THE UNIVERSITY *of* EDINBURGH

Petri nets, Probability and Event Structures

Nargess Ghahremani Azghandi

Doctor of Philosophy

Laboratory for Foundations of Computer Science

School of Informatics

University of Edinburgh

2014

Abstract

Models of true concurrency have gained a lot of interest over the last decades as models of concurrent or distributed systems which avoid the well-known problem of state space explosion of the interleaving models. In this thesis, we study such models from two perspectives.

Firstly, we study the relation between Petri nets and stable event structures. Petri nets can be considered as one of the most general and perhaps wide-spread models of true concurrency. Event structures on the other hand, are simpler models of true concurrency with explicit causality and conflict relations. Stable event structures expand the class of event structures by allowing events to be enabled in more than one way. While the relation between Petri nets and event structures is well understood, the relation between Petri nets and stable event structures has not been studied explicitly. We define a new and more compact unfoldings of safe Petri nets which is directly translatable to stable event structures. In addition, the notion of complete finite prefix is defined for compact unfoldings, making the existing model checking algorithms applicable to them. We present algorithms for constructing the compact unfoldings and their complete finite prefix.

Secondly, we study probabilistic models of true concurrency. We extend the definition of probabilistic event structures as defined by Abbes and Benveniste to a newly defined class of stable event structures, namely, *jump-free* stable event structures arising from Petri nets (characterised and referred to as *net-driven*). This requires defining the fundamental concept of *branching cells* in probabilistic event structures, for jump-free net-driven stable event structures, and by proving the existence of an isomorphism among the branching cells of these systems, we show that the latter benefit from the related results of the former models. We then move on to defining a probabilistic logic over probabilistic event structures (PESL). To our best knowledge, this is the first probabilistic logic of true concurrency. We show examples of expressivity achieved by PESL, which in particular include properties related to synchronisation in the system. This is followed by the model checking algorithm for PESL for finite event structures.

Finally, we present a logic over stable event structures (SEL) along with an account of its expressivity and its model checking algorithm for finite stable event structures.

Lay Summary

The model checking problem in computer science is defined as the procedure of verifying if a given model, representing a system, has certain desirable or undesirable properties. This is very useful as, for example, using model checking one can highlight any safety or security issue within the system. The properties to be check for are usually defined through a logical formula. This thesis studies model checking of true-concurrent systems, i.e. concurrent systems which are loyal to the true notion of concurrency, from two aspects.

Firstly, we study the relation between two models of such concurrent systems, namely Petri nets and Stable Event Structures. Petri nets are well-known and widely used models of true concurrency. However, to our best knowledge, the relation between Petri nets and stable event structures is not studied. Thus, given a safe Petri net, we produce a more compact structure, along with its mapping to Stable Event Structures. We also show some of the important properties of the new compact structure, including its suitability for verification purposes.

Secondly, we study the model checking of probabilistic models of true concurrency. For that purpose, we introduce a logic capable of expressing probabilistic properties for event structures. We also present the model checking produce for finite structures. As a result, for example, we can express and verify that the likelihood of a system modelled by event structures encountering an error is less than two per cent.

Acknowledgements

First and foremost, I would like to thank my eternal companion and guide, Chariji, for showing me everyday how to live, grow and love and for introducing me to a new, much vaster universe inside.

I am grateful to my supervisor Julian Bradfield for all that I learned from him, but also for supporting me throughout all the ups and downs of these years and especially towards the end of this long journey. I would also like to thank Gordon Plotkin and Maciej Koutny for examining my thesis and giving valuable suggestions and corrections to this work, which greatly improved the final outcome. There are many others that I would like to thank in the LFCS and Informatics, including Vincent Danos, Ian Stark, Colin Stirling, Kousha Etessami and Kyriakos Kalorkoti.

I would have never got here without the constant support of my family. I am indebted to my mother, for teaching me the most important values in life and for the never-ending care, love and support I have received from her all my life; to my father, for teaching me to dare to think differently, philosophically and perhaps artistically(!); to my dear husband, for his passion and strive to be the best, for his moral support, for reminding me not to lose perspective of the most important things in life, and for all the cooking, cleaning and washing during the last months!

Last but not least, I would like to thank my extended family, my dearest friends Azar, Maryam, and Taraneh for years of growing up together, sharing the fun and stress of life as PhD students and my Sahaj Marg family for making me feel at home all along.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Nargess Ghahremani Azghandi*)

Table of Contents

Chapter 1

Introduction

1.1 Motivation and Context

Concurrent systems appear nowadays in various different fields such as telecommunication networks, operating systems, database management systems, communication systems, product systems, monitoring and controlling systems, etc. It is therefore not surprising that there has been a vast amount of interest in modelling and verifying such systems so that they can be analysed, their performance be measured and perhaps most importantly the correctness of their behaviour can be verified. The importance of verification of concurrent systems lies not only in the fact that their failure can be expensive and even fatal, but also since generally they are much harder to test by the usual testing techniques for sequential systems. That is because for example, errors may appear rarely and therefore remain unidentified during the testing process and even if they are identified, it is almost impossible to repeat them and find their roots.

In concurrency theory there are generally two main approaches in modelling concurrency, namely, the *interleaving* or *sequential* approach and the *partial order* or *true concurrency* one. In the former, concurrency of two events is *implicitly* modelled as a nondeterministic choice between different possible orderings of the events, while in the latter, such events are *explicitly* defined to be concurrent and need not be in any particular order, hence the term 'partial order' approach. It can therefore be said that the latter approach is more faithful to the concept of concurrency, hence the terms 'true concurrency'. Examples of the interleaving model include but are not limited

to Kripke structures, labelled transition systems, infinite trees while examples of true concurrency models include Petri nets [53], (different kinds of) event structures [76], Mazurkiewicz trace languages [49], and transition systems with independence [78].

On the one hand, the interleaving approach benefits from being less complex and also from the massive amount of existing studies related to sequential systems which are also applicable to concurrent systems (under such semantics). On the other hand, it suffers from the *state explosion problem* [70] in verification, namely, the combinatorial explosion of state space. Different techniques exist for alleviating this problem such as using *binary decision diagrams* or BDDs [50], *abstraction techniques* [14] and *partial order reduction techniques* [24]. The true concurrency models, although much more complex, are far less subject to the state explosion problem. Even then, partial order techniques can be applied to them naturally.

A prominent technique specifically defined on true concurrency models, is that of complete finite prefixes of Petri net unfoldings defined by McMillan [51]. Unfoldings of Petri nets were first introduced in [53] and further formalised in [76] and are well-known structures used both for verification of Petri nets and also for describing the behaviour of a net in a simpler, more understandable manner. The latter is achieved by representing the full state space, under the true concurrency semantics. Even without considering all the interleavings, unfoldings of Petri nets grow exponentially. This issue can be addressed by introducing other types of unfoldings, such as Merged Processes [37], Trellis Processes[23] and R_i unfoldings [58], each of which have different objectives including that of compactness. In this thesis we also attempt to address this issue by defining a new unfolding which does not unfold conflicts and is therefore more compact.

