. ◇

The result below shows that the encoding of a cascading well-formed CSA-net produces a cluster-acyclic net.

Proposition 4.10.1. Let *csan* be as in Definition 4.10.1. Then acyclicnet(*csan*) is cluster-acyclic. Moreover, *csan* has no symmetric confusion, and each syn-cycle is a singleton set.

Proof. It follows directly from the definitions. □

Since the syn-cycles of cascading CSA-nets are singletons, we will often refer to the transitions of cascading CSA-nets instead of their syn-cycles in the next section.

4.10.1 Approach C: removing confusion from cascading CSA-nets

In previous section, we introduced an approach to remove confusion from CSA-nets by translating them into single acyclic nets.

In this section, we assume that $csan = (acnet_1, \ldots, acnet_n, Q, W)$ is a cascading CSA-net with asymmetric confusion caused by asynchronous communications (i.e., the component acyclic nets are confusion-free). In such a case, one can remove confusion from *csan* by adding new buffer places between $acnet_i$ and $acnet_j$, for $i < j$, inducing asynchronous communication. Such an approach of removing confusion avoids generating negative places and translating the whole CSA-net into a single acyclic net which might be very large. Also, the transformation retains the structure of being cascading CSA-net and utilizes the buffer places to handle confusion.

Example 46. Consider the confused cascading CSA-net in Figure 4.10, where a confusion arises due to the possibility of executing d before c is enabled (the same holds for e and f). This behaviour is excluded in Figure 4.11, where new buffer places and arcs are shown in gray. More precisely, in Figure 4.11, the execution of d is delayed until the conflict between a and b is resolved. As a result, d and c are enabled together. Basically, if a is chosen over b, then this yields $\{p_2, p_4, q_1'\}$, which leads to enabling d only. On the other hand, choosing b yields $\{p_3, p_4, q_1', q_1\}$, which enables both c and d together.

Note that e and f are treated similarly as their firing is postponed until the decision at $acnet_2$ has been taken. ◇

Example 46 shows how adding asynchronous communication for cascading CSA-net produces effect similar to that present in *extended free-choice* nets. In this case, the presets (including the buffer places) of all the transitions involved are not *exactly* identical, but they are fed with tokens coming from the same sources.

The construction of a confusion-free cascading CSA-net is formally defined below (see Proposition 4.10.1).

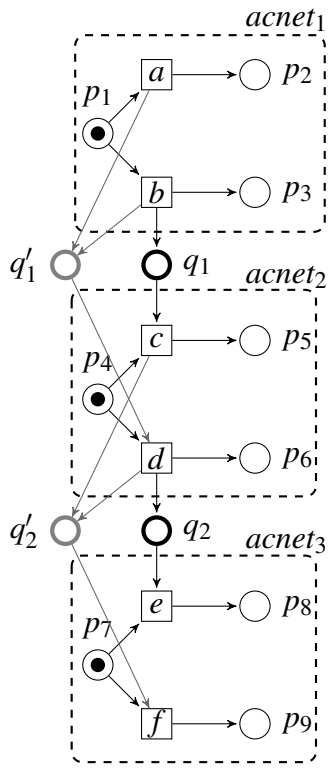


Fig. 4.11 Encoding a confused cascading CSA-net in Figure 4.10 according to Approach *C*.

Definition 4.10.2 (encoding cascading CSA-net)**.** Let *csan* be a cascading CSA-net as in Definition 4.10.1 and, for every transition $t \in T_{csan}$, let $\#_{csan}(t) \triangleq \{w \in T_{csan} \mid t\#_0 w\}$. Then, the *confusion-free encoding* of *csan* is

$$\text{confreeC}\,(csan) = (acnet_1, \ldots, acnet_n, Q \cup Q', W \cup W')$$

constructed as follows. For every asymmetric confusion $\text{asymconfused}_{csan}(M,t,u,v)$, we add to Q' new buffer places $Q_{tuv} = \{q^w_{tuv} \mid w \in \#_{csan}(u) \setminus \{v\}\}$ such that:

$$\text{pre}_{\text{confreeC}\,(csan)}\,(q^w_{tuv}) = \#_{csan}(t) \;\; \textit{and} \;\; \text{post}_{\text{confreeC}\,(csan)}\,(q^w_{tuv}) = \{w\}\ ,$$

for every $q^w_{tuv} \in Q_\alpha$. ◇

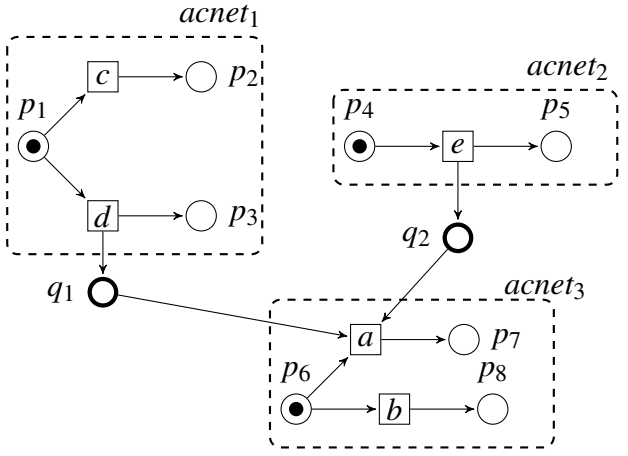
Note that syn-cycles (i.e., transitions, as stated in Proposition 4.10.1) are the same in *csan* and $\text{confreeC}\,(csan)$, and all the places of *csan* with their markings are retained.

In Definition 4.10.2, according to Definition 4.8.1, $t,u \in \text{enabled}_{csan}(M)$, $u\#_0 v$ and

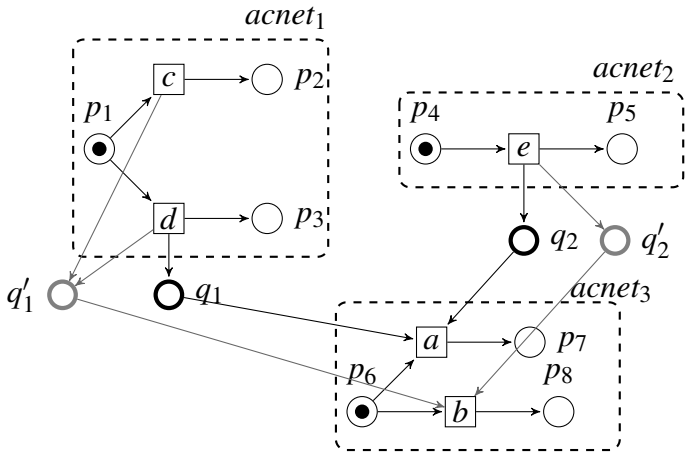
$$v \in \text{enabled}_{csan}(M') \setminus \text{enabled}_{csan}(M), \;\; \textit{where} \;\; M[t\rangle_{csan} M'.$$

Adding new buffer places Q_{tuv}, implies that both u and v are now enabled together at the marking M', and avoiding the situation where u is executed before v is enabled.

The examples below show the construction steps of confreeC$(csan)$ together with probabilities calculation.



(a)



(b)

Fig. 4.12 Cascading *csan* with asymmetric confusion (a) and its encoding according to Approach C (b).

Example 47. The cascading CSA-net in Figure 4.12 (a) has the two confusions:

$$\text{asymconfused}_{csan}(\{p_1, p_5, p_6, q_2\}, d, b, a) \;\; \textit{and} \;\; \text{asymconfused}_{csan}(\{p_3, p_4, p_6, q_1\}, e, b, a).$$

Hence, in this case, $Q_{dba} = \{q^b_{dba}\}$ and $Q_{eba} = \{q^b_{eba}\}$, where $q^b_{dba} = q'_1$ and $q^b_{eba} = q'_2$ are as in Figure 4.12 (b). Adding these buffer places postpones the execution of b until the conflict between c and d is resolved. ◇

The following example shows how probabilities can calculated in confreeC$(csan)$.

Example 48. Consider the confused cascading *csan* in Figure 4.13 (a), where asymmetric confusion arises due to the possibility of executing b before a is enabled. (The same holds for the firing of h or g before the enabledness of f.)
To remove confusion, a buffer place q'_1 is added, as shown in Figure 4.13 (b). Similarly, q'_2 and q'_3 are added to remove other cases of confusion (note that some simplifications have also been applied).
Let the associated weights be as follows: $\omega(a) = 5$, $\omega(b) = 5$, $\omega(c) = 3$, $\omega(d) = 7$, $\omega(e) = 1$, $\omega(f) = 2$, $\omega(g) = 2$, and $\omega(h) = 6$.
One of the maximal scenarios is $cson_1 = \text{scenario}_{csan}(\{d,b,e,f\})$ which can be executed by different maximal step sequences, e.g., $\sigma_1 = \{d\}\{b\}\{e\}\{f\}$ and $\sigma_2 = \{b\}\{d\}\{e\}\{f\}$. In σ_1, d is executed first which enables a in conflict with b. Hence, $\mathbf{P}_{csan}(\sigma_1) = \frac{7}{10} \cdot \frac{5}{10} \cdot \frac{2}{10} = \frac{70}{1000}$. On the other hand, $\mathbf{P}_{csan}(\sigma_2) = \frac{7}{10} \cdot \frac{2}{10} = \frac{14}{100}$. In Figure 4.13($b$), the same scenario can be simulated in only one maximal step sequence which is σ_1 as the possibility of firing b before d (and before a becomes enabled) is excluded due to the additional causality between d and b captured by the new buffer place q'_1.
In this way, the maximal scenario $cson_1$ in the original CSA-net has a corresponding maximal scenario in confreeC$(csan)$ with only one execution and its probability is always the same. The rest of the scenarios are treated similarly. ◇

Considering the cascading CSA-nets as a sub-class of CSA-nets has two advantages: (i) the encoding into acyclicnet$(csan)$ always produces a cluster-acyclic net (see Proposition 4.10.1), and (ii) the acyclicity constraint can be lifted to the level of CSA-net to produce a confusion-free version (see Definition 4.10.2). Also, producing confusion-free version of a CSA-net using Approach C as in Definition 4.10.2 is superior to the technique of removing confusion according to Definitions 4.9.1 and 3.6.1. The reason is that Approach C maintains the structure of cascading CSA-nets and no new transitions nor (negative) places are added. Finally, it is applicable even if the confused CSA-net is not backward-deterministic whereas obtaining confusion-free version of CSA-net using Approaches A and B requires backward-determinism.

The next example discusses Approaches A and C of handling confusion in cascading CSA-nets.

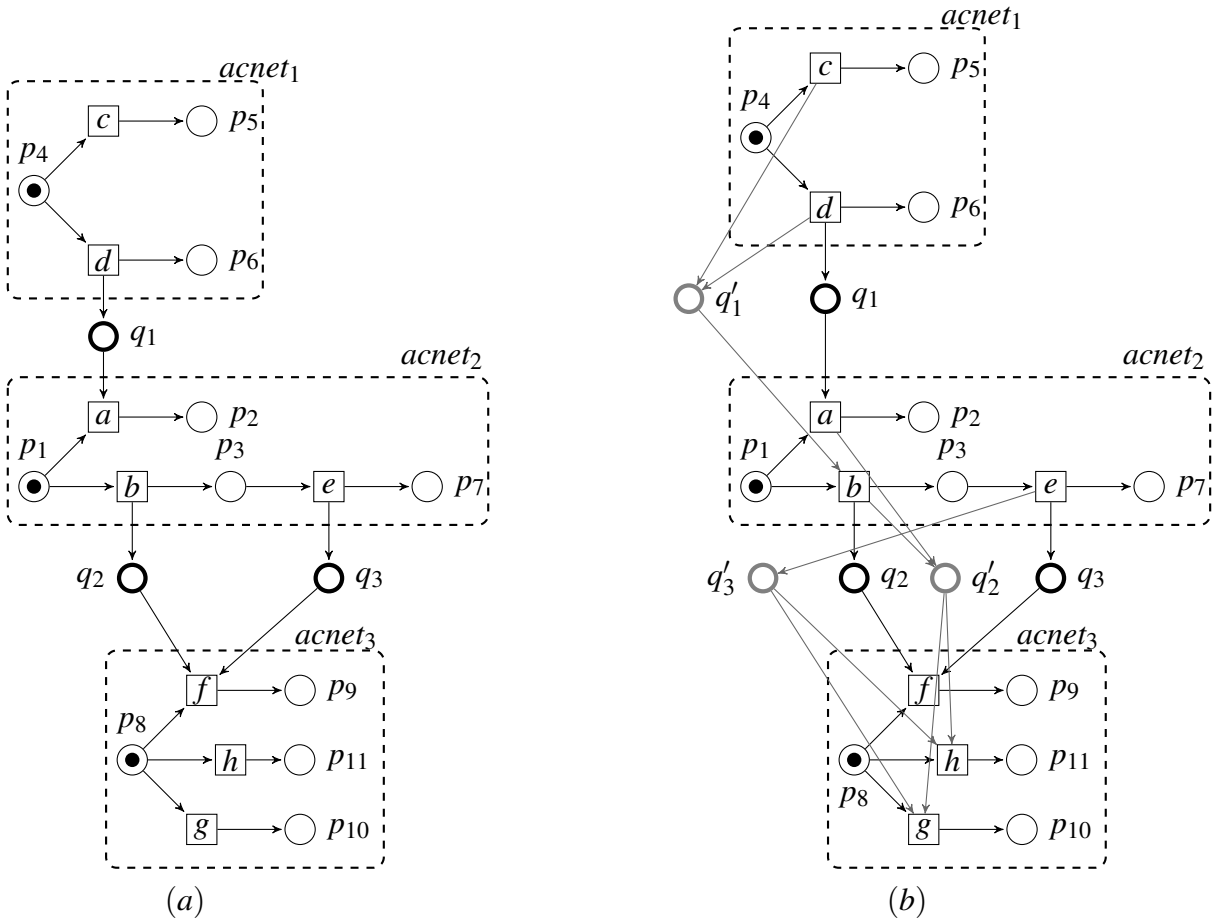


Fig. 4.13 Cascading *csan* with asymmetric confusion (a) and its encoding according to Approach *C* (b).

Example 49. Figure 4.13(b) and Figure 4.14 show the confusion-free versions of the cascading CSA-net *csan* in Figure 4.13(a). The former version is obtained by using Approach *C* according to Definition 4.10.2, where only buffer places are added and generating acyclicnet$(csan)$ is avoided. The latter version is obtained according to Approach *A* and Definitions 4.9.1 and Definition 3.6.1. In both techniques the *acyclicity* constraint is essential. Notice that Figure 4.14 produces a large acyclic net due to generating additional (negative) places and transitions. More importantly, the maximal scenario $cson_1 = \text{scenario}_{csan}(\{d,b,e,f\})$ in the original CSA-net in Figure 4.13(a) can be mapped to its corresponding maximal scenario $\text{scenario}_{\text{confreeA}(\text{acyclicnet}(csan))}(\{t_{\tau_d}, t_{\tau_b}, t_{\tau_e}, t_{\tau_f}\})$ in Figure 4.14 which is generated by only one maximal step sequence $\sigma = \{t_{\tau_d}\}\{t_{\tau_b}\}\{t_{\tau_e}\}\{t_{\tau_f}\}$ with probability $\mathbf{P}_{\text{confreeA}(\text{acyclicnet}(csan))} = \frac{7}{10} \cdot \frac{5}{10} \cdot \frac{2}{10} = \frac{70}{1000}$, where the weights are as in Example 48. $\diamond$

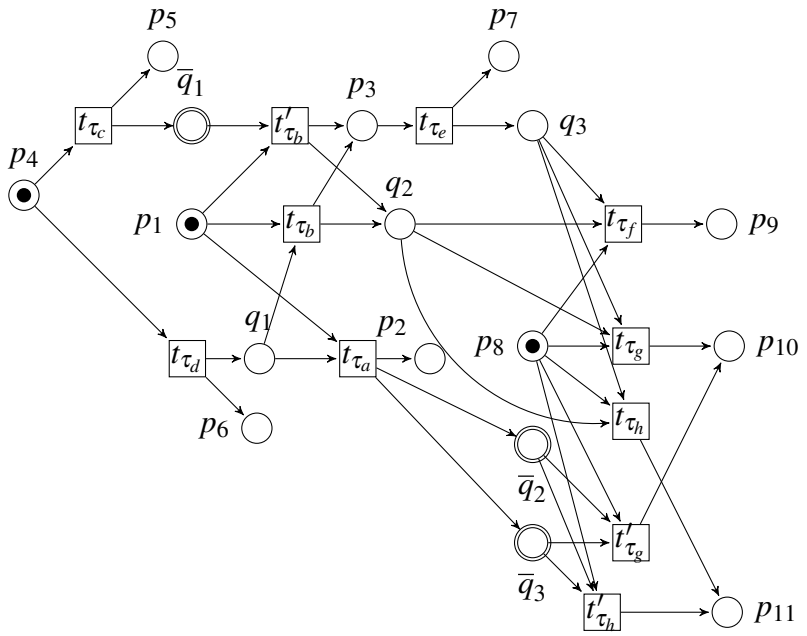


Fig. 4.14 Encoding confused cascading CSA-net in Figure 4.13 (a) according to Approach A.

Next we show an important structural property for the construction in Definition 4.10.2.

Proposition 4.10.2. Let $csan' = \text{confreeC}(csan)$ be a cascading CSA-net as in Definition 4.10.2, and $\text{asymconfused}_{csan}(M,t,u,v)$ be a confusion of $csan$. Moreover, let M' be a reachable marking of $csan'$ such that $\#_{csan}(u) \cap \text{enabled}_{csan'}(M') \neq \varnothing$. Then, for every marking M'' reachable from M' satisfying $\#_{csan}(u) \cap \text{enabled}_{csan'}(M'') \neq \varnothing$, we have:

$$\text{enabled}_{csan'}(M') \cap \#_{csan}(u) = \text{enabled}_{csan'}(M'') \cap \#_{csan}(u) \ .$$

Proof. It follows directly from Definition 4.10.1 and Definition 4.10.2. □

The result below shows that $\text{confreeC}(csan)$ is confusion-free for every cascading $csan$.

Theorem 4.10.1. $\text{confreeC}(csan)$ is confusion-free for every cascading CSA-net $csan$.

Proof. It follows directly from Proposition 4.10.2 that for every asymmetric confusion $\text{asymconfused}_{csan}(M,t,u,v)$ in $csan$ there is no asymmetric confusion of the the form $\text{asymconfused}_{csan}(M',z,w,v)$ or $\text{asymconfused}_{csan}(M',z,u,w)$ (for $w \in \{u,v\}$). Moreover, there is no confusion in $csan'$ involving choices not affected by confusion in $csan$ since the construction in Definition 4.10.2 does not introduce new asymmetric confusions. Hence $\text{confreeC}(csan)$ is confusion-free. □

Next we show that the behaviours of a well-formed confused cascading CSA-net *csan* and $csan' = \text{confreeC}(csan)$ are closely related. In particular, we demonstrate that each maximal scenario of cascading *csan* corresponds to at least one maximal scenario of $\text{confreeC}(csan)$. Hence, the executions of the confusion-free $\text{confreeC}(csan)$ can be used to assign probabilities to the maximal scenarios of *csan*.

Theorem 4.10.2. Let *csan* and $csan' = \text{confreeC}(csan)$ be as in Definition 4.10.2.

1. All step sequences of $csan'$ are also step sequences of *csan*.
2. For every $cson \in \text{maxscenarios}(csan)$, there is a maximal step sequence of $csan'$ which is a maximal step sequence of *cson*.

Proof. (1) This follows directly from the definitions.

(2) Let σ be any maximal step sequence of *cson*. First we can split all non-singleton steps of σ so that each step is now a sigleton obtaining a maximal step sequence σ'. Then, due to the acyclicity of communication in *csan*, one can rearrange the order of steps in σ' so that it has the form $\sigma_1 \dots \sigma_k$, where each σ_i is a step sequence of $acnet_i$. Then, from the acyclicity of communication in $csan'$ it follows that $\sigma_1 \dots \sigma_k$ is a step sequence of $csan'$. □

Example 50. Consider again the confusion-free cascading CSA-net in Figure 4.11 whose behaviour *closely* simulates the behaviour of the original CSA-net in Figure 4.10. The transitions a and b are enabled at the initial marking. Executing a enables d only. In the original *csan* in Figure 4.10, d is enabled at the initial marking and can be executed before a, however, as they are concurrent, there is also possibility that it can executed after a or they are executed together. Hence, the confusion-free version in (b) simulates only one execution that eliminates the confusion. In other words, the additional global causality between a and d is captured by the new buffer place q_1' excludes the possibility of executing them concurrently or d before a, which is in fact has no significant effects on the behaviour in general. Similarly, the enabledness of e and f is based on arriving the tokens at buffer places q_2 and q_2' which are generated by executing d. Executing c, on the other hand, leads to enable f, which is also enabled at the initial marking in the original *csan* in Figure 4.10. In spite of that, the confusion-free *csan* in Figure 4.11 still produces one possible execution of the all the maximal scenarios of the original *csan*. ◇

Based on the above discussion, the confusion-free versions of CSA-nets that are produced by Definitions 4.9.1, 3.6.1, and 4.10.2 maintain the behaviour of the original one by considering all possible maximal scenarios. Basically, if a cascading *csan* is confused, then

its confusion-free versions confreeA(acyclicnet(*csan*)) and confreeC(*csan*) can still mimic *csan*'s behaviour at the level of the *processes* that can be generated.

4.11 Unfolding CSA-nets

In Section 3.7, we introduced unfolding as an additional step for the confused acyclic nets that are not backward deterministic before the encoding into confusion-free acyclic nets. Similarly, if confused CSA-net is not backward deterministic, the unfolding should be performed before removing the confusion. [84] proposed the unfolding algorithm for *Communication Structured Place Transitions Nets* (CSPT-nets) introduced in [73]. The branching processes of the unfolding are the processes of CSPT-nets that are represented by CSO-nets. Like for the unfolding of the standard Petri nets, CSPT-nets unfolding provides complete details of the reachability of the original net. However, in contrast with the standard unfolding, buffer places are taken into consideration in the process of producing CSPT-nets unfolding. Hence, as an interesting observation, it is efficient to construct the acyclicnet(*csan*) and then generate the standard unfolding, instead of generating the unfolding for each component acyclic net of CSA-net then transforming the resulting net into an acyclic net to remove the confusion using the approaches in Sections 3.6. More precisely, if *csan* is a confused non-backward deterministic CSA-net, then encoding it into acyclicnet(*csan*) and then applying the standard unfolding is more effective than unfolding each component acyclic net separately. The next example illustrates the difference.

Example 51. Consider the confused CSA-net *csan* in Figure 4.15(a) and its unfolding in Figure 4.15(b) that has been generated for each component acyclic net separately (according to the unfolding algorithm in [84]). Then, extra buffer places and arcs are added to each split transition involved in the syn-cycle $S = \{e, f\}$. Figure 4.16 represents the transformation into an acyclic net for the unfolding *csan* in Figure 4.15(b). The net in Figure 4.17 is the transformed acyclic net acyclicnet(*csan*) according to Definition 4.9.1, for *csan* in Figure 4.15(a), where the transitions in the syn-cycle S are combined into a synchronised transition ef and S is removed together with two buffer places. Moreover, since any confused *csan* is transferred into an acyclicnet(*csan*) to remove confusion using the approaches in Section 3.6, then any additional buffer places together with their arcs that are generated when constructing the unfolding for each acyclic net would be collapsed. Therefore, constructing unfolding requires extra steps especially in terms of generating many new nodes which eventually would be collapsed.