Another wide-spread approach in verification is that of model checking, where given a model of the system ($\mathcal{M}$) and a logical formula (ϕ) describing a property of such models, it is verified if the formula is satisfied by the model ($\mathcal{M} \models \phi$). Some of the most prominent work and references include but are not limited to [13, 15, 21]. In general, some of the most typical properties that are verified include those of *safety* and *liveness*. Informally speaking, safety properties express that an undesirable state or scenario does not occur, while liveness properties express that a desirable state or

scenario will be reached. Such properties are classified as qualitative properties of a system. However, there are many systems for which such qualities are not guaranteed to hold or even not hold, this arising mainly from the notion of unreliability in such systems. For example, in telecommunication systems there is a possibility that occasionally messages are lost and not delivered. Therefore, rather than requiring a system to always satisfy a property, we are interested in determining the likelihood that the system does so, i.e. in verification of quantitative properties of a system. This requires representing a concurrent system by probabilistic models and expressing properties by probabilistic logical formula.

Similar to the non-probabilistic case, major work has been achieved for modelling and verification of probabilistic interleaving models, naming only a few would include [64, 63, 73, 62, 79, 66, 18, 7]. However, the work on probabilistic true concurrency models is relatively more recent [74, 26, 71, 72, 9, 1, 2] and perhaps for the same reason probabilistic logics with true concurrent semantics are, to our best knowledge, non-existent. In addition to the differences between the two approaches to concurrency mentioned above, there is another crucial difference between the two in the probabilistic framework, in particular when considering distributed concurrent systems. Generally, temporal stochastic processes and models capturing concurrency through nondeterminism such as [33, 19, 64, 65, 20] have a global state corresponding to a global time. In a distributed system, however, this is neither feasible nor natural. Thus, in the true concurrency approaches there is no notion of global time or state, but rather local ones. In other words, the local components have their own local states and act in their own local time until they communicate together. This results in a highly desirable match between concurrency and probability, more concretely, following this point of view concurrent choices can be made probabilistically independent.

In our opinion, among the existing probabilistic truly concurrent systems, probabilistic event structures as defined by [1] can be considered as the state of the art. Being mainly based on the work carried out in [9], not only they achieve the desirable objective described above, they also generalise other definitions of probabilistic event structures ([74, 72]) by resolving the crux of the problem, namely, *confusion* [53, 67]. Confusion arises when occurrence of an event disables two otherwise concurrent events. It is therefore challenging to assign probabilities to occurrence of such events, as even

though they are concurrent, they are not probabilistically independent in the real sense. Thus, the other approaches do not consider confusion. This challenge is tackled in [1] by decomposing the global state of the system into the local ones of sub-event structures called *branching cells* which resolve confusions internally and each have a *local transition probability*. The probability of a global state is then defined as the product of the probability of its consisting local states.

For all the reasons mentioned above, we focus on probabilistic event structures of [1] in this thesis and develop a probabilistic logic with true concurrent semantics. We also study the case of probabilistic stable event structures by showing that the technique and results of [1] for probabilistic event structures can be applied to a subclass of stable event structures, producing probabilistic *jump-free* stable event structures. Moreover, we develop a logic for stable event structures to better understand these structures. A detailed summary of our contribution is described in the next section.

1.2 Contributions and Outline

Our contribution in this thesis can be summarised as development of the following.

1. A new, more compact unfolding of safe Petri nets, including:
   - characterisation of compact unfoldings
   - properties of compact unfoldings
   - complete finite prefix for compact unfoldings which is suitable for existing verification techniques
   - an algorithm for constructing compact unfoldings and complete finite prefixes
   - relation between compact unfoldings and *stable* event structures.
2. A new probabilistic logic, PESL, interpreted over probabilistic event structures of [1], including syntax, semantics and expressivity of PESL. To our best knowledge, this is the first logic with true concurrency semantics. Furthermore, PESL can explicitly describe synchronisation properties, which is not possible in general by other truly concurrent logics.

3. A new logic, SEL, interpreted over stable event structures, including syntax, semantics and expressivity of SEL.
4. Analogous definition of probabilistic stable event structures for the new class of *jump-free* event structures.

The thesis is organised as follows.

Chapter 2. This chapter introduces the basic concepts, definitions and relations referred to throughout this thesis. Definitions of different types of event structures are given, along with the less obvious mappings between them. Moreover, Petri nets and morphisms between nets are defined.

Chapter 3 This chapter introduces a new, more compact, unfolding of safe Petri nets. As mentioned before, the conventional unfoldings of Petri nets while being highly useful for both the semantics and model checking of Petri nets, grow exponentially in the size of the original Petri net. A major reason is that every possible firing of a transition of the net has a unique corresponding event in the unfolding. Therefore, compact unfoldings are defined in such a way that they account for every possible marking of the net without keeping track of how exactly a transition fires in terms of the choices made in its history. In other words, conflicts are not unfolded as far as possible.

We start with the definition of required background concepts and present the characterisation of compact unfoldings or unfolding$^-$s. We describe properties of unfolding$^-$s, in particular their relation to the traditional unfoldings of Petri nets. This includes proving isomorphism of configurations of the conventional unfoldings and the compact ones.

We then present an algorithm for (deterministically) computing an unfolding$^-$of a given safe Petri net and prove its correctness. This is followed by defining the concept of complete finite prefix for unfolding$^-$s, proving that the existing verification algorithms are applicable to it and adjusting the above algorithm to compute complete finite prefixes.

Finally, we define the relation between unfolding^{-}s and stable event structures, a family of event structures which allow for multiple (conflicting) causes and give a brief and high-level view of other existing approaches.

Chapter 4. We start this chapter by characterising the stable event structures arising from safe Petri nets as defined in the previous chapter. This leads to defining the class of net-driven stable event structures.

We then present the definition of probabilistic event structures as in [1], by first defining the required background concepts, such as *stopping prefixes*, *recuresively stopped prefixes* and *branching cells*. We then define the analogous concepts for net-driven stable event structures and prove that the concept of branching cells can be defined in a similar way for a certain newly defined class of stable event structures, namely, *jump-free* event structures.

Informally speaking, in jump-free event structures the confusions occur for events which are not causally related. Therefore, none of the events in the conflict-closed sets of events are causally related. This constraint allows us to defined branching cells on stable event structures analogous to those of event structures and by proving the isomorphism between the branching cells in the two structures, the rest of the results follows for the stable case.

Chapter 5. In this chapter we define the syntax and semantics for PESL, the first probabilistic logic with truly concurrent semantics. Since PESL is defined on R-stopped configurations (as defined in chapter 4), the logic operates at different levels, namely, that of the events, configurations and in-between configurations. The expressivity of PESL is described along with the proof that it encodes the behaviour of pCTL. More interestingly, synchronisation properties expressible by PESL are discussed. Finally, the model checking algorithm for finite event structures is presented.

Chapter 6. This chapter introduces the new logic SEL, interpreted over stable event structures. Stable event structures have not been studied or used as extensively as event structures. Therefore, the aim is to better understand stable event structures and express different types of relations between groups of events. The syntax and semantics

of SEL is presented, followed by examples of its expressivity and a full comparison with ESL-based logics [56], one of the few family of logics specifically defined for event structures. Finally, the model checking algorithm for finite stable event structures is presented.

Chapter 7. This chapter concludes this thesis with some final remarks and directions for future work.

Chapter 2

Preliminaries

In this chapter we introduce the background definitions and concepts referred to throughout this thesis.

2.1 Notations

- $x \subseteq_{\text{fin}} y \Leftrightarrow x \subseteq y$ & x is finite
- x is minimal$^{\subseteq}$ satisfying a property $p \Leftrightarrow x$ satisfies p & $\nexists x' \subset x.\ x'$ satisfies p.
- $\mathbb{N} = \{0, 1, 2, \ldots\}$
- $\exists! x \Leftrightarrow$ there is a unique x
- Given a set X and a binary relation R on X, $X' \subseteq X$ is *left-closed* iff $x \in X'$ & $yRx \Rightarrow y \in X'$

2.2 Multi-sets

Definition 2.2.1. A multiset μ over a set X is a function $\mu : X \to \mathbb{N}$.

For $x \in X$, we write $x \in \mu$ to denote $\mu(x) \geq 1$ and for two multi-sets μ and μ' over X, we define $\mu \leq \mu' \Leftrightarrow_{def} \forall x \in X.\ \mu(x) \leq \mu'(x)$. We use $\{\!|\,|\!\}$ to explicitly represent a multi-set (when it is finite), e.g. $\{\!|a, a, b, b, b|\!\}$ denotes a multi-set over $X = \{a, b, c\}$ where $\mu(a) = 2$, $\mu(b) = 3$ and $\mu(c) = 0$.

Definition 2.2.2. The sum of two multi-sets μ and μ' over X is defined as

$$(\mu+\mu')(x) =_{def} \mu(x)+\mu'(x).$$

Similarly, the difference of μ and μ' is defined as

$$(\mu-\mu')(x) =_{def} \max\{0,\ \mu(x) \dot{-} \mu'(x)\}$$

Definition 2.2.3. A *multi-relation* between sets X and Y is a function $R : X \times Y \to \mathbb{N}$, denoted by $R \subseteq_{\mu} X \times Y$. The number of times element $x \in X$ is related to element $y \in Y$ is denoted by $R[x,y]$. Given a multi-set μ over X, the application $R.\mu$ of R to μ is the multi-set given by $\mu'(y) = \sum_{x \in X} R[x,y].\mu(x)$.

2.3 Event Structures

In this section we define different types of event structures, namely, elementary event structures, stable event structures, prime event structures and event structures. As we shall see, among these, stable event structures are a refinement of elementary event structures (by adding a stability axiom), and event structures are a refinement of prime event structures (by restricting the consistency to correspond to a binary conflict relation). We also present translations from stable event structures into a prime event structures and from event structures into stable event structures.

2.3.1 Elementary Event Structures [76]

Definition 2.3.1. An *elementary event structure* $\mathcal{E}$ is a triple $(E, Con, \vdash)$ where:

1. E is the set of events.
2. $Con \subseteq_{fin} \wp(E)$ is a non-empty consistency predicate satisfying:

$$X \in Con \ \& \ Y \subseteq X \Rightarrow Y \in Con.$$

3. $\vdash \subseteq Con \times E$ is the enabling relation satisfying:

$$X \vdash e \ \& \ X \subseteq Y \ \& \ Y \in Con \Rightarrow Y \vdash e.$$

Given an elementary event structure, its configurations are defined as follows.

Definition 2.3.2. Let $\mathcal{E} = (E, Con, \vdash)$ be an elementary event structure. A *configuration* of $\mathcal{E}$ is defined as a subset $X \subseteq E$ of events such that:

1. X is consistent: $\forall X' \subseteq_{fin} X.\, X' \in Con$.
2. X is secured: $\forall e \in X.\, \exists e_0, \ldots, e_n \in X.\, e_n = e$ and $\forall i \leq n.\, \{e_0, \ldots, e_{i-1}\} \vdash e_i$

The set of all configurations of $\mathcal{E}$ is represented by $\mathcal{V}(\mathcal{E})$ or $\mathcal{V}_{\mathcal{E}}$ or $\mathcal{V}$ when no confusion arises.

We now define the notion of *compatability* in order to characterise family of configurations of elementary event structures in the following theorem.

Definition 2.3.3. Let $(P, \sqsubseteq)$ be a partial order. Then $S \subseteq P$ is *compatible*, represented by $S\uparrow$ iff $\exists p \in P.\, \forall s \in S.\, s \sqsubseteq p$. A subset is finitely compatible, written as $S \uparrow^{fin}$ iff $\forall S_0 \subseteq_{fin} S.\, S_0 \uparrow$.

Theorem 2.3.4. [76] Let $\mathcal{E} = (E, Con, \vdash)$ be an elementary event structure. Then its configurations $\mathbf{V} = \mathcal{V}(\mathcal{E})$ form a family of subsets of E with the following properties.

1. Finite-Completeness:
$$\mathbf{A} \subseteq \mathbf{V} \;\&\; \mathbf{A} \uparrow^{fin} \Rightarrow \bigcup \mathbf{A} \in \mathbf{V}$$
2. Finiteness:
$$\forall u \in \mathbf{V}.\, \forall e \in u.\, \exists v \in \mathbf{V}.\, (v \text{ is finite } \&\; e \in v \;\&\; v \subseteq u).$$
3. Coincidence-Freeness:
$$\forall u \in \mathbf{V}.\, \forall e, e' \in u.\, e \neq e' \Rightarrow (\exists v \in \mathbf{V}.\, v \subseteq u \;\&\; (e \in v \Leftrightarrow e' \notin v)).$$

The next definition extends the notion of family of configurations to sets of subsets of an arbitrary set, satisfying the axioms above; the following definition and theorem show how to build an event structure from such a family.

Definition 2.3.5. Let $\mathbf{V}$ be a set of subsets. Then $\mathbf{V}$ is defined to be a *family of configurations* if it satisfies the axioms of theorem 2.3.4. We say $\mathbf{V}$ is a family of configurations of E if $E = \bigcup \mathbf{V}$.

Definition 2.3.6. Let $\mathbf{V}$ be a family of configurations of a set E. Then define $\mathcal{E}(\mathbf{V}) = (E, Con, \vdash)$ to be the structure such that:
$$X \in Con \Leftrightarrow_{def} X \text{ is finite } \&\; \exists u \in \mathbf{V}.\, X \subseteq u,$$

$$X \vdash e \Leftrightarrow_{def} X \in Con \;\&\; \exists u \in \mathbf{V}.\, e \in u \;\&\; u \subseteq X \cup \{e\}.$$

Theorem 2.3.7. [76] If $\mathbf{V}$ is a family of configurations on a set of events E, then $\mathcal{E}(\mathbf{V})$, as in definition 2.3.6, is an event structure such that $\mathcal{V}(\mathcal{E}(\mathbf{V})) = \mathbf{V}$.

2.3.2 Stable Event Structures [76]

Elementary event structures are one of the most general classes of event structures; they allow an event to have different causes, which is a desirable property. However, problems arise when dealing with configurations in which an event does not have a unique cause.Such configurations can be excluded by applying a *stability* constraint, leading to the definition of stable event structures. It is worth noting that for stable event structures there is no global partial order of causal dependency on events, but each configuration has its own local partial order of causal dependency. This is clarified in definition 2.3.12 and example 2.3.14.

Definition 2.3.8. A *stable event structure*, is an elementary event structure, satisfying the *stability axiom*:

$$X \vdash e \;\&\; Y \vdash e \;\&\; X \cup Y \cup \{e\} \in Con \Rightarrow X \cap Y \vdash e.$$

Thus, the stability axiom ensures events are caused in a unique way.

Definition 2.3.9. A minimal enabling relation $\vdash_{min}$ is defined as:

$$X \vdash_{min} e \Leftrightarrow X \vdash e \;\&\; (\forall Y \subseteq X.\, Y \vdash e \Rightarrow Y = X)$$

Then for any event structure

$$Y \vdash e \Rightarrow \exists X \subseteq Y.\, X \vdash_{min} e$$

and for consistent enabling sets of stable event structures such an X is unique:

$$Y \vdash e \;\&\; Y \cup \{e\} \in Con \Rightarrow \exists! X \subseteq Y.\, X \vdash_{min} e$$

The following theorem characterises the family of configurations of stable event structures, while the following definition does so for a family of configurations in general.