It is also worth noting that removing confusion in this CSA-net using Approach *C* is even more efficient as generating the unfolding is not essential. ◇

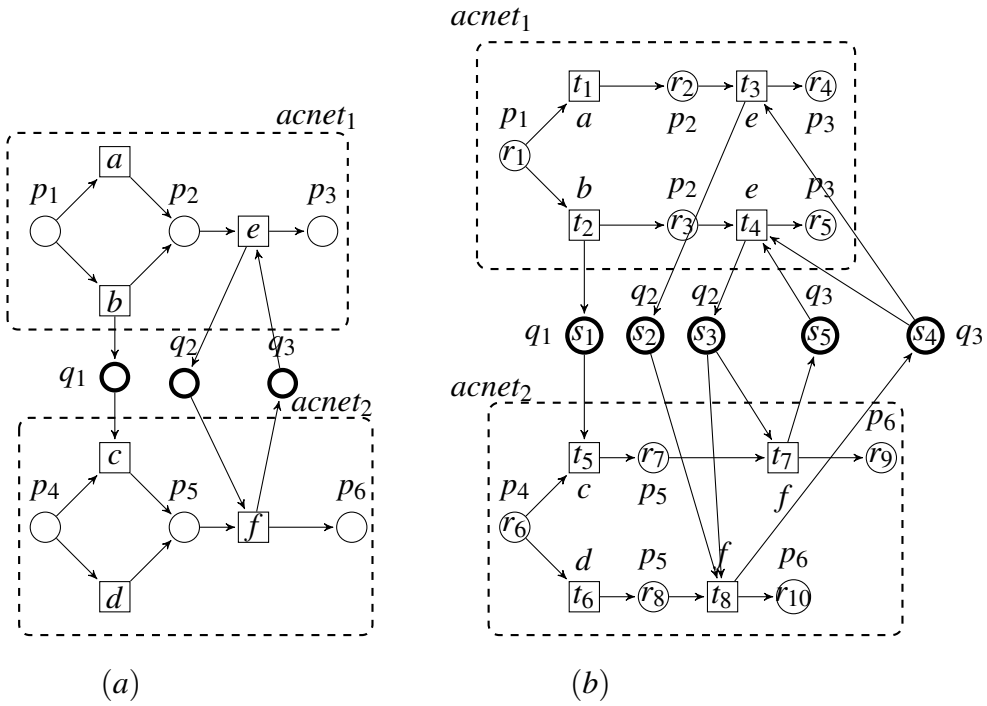


Fig. 4.15 CSA-net with confusion (a) and its unfolding in (b).

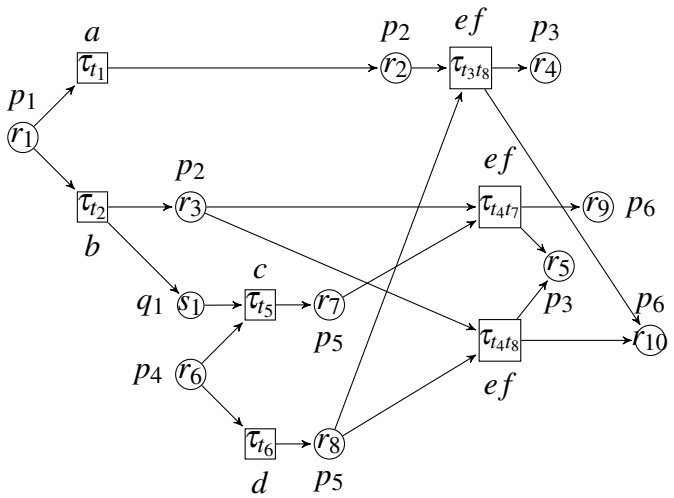


Fig. 4.16 Constructed acyclicnet$(csan)$ for the *csan* unfolding in Figure 4.15(b).

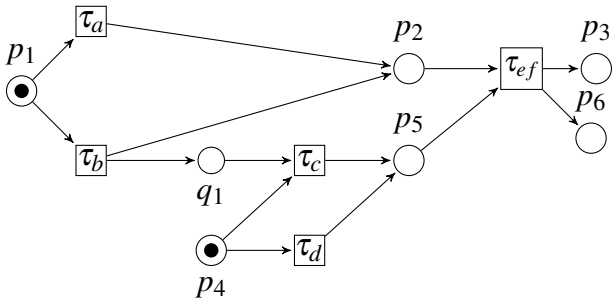


Fig. 4.17 Constructed acyclicnet($csan$) for $csan$ in Figure 4.15(a)

4.12 Conclusion

In this chapter, we formally defined the notion of CSA-net with its execution semantic and other behavioural properties. In particular, asynchronous and synchronous communications are introduced that are captured by the new nodes, namely buffer places which are responsible of transferring the tokens among several components acyclic nets. Hence, in contrast with the enabled steps in a single acyclic net defined in Chapter 3, the enabled steps in CSA-nets can involve transitions from different acyclic nets executed synchronously. This was the motivation for introducing the notion of syn-cycles. Then, the definitions of conflict, confusion, and scenarios are extended to be based on syn-cycles rather than transitions. Also, calculating the probabilities of alternative scenarios is extended. The core contribution of this chapter is the proposed approach of handling confusion in CSA-nets as presented in Section 4.9. More precisely, a confused CSA-net is translated into a single acyclic net such that transitions involved in asynchronous communication and their buffer places are replaced by regular transitions and places, whereas transitions involved in each syn-cycle representing synchronous communications are glued together into one synchronised transition. Our approach of preserving the behaviour in the original confused net and the new one is similar to the approach used in Chapter 3. Basically, the acyclic net obtained from a confused CSA-net simulates the original behaviour as long as it can generate representations of all possible scenarios in the original CSA-net. After obtaining the single acyclic net, the cluster-acyclic net concept from Chapter 3 is reused to apply Approach *A* or Approach *B* that are concerned with removing confusion from a single backward-deterministic acyclic net.

It turns out that cluster-acyclicity is guaranteed when a sub-class of CSA-nets is considered. Cascading CSA-nets are formally defined when asynchronous communications are unidirectional. One of the observations about this class of CSA-nets is that translating confused

cascading CSA-net into a single acyclic net always produces a cluster-acyclic net. Imposing this constraint structure on CSA-nets allows to check the acyclicity globally. That led to formal definition of another approach for removing confusion in CSA-nets. Approach *C* was introduced to handle confusion in cascading CSA-nets via adding asynchronous communications with new buffer places so that the execution of some transitions is postponed. This technique of dealing with confusion avoids generating new transitions and negative places. Also, as a significant feature, the structure of CSA-nets is retained and the backward-determinism is not mandatory to produce a confusion-free version.

In the next chapter, we intend to extend our probabilistic framework by considering behavioural abstraction relation.

Chapter 5

Behavioural Structured Acyclic Nets

5.1 Introduction

The structures of nets in Chapter 4 capture the notion of interaction between different systems via a/synchronous communication, however, the evolution of systems is not represented. In this chapter we introduce *behavioural relation* as a way of capturing the evolution of a set of related acyclic nets. This relation provides a mechanism to abstract part of a complex activity by another system. In other words, behaviour relation models the *duality* of the system-state as the behaviour can be embodied at two levels of abstraction, upper-level and lower-level, to depict both a system and state of an activity of that system [92].

In this chapter, *Behavioural Structured Acyclic Nets* (or BSA-nets) are defined as we formally extend the version of behavioural abstraction in [106, 103, 84] from being based on occurrence nets to one based on acyclic nets. That includes relaxing the line-like structural constraint on the upper-level net. However, only *free-choice* structure is allowed for the upper-level net. Moreover, we investigate how to extend the probabilistic framework to consider behavioural relation. Basically, probability assigned to an upper-level transition is the product of the weights of transitions that are ascribed to it.

Since the upper-level net is free-choice, behavioural relation can be used to control the occurrence of confusion at the lower-level. More precisely, only the confusion-free behaviour at the lower-level can be abstracted by the upper-level.

This chapter is organised as follows. The formal definition of BSA-nets with their behavioural and structural properties is introduced in Section 5.2 – Section 5.4. Calculating probabilities in BSA-nets is discussed in Section 5.5. Section 5.6 provides an illustrative example to show a preliminary approach of how confusion can be controlled via behavioural relation. Finally, the chapter is concluded in Section 5.7.

5.2 Behavioural structured acyclic nets

An abstract representation is the process of translating a complex representation of a problem into a new simpler representation [50] by removing irrelevant details. The aim is to solve the problem by retaining certain desirable properties from the original representation. It can be seen as a process of many-to-one mapping with ignoring details and considering unlike objects as if they were identical [86]. It also includes the ability of distinguishing between relevant and irrelevant details to solve a complex problem. Hence, abstraction can be seen as cognitive skills to cope with complexity in, e.g., mathematics and software engineering [59]. In fact, abstraction is a fundamental concept to mathematics, computer science, and software engineering [58, 59, 79]. Beside handling the complexity, other justifications include the intricate characteristic of systems and the flexibility of exploring them at multiple level of detail and abstraction based on the intention [79]. Several studies have explored abstraction in different contexts [50, 75, 93, 131].

There is also a large body of work regarding Petri nets and abstraction. [49] shows that Petri nets are suitable for modelling large scale systems as they offer techniques for abstraction and refinement. Sets of *place-bordered* and *transition-bordered* sub-nets are identified so that a net can be abstracted by replacing these sub-nets by a single element. It also shows that coloured Petri nets are another form of abstraction. Hierarchical Petri nets are introduced in [47] to support modelling of large systems and handling place and transition refinement. Building blocks are also purposed to hierarchical structure Petri nets. However, the formal definition of hierarchical Petri nets considers the structure only, without addressing the semantic information.

Multi-level abstraction in Petri nets is investigated in [126] to integrate Petri nets and object-orientation. Two-level modelling is introduced, using *system nets* and *object nets*. A token in a system net represents an object Petri net. In other words, object nets are mapped to places, and markings of the system net are represented by the processes of object nets. The approach contributes to reducing the size of the model as well as exploring the properties of the system net through the object net. In the same vein, [127] proposed *tokens as nets* concept to abstract objects nets by representing them as tokens of systems net by constructing multi-level hierarchical abstraction. Abstraction of Petri nets is also used for analysis in [122]. Verifying properties such as deadlocks, liveness, and boundedness is obtained for a subnet abstracted by a single transition. An abstraction representation based analysis is developed in [48] where the state spaces are reduced by gluing unimportant places for a certain verified property. In the following, we present the notion behinds *behavioural abstraction*.

In the initial work published in [103], Brian Randell mentioned that his attempt to illustrate *failure / fault / error* chains in an evolved complex system was the beginning of the story of behavioural abstraction. The interchangeable meaning of system and state allows to represent systems and states of different interacting occurrence nets using one *place* [105]. In this abstraction relation, many-to-one mapping is also applied as the states of one occurrence net are linked to sets of states in other occurrence net. For instance, a state of an occurrence net can be expanded into an another occurrence net that has its own places and transitions. More precisely, the hierarchical structure of behavioural abstraction involves two-level view of a system's history. The upper-level represents an abstract view of the details regarding the evolution of the system at the lower-level. Moreover, the dual representation of states and systems supports multiple abstraction of provenance [92]. [105] formalises *evolutional occurrence nets* in which occurrence net captured modification or creation of system components through evolution abstraction.

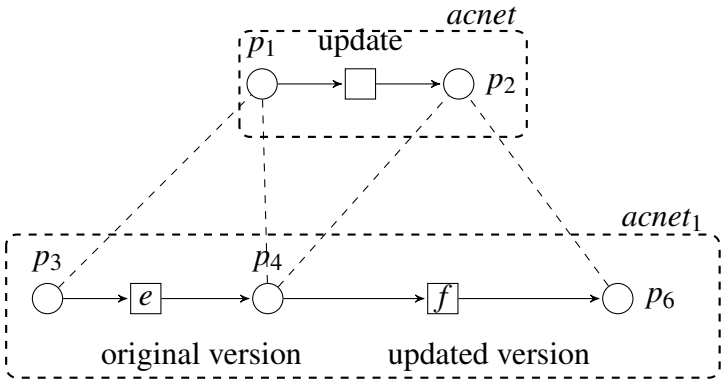


Fig. 5.1 Behavioural structured occurrence net.

The aim of the hierarchical structure is to emphasize that upper level always provides an upper-level view and hides the details of the behaviour of lower level. The lower-level, on the other hand, shows the full details of behaviour during different evolution stages which are represented by the phases of the general activity [78].

Example 52. Figure 5.1 shows the concept of the behavioural relation in a system update. The upper-level acyclic net *acnet* represents a version change caused by an update event. The lower-level acyclic net $acnet_1$ provides the detailed behaviour of the system before and after the update. The dashed lines between the two levels are used to capture the relevant relationships between the two types of behaviours. The divisions of the lower-level behaviours are captured by the *phases*, which are sets of states that are mapped to the upper-level places. ◇

The version of behavioural relation in [78, 106, 105, 84, 11, 92] does not support the notion of conflict at the upper level. Hence, applying the probability analysis for behavioural relation is trivial when the upper-level nets are always *line-like*.

Example 53. The net in Figure 5.2 is taken from [84]. The lower-level acyclic net involves two alternative scenarios: $\text{scenario}_{acnet}(\{e_0, e_1, e_2, e_6\})$ and $\text{scenario}_{acnet}(\{e_0, e_3, e_4, e_5, e_6\})$. Moreover, different phases in the lower-level acyclic net map to a single place in the upper-level acyclic net. In this case, we have place p_2 pointing to places p_6 and p_7 which are both included in two conflict scenarios. This illustrates the 'one-to-many' relationship between the upper-level places and phases of the lower-level. ◇

Still referring to the acyclic net in Figure 5.2, let us assume that the weights of transitions e_1 and e_3 are 3 and 7, respectively. Since these two scenarios are mapped into one scenario at the upper-level acyclic net, then all the upper-level transitions can be considered as certain ones as they abstract all the lower-level scenarios. Basically, what the upper-level net can tell is that *there is a scenario*, without any details of the probability of the actually executed one.

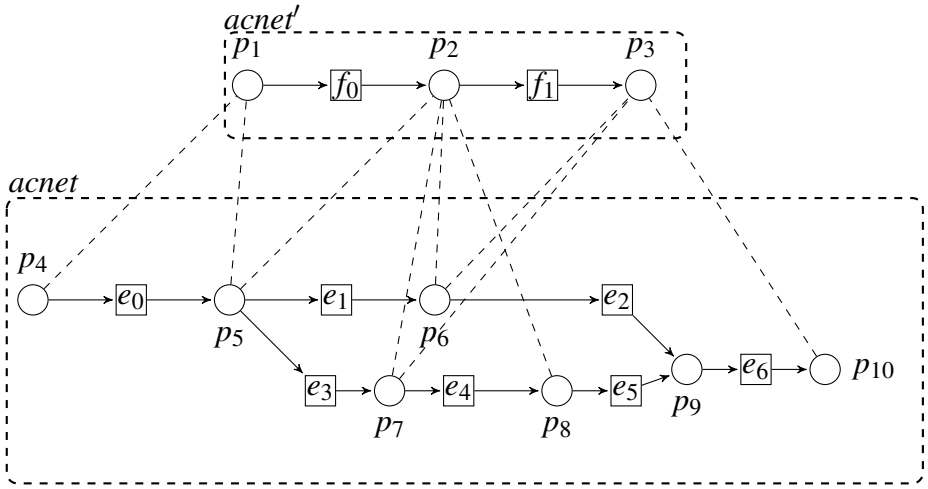


Fig. 5.2 Behavioural structured occurrence net in [84] (Figure 3.7).

Therefore, extending the current version where probabilities can be estimated for the entire net requires relaxing the assumption of upper-level net as being line-like. In addition to that restriction, behavioural structured occurrence nets are generalised by being based on *acyclic nets* instead of occurrence nets. However, this generalisation is restricted as it permits only *free-choice* structure of the upper-level acyclic nets. The new version of behavioural relation is formally defined below.

Definition 5.2.1 (BSA-net [8])**.** A *behavioural structured acyclic net (or* BSA*-net)* is a triple $bsan = (lcsan, hcsan, \beta)$ such that

$$lcsan = (lcsan_1, \ldots, lcsan_n, lQ, lW) \quad \textit{and} \quad hcsan = (hacnet_1, \ldots, hacnet_n, hQ, hW)$$

are as follows:

- $hcsan = (hacnet_1, \ldots, hacnet_n, hQ, hW)$ is a well-formed CSA-net ($n \geq 1$).
- For every $1 \leq i \leq n$, $lcsan_i = (acnet_1^i, \ldots, acnet_{m_i}^i, Q_i, W_i)$ is a CSA-net such that

  $$\text{csan}(lcsan) = (acnet_1^1, \ldots, acnet_{m_1}^1, \ldots, acnet_1^n, \ldots, acnet_{m_n}^n, lQ \cup \bigcup_{i=1}^{n} Q_i, lW \cup \bigcup_{i=1}^{n} W_i)$$

  is a well-formed CSA-net.
- *hcsan* and *lcsan* have disjoint sets of nodes, and we denote:

$$\begin{array}{lcllcl} P_{lcsan} & = & \bigcup_{i=1}^{n} P_{lcsan_i} & \qquad T_{lcsan} & = & \bigcup_{i=1}^{n} T_{lcsan_i} \\ P_{hcsan} & = & \bigcup_{i=1}^{n} P_{hacnet_i} & \qquad T_{hcsan} & = & \bigcup_{i=1}^{n} T_{hacnet_i} \end{array}$$

Moreover, $\beta \subseteq \bigcup_{i=1}^{n} P_{lcsan_i} \times P_{hacnet_i}$, and the following are satisfied, for all $1 \leq i \leq n$ and $t \in T_{hacnet_i}$:

1. $|M_{hacnet_i}^{init}| = 1$ and $|\,\text{pre}_{hacnet_i}(t)| = |\,\text{post}_{hacnet_i}(t)| = 1$.
2. $\beta_{M_{hacnet_i}^{init}} = M_{lcsan_i}^{init}$ and $\beta_{\text{pre}_{hacnet_i}(t)}[\rangle_{lcsan_i}\beta_{\text{post}_{hacnet_i}(t)}$, where $\beta_p = \beta_{\{p\}} = \{r \mid r\beta p\}$, for every $p \in P_{hcsan}$.

The default *initial* marking of *bsan* is $M_{bsan}^{init} = M_{hcsan}^{init} \cup \bigcup_{i=1}^{n} M_{lcsan_i}^{init}$, and the set of all BSA-nets is *BSAN*. ◇

Intuitively, *lcsan* provides a detailed 'lower-level' view and *hcsan* provides an 'upper-level' view of the system evolution. The role of β is to identify in the lower-level view the divisions of behaviours into 'phases', and each β_p indicates a 'boundary' between two consecutive phases. In other words, β is responsible for showing dependencies between the system's detailed behaviour and its evolution [84].

Example 54. The new (generalised) version of BSA-net is illustrated in Figure 5.3. The initial marking of the lower-level CSA-net belongs to the initial marking of *bsan*. The two maximal scenarios at the lower-level are explicitly represented by distinct scenarios at the

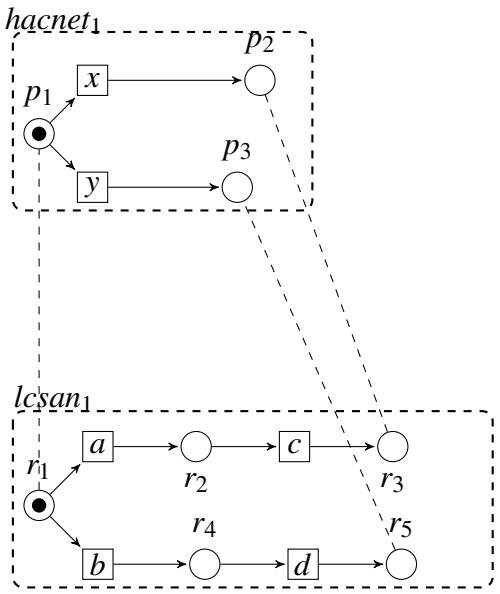


Fig. 5.3 BSA-net $bsan = (lcsan, hcsan, \beta)$ where $lcsan = (lcsan_1, \varnothing, \varnothing)$ with no lQ, lW, and $hcsan = (hacnet_1, \varnothing, \varnothing)$ with no hQ, hW, and $\beta = \{(p_1, r_1), (p_2, r_3), (p_3, r_5)\}$.

upper-level. Basically, the two maximal scenarios at the upper-level $\text{scenario}_{hacnet}(\{x\})$ and $\text{scenario}_{hacnet}(\{y\})$ are the abstracted versions of the lower-level maximal scenarios $\text{scenario}_{lacnet}(\{a,c\})$ and $\text{scenario}_{lacnet}(\{b,d\})$, respectively. ◇

Proposition 5.2.1. Assume the notation as in Definition 5.2.1, and let $1 \leq i \leq n$.

1. $\text{reachable}(hacnet_i) = \{\{p\} \mid p \in P_{hacnet_i}\}$.
2. $|M \cap P_{hacnet_i}| = 1$, for every $M \in \text{reachable}(hcsan)$.
3. $\beta_p \in \text{reachable}(lcsan_i)$, for every $p \in P_{hacnet_i}$.

Proof. It follows directly from Definition 5.2.1. □

That is, a reachable marking of the upper-level CSA-net includes exactly one place from each of its component acyclic nets. Moreover, all the boundaries between different phases are reachable markings of the lower-level CSA-nets.

The next definition shows how a BSA-net is underpinned by a CSA-net combining the two levels.

Definition 5.2.2 (underlying CSA-net)**.** Assuming the notation as in Definition 5.2.1,

$$\text{csan}(bsan) = (hacnet_1, \ldots, hacnet_n, acnet_1^1, \ldots, acnet_{m_1}^1, \ldots, acnet_1^n, \ldots, acnet_{m_n}^n, Q, W) ,$$

where $Q = hQ \cup lQ \cup \bigcup_{i=1}^{n} Q_i$ and $W = hW \cup lW \cup \bigcup_{i=1}^{n} W_i$, is the CSA-net *underlying bsan*.
◇

Proposition 5.2.2. Let $bsan \in BSAN$. Then $\mathrm{csan}(bsan)$ is a well-formed CSA-net.

Proof. It follows directly from Definition 5.2.1 and Definition 5.2.2. □

Definition 5.2.3 (phase and phase-consistent marking)**.** Assuming the notation as in Definition 5.2.1.

1. $\mathrm{phase}(p) = \{\beta_p\} \cup \{M \mid \exists t \in \mathrm{post}_{hacnet_i}(p) : \beta_p[\rangle_{lcsan_i} M[\rangle_{lcsan_i} \beta_{\mathrm{post}_{hacnet_i}(t)}\}$ is the *phase* of $p \in P_{hacnet_i}$ $(1 \leq i \leq n)$.

2. The *phase-consistent* markings of *bsan* are as follows:

$$\begin{aligned}\mathrm{phcmarkings}(bsan) \;=\; \{M \in \mathrm{reachable}(\mathrm{csan}(bsan)) \mid \\ \forall 1 \leq i \leq n : M \cap P_{lcsan_i} \in \mathrm{phase}(\beta_{M \cap P_{hacnet_i}})\}\end{aligned}$$

◇

Intuitively, Definition 5.2.3(1) implies that $\mathrm{phase}(p)$ of the upper-level net is a continuous 'chunk' of $lcsan_i$ delimited by the marking corresponding to p (start) and all markings (ends) corresponding to the places obtained by executing one output transition of p (such a transition indicates a 'phase change'). All markings in-between belong to the delimited phase. Hence, each transition of a upper-level acyclic net maps to a single phase of the corresponding lower-level CSA-net.