Theorem 2.3.10. [76] The family of configurations of a stable event structures $\mathcal{E}$ satisfies:

$$\forall X \subseteq \mathcal{V}(\mathcal{E}).\, X \neq \emptyset \,\&\, X \uparrow \Rightarrow \bigcap X \in \mathcal{V}(\mathcal{E}).$$

Definition 2.3.11. A family of configurations $\mathbf{V}$ is *stable* when it satisfies the stability axiom:

$$\forall X \subseteq \mathbf{V}.\, X \neq \emptyset \,\&\, X \uparrow \Rightarrow \bigcap X \in \mathbf{V}.$$

Definition 2.3.12. Let u be a configuration of a stable family of configurations $\mathbf{V}$. For events $e, e' \in u$ we define

$$e \leq_u e' \Leftrightarrow \forall v \in \mathbf{V}.\, e' \in v \,\&\, v \subseteq u \Rightarrow e \in v.$$

For $e \in u$ we define

$$\lceil e \rceil_u = \bigcap \{v \in \mathbf{V} \mid e \in v \,\&\, v \subseteq u\}$$

Proposition 2.3.13. Let u be a configuration of a stable family of configurations $\mathbf{V}$. Then $\leq_u$ is a partial order on u and $\lceil e \rceil_u$ is a configuration such that $\lceil e \rceil_u = \{e' \in u \mid e' \leq_u e\}$. Moreover, the configurations $v \subseteq u$ are exactly the left-closed subsets of $\leq_u$.

Example 2.3.14. [76] Let $\mathcal{E} = (E, Con, \vdash)$ be an event structure where $E = \{e_0, e_1, e_2\}$, $Con = \wp(E) \setminus \{e_0, e_1, e_2\}$ and the enabling relation is the least one including:

$$\emptyset \vdash e_0,\ \emptyset \vdash e_1,\ \{e_0\} \vdash e_2,\ \{e_1\} \vdash e_2$$

Then it is easy to see that $\mathcal{E}$ is a stable event structure its family of configurations $\mathcal{V}(E)$ are a stable family of configurations.

Let $x = \{e_0, e_2\}$ and $y = \{e_1, e_2\}$ be two configurations of $\mathcal{E}$. Then $e_0 \leq_x e_2$ and $e_1 \leq_y e_2$, however $e_0 \not\leq_y e_2$ and $e_1 \not\leq_y e_2$. Thus, $\leq_x$ and $\leq_y$ cannot be restrictions of a global partial order on events. □

2.3.3 Prime Event Structures [76]

Prime event structures are event structures whose enabling relation provides a global partial order of causal dependency on events, as each event can be enabled in only one way.

Definition 2.3.15. A *prime event structure* is a triple $\mathcal{E} = (E, Con, \leq)$, where $\leq$ is a partial order on the set of events E, called the causality relation and Con is the consistency relation as defined previously in definition 2.3.1. For all $e \in E$ and finite subsets x, y of E it satisfies:

1. $\{e' \mid e' \leq e\}$ is finite.
2. $\{e\} \in Con$.
3. $Y \subseteq X \;\&\; X \in Con \Rightarrow Y \in Con$.
4. $Y \in Con \;\&\; \exists e' \in X.\, e \leq e' \Rightarrow X \cup \{e\} \in Con$.

Definition 2.3.16. We define the history of e by $\lceil e \rceil =_{def} \{e' \in E \mid e' \leq e\}$.

Definition 2.3.17. The set of configurations of a primary event structure is denoted by $\mathcal{V}(\mathcal{E})$ (or $\mathcal{V}_{\mathcal{E}}$ or $\mathcal{V}$ if no confusion arises) and is defined as the set of subsets $X \subseteq E$ which are consistent and left-closed under $\leq$.

It can be proved that $\mathcal{V}(\mathcal{E})$ is a stable family of configurations for $\mathcal{E}$. The translation from stable event structures to prime event structures is presented in §2.3.5.

A stable event structure can be translated into a prime structure by extending and renaming the events, so that each event is augmented with the history of how it occurred in a configuration. Then, even though the events and as such the configurations are different, the domains of configurations of both event structures are isomorphic as partial orders [76].

2.3.4 Event Structures [76, 78]

Definition 2.3.18. An *event structure* is a triple $\mathcal{E} = (E, \leq, \#)$, where $\leq$ is a partial order on the set of events E, called the causality relation and $\# \subseteq E \times E$ is a binary, symmetric and irreflexive relation called the conflict relation, satisfying the following for all $e, e', e'' \in E$.

1. $\{e' \mid e' \leq e\}$ is finite
2. $e \# e' \;\&\; e' \leq e'' \Rightarrow e \# e''$

Event structures can also be seen as prime event structures which satisfy:

$$X \in Con \Leftrightarrow X \subseteq_{fin} E \;\&\; \forall e, e' \in X. \neg(e \# e')$$

Definition 2.3.19. Let $\mathcal{E} = (E, \leq, \#)$ be an event structure. A configuration of $\mathcal{E}$ is defined as a subset of events $X \subseteq E$ such that:

1. X conflict-free: $\forall e, e' \in X.\ \neg(e \# e')$.
2. X left-closed: $\forall e \in X.\ e' \leq e \Rightarrow e' \in X$.

As before we let $\lceil e \rceil =_{def} \{e' \in E \mid e' \leq e\}$ and with abuse of notation, for a set X of events let $\lceil X \rceil =_{def} \{e' \in \lceil e \rceil \mid e \in X\}$.

2.3.5 From Stable Event Structures to Prime Event Structures [76]

Given a stable event structure, for which a global partial order on events does not necessarily exist (example 2.3.14), a prime event structure can be derived with global partial order of causal dependency. This requires restructuring the events to include their history, therefore, the derived prime event structure is not isomorphic to the stable event structure. However, it can be proved that their domains of configurations are isomorphic [76].

Definition 2.3.20. Given a stable event structure $\mathcal{E} = (E, Con, \vdash)$, its associated prime event structure $\mathcal{E}' = (P, Con_P, \leq)$ is defined as follows.

- The events are
$$P = \{\lceil e \rceil_x \mid e \in x \in \mathcal{V}(\mathcal{E})\}.$$
- The global causal dependency is given by
$$p' \leq p \Leftrightarrow p' \subseteq p.$$
- The consistency predicate on P is defined as
$$X \in Con_P \Leftrightarrow X \subseteq_{\text{fin}} P \ \&\ X \uparrow .$$

The causality relation is defined based on the idea that an event p can occur only after all events p' included in it (representing its history) have occurred. The consistency predicate checks if a set of events is compatible as a configuration.

Theorem 2.3.21. [76] The domain of configurations of a stable event structure $\mathcal{E}$ and its associated prime event structure $\mathcal{E}'$ as defined in definition 2.3.20 are isomorphic.

2.4 Petri nets [57, 53]

Event structures can be viewed as structures giving semantics to Petri nets which are well-known models of true concurrency with a variety of applications. In this section we define Petri nets along with some of the relevant concepts.

Definition 2.4.1. A *general Petri net* is a tuple $\mathcal{G} = (P, T, Pre, Post, M)$, where P is a set of places, T is a set of transitions s.t. $P \cap T = \emptyset$, $Pre \subseteq_\mu T \times P$ is the pre-place multi-relation, $Post \subseteq_\mu T \times P$ is the post-place multi-relation and M is a non-empty finite multi-set of P called the *initial marking* of $\mathcal{G}$.

A graphical representation of a Petri net consists of circles for places, rectangles for transitions, connecting arcs between transitions and places denoting *Pre* and *Post*, annotated by their corresponding multi-relation weight and dots in places denoting the markings.

Definition 2.4.2. Define multi-sets ${}^\bullet x =_{def} Pre.\{x\}$ and $x^\bullet =_{def} Post.\{x\}$ where $x \in P \cup T$. By abuse of notation we define the same notion for a multi-set X of transitions or places. Namely, ${}^\bullet X =_{def} Pre.X$ and $X^\bullet =_{def} Post.X$.

The token game of Petri nets abides by the following rules. A transition t is *enabled* at a marking M iff ${}^\bullet t \leq M$.

More specifically, an enabled transition consumes $Pre[t,p]$ tokens from p and deposits $Post[t,p']$ tokens in p'. This defines a relation between markings represented by $M \xrightarrow{X} M' \Leftrightarrow_{def} {}^\bullet X \leq M$ and $M' = M - {}^\bullet X + X^\bullet$ where X is a finite multi-set of events. A marking M is said to be *reachable* iff there is an initial marking M' from which M can be reached by a finite sequence of transitions $\xrightarrow{X_i}$.

Definition 2.4.3. A Petri net is *k-bounded* iff for every reachable marking M and every place $p \in P$, $M(p) \leq k$.

Another important concept is that of safety, as defined below.

Definition 2.4.4. A Petri net is *safe* iff it is 1-bounded.

Safe nets can be represented as (B, E, F, M) where B is the set of conditions, E is the set of events and $b\ F\ e$ and $e\ F\ b$ represent $b \in B$ being a pre-condition and post-condition of $e \in E$, respectively.

A morphism between general nets preserves the token game and is defined as follows.

Definition 2.4.5. [76] A *morphism* $(\eta,\beta) : \mathcal{G} \to \mathcal{G}'$ consists of a partial function $\eta : T \to T'$ and a multi-relation $\beta \subseteq_\mu P \times P'$ such that

1. $\beta.M = M'$
2. $\forall t \in T.\ \beta({}^\bullet t) = {}^\bullet(\eta(t))$
3. $\forall t \in T.\ \beta(t^\bullet) = (\eta(t))^\bullet$

where we identify η with its graph viewed as a multi-relation, and write $*$ for the empty multiset and we denote the application of η to a set X by $\eta.X$.

Definition 2.4.6. [76] A morphism between safe Petri nets is a *folding* iff η is total and β can be identified by a total function.

The following defines a subnet of a Petri net.

Definition 2.4.7. [76] Let $(\eta,\beta) : \mathcal{G} \to \mathcal{G}'$ be a morphism between safe Petri nets. Then $\mathcal{G}$ is a *subnet* of $\mathcal{G}'$ iff

$$\eta(e) = e' \Leftrightarrow e = e' \text{ and } \beta[b,b'] = 1 \Leftrightarrow b = b'.$$

The following theorem describes the fact that morphisms preserves markings of nets.

Theorem 2.4.8. [76] Let $(\eta,\beta) : \mathcal{G} \to \mathcal{G}'$ be a morphism of Petri nets. Then β preserves the initial and reachable markings, i.e. if M is a reachable marking of $\mathcal{G}$ then $\beta.M$ is a reachable marking of $\mathcal{G}'$. Furthermore, if $M \xrightarrow{X} M'$ and M is reachable in $\mathcal{G}$ then $\beta.M \xrightarrow{\eta.A} \beta.M'$ in $\mathcal{G}'$.

In the following, the properties of morphism between **safe** Petri nets are described.

Proposition 2.4.9. [76] Let $\mathcal{G}_0 = (B_0,E_0,F_0,M_0)$ and $\mathcal{G}_1 = (B_1,E_1,F_1,M_1)$ be two safe Petri nets. Then $f = (\eta,\beta) : \mathcal{G}_0 \to \mathcal{G}_1$ is a morphism iff η is a partial function from E_0 to E_1 and β is a relation between B_0 and B_1 satisfying all the following.

1. $\beta(M_0) = M_1$ and $\forall b_1 \in M_1.\ \exists! b_0 \in M_0.\ b_0 \beta b_1$
2. if $\eta(e_0) = e_1$ then for any $b_1 \in B_1$

$$b_1 F_1 e_1 \Rightarrow \exists! b_0 \in B_0.\ b_0 F_0 e_0 \ \&\ b_0 \beta b_1$$

$$e_1 F_1 b_1 \Rightarrow \exists! b_0 \in B_0.\ e_0 F_0 b_0 \ \&\ b_0 \beta b_1$$

3. if $b_0 \beta b_1$ then for any $e_0 \in E_0$

$$e_0 F_0 b_0 \Rightarrow \exists e_1 \in E_1 .\ e_1 F_1 b_1 \ \&\ \eta(e_0) = e_1$$

$$b_0 F_0 e_0 \Rightarrow \exists e_1 \in E_1 .\ b_1 F_1 e_1 \ \&\ \eta(e_0) = e_1$$

Chapter 3

Compact Unfoldings

Petri nets are well-known models of true concurrency, as models which capture the behaviour of a system in a fairly intuitive and compact manner. However, the complete behaviour of the system is not immediately obvious from the original Petri net, as different choices (conflicts) and concurrent components create various possibilities for different runs of the net - even within the true concurrency framework.

Unfoldings of Petri nets, first introduced in [53] and further formalised in [76] for safe Petri nets, describe the behaviour of the net by unfolding its conflicts and cycles. Unfolding the conflicts results in a unique event being created for every possible history of a transition, namely, the set of elements leading to firing of that transition. In addition to unfolding the cycles, unfoldings of Petri nets can be viewed as true concurrency semantics for the behaviour of Petri nets. However, creating an event for every possible history results in a fast expansion of the unfolding, growing exponentially with the choices available in the net. The problem becomes worse when when dealing with infinite unfoldings which are usually the cases of interest.

The largeness of unfoldings of Petri nets raises the question that can unfoldings be more compact while describing the behaviour of the net? There are other interesting approaches attempting to achieve compactness, including Merged Processes [37], Trellis Processes [23] and R-unfoldings [58]. While the nature of the problem is the same, the main difference between the above approaches lies in the objective that each intends to achieve. For example, merged processes are the most compact structures handling unsafeness, Trellis processes have the factorisation property and R-unfoldings give semantics for both individual and collective token philosophy and can forget history step

by step.

In our approach, we primarily deal with safe Petri nets. However, our approach is extensible to the general Petri nets as we discuss in the conclusion chapter. Our aim is to unfold the cycles within a net and keep the conflict relation as folded as possible. In other words, we are more interested in representing what events can be fired, how many of them and in what order, rather than how exactly an event occurs in terms of the choices made in its history. It turns out that we can reuse the conditions created to a great extent, if we forget about the exact history of how an event occurred. Therefore, by allowing backwards conflict, instead of creating a new condition for every single possible history of an event firing, the conflicting events firing to the same place are grouped together, as only one of them can occur at a time.

In this chapter we introduce compact unfoldings of safe Petri nets as described above, which are occurrence nets with backwards conflict preserving the token game of the Petri nets. We also present an algorithm for constructing the compact unfoldings of a safe Petri net (with finitely many reachable markings) in a deterministic and more direct manner.