Phase-consistency in Definition 5.2.3(2) identifies implicitly the markings of lower-level acyclic nets which belong to the phases corresponding to the markings of the upper-level acyclic nets. This will guarantee that the ordering of the upper-level places matches the consecutive phases of the lower-level CSA-nets.

Proposition 5.2.3. M_{bsan}^{init} is a phase-consistent marking.

Proof. It follows directly from Definition 5.2.3. □

Example 55. For the *bsan* in Figure 5.3, the phases are as follows:

$$\begin{aligned}\mathrm{phase}(p_1) &= \{\{r_1\},\{r_2\},\{r_3\},\{r_4\},\{r_5\}\} \\ \mathrm{phase}(p_2) &= \{\{r_3\}\} \\ \mathrm{phase}(p_3) &= \{\{r_5\}\}.\end{aligned}$$

Also, $\beta_{M^{init}_{hacnet_1}} = \{r_1\}$ and $\{p_2, r_2\}, \{p_3, r_4\} \in \text{reachable}(csan(bsan))$. However, neither marking is phase-consistent due to the fact that $\{r_2\} \notin \text{phase}(p_2)$ and $\{r_4\} \notin \text{phase}(p_3)$. ◇

5.3 Structure and semantics of BSA-nets

In this section, $bsan \in BSAN$ is as in Definition 5.2.1. All basic notions and notations concerning the structure and semantics of *bsan* are taken from the underlying CSA-net $\text{csan}(bsan)$.

Definition 5.3.1 (BSO-net [8])**.** A *behavioural structured occurrence net (or* BSO*-net)* is $bsan \in BSAN$ such that $\text{csan}(bsan) \in CSON$. The set of all BSO-nets is *BSON*. ◇

Note that in BSO-nets, the upper-level acyclic nets are 'line-like' occurrence nets.

Definition 5.3.2 (BDBSA-net [8])**.** A *backward deterministic behavioural structured acyclic net (or* BDBSA*-net)* is $bsan \in BSAN$ such that $\text{csan}(bsan) \in BDCSAN$. The set of all BDBSA-nets is *BDBSAN*. ◇

Proposition 5.3.1. $BSON \subset BDBSAN \subset BSAN$.

Proof. It follows directly from the definitions. □

Example 56. Figure 5.3 shows an example of a BDBSA-net. Moreover, a BSO-net is shown in Figure 5.5. ◇

Definition 5.3.3 (step and marking [8])**.** Assuming the notation as in Definition 5.2.2,

1. $\text{steps}(bsan) = \text{steps}(\text{csan}(bsan))$ are the *steps*.
2. $\text{markings}(bsan) = \text{markings}(\text{csan}(bsan))$ are the *markings*.
3. $P_{bsan} = P_{\text{csan}(bsan)}$, $T_{bsan} = T_{\text{csan}(bsan)}$, $F_{bsan} = F_{\text{csan}(bsan)}$, $Q_{bsan} = Q_{\text{csan}(bsan)}$, and $W_{bsan} = W_{\text{csan}(bsan)}$.
4. $\text{pre}_{bsan}() = \text{pre}_{\text{csan}(bsan)}()$ and $\text{post}_{bsan}() = \text{post}_{\text{csan}(bsan)}()$. ◇

Definition 5.3.3 means that the basic characteristics of *bsan* are as in $\text{csan}(bsan)$. The notion of enabled step is, however, different. It is based on the phases of lower-level CSA-nets induced by the phase boundaries defined by the relation β.

Definition 5.3.4 (enabled and executed step)**.** Let M and M' be markings of *bsan*, and U be a step of *bsan*. Then,

$$U \in \text{enabled}_{bsan}(M) \quad \textit{and} \quad M[U\rangle_{bsan} M'$$

if $M, M' \in \text{phcmarkings}(bsan)$ and $M[U\rangle_{\text{csan}(bsan)} M'$. ◇

Intuitively, *bsan* is executed in exactly the same way as its underlying CSA-net provided that the markings involved are phase-consistent.

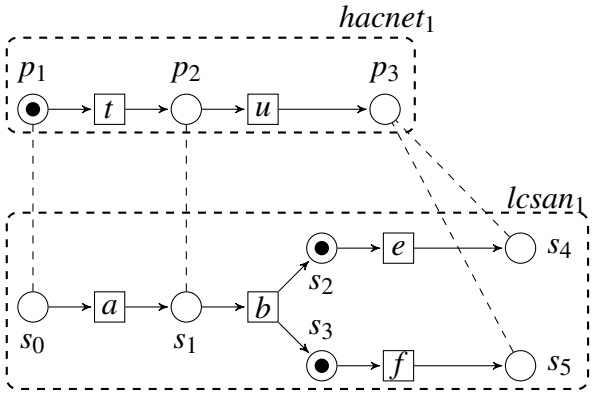


Fig. 5.4 BSA-net with no enabled step.

Example 57. Figure 5.4 illustrates phase-consistency and step enabledness. The phases are:

$$\begin{aligned}
\text{phase}(p_1) &= \{\{s_0\},\{s_1\}\} \\
\text{phase}(p_2) &= \{\{s_1\},\{s_2,s_3\},\{s_4,s_3\},\{s_2,s_5\},\{s_4,s_5\}\} \\
\text{phase}(p_3) &= \{\{s_4,s_5\}\}
\end{aligned}$$

The marking $\{p_1, s_2, s_3\}$ is not phase-consistent because $\{s_2, s_3\} \notin \text{phase}(p_1)$. Consequently, according to Definition 5.3.4, $U = \{e, f\} \notin \text{enabled}_{bsan}(\{p_1, s_2, s_3\})$. Note, however, that U is enabled in the reachable marking $\{p_2, s_2, s_3\}$ of the underlying CSA-net. ◇

Definition 5.3.5 ((mixed) step sequence)**.** Assume the notation as in Definition 5.2.2. Let $\mu = M_0 U_1 M_1 \ldots M_{k-1} U_k M_k$ ($k \geq 0$) be a sequence such that $M_0, \ldots, M_k$ are markings and $U_1, \ldots, U_k$ are steps of *bsan*.

1. μ is a *mixed step sequence from M_0 to M_k* if $M_{i-1}[U_i\rangle_{bsan} M_i$, for every $1 \leq i \leq k$.
2. If μ is a mixed step sequence from M_0 to M_k, then $\sigma = U_1 \ldots U_k$ is a *step sequence from M_0 to M_k*.

This is denoted by $M_0[\mu\rangle\rangle_{bsan} M_k$ and $M_0[\sigma\rangle_{bsan} M_k$, respectively. Also, $M_0[\rangle_{bsan} M_k$ denotes that M_k is reachable from M_0. ◇

In the last definition, the starting point was an arbitrary marking. The next definition assumes that the starting point is the default initial marking.

Definition 5.3.6 (behavioural notions)**.** The following sets capture various behavioural notions related to step sequences and reachable markings of a BSA-net *bsan*.

1. $\text{sseq}(bsan) = \{\sigma \mid M^{init}_{bsan}[\sigma\rangle_{bsan} M\}$ *step sequences*.
2. $\text{mixsseq}(bsan) = \{\mu \mid M^{init}_{bsan}[\mu\rangle\rangle_{bsan} M\}$ *mixed step sequences*.
3. $\text{maxsseq}(bsan) = \{\sigma \in \text{sseq}(bsan) \mid \neg\exists U : \sigma U \in \text{sseq}(bsan)\}$ *maximal step sequences*.
4. $\text{maxmixsseq}(bsan) = \{\mu \in \text{mixsseq}(bsan) \mid \neg\exists U, M : \mu U M \in \text{mixsseq}(bsan)\}$ *maximal mixed step sequences*.
5. $\text{reachable}(bsan) = \{M \mid M^{init}_{bsan}[\rangle_{bsan} M\}$ *reachable markings*.
6. $\text{finreachable}(bsan) = \{M \mid \exists \sigma \in \text{maxsseq}(bsan) : M^{init}_{bsan}[\sigma\rangle_{bsan} M\}$ *final reachable markings*.

◇

Example 58. A maximal mixed step sequence of the BSA-net depicted in Figure 5.5 (*a*) is:

$$\mu = \{p_1, r_1\}\{b\}\{p_1, r_4\}\{y, d\}\{p_3, r_5\}$$

and the corresponding maximal step sequence is $\sigma = \{b\}\{y, d\}$. ◇

As before, what matters is to identify in a BSA-net all deterministic behaviours (scenarios) which can then be investigated.

Definition 5.3.7 (scenario and maximal scenario)**.** Assume the notation as in Definition 5.2.2.

1. A *scenario* of *bsan* is a BSO-net $bson = (lcson, hcson, \beta')$ such that:
   (a) $\text{csan}(lcson) \in \text{scenarios}(\text{csan}(lcsan))$ and $hcson \in \text{scenarios}(hcsan)$.
   (b) $\beta' = \beta \cap (P_{lcson} \times P_{hcson})$.

2. A *maximal scenario* of *bsan* is a scenario *bson* such that there is no scenario *bson*′ satisfying $T_{bson} \subset T_{bson'}$.

The set of all scenarios of *bsan* is scenarios(*bsan*), and the set of all maximal scenarios of *bsan* is maxscenarios(*bsan*). ◇

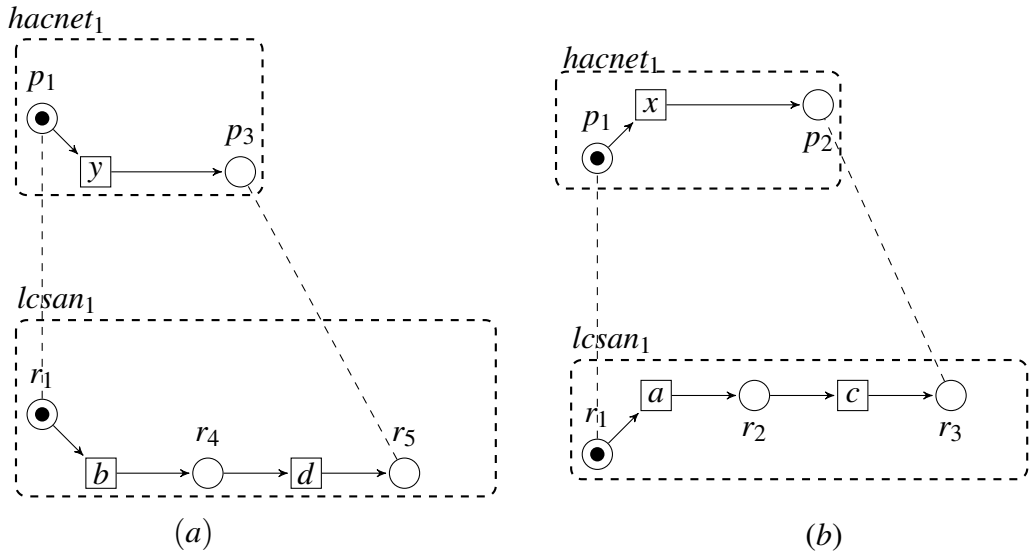


Fig. 5.5 All the maximal scenarios for the BSA-net in Figure 5.3.

Example 59. Figure 5.5 depicts the only two maximal scenarios of the BSA-net shown in Figure 5.3. ◇

5.4 Well-formed BSA-nets

The general definition of BSA-net given above does not guarantee that the step sequences of *bsan* cover all possible scenarios of the lower-level CSA-net. Indeed, in the extreme case, we can take *hcsan* which contains no transitions at all, and the resulting BSA-net generates then only the empty step sequence. It is therefore crucial to identify cases where *bsan* generates at least one step sequence for every scenario of *lcsan*. As a result, the definition of a well-formed BSA-net is more demanding than the definition of a well-formed CSA-net.

Definition 5.4.1 (well-formedness)**.** A BSA-net $bsan = (lcsan, hcsan, \beta)$ is *well-formed* if the following hold:

1. $\text{sseq}(bsan) = \bigcup \text{sseq}(\text{scenarios}(bsan))$.
2. For every $cson \in \text{maxscenarios}(\text{csan}(lcsan))$, there is $U_1 \dots U_k \in \text{maxsseq}(bsan)$ such that $(U_1 \cap T_{lcsan}) \dots (U_k \cap T_{lcsan}) \in \text{maxsseq}(cson)$.

The set of all well-formed BSA-nets is denoted by *WFBSAN*. ◇

The well-formedness of BSA-net ensures that for each scenario of the lower-level *csan* [8], there is at least one step sequence that can be generated by the whole *bsan*.

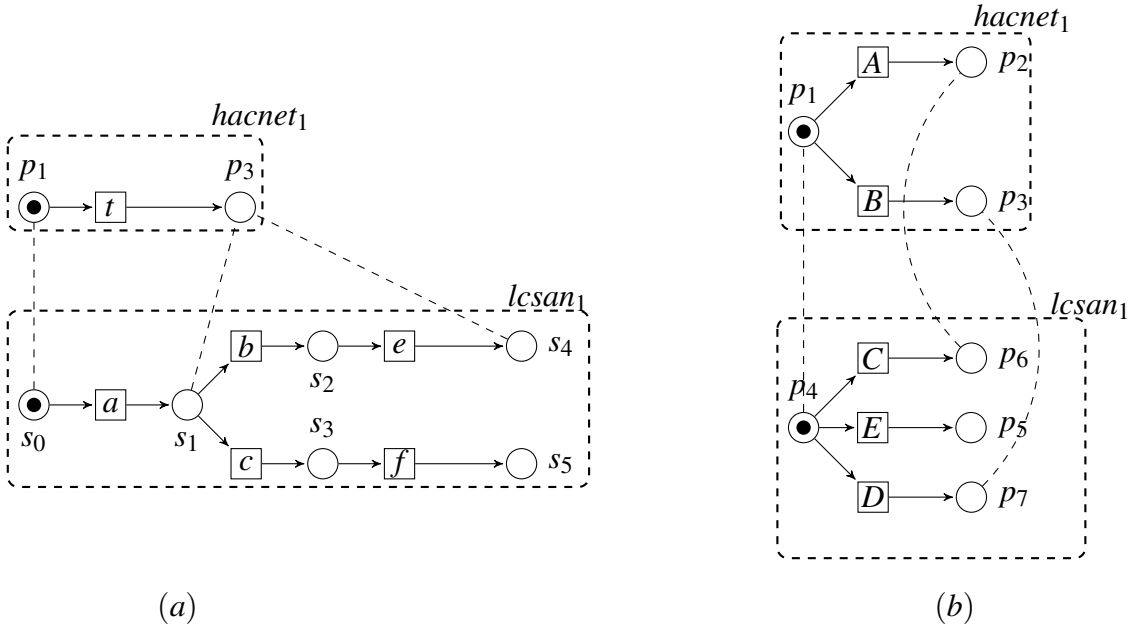


Fig. 5.6 BSA-nets that are not well-formed.

Example 60. Consider the *bsan* in Figure 5.6(*a*). Note that there are two scenarios in $lcsan_1$:

$$\text{scenario}_{lcsan_1}(\{a,b,e\}) \ \ and \ \ \text{scenario}_{lcsan_1}(\{a,c,f\}).$$

There is a step sequence $\sigma = \{a\}\{t,b\}\{e\} \in \text{maxsseq}(bsan)$ that can be generated for the former scenario. However, there is no similar step sequence for the latter one. That is because

$$\text{post}_{lcsan_1}(c) = \{s_3\} \notin \text{phase}(p) \ \ and \ \ \text{post}_{lcsan_1}(f) = \{s_5\} \notin \text{phase}(p),$$

for every $p \in P_{hacnet_1}$. Moreover, we have

$$\{s_3,p_3\}[f\rangle_{bsan}\{s_5,p_3\} \ \ and \ \ \{f\} \notin \text{enabled}_{bsan}(\{s_3,p_3\})$$

because the phase-consistency is not satisfied: $\{s_3\} \notin \text{phase}(p_3)$. Hence, f is a 'useless' transition. In the same vein, the upper-level *hcsan* in Figure 5.6(b) does not provide a complete view of the lower-level. Indeed, what the upper-level can tell is that there are only two scenarios, corresponding to A and B, whereas three scenarios are included in the lower-level: $\text{scenario}_{lcsan_1}(\{C\})$, $\text{scenario}_{lcsan_1}(\{D\})$ and $\text{scenario}_{lcsan_1}(\{E\})$. That is because the last scenario is not abstracted at $hacnet_1$. Hence, the behaviour captured by $hacnet_1$ is not complete and, e.g., the probabilistic analysis is not accurate due to the missing information. $\diamond$

In the above sections we considered BSA-net properties. Next we illustrate how probabilities can be added to BSA-net. Basically, the probabilities are assigned to the upper-level transitions using the weights of the lower-level transitions with respecting the phases between the levels.

5.5 Calculating probabilities in BSA-nets

In this section, we assume that the high-level nets have no communications. This section extends the probabilities calculation in CSA-nets so that behavioural relation is taken into consideration. More precisely, the β relation is used to represent the probabilities at the upper-level that are derived from the weights associated with the lower-level syn-cycles. In this case, the probability of a whole lower-level scenario that is calculated as in Chapter 4 is reflected by its abstracted upper-level syn-cycle. Therefore, in addition to defining the probabilities of syn-cyclesas in Chapter 4, in this chapter the upper-level syn-cycles, are assigned probabilities derived from the weight of conflict syn-cycles at the lower-level Basically, extending the formula in Section 4.7 on Page 95 includes ensuring that all the reachable markings obtained by executing lower-level CSA-nets should belong to the phases corresponding to the markings obtained by executing upper-level CSA-nets. This means that the phase-consistency of markings should be satisfied. In this thesis, we will not introduce the formal formulas for calculating the probabilities in BSA-nets. We plan to extend the work in [13] and address this part in the future work.

Example 61. Consider the BSA-net in Figure 5.7. The low-level syn-cycles in *lcsan* are: $S_1^{lcsan} = \{a\}$, $S_2^{lcsan} = \{b\}$, $S_3^{lcsan} = \{c\}, S_4^{lcsan} = \{d\}$. The high-level syn-cycles are: $S_1^{hcsan} = \{x\}$, $S_2^{hcsan} = \{y\}$, $S_3^{hcsan} = \{z\}$. Then the three maximal scenarios are as follows:

$$bson_1 = \text{scenario}_{bsan}(\{a,c,x\}) \quad bson_2 = \text{scenario}_{bsan}(\{a,d,y\})$$
$$bson_3 = \text{scenario}_{bsan}(\{b,z\})$$

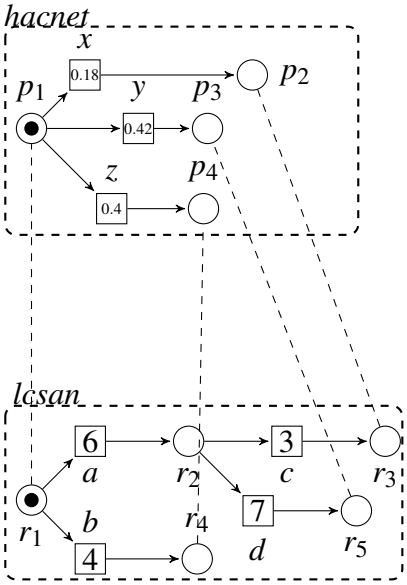


Fig. 5.7 BSA-net with weights.

According to the β relation between the two levels, we have $\beta_{p_1} = \{r_1\}$, and $\text{post}_{hcsan}(p_1) = \{x, y, z\}$. The phases are as follows:

$$\begin{array}{lcllcl} \text{phase}(p_1) & = & \{\{r_1\},\{r_2\},\{r_3\},\{r_4\},\{r_5\}\} & \text{phase}(p_2) & = & \{\{r_3\}\} \\ \text{phase}(p_3) & = & \{\{r_5\}\} & \text{phase}(p_4) & = & \{\{r_4\}\} \end{array}$$

Initially, only the weights of lower-level transitions are given. Then, the probability of each low-level maximal scenario is represented at the high-level syn-cycles such that β relation is taken into account. For example, $\text{scenario}_{lcsan}(\{a,c\})$ can be executed as $\sigma_1 = S_1^{lcsan} S_3^{lcsan}$ with the 0.18 probability (calculated as in Section 4.7) which is then assigned to its corresponding high-level syn-cycle $S_1^{hcsan} = \{x\}$. ◇

The powerful representation of the *systems* and *states* in the behavioural relation using places [103] allows us to represent the probabilities at the upper-level acyclic nets assuming that the probabilities in CSA-nets introduced in Chapter 4 can capture the probabilities of the scenarios at the lower-level. Intuitively, the upper-level acyclic nets do not only provide an abstraction of the detailed behaviour at the lower-level, but also can be used as a means to analyse the probabilistic aspects of executions.

Example 62. Consider the BSA-net *bsan* in Figure 5.8. The low-level nets are as *csan* in Figure 4.4 on Page 96. However, here they are abstracted by the high-level acyclic net *hacnet*. Recall from Example 39, the low-level syn-cycles: $S_1^{lcsan} = \{e\}$, $S_2^{lcsan} = \{f\}$, $S_3^{lcsan} = \{A, D\}$,

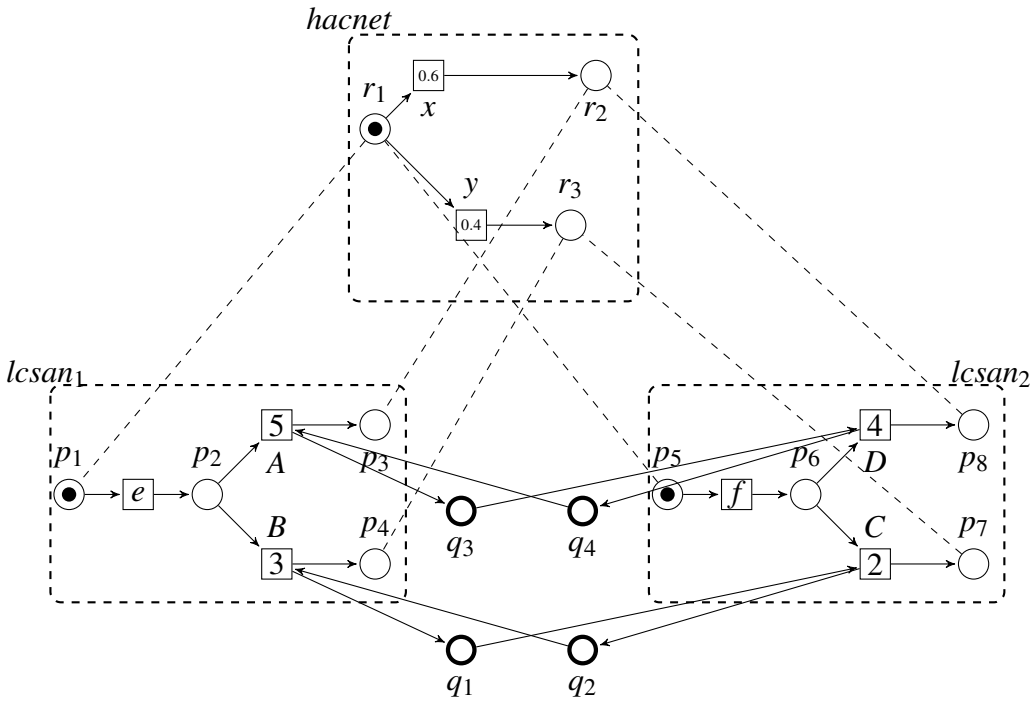


Fig. 5.8 BSA-net with weights.

and $S_4^{lcsan} = \{B,C\}$. Note that the probabilities of the maximal scenarios (based on the calculation in Example 39) are represented at their corresponding high-level syn-cycles $S_1^{hcsan} = \{x\}$, and $S_2^{hcsan} = \{y\}$. Basically, scenario$_{lcsan}(\{e,f,A,D\})$ and scenario$_{lcsan}(\{e,f,B,C\})$ with the probability of their executions are assigned to the syn-cycles of the upper-level net. ◇

In the above example, we have shown the initial idea of employing behavioural relation to provide probabilistic analysis. Note that the lower-level CSA-net in the above example are *confusion-free*.

5.6 Confusion in behavioural structured acyclic nets

In Section 5.1, we restricted the structure of upper-level acyclic nets as being free-choice only. Therefore, with the assumption that there are no communications at the higher-level nets as in Section 5.5, any potential a/symmetric confusion *within* the upper-level acyclic net is excluded. In this section, we propose a new approach of handling confusion via behavioural relation. The aim is to take advantage of the structural constraint at the upper-level so that the confused behaviour at the lower-level is mapped to a probabilistic confusion-free

representation at the upper-level. In this case, the upper-level net can be seen as a controller of the lower-level confused behaviour. In this section, we stipulate that the behavioural relation is a bottom-up abstraction approach. This means the maximal scenarios of lower-level CSA-nets with their weights and calculated probabilities for certain steps sequences which generate these maximal scenarios are defined first. Then the calculated probabilities are represented at the high-level. In this case,we first ensure that all the maximal scenarios are represented by the abstract nets, second; since the abstraction can play a role in analysing nets [82], the abstracted confusion-free scenarios allow probabilistic analysis of concurrent nets.

Example 63. Figure 5.9 shows a lower-level CSA-net *lcsan* with a symmetric confusion. According to Approach *A*, to remove confusion transitions *A* and *C* are combined together in one synchronised transition *AC* in conflict with *B* (similar to Example 16 and Figure 3.10 on Page 51). However, in the proposed approach lower-level CSA-net *lcsan* is abstracted by a confusion-free upper-level acyclic net *hacnet*. Basically, the maximal scenarios $\text{scenario}_{lcsan}(\{A,C\})$ and $\text{scenario}_{lcsan}(\{B\})$ of *lcsan* are represented by $\text{scenario}_{hacnet}(\{AC\})$ and $\text{scenario}_{hacnet}(\{B\})$ at the upper-level, respectively, through the β relation. Hence, the phases are as follows:

$$\text{phase}(s_1) = \{\{p_1,p_2\},\{p_3,p_5\},\{p_1,p_5\},\{p_3,p_2\},\{p_4\}\}$$
$$\text{phase}(s_2) = \{\{p_3,p_5\}\} \quad \text{phase}(s_3) = \{\{p_4\}\}.$$

Mapping places $p_3 \in \text{post}_{lacnet}(A)$ and $p_5 \in \text{post}_{lacnet}(C)$ to the same place s_2 captures the fact that *A* and *C* are executed together. Hence, the probability that $\{A,C\}$ are chosen over *B* at the lower-level is reflected by executing *AC* at the upper-level with probability 0.8 (based on their weights and probability calculation in Example 16). ◇

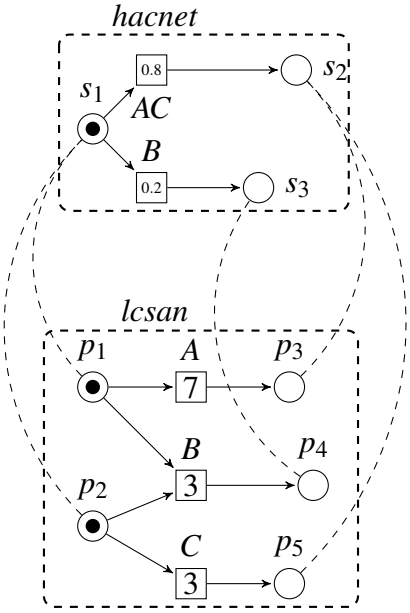


Fig. 5.9 BSA-net with a symmetric confusion.

Using behavioural relation to manage confusion is a powerful aspect of abstraction. Example 63 illustrates how abstraction allows us to construct a clearer representation of the maximal scenarios. For example, the conflict between A, C and B at the lower-level in Example 63 can be resolved with accurate probability estimation when $\{A, C\}$ are executed together as one step (more precisely, not their interleaving execution). Such representation is reflected at the abstract level. In other words, behavioural abstraction is filtering out the undesirable behaviour (the interleaving execution of A and C in the above example) without losing significant information. Essentially, handling confusion using behavioural relation requires to modify the step sequence semantics and consider the maximal step in each of phases. We aim to address this in future work.

The last topic related to behavioural abstraction concerns the situation in which a scenario is linked to different interpretations. This case accounts for contradicting views of what actually happened, which can be represented by several alternative abstractions of the same activity. More precisely, investigators or judges might agree about what actually happened, however, their analysis perspectives are different. In this case, the system represented by lower-level nets can be seen as source of information for alternative abstractions, each of which is possible analysis of the system [106].

Example 64. Figure 5.10 shows BSA-net whose upper-level acyclic net *hacnet* shows two different abstractions of the lower-level scenario. This can be interpreted as investigators

being in disagreement of the probability of a single scenario. More precisely, through β relation, we can infer that the scenario at the lower-level can be analysed probabilistically via transition t' with probability 0.7 or via transition u' with probability 0.3. ◇

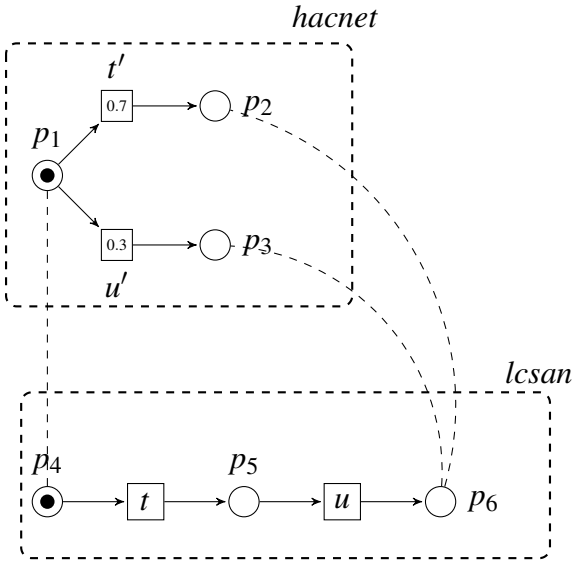


Fig. 5.10 Alternative abstractions for one scenario.

5.7 Conclusion

In this chapter, a formalisation for *behavioural abstraction relation* is introduced. In particular, in BSA-nets the evolution of a set of interacting acyclic nets is captured through behavioural abstraction. The hierarchical structure of this relation allows to represent the behaviour of a complex evolving system at two levels: the upper level which captures an abstract view and the lower level in which it represents the full details of the system behaviour. In particular, the β relation can map a set of places from the lower level to a place at the upper level. This results in dividing the lower-level view into *phases*.

This chapter extends the behavioural abstraction in [106, 103, 84] as being based on acyclic nets instead of occurrence nets so that the upper-level *free-choice* nets reflect the distinct scenarios at the lower level. Then, calculating probabilities in Chapter 4 is represented at the upper-level syn-cycles. Through phases, which are delimited by the β relation, we provide an initial idea of how the probability of a low-level maximal scenario can be assigned to its corresponding transition at the high-level. This mechanism of representing the probabilities at

the abstract view emphasizes that abstraction is not only a tool to simplify the representation of complex scenarios, but also it can contribute to the analyses of these scenarios. This is the motivation for proposing a new approach of handling *confusion* in BSA-net. Exploiting the structural constraint of the upper-level acyclic nets allows to propose a preliminary approach of controlling the evolving of a confusion at the lower level. More precisely, behavioural abstraction is seen as a filtering to distinguish desirable and undesirable behaviours at the lower level. Hence, only confusion-free behaviour is represented at the upper-level net. An illustrative example was provided to show how confusion can be handled by behavioural abstraction relation.

In general, this chapter is introduced at the exploratory level, especially the sections related to calculating probabilities and handling confusion. Providing formal definitions concerned with these sections requires extending the current definitions of BSA-nets, which goes beyond the time frame of this PhD thesis. Intuitively, the probability of a single process at the lower-level should be represented at the higher-level. Also, we would need to define the probabilities of phases (not only of processes) which may include multiple different alternatives at the lower-level. Relaxing the structural constraints imposed on the higher-level nets would also extend this chapter. Moreover, additional results and proofs are essential for the approach of handling confusion through behavioural relation.

Chapter 6

Verifying properties using SAT-solvers

6.1 Introduction

In the work so far, we have proposed approaches to remedy *confusion*. Generally, the aim was to extend probabilistic analysis from acyclic nets to CSA-nets and BSA-nets. However, a satisfactory property verification for CSA-nets and BSA-nets is still missing. To address this, our main contribution in this chapter is to fill this gap by providing SAT-encoding of the most significant behavioural properties of CSA-nets and BSA-nets, such as reachability and deadlock scenarios. Since CSA-nets and BSA-nets are formalisms to model concurrent and distributed systems, providing formal verification is crucial and poses a significant challenge [46]. That is because in distributed systems software, meagre errors may lead to significant collapse of systems functionalities [70]. In fact, proving the correctness of such systems is not guaranteed through ordinary testing. Employing mathematical proofs of correctness in the sense of formal verification is therefore needed.

In this chapter, we discuss the notion of a SAT-basic verification. It is a technique that determines if a given logical formula represented in propositional logic is satisfiable, i.e., if there is a truth assignment to the variables that makes the statement true. Basically, first, we formally translate behavioural specifications of an acyclic net. That includes constructing a formula to check whether or not a set of transitions represents a scenario. Conditions, such as being causally related, forward-deterministic, and backward-deterministic, presented in Definition 3.2.11, are translated into SAT constraints. Moreover, investigating the enabledness property is essential to verify that a given scenario is maximal. It turns out that constructing a scenario formula is a basis for providing a formal checking for other behavioural specifications. For example, converting the well-formedness definition into a SAT-encoding is obtained by identifying a scenario and an enabled transition whose firing violates scenario's

definition. In the same vein, checking deadlock scenarios and dead of transitions is based on the scenario formula. Simplified formats of these formulas are presented in the case of backward deterministic acyclic nets. Extending the work of [14], a SAT-based formula is proposed to detect confusion in an acyclic net. Detecting confusion using SAT-encoding considers both structural and dynamic aspects. To our knowledge, no such technique has been proposed in the literature so far.

Then, extended versions of the above formulas are provided for CSA-nets. This is the first work to propose a model checking solution for CSA-nets. It turns out that translating most of SAT-based formulas to the level of CSA-nets is straightforward. Basically, buffer places, not only places like in the case of acyclic nets, and syn-cycles instead of individual transitions are considered. SAT-based model checking for BSA-nets is also introduced. The most important constraints are related to Definition 5.2.3. Intuitively, we investigate how to translate the notions of a *phase* and *phase consistent marking* into a SAT formula (they are both required to formally verify the properties of BSA-nets). Finally, we explain the SAT-encoding constraints with illustrative examples.

This chapter is organised as follows: Section 6.2 provides an overview of the SAT-based model checking. Translating behavioural specifications of acyclic nets into SAT instances is presented in Section 6.3. We investigate in Section 6.4 how to extend SAT formulas to include behavioural properties of CSA-nets. SAT-based model checking for BSA-nets is considered in Section 6.5. The chapter is concluded in Section 6.6.

6.2 SAT-based model checking

The task of a SAT-solver is to find a satisfying assignment for the variables used in a propositional boolean formula ϕ. That is, the SAT-solver looks for an assignment $f : Var \rightarrow \{0,1\}$ for the variables Var which occur in ϕ that makes ϕ true, i.e., $\phi[f] = 1$. And, if such an assignment exists, ϕ is *satisfiable*. The formula ϕ is often expressed in *conjunctive normal form (*CNF*)*, i.e., a conjunction of clauses which are disjunctions of literals (where a literal is either a variable or the negation of a variable). The size of a CNF formula is based on the number of literals in its clauses.

SAT problems became popular as Cook was able to prove that they are NP-complete [35]. Reducing well-known NP-complete problems into SAT formulas increase the applicability of the SAT technique in the practically based research [34, 95]. In particular, scheduling problems [36], fault tolerance in circuits [102, 20], and an automatic verification of both

hardware and software systems [28] are all examples of exploiting the powerful technique of SAT-solvers in real-world applications.

Verifying formally the structural or dynamical aspects requires to model a system using a formalism such as Petri nets [98]. Verifying Petri net properties is one of the uses of SAT-solvers in Petri net analysis. The properties of a Petri net can be characterized as logical formulas that describe the conditions under which the behaviour of the net should operate. Some of these properties can be verified with SAT-solvers by encoding the Petri net behaviours and the negation of the property as a logical formula and then evaluating the formula's satisfiability. If the formula is satisfiable, then there is a Petri net behaviour that violates the property, and the SAT-solver can offer evidence of such a behaviour. If the formula is unsatisfiable, then the property holds true for all Petri net behaviours. This method can be used to validate a variety of properties of safe Petri nets, including reachability, liveness, and boundedness. There exists an extensive literature on verifying a Petri net model using SAT-solvers. For instance, [72] proposed an approach using SAT-solver to detect state encoding conflicts in Signal Transition Graphs which are converted into equivalent Petri nets and the verification is performed on the finite complete prefixes of their unfoldings ([71] extends this approach to model-checking for merged processes). [46] introduced a general study of model-checking based on Petri net unfoldings. A similar work has been done for the verification of contextual nets in [110]. Deadlock and reachability checking in Elementary Object Nets System (EOS) was introduced in [66].

6.3 Verifying properties of acyclic nets

In next sections, we provide a satisfiability checking for key behavioural properties related to acyclic nets. First, we construct a formula to find the scenarios. Then, such a formula is used as a basis to encode other properties.

6.3.1 Identifying scenarios and maximal scenarios for acyclic nets

In Chapter 3, we formally defined a scenario of an acyclic net. Then, its definition was extended in the subsequent chapters, Chapter 4 and Chapter 5. Scenarios are a significant notion of system's behaviour. Scenarios can be used to inspect, simulate, and analyse system properties. For instance, in the previous chapters, CSA-nets are analysed probabilistically through their scenarios. As scenarios' features include forward and backward determinism, they are suitable to provide a unique representation of the execution history. In the same vein,

in this section, we continue to show that scenarios are crucial concepts which can be used to verify some important behavioural properties.

Definition 3.2.11 asserts that a scenario structurally demonstrates causal and conflict relationships. Recall that, for a given acyclic net $acnet = (P,T,F)$, a subset of transitions $T' \subseteq T$ induces a scenario whenever the following three properties hold:

- *causality*: all non-initial pre-places of T' received tokens from T'.
- *forward deterministic*: the transitions in T' are free from forward conflicts.
- *backward deterministic*: the transitions in T' are free from backward conflicts.

The next definition and proposition identify the scenarios of an acyclic net with occurrence net restrictions induced by subsets of transitions T'.

Definition 6.3.1 (restricting acyclic net)**.** Let $acnet = (P,T,F)$ be an acyclic net and $T' \subseteq T$. Then $acnet|_{T'} = (P',T',F|_{(P'\times T')\cup(T'\times P')})$, where $P' = M^{init}_{acnet} \cup \text{post}_{acnet}(T')$, is the *restriction* of *acnet* to T'. ◇

Note that if $acnet|_{T'}$ is a scenario, then $acnet|_{T'} = \text{scenario}_{acnet}(T')$.

Proposition 6.3.1. The following statements are equivalent for an acyclic net $acnet = (P,T,F)$ and $T' \subseteq T$:

1. $acnet|_{T'} \in \text{scenarios}(acnet)$.
2. $acnet|_{T'}$ is an occurrence net and $\text{pre}_{acnet}(T') \setminus M^{init}_{acnet} \subseteq \text{post}_{acnet}(T')$.

Proof.
(1) ⇒ (2) Since $acnet|_{T'}$ is a scenario, then it is also an occurrence net (see Definition 3.2.11) and $\text{pre}_{acnet}(P') \setminus M^{init}_{acnet} \subseteq \text{post}_{acnet}(T')$.
(2) ⇒ (1) Obvious. □

In [14] we presented a propositional formula which can be used to identify all the scenarios of an acyclic net *acnet*. Basically, the aim is to construct a formula $Scenario_{acnet}$ which evaluates to 1 iff all the transitions assigned 1 (i.e., all transitions t such that in_t is assigned 1) induce a scenario of *acnet* (note that *acnet* is not assumed to be well-formed). The following boolean variables will be used in the construction of $Scenario_{acnet}$ and the translation into SAT instances:

- For every $t \in T$, we have a variable in_t tracing that t belongs to a scenario.

The constraints on the above variables are defined, following Proposition 6.3.1, as follows:

- To ensure that all non-initial pre-places of T' received tokens from T':

$$Causality_{acnet} \triangleq \bigwedge_{t \in T} (in_t \rightarrow \bigwedge_{p \in {}^\bullet t \setminus M^{init}_{acnet}} \bigvee_{u \in {}^\bullet p} in_u)$$

  where $\bigwedge_{p \in {}^\bullet t \setminus M^{init}_{acnet}} \bigvee_{u \in {}^\bullet p} in_u$ is set to 1 if ${}^\bullet t \subseteq M^{init}_{acnet}$.

- To ensure that scenario has no forward conflicts:

$$NoForwardConflict_{acnet} \triangleq \bigwedge_{t \in T} \bigwedge_{u \in ({}^\bullet t)^\bullet \setminus \{t\}} \neg(in_t \wedge in_u)$$

  where $\bigwedge_{u \in ({}^\bullet t)^\bullet \setminus \{t\}} \neg(in_t \wedge in_u)$ is omitted if $({}^\bullet t)^\bullet = \{t\}$.

- To ensure that scenario has no backward conflicts:

$$NoBackwardConflict_{acnet} \triangleq \bigwedge_{t \in T} \bigwedge_{u \in {}^\bullet(t^\bullet) \setminus \{t\}} \neg(in_t \wedge in_u)$$

  where $\bigwedge_{u \in {}^\bullet(t^\bullet) \setminus \{t\}} \neg(in_t \wedge in_u)$ is omitted if ${}^\bullet(t^\bullet) = \{t\}$.

Then the formula which characterises all the scenarios of *acnet* is:

$$Scenario_{acnet} \triangleq Causality_{acnet} \wedge NoForwardConflict_{acnet} \wedge NoBackwardConflict_{acnet}$$

The size of the above formula (in terms of the number of the occurrences of literals) is bounded by $|T| + 3 \cdot |T| \cdot \min\{|F|^2, |P| \cdot |T|\}$.

Proposition 6.3.2. Let *acnet* be an acyclic net.

1. If f is a satisfying assignment for $Scenario_{acnet}$ then $acnet|_{f^{-1}(1)}$ is a scenario of *acnet*.
2. If $acnet'$ is a scenario of *acnet*, then there is a satisfying assignment f for $Scenario_{acnet}$ such that $acnet|_{f^{-1}(1)} = acnet'$.

Proof. Recall that (see Section 6.3.1 on Page 145):

$$Scenario_{acnet} \triangleq Causality_{acnet} \wedge NoForwardConflict_{acnet} \wedge NoBackwardConflict_{acnet}$$

(1) Suppose that $Scenario_{acnet}$ has a satisfying assignment f. Then $Causality_{acnet}[f] = 1$, $NoForwardConflict_{acnet}[f] = 1$ and $NoBackwardConflict_{acnet}[f] = 1$. Hence, for any transition t such that $f(in_t) = 1$, it is required that each non-initial place $p \in {}^\bullet t$ should have received a token, and so at least one $u \in {}^\bullet p$ had to be fired as well which is guaranteed by $Causality_{acnet}[f] = 1$. Moreover, since we have $NoForwardConflict_{acnet}[f] = 1$ and $NoBackwardConflict_{acnet})[f] = 1$, then for any transition y in forward or backward conflict with t we have $f(in_y) = 0$. Hence, all transitions whose corresponding variable assignment is 1 induce a scenario captured by $acnet|_{f^{-1}(1)}$.

(2) Suppose that there is scenario $acnet'$ of the acyclic net $acnet$. Then, from [8] there is a step sequence $\sigma \in \text{sseq}(acnet)$ which induces $acnet'$:

$$\text{scenario}_{acnet}(\sigma) = (P', T', F|_{(P' \times T') \cup (T' \times P')}) = acnet'.$$

From Definition 6.3.1, $acnet'$ is a restriction of $acnet$ to T' where $T' = \bigcup \sigma$. Let f be variable assignment such that $f(in_t) = 1$, for every $t \in T'$, and $f(in_t) = 0$, for every $t \in T \setminus T'$. Since $acnet'$ is a scenario, then it is also an occurrence net, which satisfies forward and backward determinism. This means the transitions in T' have no forward conflict and have no backward conflict. Hence, $NoForwardConflict_{acnet}[f] = 1$ and $NoBackwardConflict_{acnet}[f] = 1$. Since for each executed transition $t \in \sigma$ we have $f(in_t) = 1$, then it is necessary that, for every $p \in {}^\bullet t$ there is $u \in {}^\bullet p$ such that u occurs in σ, which implies that $f(in_u) = 1$. Hence, $Causality_{acnet}[f] = 1$, and we can conclude that $Scenario_{acnet}$ has a satisfying assignment f such that $acnet|_{f^{-1}(1)} = acnet'$. □

Hence, we can use $Scenario_{acnet}$ to find all the subsets T' of transitions of $acnet$ inducing scenarios after translating it into CNF and feeding into a SAT-solver.

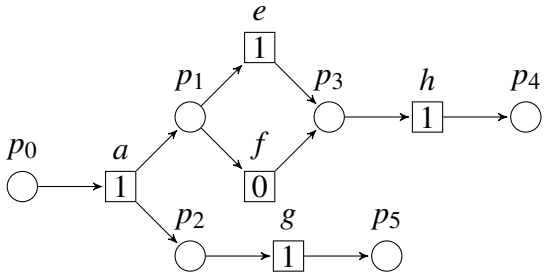


Fig. 6.1 A scenario induced by transitions for whom the corresponding variables are assigned 1 or 0 (see the acyclic net in Figure 3.1). Note that the assigned values are shown inside the boxes representing transitions.

Example 65. Figure 6.1 shows an acyclic net *acnet* and an assignment r of values to boolean variables associated with its transitions (i.e., t is associated with in_t). We have $r(in_a) = r(in_e) = r(in_g) = r(in_h) = 1$ and $r(in_f) = 0$.