Apart from defining the semantics of Petri nets, unfoldings of Petri nets are extensively used for verification of certain properties for Petri nets, e.g. detection of deadlocks. For the most common of such approaches, the notion of a complete finite prefix is crucial, which was first introduced by McMillan in [51], improved in [22, 38] and further formalised in [39]. We therefore define the complete finite prefix for the compact unfoldings, which makes the verification algorithms based on complete finite prefixes applicable to them.

Closely related to unfoldings of Petri nets, are event structures as models of true concurrency which explicitly capture concurrency and in particular conflicts. The compact unfoldings correspond directly to stable event structures as first introduced by [76]. To our best knowledge, unlike event structures, the relation between Petri nets and stable event structures has not been studied before. Thus, we conclude the chapter with introducing the translation from safe Petri nets into stable event structures and we also present a short comparison of our approach to the other existing ones.

3.1 Unfoldings

In this section we briefly present the traditional unfolding of Petri nets, which unfolds the original net into an occurrence net, as defined below. Occurrence nets can describe the behaviour of nets in a simpler manner, while being easily translatable into other models of concurrent and in particular into event structures. The definitions in this section (3.1) are mostly taken from [76].

Definition 3.1.1. An *occurrence net* $O = (B, E, F, M)$ is a *safe* net satisfying the following constraints.

1. $\forall b \in M.\ {}^\bullet b = \emptyset$
2. $\forall b' \in B.\ \exists b \in M.\ b\ F^*\ b'$
3. $\forall b \in B.\ \mid {}^\bullet b \mid \leq 1$
4. F^+ is irreflexive and $\forall e \in E.\ \{e' \mid e' F^* e\}$ is finite.
5. $\#$ is irreflexive, where for $e, e' \in E$ and $x, x' \in E \cup B$:

$$e \#_m e' \quad \Leftrightarrow_{def} \quad e, e' \in E\ \&\ e \neq e'\ \&\ {}^\bullet e \cap {}^\bullet e' \neq \emptyset$$
$$x \# x' \quad \Leftrightarrow_{def} \quad \exists e, e' \in E.\ e \#_m e'\ \&\ e F^* x\ \&\ e' F^* x'$$

The first constraint describes initial markings, while the second one ensures that there are no isolated conditions. The third constraint prevents backwards conflicts, something that we shall change to achieve a different, more compact structure. The fourth constraint guarantees that cycles are unfolded and that events have finite history. Finally, the last constraint defines the binary and irreflexive conflict relation, describing that conflicts are inherited by elements in F relation.

A conflict-free net is then defined as below.

Definition 3.1.2. Given an occurrence net $O = (E, B, F, M)$, a set $X \subseteq E \cup B$ is *#-free* or *conflict-free* iff $\nexists x, x' \in E \cup B.\ x \# x'$.

In occurrence nets, each element can be enabled in a single manner. The elements involved in enabling another element form the *history* of that element, as defined below.

Definition 3.1.3. We then denote the history of an element $x \in E \cup B$ of the occurrence net by

$$[x] =_{def} \{x' \in E \mid x' F^* x\}$$

Moreover, the notion of *concurrency* is captured by the following definition.

Definition 3.1.4. The *concurrency* relation $co \subseteq (B \cup E) \times (B \cup E)$ is defined as

$$x \; co \; y \Leftrightarrow_{def} \neg(x \# y \vee x F^{+} y \vee y F^{+} x)$$

and a set $A \subseteq E \cup B$ is pairwise concurrent written as $co\, A$ iff $\forall x \neq y \in A.\; x \; co \; y \;\&\; \forall x \in A.\; [x] \cap E$ is finite.

Unfoldings of safe Petri nets are characterised by the following theorem.

Theorem 3.1.5. [76] Given a safe Petri net $\mathcal{G} = (B, E, F, M)$, there is a unique occurrence net $\mathcal{U}(\mathcal{G}) = (B_0, E_0, F_0, M_0)$ and a folding $f = (\eta, \beta) : \mathcal{U}(\mathcal{G}) \to \mathcal{G}$ given by $\beta((e_0, b)) = b$ and $\eta((A, e)) = e$, for which the following hold.

$$\begin{aligned} M_0 &= \{(\perp, b) \mid b \in M\} \\ B_0 &= M_0 \cup \{(\{e_0\}, b) \mid e_0 \in E_0 \;\&\; b \in B \;\&\; \eta(e_0) \in {}^{\bullet}b\} \\ E_0 &= \{(A, e) \mid A \subseteq B_0 \;\&\; e \in E \;\&\; co\, A \;\&\; \beta.A = {}^{\bullet}e\} \end{aligned}$$

$$b_0 F_0 (A, e) \Leftrightarrow b_0 \in A$$

$$e_0 F_0 b_0 \Leftrightarrow b_0 = (\{e_0\}, b)$$

Definition 3.1.6. [76] The *unfolding* of a safe Petri net is defined as the unique occurrence net and the folding morphism described in theorem 3.1.5.

The following theorem describes a fundamental property of the unfolding of a Petri net, implied by theorem 3.1.5.

Theorem 3.1.7. [76] Let $\mathcal{G}$ be a safe net. Its unfolding $\mathcal{U}$ along with folding f as defined above satisfy:

$$\forall A \subseteq B_0.\; co\, A \;\&\; \beta.\, A = {}^{\bullet}e \Rightarrow \exists! e_0 \in E_0.\; A = {}^{\bullet}e_0 \;\&\; \eta(e_0) = e.$$

3.2 Compact Unfoldings

Although occurrence nets are easily translated into event structures(and thus into a sub-class of stable event structures), the obtained event structures do not take advantage of the main characteristic of stable event structures, namely, that of allowing multiple histories, uniquely enabling an event. This also implies that event structures driven from occurrence nets are not the most compact ones representing the behaviour of the

original net. In order to allow for multiple histories for an event, backwards conflict should be permitted. Allowing backwards conflicts avoids the expansion caused by creating a unique event for each history of an event. In this section, we introduce a new and more compact unfolding of safe Petri nets, where as far as possible conflicts are not unfolded.

Compact unfoldings of Petri nets are defined on structures called *occurrence$^-$nets*, defined below.

3.2.1 Definitions

Definition 3.2.1. An *occurrence$^-$net* $O^- = (B, E, F, M)$ is a *safe* net satisfying conditions 1, 2, and 4 of occurrence nets in addition to a modified version of condition 5 of occurrence nets:

$5'$. $\#^-$ is irreflexive, where for $e, e' \in E$ and $b, b' \in B$:

$$
\begin{array}{lll}
e\#^-_m e' & \Leftrightarrow_{def} & e \neq e' \;\&\; {}^\bullet e \cap {}^\bullet e' \neq \emptyset \\
e\#^- e' & \Leftrightarrow_{def} & e\#^-_m e' \text{ or } \exists b \in {}^\bullet e.\ b\#^- e' \text{ or } \exists b' \in {}^\bullet e'.\ b'\#^- e \\
b\#^- b' & \Leftrightarrow_{def} & \forall e \in {}^\bullet b, e' \in {}^\bullet b'.\ e\#^- e' \;\&\; b, b' \notin M \\
e\#^- b & \Leftrightarrow_{def} & \exists b' \in {}^\bullet e.\ b\#^- b' \text{ or } \forall e' \in {}^\bullet b.\ e\#^- e' \;\&\; b \notin M \\
b\#^- e & \Leftrightarrow_{def} & \text{defined symmetrically.}
\end{array}
$$

Note that the above recursive definition of $\#^-$ is well-defined as the predecessor relation ${}^\bullet x$ is well-founded. As explained previously, occurrence$^-$nets allow backwards conflict, which results in allowing multiple histories for an event. Therefore, a new notion of conflict needs to be defined, as now two elements can be in conflict when enabled in one way, and not in conflict if enabled in another way. Consider the following example.