Transitions a, e, g, h induce a scenario, namely $\text{scenario}_{acnet}(\{a, e, g, h\})$. That is confirmed by the formula $Scenario_{acnet}$ as is satisfied under r. First, we have:

$$\begin{aligned}
Causality_{acnet}[r] &= (in_a \to 1)[r] \wedge (in_f \to in_a)[r] \wedge (in_e \to in_a)[r] \wedge \\
&\quad (in_g \to in_a)[r] \wedge (in_h \to (in_e \vee in_f))[r] \\
&= (1 \to 1) \wedge (0 \to 1) \wedge (1 \to 1) \wedge (1 \to 1) \wedge (1 \to (1 \vee 0)) = 1 \\
NoForwardConflict_{acnet}[r] &= \neg(in_e \wedge in_f)[r] = \neg(1 \wedge 0) = 1 \\
NoBackwardConflict_{acnet}[r] &= \neg(in_e \wedge in_f)[r] = \neg(1 \wedge 0) = 1
\end{aligned}$$

Hence, we have $Scenario_{acnet}[r] = 1$. ◇

Identifying scenarios of a given acyclic net leads also to determining the maximal scenarios. Recall that calculating probabilities introduced in Chapter 3 concerns maximal scenarios only. Moreover, Approach *A* and Approach *B* of removing confusion in Section 3.6 rely on computing *transactions*, which are maximal scenarios of clusters. Therefore, there is a clear need to provide a boolean formula that computes maximal scenarios.

A scenario is maximal if it cannot be extended. Intuitively, it means that no enabled transition exists. Verifying the enabledness property is therefore essential to determine the maximality of a scenario. Recall that a transition t is enabled if each of its non-initial input places received a token, and that such tokens have not yet been consumed. This constraint can be formally translated into a boolean formula $Enabled_t$, as follows:

$$Enabled_t \triangleq \bigwedge_{u \in ({}^\bullet t)^\bullet} \neg in_u \ \wedge \bigwedge_{p \in {}^\bullet t \setminus M^{init}_{acnet}} \bigvee_{u \in {}^\bullet p} in_u$$

where the second clause is omitted if ${}^\bullet t \subseteq M^{init}_{acnet}$.
The first part of the above formula implies that transition t along with all the output transitions u of places which belongs to the preset of t have not been executed (this is captured by $\neg in_u$). The second part of this formula ensures that each non-initial input place p of t received a token produced by some transition u (this is represented by in_u).

Example 66. Consider the acyclic net in Figure 6.2 and the variable assignment r indicated there. Such an assignment induces scenario. We then can examine the enabledness of e, as follows:

$$Enabled_e[r] = \neg(in_e \wedge in_f)[r] \wedge in_a[r] = \neg(0 \wedge 0) \wedge 1 = 1$$

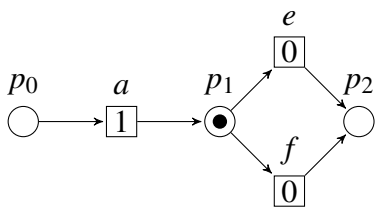


Fig. 6.2 Verifying enabledness for transitions e and f of the acyclic net in Example 66.

Similarly, $Enabled_f[r] = 1$. ◇

After introducing the $Enabled_t$ formula, a formula for maximal scenario can be given. As we mentioned earlier, a scenario is maximal if there are no enabled transitions. Then the negation of $Enabled_t$ (for all t) together with the scenario formula $Scenario_{acnet}$ produces the required constraint for maximal scenarios. Hence, the satisfying assignment for the following formula give all the maximal scenarios:

$$MaxScenario_{acnet} \triangleq Scenario_{acnet} \wedge \bigwedge_{t \in T} \neg Enabled_t$$

The size of $MaxScenario_{acnet}$ is bounded by $|T| + 5 \cdot |T| \cdot \min\{|F|^2, |P| \cdot |T|\}$. Intuitively, it is satisfiable for all the scenarios that do not enable any transitions.

Proposition 6.3.3. Let *acnet* be an acyclic net.

1. If f is a satisfying assignment for $MaxScenario_{acnet}$ then $acnet|_{f^{-1}(1)}$ is a maximal scenario of *acnet*.
2. For every maximal scenario $acnet'$ of *acnet*, there is a satisfying assignment f for $MaxScenario_{acnet}$ such that $acnet|_{f^{-1}(1)} = acnet'$.

Proof. It follows directly from Proposition 6.3.2. □

Hence we can use $MaxScenario_{acnet}$ to find the subsets T' of transitions of *acnet* inducing maximal scenarios after translating it into CNF and feeding into a SAT-solver.

Example 67. Figure 6.3 shows a well-formed acyclic net that exhibits both forward and backward conflict. Observe that the transitions in $\{x, e, f, h\}$ satisfy the causality condition, and have no forward nor backward conflict. Hence, $Scenario_{acnet}$ is satisfied by the assignment r indicated in Figure 6.3. The scenario leads to marking $\{p_5\}$, where no transitions

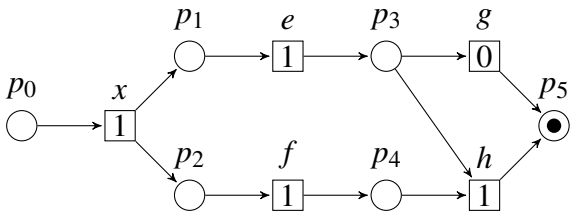


Fig. 6.3 A maximal scenario for a well-formed acyclic net in Example 67.

are enabled. Therefore, the set of transitions $\{x,e,f,h\}$ induces a maximal scenario. This is confirmed by the formula $MaxScenario_{acnet}$, as we have the following:

$$\begin{aligned} MaxScenario_{acnet}[r] &= Scenario_{acnet}[r] \wedge \neg Enabled_x[r] \wedge \neg Enabled_e[r] \wedge \\ &\quad \neg Enabled_f[r] \wedge \neg Enabled_g[r] \wedge \neg Enabled_h[r] \\ &= 1 \wedge 1 \wedge 1 \wedge 1 \wedge 1 \wedge 1 = 1 \end{aligned}$$

Note that $MaxScenario_{acnet}$ also evaluates to 1 if the values for in_g and in_h are swapped. $\diamond$

As shown by the above example, providing a formula to capture maximal scenarios is derived from $Scenario_{acnet}$. Similarly, in the next sub-sections, $Scenario_{acnet}$ is used as a basis to capture other properties.

6.3.2 Well-formedness

The property of well-formedness of an acyclic net is given in Definition 3.2.12. It asserts, in particular, that no place is filled with a token more than once. To characterise acyclic nets which are not well-formed, we use the $Scenario_{acnet}$ and $Enabled_t$ formulas, as follows:

$$NotWellFormed_{acnet} \triangleq Scenario_{acnet} \wedge \bigvee_{t \in T} \left(Enabled_t \wedge \bigvee_{u \in {}^\bullet(t^\bullet)} in_u \right)$$

Intuitively, in the above formula, $Scenario_{acnet}$ 'selects' one of the executions (scenarios) of *acnet* which does not violate well-formedness, and the second part means that there is a transition which can be executed after that violating well-formedness.
The size of the above formula is bounded by $|T| + 6 \cdot |T| \cdot \min\{|F|^2, |P| \cdot |T|\}$.

Hence we can use $NotWellFormed_{acnet}$ to find out whether *acnet* is well-formed after translating it into CNF and feeding into a SAT-solver.

Example 68. The acyclic net in Figure 6.4 is not well-formed as $\text{scenario}_{acnet}(\{e,b,d\})$ enables transition a which violates well-formedness (firing a violates Definition 3.2.11 as

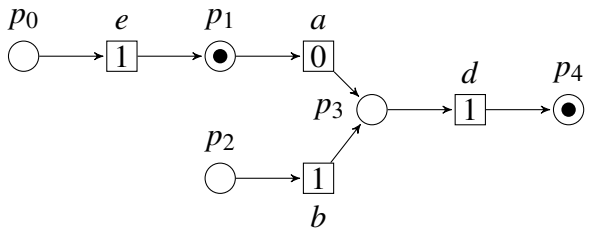


Fig. 6.4 A non-well-formed acyclic net with assignment variables.

that causes p_3 to be filled twice). This is confirmed by the the assignment f illustrated in Figure 6.4, as we have the following:

- $Scenario_{acnet}[f] = 1$.
- $Enabled_a[f] = 1$.
- $(\bigvee_{u \in {}^\bullet(a^\bullet)} in_u)[f] = (in_a \vee in_b)[f] = (0 \vee 1) = 1$.

Hence, $NotWellFormed_{acnet}[f] = 1$. ◇

Proposition 6.3.4. An acyclic net *acnet* is well-formed iff $NotWellFormed_{acnet}$ does not have a satisfying assignment.

Proof. Recall that we have (see Section 6.3.2 on Page 149):

$$NotWellFormed_{acnet} \triangleq Scenario_{acnet} \wedge \bigvee_{t \in T} \left(Enabled_t \wedge \bigvee_{u \in {}^\bullet(t^\bullet)} in_u \right)$$

($\Rightarrow$) Assume that $NotWellFormed_{acnet}$ has a satisfying assignment f. Then, by Proposition 6.3.2(1), $acnet' = acnet|_{f^{-1}(1)}$ is a scenario of *acnet*. Then there is a step sequence σ which induces $acnet'$. Moreover, there is transition t such that $Enabled_t[f] = 1$ and there is $u \in {}^\bullet(t^\bullet)$ such that $f(in_u) = 1$. This means that $\sigma\{t\}$ is a step sequence of *acnet* and u occurs in σ. Hence *acnet* is not well-formed.

($\Leftarrow$) Assume that *acnet* is not well-formed. Then there is a step sequence $\sigma = U_1 \ldots U_k$ such that σ does not meet the conditions in Definition 3.2.12, but $\sigma' = U_1 \ldots U_{k-1}$ meets the conditions in Definition 3.2.12. Hence, σ' induces a scenario captured by $\text{scenario}_{acnet}(\sigma')$ and there are two transitions, $t \in U_k$ and $u \in \sigma'$ such that $t^\bullet \cap u^\bullet \neq \varnothing$. From Proposition 6.3.2(2) there is a satisfying assignment f for the formula $Scenario_{acnet}$ such that $acnet|_{f^{-1}(1)} = \text{scenario}_{acnet}(\sigma')$, where the assignment function f is such that $f(in_z) = 1$ for $z \in \sigma'$, and $f(in_z) = 0$ for $z \notin \sigma$. Therefore, the conjunction of $Scenario_{acnet}$, $Enabled_t$, and

in_u evaluates to 1 under f. Hence, we conclude that the $NotWellFormed_{acnet}$ formula has a satisfying assignment. □

6.3.3 Not dead transitions

Definition 3.2.12(2) asserts that a well-formed *acnet* has non-dead transitions if each transition is guaranteed to occur in at least one of its steps sequences. In other words, if for each transition t there is a scenario that contains t.

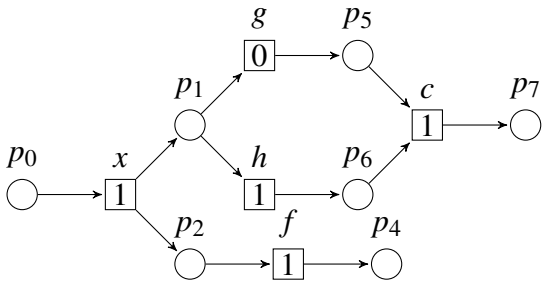


Fig. 6.5 A well-formed acyclic net with a dead transition c (see Example 69).

Example 69. Consider the backward deterministic acyclic net *acnet* in Figure 6.5. The two transitions g and h are in direct conflict. Also, they are both required for c to be enabled. Hence, c is in a self-conflict. This means that c can never be executed, i.e., there is no step sequence $\sigma \in \text{sseq}(acnet)$ such that $c \in \bigcup \sigma$. Therefore, c is not included in any scenario of $\text{scenarios}(acnet)$, and can be seen as an irrelevant part of *acnet*.

This is confirmed by evaluating the relevant boolean formulas. For the acyclic net in Figure 6.5, we can see that $Scenario_{acnet}$ is not satisfiable for any assignment r such that $r(in_c) = 1$ since

$$\begin{aligned} Causality_{acnet} &= (in_x \to 1) \wedge (in_g \to in_x) \wedge (in_h \to in_x) \wedge \\ &\quad (in_f \to in_x) \wedge (in_c \to (in_g \wedge in_h)) \\ NoForwardConflict_{acnet} &= \neg(in_g \wedge in_h) \\ NoBackwardConflict_{acnet} &= \neg(in_h \wedge in_g) \end{aligned}$$

Hence, if $r(in_c) = 1$ and $Causality_{acnet}[r] = 1$ then we have $r(in_g) = r(in_h) = 1$, and so $NoForwardConflict_{acnet}[r] = 0$. Therefore, $Scenario_{acnet}[r] = 0$. This means there is no scenario such that c belongs to it. ◇

Checking whether a transition t belongs to at least one scenario can be done using the following formula:

$$NotDead^{t}_{acnet} \triangleq in_t \wedge Scenario_{acnet} \ .$$

Example 70. Consider again the acyclic net in Figure 6.5. In Example 69, we showed that there is no scenario satisfiable by $Scenario_{acnet}$ in which c is included. This can be tested using the formula:

$$NotDead^{c}_{acnet} = in_c \wedge Scenario_{acnet}$$

As another example, verifying the not dead property for transition h in Figure 6.3 can be done by observing that with the assignment r in Figure 6.3 the corresponding formula evaluates to 1 as shown below:

$$NotDead^{h}_{acnet}[r] = in_h[r] \wedge Scenario_{acnet}[r] = 1 \wedge 1 = 1 \ .$$

◇

Proposition 6.3.5. A well-formed acyclic net *acnet* has only non-redundant transitions iff the formula $NotDead^{t}_{acnet}$ is satisfiable for every $t \in T$.

6.3.4 Marked places and deadlocked scenarios

Consider a scenario of a well-formed acyclic net *acnet*. Then a place $p \in P$ is marked when at least one transition in its preset has been executed and no transition in its postset has consumed the token from p. This can be captured as follows:

$$Mark_p \triangleq \bigvee_{t \in {}^\bullet p} in_t \ \wedge \ \bigwedge_{u \in p^\bullet} \neg in_u$$

where $\bigvee_{t \in {}^\bullet p} in_t$ is omitted if p has no input transitions, and $\bigwedge_{u \in p^\bullet} \neg in_u$ is omitted if p has no output transitions. The size of the above formula is bounded by $2 \cdot |T|$, and it can be used to answer questions such as "is a specific marking reachable?".

The following formula can be used to check whether there is a reachable marking in which all places in a non-empty set $M \subseteq P$ are marked:

$$Reach^{M}_{acnet} \triangleq Scenario_{acnet} \ \wedge \ \bigwedge_{p \in M} Mark_p$$

Example 71. To illustrate how to use $Mark_p$ to verify that place p_1 in Figure 6.2 is marked under the assignment r indicated there, we have the following:

$$Mark_{p_1}[r] = in_a[r] \ \wedge \ (\neg in_f \wedge \neg in_e)[r] = 1 \wedge 1 = 1$$

Also, in Figure 6.2, $M = \{p_1\}$ is reached by the scenario $\text{scenario}_{acnet}(\{a\})$ induced by r as the next formula is satisfied:

$$Reach_{acnet}^{\{p_1\}}[r] = Scenario_{acnet}[r] \wedge Mark_{p_1}[r] = 1 \wedge 1 = 1$$

$\diamond$

Proposition 6.3.6. Let *acnet* be a well-formed acyclic net and M be a non-empty set of its places. Then there is a reachable marking M' satisfying $M \subseteq M'$ iff $Reach_{acnet}^{M}$ has a satisfying assignment.

Proof. Recall that the following hold (see Section 6.3.4 on Page 152):

$$Reach_{acnet}^{M} \triangleq Scenario_{acnet} \wedge \bigwedge_{p \in M} Mark_p \qquad \text{and} \qquad Mark_p \triangleq \bigvee_{t \in {}^\bullet p} in_t \wedge \bigwedge_{u \in p^\bullet} \neg in_u.$$

($\Rightarrow$) Let M' be a marking reachable through step sequence σ such that $M \subseteq M'$. Since *acnet* is well-formed, from [8], σ induces scenario $acnet' = Scenario_{acnet}(\sigma)$ Since $acnet'$ is a scenario, then from Proposition 6.3.2(2), there is a satisfying assignment for $Scenario_{acnet}$ such that $acnet|_{f^{-1}(1)} = acnet'$. This implies that the assignment function f is such that $f(in_t) = 1$ for $t \in \sigma$, and $f(in_t) = 0$ for $t \notin \sigma$.

Also, a place $p \in M$ is marked when a transition $t \in {}^\bullet p$ is fired and belongs to σ and no transitions $u \in p^\bullet$ have consumed such token, which implies that $f(in_t) = 1$, and $f(in_u) = 0$ for each $u \in p^\bullet$. Hence, $Mark_p[f] = 1$, which implies $Scenario_{acnet}[f] = 1$ and $\bigwedge_{p \in M} Mark_p[f] = 1$. We then conclude that $Reach_{acnet}^{M}$ has a satisfying assignment.

($\Leftarrow$) Let f be an assignment such that $Reach_{acnet}^{M}[f] = 1$. Since $Scenario_{acnet}[f] = 1$, then from Proposition 6.3.2(1) $acnet|_{f^{-1}(1)}$ is a scenario of *acnet*. Also, *acnet* is well-formed and so there is a step sequence $\sigma = U_1 \ldots U_k$ such that $acnet|_{f^{-1}(1)} = \text{scenario}_{acnet}(\sigma)$, and $f(in_t) = 1$ for every $t \in \sigma$, and $f(in_t) = 0$ for every $t \notin \sigma$. Since $\bigwedge_{p \in M} Mark_p[f] = 1$, then all the places $p \in M$ are marked by $\text{scenario}_{acnet}(\sigma)$. Hence, M' reached by $\text{scenario}_{acnet}(\sigma)$ is such that $M \subseteq M'$. $\square$

No execution of an acyclic net $acnet = (P, T, F)$ can be extended indefinitely. However, one may consider a scenario as *deadlocked* if it is maximal and some of the places it marks do

not belong to M^{fin}_{acnet}. (Verification of deadlock-freeness using SAT-solvers has been studied in, e.g., [106, 1, 90].) All the deadlocked scenarios can be captured by the following formula:

$$DeadlockScenario_{acnet} \triangleq MaxScenario_{acnet} \wedge \bigvee_{p \in P \setminus M^{fin}_{acnet}} Mark_p$$

The size of $MaxScenario_{acnet}$ is bounded by $|T| + 5 \cdot |T| \cdot \min\{|F|^2, |P| \cdot |T|\} + 2 \cdot |P| \cdot |T|$.

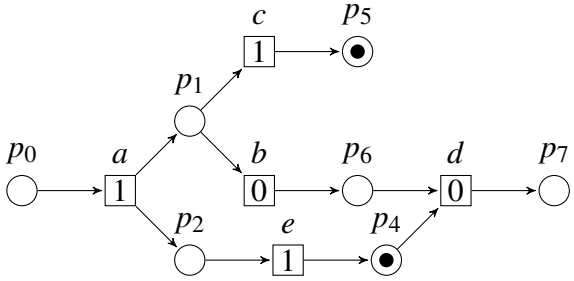


Fig. 6.6 A deadlocked maximal scenario for the acyclic net in Example 72.

Example 72. According to the the assignment f illustrated in Figure 6.6 $MaxScenario_{acnet}[f] = 1$ and so $\{a, c, e\}$ induce a maximal scenario. This maximal scenario is deadlocked as it leads to the reachable marking $\{p_4, p_5\}$ which marks a non-final place p_4. We also have $Mark_{p_4}[f] = 1$ and $p_4 \in P \setminus M^{fin}_{acnet}$, and so $DeadlockScenario_{acnet}[f] = 1$. $\diamond$

6.3.5 Backward deterministic acyclic nets

In this section, we assume that each acyclic net is backward deterministic. In this case, some formulas defined earlier can be simplified. For instance, the formula capturing causality in Section 6.3.1, can be reduced by only considering the conjunction of all the transitions causally related to a certain transition t. Also, the $Scenario_{acnet}$ formula would be the conjunction of $Causality_{acnet}$ and $NoForwardConflict_{acnet}$ only as there is no backward conflict. $Enabled_t$ and $Mark_p$ formulas are similarly modified. Finally, since this section concerns only backward deterministic nets, then there is no need for checking well-formedness property as by Proposition 3.2.1 well-formedness holds for each backward deterministic acyclic net. Below we introduce all the important modifications for these formulas.

- Causality: $Causality_{acnet} \triangleq \bigwedge_{t \in T} \left(in_t \rightarrow \bigwedge_{u \in {}^\bullet({}^\bullet t)} in_u \right)$.
- Enabling: $Enabled_t \triangleq \bigwedge_{u \in ({}^\bullet t)^\bullet} \neg in_u \wedge \bigwedge_{u \in {}^\bullet({}^\bullet t)} in_u$.

- Scenario: $Scenario_{acnet} \triangleq Causality_{acnet} \wedge NoForwardConflict_{acnet}$.
- Place marking (for $p \in P$ and $t \in T$ satisfying $\{t\} = {}^\bullet p$): $Mark_p \triangleq in_t \wedge \bigwedge_{u \in p^\bullet} \neg in_u$.
- Checking well-formedness is not needed as *acnet* is well-formed.

6.3.6 Detecting confusion in acyclic nets using SAT

The papers [90] and [24] used FDR model-checker to detect conflict and confusion after translating concurrent Petri nets into Communicating Sequential Processes (CSP). Their approach of detecting confusion is related to trace theory and it addressed only the structural properties of confusion. Also, [31] proposed an algorithm for detecting confusion by analysing its structure. In this section, an extended version of the work in [14] is introduced by constructing a satisfiability checking for detecting confusion in a given acyclic net. Our approach however considers both aspects of confusion: structural and dynamic. In particular, the structural constraints are captured by the conditions identified according to Definition 3.4.1 on Page 34. This static information is related to the set of distinct transitions e, f, h involved in a potential confusion. For instance, the below captures the static conditions of symmetric confusion:

$$PotSymConfused_{acnet} = \{(e,h,f) \mid e\#_0 h \ \wedge \ f\#_0 h \ \wedge \ {}^\bullet e \cap {}^\bullet f = \varnothing\}$$

With the above, the dynamic constrains of symmetric confusion can be obtained as shown below:

$$Symconfused_{acnet} \triangleq Scenario_{acnet} \ \wedge \bigvee_{(e,f,h) \in PotSymConfused_{acnet}} Enabled_e \ \wedge \ Enabled_f \ \wedge \ Enabled_h$$

Similarly, the asymmetric case can be detected using the following formula:

$$Asymconfused_{acnet} \triangleq Scenario_{acnet} \ \wedge \bigvee_{(e,f,h) \in PotAsymConfused_{acnet}} \left(Enabled_e \wedge Enabled_f \wedge \neg Enabled_h \wedge \bigwedge_{p \in {}^\bullet h \setminus f^\bullet} Mark_p\right)$$

where $PotAsymConfused_{acnet} = \{(e,h,f) \mid e\#_0 h \wedge {}^\bullet f \cap {}^\bullet h = {}^\bullet f \cap {}^\bullet e = \varnothing\}$ is set of distinct transitions $e, f, h \in T$ involved in a potential asymmetric confusion. Hence, detecting any confusion for a given *acnet* can in general be done using the following formula:

$$Confusion_{acnet} \triangleq Symconfused_{acnet} \vee \ Asymconfused_{acnet}$$

Proposition 6.3.7. An acyclic net *acnet* is confused iff $Confusion_{acnet}$ has a satisfying assignment.

Proof. Follows directly form Definition 3.4.1 in Section 3.4 on Page 34. □

Example 73. Consider the acyclic nets in Figure 6.7 which shows the two types of confusion. In Figure 6.7(a), we have a symmetric confusion for $\text{scenario}_{acnet}(\{d\})$. The structural constraints are represented by $PotSymConfused_{acnet} = \{(a,b,c)\}$. Then, the symmetric confusion is detected by the following formula:

$$Symconfused_{acnet} \triangleq Scenario_{acnet} \wedge Enabled_a \wedge Enabled_b \wedge Enabled_c$$

which evaluates to $1 \wedge 1 \wedge 1 \wedge 1 = 1$ under the assignment shown in Figure 6.7(a).

Similarly, the acyclic net in Figure 6.7(b) exhibits an asymmetric confusion for $\text{scenario}_{acnet}(\{b\})$. The structural constraints are represented by $PotAsymConfused_{acnet} = \{(a,b,c)\}$. Then, the asymmetric confusion is detected by the following formula:

$$Asymconfused_{acnet} \triangleq Scenario_{acnet} \wedge Enabled_b \wedge Enabled_a \wedge \neg Enabled_c \wedge Mark_{p_2}$$

which evaluates to $1 \wedge 1 \wedge 1 \wedge 1 \wedge 1 = 1$ under the assignment shown in Figure 6.7(b). ◇

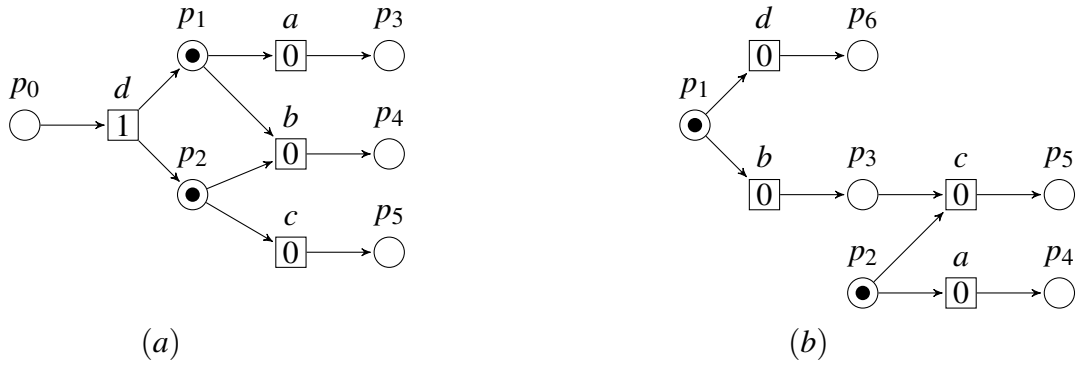


Fig. 6.7 Symmetric confusion (a) and asymmetric confusion (b) in an acyclic net with assignment variables.

6.4 Verifying properties of CSA-nets

In this section, we extend SAT-encodings concerning the properties of acyclic nets to include CSA-nets. It is worth noting that this is the first work aimed at transforming CSA-nets behavioural properties into SAT problems.

The development and intuitive meaning of formulas needed to check the basic properties of CSA-nets are similar as in the case of acyclic nets. Therefore, we will avoid repeating some of the explanations and formal properties. Also, the sizes of formulas introduced for CSA-nets are similar to those developed for acyclic nets with the number of transitions being sometimes replaced by the number of syn-cycles.

Identifying scenarios and maximal scenarios

Similarly to identifying a scenario in a given acyclic net, we say that a set of transitions $T' \subseteq T_{csan}$ induces a scenario of *csan* if the following three constraints are satisfied: *Causality*, *No Forward Conflict*, and *No Backward Conflict*. The next definition and proposition make this more formal.

Definition 6.4.1 (restricting CSA-net)**.** The *restriction* of a CSA-net

$$csan = (acnet_1, \ldots, acnet_n, Q, W)$$

to $T' \subseteq T_{csan}$ is

$$csan|_{T'} \triangleq (acnet_1|_{T'}, \ldots, acnet_n|_{T'}, Q', W|_{(Q' \times T') \cup (T' \times Q')}),$$

where $Q' = Q \cap \mathrm{post}_{acnet}(T')$. ◇

Proposition 6.4.1. The following are equivalent for $T' \subseteq T_{csan}$:

1. $csan|_{T'} \in \mathrm{scenarios}(csan)$.
2. $csan|_{T'}$ is a CSO-net and $\mathrm{pre}_{csan}(T') \setminus M^{init}_{csan} \subseteq \mathrm{post}_{csan}(T')$.

Proof. This result can be shown in a similar way as Proposition 6.3.1. The only important modification is to consider the syn-cycles rather than individual transitions. □

The following boolean variables will be used in the construction of $Scenario_{csan}$ and the translation into SAT problem.

- For every $t \in T_{csan}$, we have a variable in_t tracing that t belongs to a scenario.

The constraints on the above variables are defined, following Proposition 6.3.1, as follows:

- To ensure that all non-initial pre-places of T' received tokens from T':

$$Causality_{csan} \triangleq \bigwedge_{t \in T_{csan}} \Big(in_t \rightarrow \bigwedge_{p \in \mathrm{pre}_{csan}(t) \setminus M^{init}_{csan}} \bigvee_{u \in \mathrm{pre}_{csan}(p)} in_u\Big)$$

- To ensure that scenario has no forward conflicts:

$$NoForwardConflict_{csan} \triangleq \bigwedge_{t \in T_{csan}} \bigwedge_{u \in \mathrm{post}_{csan}(\mathrm{pre}_{csan}(t)) \setminus \{t\}} \neg\big(in_t \wedge in_u\big)$$

- To ensure that scenario has no backward conflicts:

$$NoBackwardConflict_{csan} \triangleq \bigwedge_{t \in T_{csan}} \bigwedge_{u \in \mathrm{pre}_{csan}(\mathrm{post}_{csan}(t)) \setminus \{t\}} \neg\big(in_t \wedge in_u\big)$$

Note that sometimes parts of the above formulas may be omitted, similarly as in the case of acyclic nets. Then the formula which characterises all the scenarios of *csan* is:

$$Scenario_{csan} \triangleq Causality_{csan} \wedge NoForwardConflict_{csan} \wedge NoBackwardConflict_{csan}$$

The satisfying assignments of $Scenario_{csan}$ identify precisely all the scenarios of *csan* which is not necessary well-formed.

Example 74. All the transitions in Figure 6.8 which have value 1 assigned by the indicated assignment r represent a scenario for *csan* because we have:

$$\begin{aligned}
&Causality_{csan}[r] \\
&\quad = \ (in_f \rightarrow 1)[r] \wedge ((in_d \rightarrow in_A)[r] \wedge (in_e \rightarrow in_f)[r] \wedge (in_A \rightarrow in_e)[r] \wedge \\
&\qquad (in_B \rightarrow (in_e \wedge in_C))[r] \wedge ((in_C \rightarrow (in_f \wedge in_B)))[r]) \\
&\quad = \ (0 \rightarrow 0) \wedge (1 \rightarrow 1) \wedge (0 \rightarrow 1) \wedge (1 \rightarrow (1 \wedge 1)) \wedge (1 \rightarrow (1 \wedge 1)) \wedge (1 \rightarrow 1) = 1 \\
&NoForwardConflict_{csan}[r] \\
&\quad = \ (\neg(in_A \wedge in_B)[r] \wedge \neg(in_B \wedge in_A)[r]) = \neg(0 \wedge 1) \wedge \neg(1 \wedge 0) = 1. \\
&NoBackwardConflict_{csan}[r] \\
&\quad = \ (\neg(in_A \wedge in_B)[r] \wedge \neg(in_B \wedge in_A)[r]) = \neg(0 \wedge 1) \wedge \neg(1 \wedge 0) = 1
\end{aligned}$$

Hence, $Scenario_{csan}[r] = 1 \wedge 1 \wedge 1 = 1$. ◇

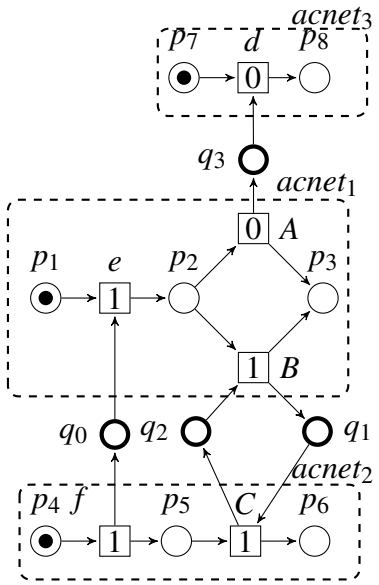


Fig. 6.8 A CSA-net with a scenario indicated by transitions assigned 1.

Proposition 6.4.2. Let *csan* be a CSA-net.

1. If f is a satisfying assignment for $Scenario_{csan}$ then $csan|_{f^{-1}(1)}$ is a scenario of *csan*.
2. For every scenario $csan'$ of CSA-net *csan*, there is a satisfying assignment f for $Scenario_{csan}$ such that $csan|_{f^{-1}(1)} = csan'$.

Proof. This result can be shown in a similar way as Proposition 6.3.2. The only important modification is to consider the syn-cycles rather than individual transitions. □

The formula $Scenario_{csan}$ represents scenarios which are not necessarily maximal. In order to capture the maximality of a scenario, we need to evaluate the enabledness property as any scenario is maximal iff there are no enabled steps that can be executed. In this case, however, sync-cycles (which include individual transitions) rather than individual transitions need to be checked for enabledness.

The following formula captures the enabledness of a syn-cycle $S \in \text{syncycles}(csan)$:

$$Enabled_S \triangleq \bigwedge_{u \in \text{post}_{csan}(\text{pre}_{csan}(S))} \neg in_u \ \wedge \bigwedge_{p \in \text{pre}_{csan}(S) \setminus (M^{init}_{csan} \cup (Q \cap \text{post}_{csan}(S)))} \bigvee_{u \in \text{pre}_{csan}(p)} in_u$$

Thus S is a set of synchronised transitions whose enabledness depends on each other. For instance, in Figure 6.8, B and C are transitions involved in a synchronous communication and $S = \{B, C\}$ is a syn-cycle.

Finally, the following formula represents the set of transitions that induce a maximal scenario of a *well-formed csan*:

$$MaxScenario_{csan} \triangleq Scenario_{csan} \wedge \bigwedge_{S \in \text{syncycles}(csan)} \neg Enabled_S \,.$$

Example 75. The scenario induced by $\{f, e, B, C\}$ in Figure 6.8 is also a maximal scenario as there is no enabled syn-cycle. Moreover, $MaxScenario_{csan}[r] = 1$ for the assignment r indicated there. ◇

6.4.1 Well-formedness

Translating well-formedness into SAT formula requires the two formulas $Scenario_{csan}$ and $Enabled_S$ introduced earlier. Informally speaking, *csan* is not well-formed if there is a subset of transitions that induce a scenario which does not violate well-formedness and there is an enabled sync-cycle S such that its execution violates well-formedness. The following formula detects whether a given *csan* is not well-formed:

$$NotWellFormed_{csan} \triangleq Scenario_{csan} \wedge \bigvee_{S \in \text{syncycles}(csan)} \left(Enabled_S \wedge \bigvee_{u \in \text{pre}_{csan}(\text{post}_{csan}(S))} in_u\right) .$$

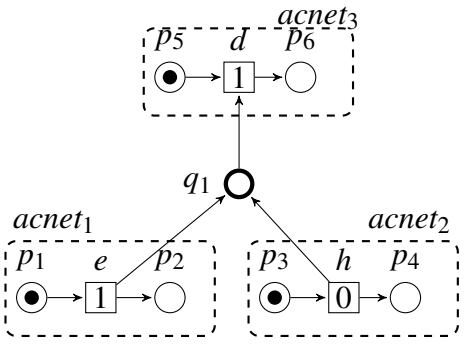


Fig. 6.9 A non-well-formed CSA-net with assignment variables.

Example 76. Figure 6.9 displays a non-well-formed CSA-net (the same in Figure 4.3) with the assignment r. According to r, $Scenario_{csan}[r] = 1$. Also, $Enabled_h[r] = 1$, $r(in_e) = 1$ and $e \in \text{pre}_{csan}(\text{post}_{csan}(h))$. Hence, $NotWellFormed_{csan}[r] = 1 \wedge (1 \wedge 1) = 1$. ◇

Proposition 6.4.3. A CSA-net *csan* is well-formed iff $NotWellFormed_{csan}$ does not have a satisfying assignment.

Proof. This result can be shown in a similar way as Proposition 6.3.4. The only important modification is to consider the syn-cycles rather than individual transitions. □

6.4.2 Not dead transitions, marked places, and deadlocked scenarios

The absence of dead transitions of *csan* can be verified in the same way as for an acyclic net, by checking that, for every $t \in T_{csan}$, the following formula has a satisfying assignment:

$$NotDead^{t}_{csan} \triangleq in_t \ \wedge \ Scenario_{csan}$$

The formula for checking whether a place p is marked by a scenario is also similar as before:

$$Mark^{csan}_{p} \triangleq \bigvee_{t \in \mathrm{pre}_{csan}(p)} in_t \ \wedge \bigwedge_{u \in \mathrm{post}_{csan}(p)} \neg in_u$$

Moreover, checking whether a set of places M is marked by a scenario is achieved by:

$$Reach^{M}_{csan} \triangleq Scenario_{csan} \ \wedge \bigwedge_{p \in M} Mark^{csan}_{p}$$

Detecting deadlocked scenarios can then be done using the following formula:

$$DeadlockScenario_{csan} \triangleq MaxScenario_{csan} \ \wedge \bigvee_{p \in P_{csan} \setminus M^{fin}_{csan}} Mark_p$$

Note that it is not required that the buffer places are empty in non-deadlocked markings.

Proposition 6.4.4. Let *csan* be a well-formed CSA-net and M be a non-empty set of its places. Then there is a reachable marking M' satisfying $M \subseteq M'$ iff $Reach^{M}_{csan}$ has a satisfying assignment.

Proof. This result can be shown in a similar way as Proposition 6.3.6. The only important modification is to consider the syn-cycles rather than individual transitions. □

6.4.3 Backward deterministic CSA-nets

When *csan* is a backward deterministic CSA-net, some formulas defined earlier can be simplified, for example:

$$\begin{array}{lcl} Causality_{csan} & \triangleq & \bigwedge_{t\in T_{csan}} \left(in_t \rightarrow \bigwedge_{p\in \mathrm{pre}_{csan}(\mathrm{pre}_{csan}(t))} in_u\right) \\ Enabled_S & \triangleq & \bigwedge_{u\in \mathrm{post}_{csan}(\mathrm{pre}_{csan}(S))} \neg in_u \ \wedge \ \bigwedge_{u\in \mathrm{pre}_{csan}(\mathrm{pre}_{csan}(S))\setminus S} in_u \end{array}$$

Then, the $Scenario_{csan}$ formula is reduced to:

$$Scenario_{csan} \triangleq Causality_{csan} \wedge NoForwardConflict_{csan}$$

and the formula for marking place p with $\mathrm{pre}_{csan}(p) = \{t\}$ is:

$$Mark_p \triangleq in_t \ \wedge \bigwedge_{u\in \mathrm{post}_{csan}(p)} \neg in_u$$

6.4.4 Detecting confusion in CSA-nets using SAT

This section extends the approach of detecting confusion in acyclic nets to CSA-nets. As in the strategy of verifying the static constraints of confusion in acyclic nets, we provide SAT-encoding for the structural features of confusion in CSA-net. In particular, the set of syn-cycles involved in potential confusions are considered instead of individual transitions. In the case of symmetric confusion, potential confusions are defined as follows, according to the first part of Definition 4.8.1:

$$PotSymConfused_{acnet} = \{(S_1,S_2,S_3) \mid S_1 \#_0 S_3 \ \wedge \ S_2 \#_0 S_3 \ \wedge \ {}^\bullet S_1 \cap {}^\bullet S_2 = \varnothing\}$$

Then the following formula detects symmetric confusion in a given *csan*:

$$Symconfused_{csan} \triangleq Scenario_{csan} \ \wedge \bigvee_{(S_1,S_2,S_3)\in PotSymConfused_{csan}} \left(Enabled_{S_1} \wedge Enabled_{S_2} \wedge Enabled_{S_3}\right)$$

The asymmetric case can be detected using the following formula:

$$Asymconfused_{csan} \triangleq Scenario_{csan} \wedge \bigvee_{(S_1,S_2,S_3)\in PotAsymConfused_{csan}} \left(Enabled_{S_1} \wedge Enabled_{S_2} \wedge \neg Enabled_{S_3} \wedge \bigwedge_{p\in \mathrm{pre}_{csan}(S_3)\setminus \mathrm{post}_{csan}(S_2)} Mark_p\right)$$

where $PotAsymConfused_{csan}$ captures the static constraints, according to the second part of Definition 4.8.1, for syn-cycles involved in potential asymmetric confusions:

$$PotAsymConfused_{csan} = \{(S_1, S_2, S_3) \mid S_1 \#_0 S_3 \ \wedge \ {}^\bullet S_1 \cap {}^\bullet S_2 = \varnothing \ \wedge \ {}^\bullet S_2 \cap {}^\bullet S_3 = \varnothing\}$$

Then, detecting any confusion can be done using the following formula:

$$Confusion_{csan} \triangleq Symconfused_{csan} \vee Asymconfused_{csan}$$

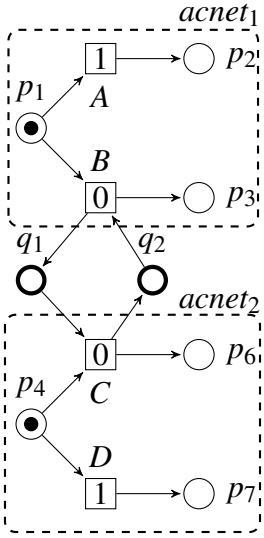


Fig. 6.10 A symmetric confusion in CSA-net with assignment variables.

Example 77. A symmetric confusion is depicted in Figure 6.10 (for the confused CSA-net in Figure 4.5) together with a variable assignment r. From Example 40, we have

$$PotSymConfused_{acnet} = \{(S_1, S_2, S_3)\},$$

where $S_1 = \{A\}$, $S_2 = \{D\}$ and $S_3 = \{B, C\}$. Also, we have

$$Symconfused_{csan}[r] = Scenario_{csan}[r] \wedge Enabled_{S_1}[r] \wedge Enabled_{S_2}[r] \wedge Enabled_{S_3}[r] = 1.$$

$\diamond$

Proposition 6.4.5. A CSA-net *csan* is confused iff $Confusion_{csan}$ has a satisfying assignment.

Proof. It follows directly from Definition 4.8.1 in Section 4.8 on Page 96. □

6.5 Verifying properties of BSA-nets

In this section, we discuss how to extend the SAT-based verification approach developed for acyclic nets and CSA-nets to BSA-nets.

In BSA-nets, there are some additional constraints that are not required in acyclic nets and CSA-nets. For instance, Definition 5.2.3 asserts that all markings of lower-level and upper-level nets must respect the start and end of the phases. In turn, that imposes restrictions over the steps enabledness. Therefore, before constructing boolean formulas for checking behavioural properties of BSA-nets, translating *phase* and *phase-consistent* markings into SAT-encodings is essential. The underlying CSA-net of a BSA-net *bsan*, i.e., $csan = \text{csan}(bsan)$ as in Definition 5.2.2, will be used in constructing verification formulas.

In the rest of this section, we assume that *bsan* and $csan = \text{csan}(bsan)$ are as Definitions 5.2.1 and 5.2.2, together with other associated definitions.

6.5.1 Identifying valid executions for BSA-nets

The phase of a place p belonging to $hacnet_i$ is the set of markings in $lcsan_i$ reached along a step sequence starting from β_p and ending at one of the markings β_r, for $r \in \text{post}_{hacnet}(\text{post}_{hacnet}(p))$. The following formula captures this constraint:

$$Phase_p \triangleq \bigvee_{R \in \text{phase}(p)} \Big(\bigwedge_{q \in R} Mark_q \wedge \bigwedge_{q \in P_{lcsan_i} \setminus R} \neg Mark_q \Big)$$

The idea of verification of phase-consistent marking in BSA-nets is to separate the static and the dynamic part. The static part is related to calculating markings belonging to the phase of a place p which is captured by the formula $Phase_p$.

The dynamic part is to verify that a marked place p in *hacnet* is matched by a marking in $\text{phase}(p)$. After introducing the $Phase_p$ formula, it is possible now to capture formally this constraint as it is shown below:

$$PhConsMarking_p \triangleq \; Mark_p \rightarrow Phase_p$$

And for the whole BSA-net we have:

$$PhConsMarking_{bsan} \triangleq \bigwedge_{p \in P_{hacnet}} Mark_p \rightarrow Phase_p$$

The above formula translates the second part of Definition 5.2.3. Basically, this formula ensures that the makings of lower-level acyclic nets belong to the phases corresponding to the markings of the upper-level acyclic nets.

In this section we do not provide a formula for characterising exactly all the scenarios of *bsan*. We leave this for the future work. Instead, we discuss some initial findings and ideas.

We first observe that the step sequences of *bsan* (behaviours) are those step sequences of csan($bsan$) which are based on phase-consistent marking. In particular, each such step sequence must end at a phase-consistent marking. It is therefore the case that the set of all step sequences of csan($bsan$) which end at phase-consistent markings provide an *over-approximation* of the step sequences of *bsan*. Such a characterisation can be clearly useful to verify various behavioural properties of *bsan*'s behaviours. The (over-approximated) characterisation of the behaviours of *bsan* is given by the following formula:

$$PhConsScenario_{bsan} \triangleq Scenario_{\text{csan}(bsan)} \wedge PhConsMarking_{bsan}$$

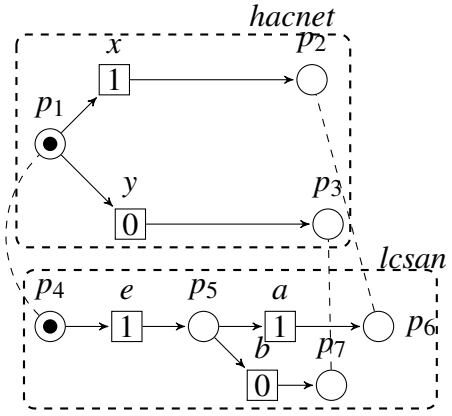


Fig. 6.11 BSA-net with assignment variables.