Example 3.2.2. Consider two simple occurrence$^-$nets, O_1^- (left) and O_2^- (right) in figure 3.1. In O_1^-, we have $b_3\#^- b_4$ since $\forall e \in {}^\bullet b_3, e' \in {}^\bullet b_4.\ e\#^- e'$, namely, $e_1\#^- e_3$ & $e_2\#^- e_3$. However, in O_2^- we no longer have $b_3'\#^- b_4'$, since $e_2' \in {}^\bullet e_3'$ & $e_2' \in {}^\bullet e_4'$ and clearly $\neg(e_2'\#^- e_2')$ as $\#^-$ is irreflexive. More intuitively, in O_1^-, no matter how b_3 is enabled, b_4 cannot be enabled at the same time. However, in O_2^-, if b_3' is enabled through e_2', then b_4' is also enabled, though if b_3' is enabled through e_1' then b_4' cannot be enabled at the same time. Thus, b_3' and b_4' are not *always* in conflict and therefore, not in $\#^-$.

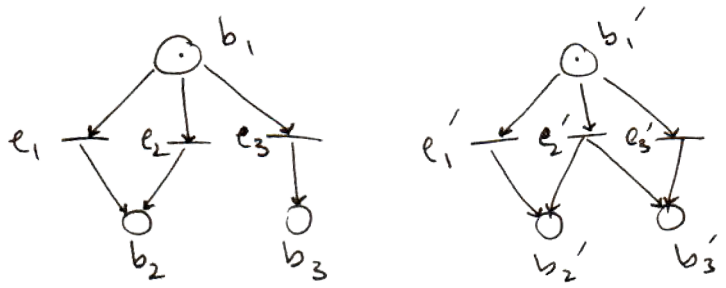

Figure 3.1: Two simple occurrence$^-$nets, O_1^- (left) and O_2^- (right)

As the above example shows, we have defined two elements to be in conflict iff they are *always* in conflict, i.e. no matter how they are enabled, they are in conflict.

A conflict$^-$-free occurrence$^-$net is then defined as follows.

Definition 3.2.3. Given an occurrence$^-$net $O^- = (E,B,F,M)$, a set $X \subseteq E \cup B$ is $\#^-$-*free* or *conflict$^-$-free* iff $\nexists x,x' \in E \cup B.\ x\#^-x'$.

We have explained that in occurrence$^-$nets, elements can be enabled in more than one way. We now formalise this in the definition of the history of an element and that of a set of elements below.

Definition 3.2.4. A *history* of an element $x \in E \cup B$ is a minimal$^{\subseteq}$ $\#^-$-free subset $H \subseteq \{y \in E \cup B \mid yF^+x\}$ satisfying the following.

1. $\forall y \in (H \cup \{x\}) \cap B.\ {}^\bullet y \neq \emptyset \Rightarrow \exists y' \in {}^\bullet y.\ y' \in H$
2. $\forall y \in (H \cup \{x\}) \cap E.\ \forall y' \in {}^\bullet y.\ y' \in H$
3. if $T = \emptyset$ then ${}^\bullet x = \emptyset$

The set of histories of x is then referred to by ${}^*x$ and $[x]_H =_{def} H \cup \{x\}$ denotes a history of x enabling it.

The first two constraints in the definition above describe that for each condition (b), one event is selected is its predecessor (${}^\bullet b$) and that for each event (e), all its predecessors (${}^\bullet e$) should exist in the history H. Note that since occurrence$^-$nets are safe, then a condition cannot have more than one predecessor in a history H, as otherwise the predecessor events cannot occur together and therefore H cannot be $\#^-$-free. The last constraint describes the history of initial conditions.

Remark 3.2.5. There is a slight yet deliberate inconsistency among the definition of a history of an element in occurrence$^-$nets and that of an event in (stable) event structures (definitions 2.3.12, 2.3.16): in occurrence$^-$nets (and hence occurrence nets) an element is not included in its history whereas in (stable) event structures, and event is included in its history. This is only to simplify some of the arguments about the occurrence$^-$nets while maintaining the standard definitions for (stable) event structures.

Example 3.2.6. Consider O_2^- in figure 3.1 and its condition b_4'. Then

$$^*b_4' = \{\{b_1', e_2'\}, \{b_1', e_3'\}\}$$

Lemma 3.2.7. If $x\#^-x'$ then for any history of x, H, and any history of x', H', $\exists e \in H \cup \{x\}, e' \in H' \cup \{x'\}.\ e\#_m^- e'$.

Proof. Follows from the definition of $\#^-$; more concretely, from the fact that for any two element x and x', they are in $\#^-$ iff there are *always* events e, e' in their past where $e\#_m^- e'$. □

Lemma 3.2.8. Let O^- be an occurrence$^-$net s.t. $\forall b \in B.\ |^\bullet b| \leq 1$. Then O^- is an occurrence net.

Proof. It is clear that all O^- satisfies conditions 1-4 in definition 3.1.1. Thus, we only need to show that $\forall e, e' \in E.\ e\#^- e' \Leftrightarrow e\#e'$ which follows from lemma 3.2.7 □

Definition 3.2.9. The *set of histories* of a set $X \subseteq E \cup B$ is defined as

$$^*S = \{H \mid H \text{ is a minimal}^{\subseteq}\ \#^-\text{-free set s.t.} \forall x \in X.\ \exists J \in {}^*x.\ J \subseteq H\}$$

Example 3.2.10. Consider the occurrence$^-$net O_5^- in figure 3.2. For e_4, e_5 in O_5^- we have:

$$^*e_4 = \{\{a_1, e_1, b_1\}, \{a_1, a_2, e_2, b_1\}\}$$

$$^*e_5 = \{\{a_2, e_3, b_2, b_3, e_5\}\}$$

$$^*\{e_4, e_5\} = \{\{a_1, e_1, b_1, a_2, e_3, b_2, b_3, e_5\}\}$$

We can now define a set to be consistent if it has a history, implying that all the elements in the set can occur in some scenario:

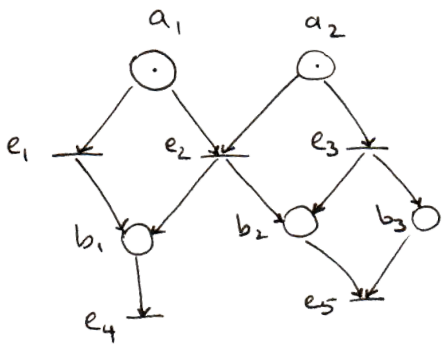

Figure 3.2: O_5^-

Definition 3.2.11. A set $X \subseteq E \cup B$ is *consistent* iff ${}^*X \neq \emptyset$.

Furthermore, a concurrent set can then be defined as follows.

Definition 3.2.12. A set $X \subseteq E \cup B$ is *concurrent*, represented by $co^- X$ iff X is consistent and $\forall x, y \in X.\ \neg(xF^+y \text{ or } yF^+x)$.

A $\#^-$-free yet inconsistent set appears in nets with *confusion*, as defined below.

Definition 3.2.13. An event $e \in E$ of an occurrence$^-$ net O^- is *confusing* iff $\exists e', e'' \in E.\ co^-\ \{e', e''\}\ \&\ \neg co^-\{e, e', e''\}$.