Example 78. Figure 6.11 illustrates a BSA-net with a variable assignment r. The set of the markings at *lcsan* included in phase(p_1) is as follows:

$$\text{phase}(p_1) = \{\{p_4\}, \{p_5\}, \{p_6\}, \{p_7\}\}.$$

To check that using the $Phase_{p_1}$ formula holds under r, we obtain:

$$\begin{aligned} Phase_{p_1}[r] \;=\; & (Mark_{p_4} \wedge \neg Mark_{p_5} \wedge \neg Mark_{p_6} \wedge \neg Mark_{p_7})[r] \vee \\ & (Mark_{p_5} \wedge \neg Mark_{p_4} \wedge \neg Mark_{p_6} \wedge \neg Mark_{p_7})[r] \vee \\ & (Mark_{p_6} \wedge \neg Mark_{p_4} \wedge \neg Mark_{p_5} \wedge \neg Mark_{p_7})[r] \vee \\ & (Mark_{p_7} \wedge \neg Mark_{p_4} \wedge \neg Mark_{p_5} \wedge \neg Mark_{p_6})[r] \\ & (1 \wedge 1 \wedge 1 \wedge 1) \vee (0 \wedge 0 \wedge 1 \wedge 1) \vee (0 \wedge 0 \wedge 1 \wedge 1) \vee (0 \wedge 0 \wedge 1 \wedge 1) \\ & 1 \vee 0 \vee 0 \vee 0 = 1 \end{aligned}$$

To ensure the marking is phase-consistent, the following are checked:

$$\begin{aligned} PhConsMarking_{p_1}[r] &= Mark_{p_1}[r] \rightarrow Phase_{p_1}[r] = 1 \rightarrow 1 = 1 \\ PhConsMarking_{p_2}[r] &= Mark_{p_2}[r] \rightarrow Phase_{p_2}[r] = 0 \rightarrow 0 = 1 \\ PhConsMarking_{p_3}[r] &= Mark_{p_3}[r] \rightarrow Phase_{p_3}[r] = 0 \rightarrow 0 = 1 \end{aligned}$$

Moreover, $Scenario_{\mathrm{csan}(bsan)}[r] = 1$ as the set of transitions $\{e,a,x\}$ induces a valid scenario $\mathrm{scenario}_{\mathrm{csan}(bsan)}(\{e,a,x\})$. Therefore, $PhConsScenario_{bsan}[r] = 1$. ◇

As before, to evaluate the maximality of behaviours in BSA-nets, the enabledness property needs to be captured. Here the problem is that, in addition to considering the constraint of phase-consistent (current) marking, executing a step should also lead to a phase-consistent marking.

Example 79. The BSA-net in Figure 6.12 shows two upper level acyclic nets with a synchronous communication which forms a syn-cycle $S_1 = \{t,u\}$. Executing S_1 requires the execution of two different syn-cycles, $S_2 = \{e\}$ and $S_3 = \{d\}$, at different lower acyclic nets due to the β relation which enforces consistency between the two levels. ◇

We want to capture conditions for the enabledness of a step U. The following formula checks its enabledness:

$$PhConsEnabled_U \triangleq Enabled_U \;\wedge\; PhConsMarking'_{bsan} \;\wedge\; \bigwedge_{t \in U} in'_t = 1 \;\wedge\; \bigwedge_{t \in T_{bsan} \setminus U} in'_t = in_t$$

where the variables in'_t represent the transitions fired before and in U, $PhConsMarking'_{bsan}$ is $PhConsMarking_{bsan}$ with each in_t replaced by in'_t, and $Enabled_U$ is the enabledness formula introduced for $\mathrm{csan}(bsan)$.

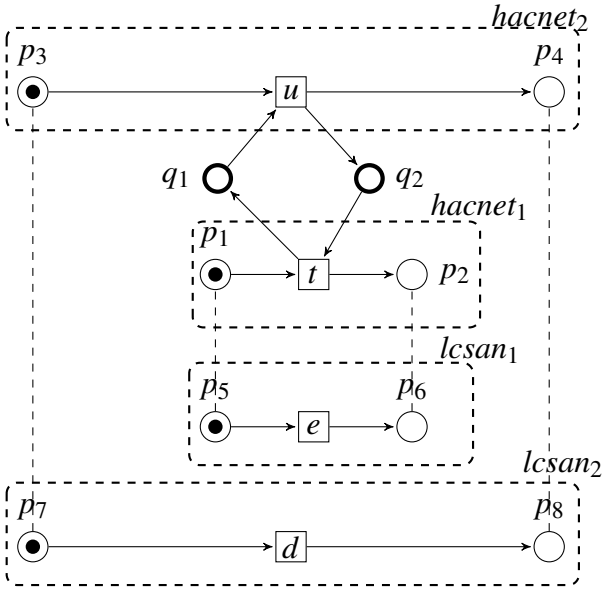


Fig. 6.12 An illustrative example of verifying enabledness property in BSA-nets.

Example 80. Consider *bsan* shown in Figure 6.13 together with the assignment r indicated there. Let $U = \{S_1, S_2, S_3\}$, where $S_1 = \{t, u\}$, $S_2 = \{e\}$ and $S_3 = \{b\}$. Then $PhConsEnabled_U[r] = 1$ and so the boolean formula correctly identifies enabled step. $\diamond$

Checking of other properties can be built upon the formulas we provided above.

6.6 Conclusion

This chapter developed a theoretical framework that concerns the fundamentals of SAT-based verification of acyclic nets, CSA-nets. Extending our work in [14], behavioural properties of BSA-nets were also investigated to enhance the applicability of the proposed verification method. Booleans variables are associated with the transitions to find a satisfying assignment for the formula that verifies certain behavioural properties. For instance, checking that a set of transitions induces a scenario is obtained by checking that they are causally dependent, forward and backward conflict free. Also, the two cases of confusion are detected via a formula that considers the structural and behavioural aspects of confusion.

The challenges in this chapter were related to verifying the behavioural properties of *bsan*. That because most of its behavioural properties are defined in terms of the underlying *csan*.The implication is that the formulas provide an over approximation of behaviour characterisation of *bsan*. In order to define formulas that check exactly *bsan* behaviour

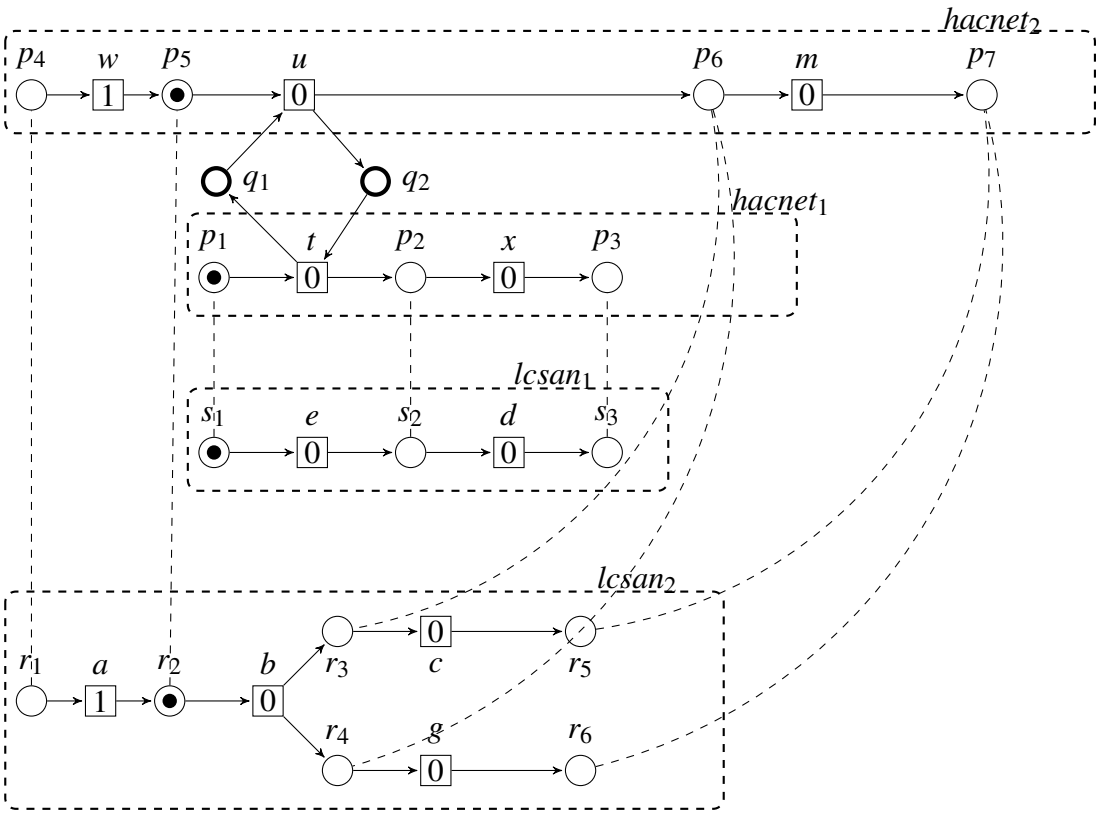


Fig. 6.13 An illustrative example of verifying an enabled step in BSA-nets.

characterisation, steps and step sequences need to be defined for *bsan*. However, we leave that for the future work.

Also, an important future direction of the current work is concerned with an implementation of formulas developed here in SONCraft tool [85]. This would allow comparisons of the efficiency of the proposed model checking technique with other approaches (note that verification problems considered here are NP-complete; see, e.g., [45]). In particular, a comparison with model checking based on finite prefixes of net unfoldings [46, 70] after adapting it to CSA-nets and their step sequence execution semantics. In this case, the branching processes of CSA-nets unfolding algorithm presented in [84] can be used. However, even the unfolding of acyclic nets, where the forward and backward conflict are allowed, would in the worst case generate exponential finite prefixes [71]. The method proposed in this chapter does not suffer from a similar problem.

Chapter 7

Concluding Remarks

7.1 Summary

In this thesis, we have developed a concurrent probabilistic framework for acyclic nets, CSA-nets, and BSA-nets.

In Chapter 3, acyclic nets and their structural and behavioural specifications are considered. Resolving conflicts probabilistically is discussed throughout providing the method of calculating probabilities in acyclic nets. The crucial contribution of this chapter are the approaches of avoiding the confusion from acyclic nets. These approaches have been formally defined and explained using several illustrative examples (see Section 3.6.1 and Section 3.6.6).

In Chapter 4, the basic concepts of Communication Structured Acyclic Nets (CSA-nets) were considered. The initial version of this concept was built on occurrence nets (see [84, 78, 105, 11, 23, 92]), and it aims to provide a notation for representing and recording the activities of complex evolving systems and the interaction of their components. In this chapter, the model is generalised by being based on acyclic nets. Calculating probabilities is extended so that conflict between set of transitions represented by syn-cycles is resolved by probability estimation. Moreover, the definition of confusion and the techniques of handling it are extended.

Chapter 5 extended the probabilistic framework so that probabilities are represented by means of behavioural relation at the upper-level. In behavioural relation, parts of a complex activity are abstracted by another system. This abstraction mechanism is embodied at two levels, and the upper-level provides a simpler abstracted representation of the details captured by the lower-level. In this chapter, BSA-nets are extended so that the structure of the upper-level nets are not line-like as in [78, 84, 105, 11, 92]. However, only free-choice structure is allowed for the upper-level nets. The probabilities of certain alternative behaviours at

the lower-level are represented by the corresponding behaviours at the upper-level. More precisely, calculating probabilities is extended in such way that the weights of the lower-level alternatives are used to assign the probabilities at the upper-level. This is a novel representation of probabilities in the area of concurrent probabilistic models. Exploiting the free-choice structural constraint of the upper-level nets motivated us to propose a new idea of handling confusion. More precisely, all the alternative behaviours of the lower-level net with the presence of confusion are abstracted by the processes of the upper-level free-choice net. Hence, only the behaviours where confusion is not present are captured.

Chapter 6 provided a method to make the probabilistic framework usable in applications. A SAT-based model checking was proposed in order to formally verify the behavioural properties of acyclic nets, CSA-nets, and BSA-nets. The proposed formulas allow to check, e.g., whether a set of transitions induce a valid behaviour, and to detect confusion.

7.2 Aim and Objectives

This thesis aimed to develop a theoretical concurrent probabilistic framework based on CAS-nets and their behavioural abstraction and propose approaches of removing *confusion*. We believe the research work presented in this dissertation makes an original and significant contribution to this aim. We have already listed key contributions in Chapter 1, Section 1.2, and we provide further context for this here by reviewing the results achieved for each objective.

1. **A survey of the existing approaches of removing confusion in probabilistic Petri nets**

   There is a large body of work in probabilistic Petri nets considering handling confusion. We have fulfilled this objective by reviewing some studies that have proposed techniques of handling confusion. For example, the work introduced by [1] and [2] provided an approach of handling confusion by means of *branching cells* in such way that a net is decomposed dynamically. The most recent theory proposed by [26] offers a new approach for removing confusion by static decomposition of branching cells. Additional investigation about other approaches for removing confusion are explored in Chapter 2. Even though that our survey of the previous studies concern removing confusion has provided us good understanding of the issue, we can't guarantee that we have covered all the work in this area.

2. **A theoretical probabilistic framework for the analysis of cluster-acyclic nets and CSA-nets, and different approaches for removing confusion**

   Satisfying this objective requires developing various results. We have developed foundation theory of cluster-acyclic nets. In particular, we have defined the formula of calculating probabilities in a single acyclic net. We showed with some illustrative examples how the presence of confusion can affect calculating the probabilities. For example, we have proved formally that in a confusion-free acyclic net the conflict set of a transition is constant for all the executions of each scenario, and hence the probabilities can be calculated. Two approaches of removing confusion are developed. Approach *A* (motivated by the work in [26]) concentrates on markings *M* included in the pre-places of a cluster. In Approach *B*, only the binary synchronised acyclic nets are considered. A key result is that the constructions produce *confusion-free* nets. Proving such a property is based on ensuring that the executions of resulting nets satisfy the structure of *free-choice or extended free-choice* nets, where the conflict transitions are always enabled together at the same marking. These results and others are proved in Chapter 3. In Chapter 4, we have formally extended these results to a set of interacting acyclic nets captured by CSA-nets. Precisely, the formula of calculating probabilities is extended to consider syn-cycles instead of transitions. Moreover, the definition of confusion is extended in a similar way. The core contribution of Chapter 4 is the proposed approach of handling confusion in CSA-nets as presented in Section 4.9.

3. **A theoretical probabilistic framework for the analysis of BSA-nets, and an approach for handling confusion in BSA-nets**

   To satisfy this objective we first extended the model of BSA-nets discussed in [78, 106, 105, 84, 11, 92]. In particular, the upper-level nets are defined as *free-choice* acyclic nets instead of being *line-like* in order to represent alternatives in a given lower-level behaviour. Being able to represent alternative behaviours motivated us to extend the probabilistic framework so that probabilities are represented at the upper-level nets. In this case, the probability of a upper-level transition is derived from the weights associated with the lower-level transitions that are ascribed to it.The formula of calculating probabilities in CSA-nets is reused and the obtained results are represented at the upper level. Moreover, we propose a preliminary idea of controlling confusion in BSA-nets. More precisely, exploiting the structural constraints of upper-level nets, behavioural relation is used to filter out undesirable representation of a given behaviour.

4. **A SAT-based verification for CSA-nets and BSA-nets**

We have fulfilled this objective by investigating the development of a SAT-based model checking technique for our framework. We formalise important behavioural specifications of CSA-nets and BSA-nets as satisfiability formulas. For instance, we show how the presence of confusion can be detected using a suitable SAT formula. Checking of other crucial behavioural properties is discussed in Chapter 6.

7.3 Challenges

1. **Fully behaviour preservation**

   Simulating all the executions of the step sequences in the confusion-free version of an acyclic net was a major challenge. Even though that our encoding preserves all the maximal scenarios, some of their executions can't be simulated. The reason was the additional causality between the transitions whose some of their postset and preset belong to the negative places. Also, in the encoding, the transactions are computed which are the maximal step sequences. In fact, we established a crucial result in Theorem 3.6.2 that all the executions of the constructed net can be regarded as the executions of the original one, however, the reverse inclusion does not hold. We believe that proposing a general approach of removing confusion in concurrent systems is quite complicated. Hence, imposing reasonable restrictions is needed. For instance, we require the ordering relation over the maximal clusters to be strict partial order. Consequently, fully behaviour preservation would be impossible.

2. **Dealing with a confused acyclic net that is not a cluster-acyclic**

   The main idea of the approaches of removing confusion presented in this thesis is to delay the execution of some transitions such that all the conflicting transitions are enabled together. That is due to the constraint imposed over the maximal clusters to be strict partially ordered. In this case, it is possible to add negative places to capture the additional causality between the transitions belong to consecutive maximal clusters. However, if the strict ordering relation over the clusters is not satisfied, then the approaches proposed in this thesis are not applicable. Figure 3.20 showed an example of this case.We proposed an initial idea to handle this situation in Section 3.6.7. The key solution was to repair the *acyclicity* over the clusters by adding *auxiliary* transitions. However, it turned out that adding more information, for instance *time* would be useful to address this issue efficiently.

3. **Providing a formula to calculate probabilities in BSA-nets**

In Chapter 3, we formally defined a formula that calculates the probabilities for the maximal steps sequences. Basically, the probabilities are calculated for the execution histories captured by the *maximal scenarios*. Similarly, in Chapter 4, the formula was extended to consider syn-cycles instead of transitions. It turned out that extending the formula in Chapter 4 to calculate probabilities in BSA-nets is a major challenge. Even though that in this thesis the definition of BSA-nets is extended by being based on acyclic nets instead of being *line-like*, their calculating probabilities were not addressed in this thesis. In fact, the the enabled steps in BSA-nets were defined in terms of the underlying CSA-nets. In order to calculate the probabilities for the BSA-nets steps, their definition need to be extended and defined explicitly. In this case, the phases and the phase-consistent markings should be considered. Since, the current definition of steps in BSA-nets is introduced in terms of csan($bsan$) and based on phase-consistent marking, then it is required that each such step sequence must end at a phase-consistent marking. Therefore it provides only over approximation of the steps sequences of *bsan*. We face this challenge again in Chapter 6 as we could not able to provide satisfiability formulas that verifies exactly the behavioural properties of *bsan*.

Hence, for probabilistic analysis of *bsan*, we reused the formula defined in Chapter 4 to represent only the probabilities at the upper-level transitions. Extending the formula would result in extending other results as well, however, that is left for the future work.

7.4 Future work

There are several directions along which the work contained in the current thesis could be extended in the future.

1. **Probability and Time**

   This thesis concerns probability estimation for transitions firing. Time can be added to the probabilistic framework. Several attempts have been published related to probabilistic Time Petri Nets (TPNs). [44] extended TPNs in a way that output arcs of the transitions are defined as probabilistic hyperarcs. Firing a transition generates tokens in its output places and one of its hyperarcs is chosen with probability distribution. Each transition has its time interval and it is fireable if it is enabled and its clock has value within its interval. If its interval period passed, then it will not be fired even if it is enabled. Combining time and probability estimation in such a way may avoid the case of confusion. In particular, we can enforce conflict transitions to have the

same time interval to ensure that they are always fireable together. In the same vein, TPNs were extended by associating a probability density function to the fixed firing interval of each nondeterministic transition [130]. Extending the current thesis by considering time can be investigated by using the work in [23] as a basis. It provides time for related occurrence nets with alternative behaviours and is aimed to model and reason about causally related and concurrent transitions whose time information is not complete. Representing time for concurrent probabilistic models would be a significant extension of CSA-nets.

2. **Probabilistic Coloured CSA-nets**

   Coloured Petri Nets (CPNs) are extension of the standard Petri nets. They are a well known formalism to represent synchronised components systems, communication protocols, and distributed systems. In CPNs, tokens are associated with data values which are the tokens' *colour*. Probabilistic Coloured Petri Nets have been investigated in [87, 113]. The work in [11] can be used to extend the current thesis to Probabilistic Coloured CSA-nets. In this thesis, we proposed approaches of handling confusion. Exploring further possibilities of how the confusion can be handled in Probabilistic Coloured CSA-nets is an attractive direction of the current thesis.

3. **Probabilistic model checking**

   The core of this thesis is the development of probabilistic framework for CSA-nets. Also, in Chapter 6, we provided a SAT-based verification approach for various properties of CSA-nets. An interesting direction of taking the current thesis forward is to develop probabilistic model checking for CSA-nets. In this case, we could infer the probability that a certain property holds. It would be an effective approach for formally checking the quantitative features of systems. Probabilistic model checking in Petri Nets have been discussed in [37, 51, 108].

4. **Unfolding model checking for CSA-nets and tool support**

   It would be interesting to develop an effective unfolding based model checking verification for CSA-nets. [84] proposed unfolding algorithm for CSPT-nets. Complete unfolding prefixes can then be generated before being fed to SAT-solvers in order to verify the relevant behavioural properties. Such a model checking technique could alleviate the state space explosion problem. Also, it would be a significant addition to the SONCraft tool to implement prefix-based verification plug-in for CSA-nets.

References

[1] Abbes, S. and Benveniste, A. (2005). Branching cells as local states for event structures and nets: Probabilistic applications. In *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, pages 95–109.

[2] Abbes, S. and Benveniste, A. (2006). True-concurrency probabilistic models: Branching cells and distributed probabilities for event structures. *Inf. Comput.*, 204(2):231–274.

[3] Abbes, S. and Benveniste, A. (2008). True-concurrency probabilistic models: Markov nets and a law of large numbers. *Theor. Comput. Sci.*, 390(2-3):129–170.

[4] Abbes, S. and Benveniste, A. (2009). Concurrency, σ-algebras, and probabilistic fairness. In de Alfaro, L., editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 380–394. Springer.

[5] Alahmadi, M. (2021). Master channel places for communication structured acyclic nets. In Köhler-Bussmeier, M., Kindler, E., and Rölke, H., editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering 2021 co-located with the 42nd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2021), Paris, France, June 25th, 2021 (due to COVID-19: virtual conference)*, volume 2907 of *CEUR Workshop Proceedings*, pages 233–240. CEUR-WS.org.

[6] Alahmadi, M. (2022). Parametrisation of csa-nets. In Köhler-Bussmeier, M., Moldt, D., and Rölke, H., editors, *Petri Nets and Software Engineering 2022 co-located with the 43rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2022), Bergen, Norway, June 20th, 2022*, volume 3170 of *CEUR Workshop Proceedings*, pages 215–216. CEUR-WS.org.

[7] Alahmadi, M. (2023). Parameterised csa-nets. In Köhler-Bussmeier, M., Moldt, D., and Rölke, H., editors, *Petri Nets and Software Engineering 2023 co-located with the 44rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2023), Lisbon, Portugal, June 27th, 2023*, volume 3430 of *CEUR Workshop Proceedings*, pages 167–182. CEUR-WS.org.

[8] Alahmadi, M., Alharbi, S., Alharbi, T., Almutairi, N., Alshammari, T., Bhattacharyya, A., Koutny, M., Li, B., and Randell, B. (2023). Structured acyclic nets. Technical report, School of Computing, University of Newcastle upon Tyne.

[9] Albanese, M., Chellappa, R., Moscato, V., Picariello, A., Subrahmanian, V., Turaga, P., and Udrea, O. (2008). A constrained probabilistic Petri net framework for human activity detection in video. *IEEE Transactions on Multimedia*, 10(8):1429–1443.

[10] Alharbi, S. (2023). Hierarchical simulation of timed behaviours of structured occurrence nets. In Köhler-Bussmeier, M., Moldt, D., and Rölke, H., editors, *Petri Nets and Software Engineering 2023 co-located with the 44rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2023), Lisbon, Portugal, June 27th, 2023*, volume 3430 of *CEUR Workshop Proceedings*, pages 143–165. CEUR-WS.org.

[11] Alharbi, T. (2020). *Analysing and visualizing big data sets of crime investigations using Structured Occurrence Nets*. PhD thesis, University of Newcastle upon Tyne.

[12] Almutairi, N. (2022). Probabilistic communication structured acyclic nets. In Köhler-Bussmeier, M., Moldt, D., and Rölke, H., editors, *Petri Nets and Software Engineering 2022 co-located with the 43rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2022), Bergen, Norway, June 20th, 2022*, volume 3170 of *CEUR Workshop Proceedings*, pages 168–187. CEUR-WS.org.

[13] Almutairi, N. (2023). Probabilistic behavioural acyclic nets. In Köhler-Bussmeier, M., Moldt, D., and Rölke, H., editors, *Petri Nets and Software Engineering 2023 co-located with the 44rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2023), Lisbon, Portugal, June 27th, 2023*, volume 3430 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[14] Almutairi, N. and Koutny, M. (2021). Verification of communication structured acyclic nets using SAT. In Köhler-Bussmeier, M., Kindler, E., and Rölke, H., editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering 2021 co-located with the 42nd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2021), Paris, France, June 25th, 2021 (due to COVID-19: virtual conference)*, volume 2907 of *CEUR Workshop Proceedings*, pages 175–194. CEUR-WS.org.

[15] Alshammari, T. (2022). Towards automatic extraction of events for SON modelling. In Köhler-Bussmeier, M., Moldt, D., and Rölke, H., editors, *Petri Nets and Software Engineering 2022 co-located with the 43rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2022), Bergen, Norway, June 20th, 2022*, volume 3170 of *CEUR Workshop Proceedings*, pages 188–201. CEUR-WS.org.

[16] Alshammari, T. (2023). Integrating nlp and structured occurrence nets for crime modelling: A pattern-based approach. In Köhler-Bussmeier, M., Moldt, D., and Rölke, H., editors, *Petri Nets and Software Engineering 2023 co-located with the 44rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2023), Lisbon, Portugal, June 27th, 2023*, volume 3430 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[17] Arbab, F. (2004). Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366.

[18] Balbo, G. (2007). Introduction to generalized stochastic Petri nets. In Bernardo, M. and Hillston, J., editors, *Formal Methods for Performance Evaluation, 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*, volume 4486 of *Lecture Notes in Computer Science*, pages 83–131. Springer.

[19] Bamberg, R. (2012). Non-deterministic generalised stochastic Petri nets modelling and analysis. Master's thesis, University of Twente, The Netherlands.

[20] Barbour, A. E. (1992). Solutions to the minimization problem of fault-tolerant logic circuits. *IEEE transactions on computers*, 41(04):429–443.

[21] Benveniste, A., Fabre, E., and Haar, S. (2001). Markov nets: probabilistic models for distributed and concurrent systems. In *40th IEEE Conference on Decision and Control, CDC 2001, Orlando, FL, USA, 4-7 Dec., 2001*, pages 5010–5015. IEEE.

[22] Best, E. (1986). Structure theory of Petri nets: the free choice hiatus. In Brauer, W., Reisig, W., and Rozenberg, G., editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, Germany, 8-19 September 1986*, volume 254 of *Lecture Notes in Computer Science*, pages 168–205. Springer.

[23] Bhattacharyya, A., Li, B., and Randell, B. (2016). Time in structured occurrence nets. In Cabac, L., Kristensen, L. M., and Rölke, H., editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering 2016, Toruń, Poland, June 20-21, 2016*, volume 1591 of *CEUR Workshop Proceedings*, pages 35–55. CEUR-WS.org.

[24] Bolton, C. (2005). Adding conflict and confusion to csp. In *Proceedings of the 2005 International Conference on Formal Methods*, FM'05, page 205–220, Berlin, Heidelberg. Springer-Verlag.

[25] Boudali, H., Crouzen, P., Haverkort, B. R., Kuntz, M., and Stoelinga, M. I. A. (2008). Architectural dependability evaluation with arcade. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 512–521. IEEE.

[26] Bruni, R., Melgratti, H. C., and Montanari, U. (2017). Concurrency and probability: Removing confusion, compositionally. *CoRR*, abs/1710.04570.

[27] Bruni, R. and Montanari, U. (2000). Zero-safe nets: Comparing the collective and individual token approaches. *Information and Computation*, 156(1-2):46–89.

[28] Bryant, R. E., German, S. M., and Velev, M. N. (1999). Microprocessor verification using efficient decision procedures for a logic of equality with uninterpreted functions. In Murray, N. V., editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX '99, Saratoga Springs, NY, USA, June 7-11, 1999, Proceedings*, volume 1617 of *Lecture Notes in Computer Science*, pages 1–13. Springer.

[29] Čepin, M. and Mavko, B. (2002). A dynamic fault tree. *Reliability Engineering & System Safety*, 75(1):83–91.

[30] Chang, Y., Wang, J., and Lee, R. (1994). Generating all maximal independent sets on trees in lexicographic order. *Information Sciences*, 76(3):279–296.

[31] Chen, X., Li, Z., Wu, N., Al-Ahmari, A. M., El-Tamimi, A. M., and Abouel Nasr, E. S. (2016). Confusion avoidance for discrete event systems by P/E constraints and supervisory control. *IMA Journal of Mathematical Control and Information*, 33(2):309–332.

[32] Chen, X.-l., Li, Z.-w., Al-Ahmari, A. M., El-Tamimi, A. M., and Abouel Nasr, E. S. (2013). Confusion diagnosis and control of discrete event systems using synchronized p etri nets. *Asian Journal of Control*, 15(6):1736–1751.

[33] Christensen, S. and Hansen, N. D. (1994). Coloured petri nets extended with channels for synchronous communication. In Valette, R., editor, *Application and Theory of Petri Nets 1994, 15th International Conference, Zaragoza, Spain, June 20-24, 1994, Proceedings*, volume 815 of *Lecture Notes in Computer Science*, pages 159–178. Springer.

[34] Codish, M., Genaim, S., and Stuckey, P. J. (2009). A declarative encoding of telecommunications feature subscription in SAT. In Porto, A. and López-Fraguas, F. J., editors, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 255–266. ACM.

[35] Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158.

[36] Crawford, J. M. and Baker, A. B. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI*, volume 2, pages 1092–1097.

[37] Dehnert, C., Junges, S., Katoen, J.-P., and Volk, M. (2017). A storm is coming: A modern probabilistic model checker. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II 30*, pages 592–600. Springer.

[38] Dellaert, F. (2012). Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology.

[39] Dellaert, F. and Kaess, M. (2017). Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139.

[40] Deng, Y. and Hennessy, M. (2013). On the semantics of Markov automata. *Inf. Comput.*, 222:139–168.

[41] Desel, J. and Esparza, J. (1995). *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Science*. Springer.

[42] Eisentraut, C. (2017). *Principles of Markov automata*. PhD thesis, Saarland University, Germany.

[43] Eisentraut, C., Hermanns, H., Katoen, J.-P., and Zhang, L. (2013). A semantics for every GSPN. In *Proceedings of the 34th International Conference on Application and Theory of Petri Nets and Concurrency*, PETRI NETS'13, page 90–109, Berlin, Heidelberg. Springer-Verlag.

[44] Emzivat, Y., Delahaye, B., Lime, D., and Roux, O. H. (2016). Probabilistic time Petri nets. In Kordon, F. and Moldt, D., editors, *Application and Theory of Petri Nets and Concurrency*, pages 261–280, Cham. Springer International Publishing.

[45] Esparza, J. (1996). Decidability and complexity of Petri net problems - an introduction. In Reisig, W. and Rozenberg, G., editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer.

[46] Esparza, J. and Heljanko, K. (2008). *Unfoldings: A Partial-Order Approach to Model Checking (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer.

[47] Fehling, R. (1991). A concept of hierarchical Petri nets with building blocks. In Rozenberg, G., editor, *Advances in Petri Nets 1993, Papers from the 12th International Conference on Applications and Theory of Petri Nets, Gjern, Denmark, June 1991*, volume 674 of *Lecture Notes in Computer Science*, pages 148–168. Springer.

[48] Ganty, P., Raskin, J.-F., and Van Begin, L. (2008). From many places to few: Automatic abstraction refinement for Petri nets. *Fundamenta Informaticae*, 88(3):275–305.

[49] Girault, C. and Valk, R. (2013). *Petri nets for systems engineering: a guide to modeling, verification, and applications*. Springer Science & Business Media.

[50] Giunchiglia, F. and Walsh, T. (1992). A theory of abstraction. *Artificial intelligence*, 57(2-3):323–389.

[51] Gribaudo, M., Horváth, A., Bobbio, A., Tronci, E., Ciancamerla, E., and Minichino, M. (2003). Fluid Petri nets and hybrid model-checking: A comparative case study. *Reliability Engineering & System Safety*, 81(3):239–257.

[52] Guck, D., Timmer, M., Hatefi, H., Ruijters, E., and Stoelinga, M. (2014). Modelling and analysis of Markov reward automata. In Cassez, F. and Raskin, J.-F., editors, *Automated Technology for Verification and Analysis*, pages 168–184, Cham. Springer International Publishing.

[53] Haar, S. (2001). Clusters, confusion and unfoldings. *Fundamenta Informaticae*, 47(3-4):259–270.

[54] Haar, S. (2002). Probabilistic cluster unfoldings for Petri nets. *Fundamenta Informaticae*, 53(3-4):281–314.

[55] Hartmanns, A. and Hermanns, H. (2019). A modest markov automata tutorial. In Krötzsch, M. and Stepanova, D., editors, *Reasoning Web. Explainable Artificial Intelligence - 15th International Summer School 2019, Bolzano, Italy, September 20-24, 2019, Tutorial Lectures*, volume 11810 of *Lecture Notes in Computer Science*, pages 250–276. Springer.

[56] Hatefi, H., Braitling, B., Wimmer, R., Fioriti, L. M. F., Hermanns, H., and Becker, B. (2015). Cost vs. time in stochastic games and Markov automata. In Li, X., Liu, Z., and Yi, W., editors, *Dependable Software Engineering: Theories, Tools, and Applications*, pages 19–34, Cham. Springer International Publishing.

[57] Hatefi, H. and Hermanns, H. (2012). Model checking algorithms for Markov automata. *Electronic Communications of the EASST*, 53.

[58] Hazzan, O. (1999). Reducing abstraction level when learning abstract algebra concepts. *Educational Studies in Mathematics*, 40:71–90.

[59] Hazzan, O., Dubinsky, Y., Hazzan, O., and Dubinsky, Y. (2008). Abstraction abstraction. *Agile Software Engineering*, pages 1–16.

[60] Heckerman, D., Horvitz, E., and Nathwani, B. N. (1992). Toward normative expert systems: Part i. the pathfinder project. *Methods of information in medicine*, 31 2:90–105.

[61] Hermanns, H. and Katoen, J. (2009). The how and why of interactive markov chains. In *FMCO*, volume 6286 of *Lecture Notes in Computer Science*, pages 311–337. Springer.

[62] Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

[63] Holmes, D. E. (2008). Toward a generalized bayesian network. In Holmes, D. E. and Jain, L. C., editors, *Innovations in Bayesian Networks: Theory and Applications*, volume 156 of *Studies in Computational Intelligence*, pages 281–288. Springer.

[64] Janicki, R. and Koutny, M. (1991). Invariants and paradigms of concurrency theory. In *Parle'91 Parallel Architectures and Languages Europe*, pages 481–496. Springer.

[65] Jiang, X., Wagner, M. M., and Cooper, G. F. (2008). Modeling the temporal trend of the daily severity of an outbreak using bayesian networks. In Holmes, D. E. and Jain, L. C., editors, *Innovations in Bayesian Networks: Theory and Applications*, volume 156 of *Studies in Computational Intelligence*, pages 169–185. Springer.

[66] Jihad, A. I. (2018). Automated verification of object Petri nets based on transformation, unfoldings and sat solving. Technical report, University of South Wales (United Kingdom).

[67] Johnson, D. S., Yannakakis, M., and Papadimitriou, C. H. (1988). On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123.

[68] Katoen, J.-P. (2012). GSPNs revisited: Simple semantics and new analysis algorithms. In *Proceedings of the 2012 12th International Conference on Application of Concurrency to System Design*, ACSD '12, page 6–11, USA. IEEE Computer Society.

[69] Katoen, J.-P. and Peled, D. (2013). *Taming Confusion for Modeling and Implementing Probabilistic Concurrent Systems*, volume 7792 of *Lecture Notes in Computer Science*, pages 411–430. Springer, Berlin.

[70] Khomenko, V. (2003). *Model checking based on prefixes of Petri net unfoldings*. PhD thesis, University of Newcastle upon Tyne.

[71] Khomenko, V., Kondratyev, A., Koutny, M., and Vogler, W. (2006). Merged processes: a new condensed representation of Petri net behaviour. *Acta Informatica*, 43(5):307–330.

[72] Khomenko, V., Koutny, M., and Yakovlev, A. (2004). Detecting state encoding conflicts in STG unfoldings using SAT. *Fundam. Informaticae*, 62(2):221–241.

[73] Kleijn, J. and Koutny, M. (2011). Causality in structured occurrence nets. In Jones, C. B. and Lloyd, J. L., editors, *Dependable and Historic Computing - Essays Dedicated to Brian Randell on the Occasion of His 75th Birthday*, volume 6875 of *Lecture Notes in Computer Science*, pages 283–297. Springer.

[74] Kleijn, J., Koutny, M., and Pietkiewicz-Koutny, M. (2012). Regions of petri nets with a/sync connections. *Theor. Comput. Sci.*, 454:189–198.

[75] Knoblock, C. A. (1991). Search reduction in hierarchical problem solving. In Dean, T. L. and McKeown, K. R., editors, *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, 1991, Volume 2*, pages 686–691. AAAI Press / The MIT Press.

[76] Köhler-Bußmeier, M. and Kudlek, M. (2008). Linear properties of zero-safe nets with debit tokens. *Fundamenta Informaticae*, 85(1-4):329–342.

[77] Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

[78] Koutny, M. and Randell, B. (2009). Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques. *Fundam. Inform.*, 97(1-2):41–91.

[79] Kramer, J. (2006). Abstraction in computer science & software engineering: A pedagogical perspective. *System Design Frontier Journal*, 3(12):1–9.

[80] Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519.

[81] Kummer, O. (1999). A petri net view on synchronous channels. *Petri Net Newsletter*, 56:7–11.

[82] Lakos, C. (1999). *The Compositionality of Abstraction for Coloured Petri Nets*. PhD thesis, Department of Computer Science, The University of Adelaide.

[83] Leung, J. Y.-T. (1984). Fast algorithms for generating all maximal independent sets of interval, circular-arc and chordal graphs. *Journal of Algorithms*, 5(1):22–35.

[84] Li, B. (2017). *Visualisation and Analysis of Complex Behaviours using Structured Occurrence Nets*. PhD thesis, University of Newcastle upon Tyne, United Kingdom.

[85] Li, B., Randell, B., Bhattacharyya, A., Alharbi, T., and Koutny, M. (2018). Soncraft: A tool for construction, simulation, and analysis of structured occurrence nets. In *18th International Conference on Application of Concurrency to System Design, ACSD 2018, Bratislava, Slovakia, June 25-29, 2018*, pages 70–74. IEEE Computer Society.

[86] Liskov, B., Guttag, J., et al. (1986). *Abstraction and specification in program development*, volume 180. MIT press Cambridge.

[87] Liu, X., Zhang, J., and Zhu, P. (2017). Modeling cyber-physical attacks based on probabilistic colored Petri nets and mixed-strategy game theory. *International Journal of Critical Infrastructure Protection*, 16:13–25.

[88] Loeliger, H.-A. (2004). An introduction to factor graphs. *IEEE Signal Processing Magazine*, 21(1):28–41.

[89] Marr, C. (2007a). Capturing conflict and confusion in CSP. In Davies, J. and Gibbons, J., editors, *Integrated Formal Methods, 6th International Conference, IFM 2007, Oxford, UK, July 2-5, 2007, Proceedings*, volume 4591 of *Lecture Notes in Computer Science*, pages 413–438. Springer.

[90] Marr, C. (2007b). Capturing conflict and confusion in CSP. In *Integrated Formal Methods, 6th International Conference, IFM 2007, Oxford, UK, July 2-5, 2007, Proceedings*, pages 413–438.

[91] Milner, R. (1989). *Communication and concurrency*. PHI Series in computer science. Prentice Hall.

[92] Missier, P., Randell, B., and Koutny, M. (2012). Modelling provenance using structured occurrence networks. In Groth, P. and Frew, J., editors, *Provenance and Annotation of Data and Processes - 4th International Provenance and Annotation Workshop, IPAW 2012, Santa Barbara, CA, USA, June 19-21, 2012, Revised Selected Papers*, volume 7525 of *Lecture Notes in Computer Science*, pages 183–197. Springer.

[93] Mozetic, I. (1990). Abstractions in model-based diagnosis. Technical report, Austrian Research Institute for Artificial Intelligence.

[94] Nagl, S., Williams, M., and Williamson, J. (2008). Objective bayesian nets for systems modelling and prognosis in breast cancer. *Studies in Computational Intelligence*, 156:131–167.

[95] Nguyen, V.-H. and Mai, S. T. (2015). A new method to encode the at-most-one constraint into sat. In *Proceedings of the 6th International Symposium on Information and Communication Technology*, pages 46–53.

[96] Niedermayer, I. D. (2008). An introduction to bayesian networks and their contemporary applications. In Holmes, D. E. and Jain, L. C., editors, *Innovations in Bayesian Networks: Theory and Applications*, volume 156 of *Studies in Computational Intelligence*, pages 117–130. Springer.

[97] Pearl, J. (1985). Bayesian networks: A model of self-activated memory for evidential reasoning. Technical Report CSD-850021, R-43, UCLA Computer Science Department.

[98] Penczek, W., Półrola, A., and Zbrzezny, A. (2009). Sat-based (parametric) reachability for distributed time Petri nets. In *Proc. of the Int. Workshop on Petri Nets and Software Engineering (PNSE'09)*, pages 133–154. Citeseer.

[99] Petri, C. A. (1962). Kommunikation mit Automaten. Dissertation, Schriften des IIM 2, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Bonn.

[100] Poliakov, I., Khomenko, V., and Yakovlev, A. (2009). Workcraft - A framework for interpreted graph models. In Franceschinis, G. and Wolf, K., editors, *Applications and Theory of Petri Nets, 30th International Conference, PETRI NETS 2009, Paris, France, June 22-26, 2009. Proceedings*, volume 5606 of *Lecture Notes in Computer Science*, pages 333–342. Springer.

[101] Rabin, M. O. (1963). Probabilistic automata. *Information and control*, 6(3):230–245.

[102] Radu, M., Pitica, D., and Posteuca, C. (2000). Reliability and failure analysis of voting circuits in hardware redundant design. In *International Symposium on Electronic Materials and Packaging (EMAP2000)(Cat. No. 00EX458)*, pages 421–423. IEEE.

[103] Randell, B. (2011). Occurrence nets then and now: The path to structured occurrence nets. In Kristensen, L. M. and Petrucci, L., editors, *Applications and Theory of Petri Nets*, pages 1–16, Berlin, Heidelberg. Springer Berlin Heidelberg.

[104] Randell, B. (2013). Incremental construction of structured occurrence nets. Technical Report 1384, School of Computing Science, University of Newcastle upon Tyne.

[105] Randell, B. and Koutny, M. (2007). Failures: Their definition, modelling and analysis. In Jones, C. B., Liu, Z., and Woodcock, J., editors, *Theoretical Aspects of Computing - ICTAC 2007, 4th International Colloquium, Macau, China, September 26-28, 2007, Proceedings*, volume 4711 of *Lecture Notes in Computer Science*, pages 260–274. Springer.

[106] Randell, B. and Koutny, M. (2009). Structured occurrence nets: Incomplete, contradictory and uncertain failure evidence. Technical Report 1170, School of Computing Science, University of Newcastle upon Tyne.

[107] Raynal, M. (2013). *Distributed Algorithms for Message-Passing Systems*. Springer.

[108] Razzaq, M. and Ahmad, J. (2015). Petri net and probabilistic model checking based approach for the modelling, simulation and verification of internet worm propagation. *PloS one*, 10(12):1–22.

[109] Reisig, W. (2013). *Understanding Petri nets: modeling techniques, analysis methods, case studies*. Springer.

[110] Rodríguez, C. and Schwoon, S. (2012). Verification of Petri nets with read arcs. In Koutny, M. and Ulidowski, I., editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 471–485. Springer.

[111] Rozenberg, G. and Engelfriet, J. (1996a). Elementary net systems. In *Advanced Course on Petri Nets*, pages 12–121. Springer.

[112] Rozenberg, G. and Engelfriet, J. (1996b). Elementary net systems. In Reisig, W. and Rozenberg, G., editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*, pages 12–121. Springer.

[113] Rozinat, A., Mans, R., Song, M., and Van der Aalst, W. M. (2008). Discovering colored Petri nets from event logs. *International Journal on Software Tools for Technology Transfer*, 10:57–74.

[114] Scholten, J. G., Arbab, F., de Boer, F. S., and Bonsangue, M. M. (2006). A component coordination model based on mobile channels. *Fundamenta Informaticae*, 73(4):561–582.

[115] Schwartz, I., Yu, S., Hazan, T., and Schwing, A. G. (2019). Factor graph attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2039–2048.

[116] Segala, R. (1995). A compositional trace-based semantics for probabilistic automata. In *International Conference on Concurrency Theory*, pages 234–248. Springer.

[117] Segala, R. and Lynch, N. (1994). Probabilistic simulations for probabilistic processes. In *International Conference on Concurrency Theory*, pages 481–496. Springer.

[118] Segala, R. and Lynch, N. (1995). Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273.

[119] Smith, E. (1996). On the border of causality: Contact and confusion. *Theoretical Computer Science*, 153(1):245–270.

[120] Stephenson, T. A. (2000). An introduction to bayesian network theory and usage. Technical report, Idiap.

[121] Stoelinga, M. (2002). An introduction to probabilistic automata. *Bulletin of the EATCS*, 78(176-198):2.

[122] Suzuki, I. and Murata, T. (1983). A method for stepwise refinement and abstraction of Petri nets. *Journal of computer and system sciences*, 27(1):51–76.

[123] Timmer, M. and Iwama, K. (2014). Efficient modelling, generation and analysis of Markov automata. *Bulletin of the European Association for Theoretical Computer Science*, 112:139–140.

[124] Timmer, M., Katoen, J., van de Pol, J., and Stoelinga, M. (2012). Efficient modelling and generation of Markov automata. In Koutny, M. and Ulidowski, I., editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 364–379. Springer.

[125] Tsukiyama, S., Ide, M., Ariyoshi, H., and Shirakawa, I. (1977). A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517.

[126] Valk, R. (1996). On processes of object Petri nets. Technical Report 185/96, University of Hamburg - Computer Science Department.

[127] Valk, R. (2003). Object Petri nets: Using the nets-within-nets paradigm. In Desel, J., Reisig, W., and Rozenberg, G., editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given*

at ACPN 2003, additional chapters have been commissioned], volume 3098 of *Lecture Notes in Computer Science*, pages 819–848. Springer.

[128] Varacca, D. and Nielsen, M. (2003). Probabilistic Petri nets and Mazurkiewicz equivalence.

[129] Varacca, D., Völzer, H., and Winskel, G. (2006). Probabilistic event structures and domains. *Theor. Comput. Sci.*, 358(2-3):173–199.

[130] Vicario, E., Sassoli, L., and Carnevali, L. (2009). Using stochastic state classes in quantitative evaluation of dense-time reactive systems. *IEEE Trans. Softw. Eng.*, 35(5):703–719.

[131] Williams, B. C. (1990). Capturing how things work: Constructing critical abstractions of local interactions. In *Workshop on the Automatic Generation of Approximations and Abstractions*.

[132] Workcraft (2018). `https://workcraft.org/`.

[133] Yim, J. and Lee, K.-Y. (2007). Fuzzy-timing Petri nets with choice probabilities for response time analysis. In Shi, Y., van Albada, G. D., Dongarra, J., and Sloot, P. M. A., editors, *Computational Science – ICCS 2007*, pages 652–659, Berlin, Heidelberg. Springer Berlin Heidelberg.