This is in fact an extension of symmetric confusion in Petri nets defined in [53], in the sense that it includes non-immediate conflict between events. Confusing events affect the outcome of the firing transitions, since if a confusing event occurs, it disables (or reflects the impossibility of) the occurrence of two or more concurrent events.

Example 3.2.14. Consider O_1^- in figure 3.1. Then event e_2 is confusing as $co^-\{e_1, e_3\}$, whereas $\neg co^-\{e_1, e_2, e_3\}$.

Example 3.2.15. Consider the following occurrence$^-$net (O_4^-). The set $\{c_1, c_2, c_3\}$ is conflict$^-$-free, as every pair of its elements can occur together under a certain history. However, these three elements cannot occur at the same time, hence they are inconsistent. (Note that all the events $e_1, \ldots, e_5$ are confusing.)

In what follows, we require to refer to the elements in the unfolding and compact unfoldings (to be defined) in an inductive manner. We therefore define the notion of depth for the elements in occurrence and occurrence$^-$nets.

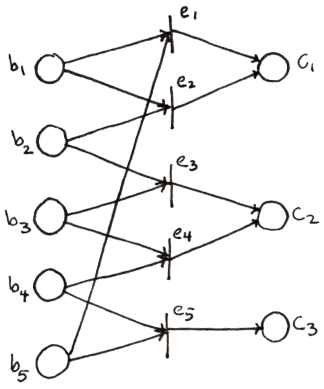

Figure 3.3: An occurrence$^-$net with O_4^- with conflict-free and inconsistent subset of elements

Definition 3.2.16. The *depth* of an element $x \in E \cup B$ represented by $d(x)$ of an occurrence net $O = (B, E, F, M_0)$ is defined as follows, where $b \in B$ and $e \in E$.

- $d(b) = 0$ if $\exists M \in M_0.\ b \in M$
- $d(b) = max\{d(e) \mid e \in E\ \&\ eFb\}$
- $d(e) = 1 + max\{d(b) \mid b \in B\ \&\ bFe\}$

The occurrence net restricted to the elements of up to depth n is denoted by $O^{(n)}$ with its events and conditions represented by $E^{(n)}$ and $B^{(n)}$, respectively.

As before, since the elements in occurrence$^-$nets can have multiple histories, they may also be assigned different depths. We use the definition of depth for occurrence nets by applying it to a history of an element. Note that given x an element of an occurrence$^-$net, $H \in {}^*x$ can be seen as an occurrence net (lemma 3.2.8) and we define $d(H) = max\ \{d(x) \mid x \in H\}$.

Definition 3.2.17. The set of *depths* of an element $x \in E \cup B$ represented by $D(x)$ of an occurrence $^-$ net $O^- = (B, E, F, M_0)$ is defined as follows, where $b \in B$ and $e \in E$.

- $D(b) = \{d(H_i) \mid H_i \in {}^*b\}$
- $D(e) = \{d(H_i) + 1 \mid H_i \in {}^*e\}$

The occurrence$^-$ net restricted to the elements which have some depth less than or equal to n is denoted by $O^{-(n)}$ with its events and conditions represented by $E^{(n)}$ and

$B^{(n)}$, respectively.

Example 3.2.18. Consider O_6^- depicted in figure 3.4. Then, $D(a) = \{0\}$, $D(e_1) = \{1\}$ and $D(d) = \{1,2\}$.

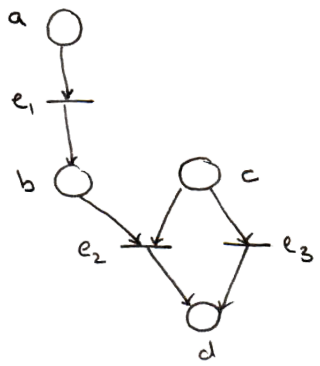

Figure 3.4: An occurrence$^-$net (O_6^-), with the highest depth of 2

3.2.2 Compact Unfoldings of Safe Petri nets

We now define the new compact unfoldings for safe Petri nets. The idea is similar to the traditional unfoldings of Petri nets with the difference that, as far as possible, we do not unfold conflicts . As it follows from the definition of traditional unfoldings of Petri nets where backward conflicts are not allowed, every condition in the unfolding of a net has one preceding event and therefore a single causal path of elements or *history*. Here, we try to avoid this fast growing expansion by forming the largest possible groups of conflicting events (which fire to the same corresponding place in the original net), which then fire to the same condition in the compact unfolding or unfolding$^-$.

Definition 3.2.19. Let $G = (B,E,F,M)$ be a safe Petri net. An unfolding$^-$ $\mathcal{U}^-(G) = (B_1 \subseteq \wp(E_1) \times B, E_1 \subseteq \wp(B_1) \times E, F_1, M_1 \subseteq B_1)$ is a minimal occurrence$^-$ net closed under the following rules, where $\beta^- : B_1 \to B$ is given by $\beta^-((X,b)) = b$, and $\eta^- : E_1 \to E$ is given by $\eta^-((A,e)) = e$.

1. $M_1 = \{(\emptyset, b) \mid b \in M\}$
2. if $A \subseteq B_1, e \in E.\ \beta^-(A) = {}^\bullet e$ & $co^-\ A$ then $(A,e) \in E_1$ and $\forall b_1 \in A.\ b_1 F_1 (A,e)$

3. if $b \in B, X \subseteq^{max} E_1.(\forall e_1, e_1' \in X.\, e_1 \#^- e_1' \,\&\, \forall e_1 \in X.\, \eta^-(e_1) \in {}^\bullet b)$ then $(X,b) \in B_1$ and $\forall e_1 \in X. e_1 F_1 (X,b)$

Example 3.2.20. As a simple example consider the safe Petri net $\mathcal{G}_1$ depicted in figure 3.5. Then $\mathcal{G}_1$ has a single unfolding$^-$which is isomorphic to $\mathcal{G}_1$.

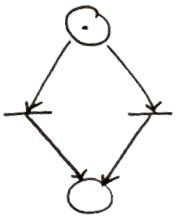

Figure 3.5: A simple safe Petri net, $\mathcal{G}_1$

Example 3.2.21. As another example, consider the safe Petri net $\mathcal{G}_2$ along with its unfolding$^-$and traditional unfolding as depicted in figure 3.6. As the conflicting events are kept folded, the exponential expansion occurring in the unfolding is avoided in the unfolding$^-$.

As we shall see in the following example, compact unfoldings are not necessarily unique, as in some cases there are choices to be made in grouping conflicting events. The following example is chosen from merged processes of [37], another unfolding semantics for Petri nets, to illustrate compact unfoldings along with their similarities and differences to the traditional unfoldings and merged processes for the interested reader.

Example 3.2.22. Consider $\mathcal{G}_3$ and its unfolding $\mathcal{U}_3$ as in figure 3.7. Then $\mathcal{G}_3$ has two unfoldings, one of which is depicted in figure 3.8. More concretely, we had a choice between merging conditions c_4 and c_7 or c_5 and c_6 in $\mathcal{U}_3$ and $\mathcal{U}_3^-$ is the result of choosing the latter.

3.2.3 Properties of Compact Unfoldings

In this section we describe some of the properties of compact unfoldings.

Remark 3.2.23. As mentioned before, safe Petri nets are generally represented by $\mathcal{G} = (B,E,F,M)$ where B is the set of conditions and E is the set of events. However, in the rest of this thesis we represent the original Petri net by $\mathcal{G} = (P,T,F,M)$, referring

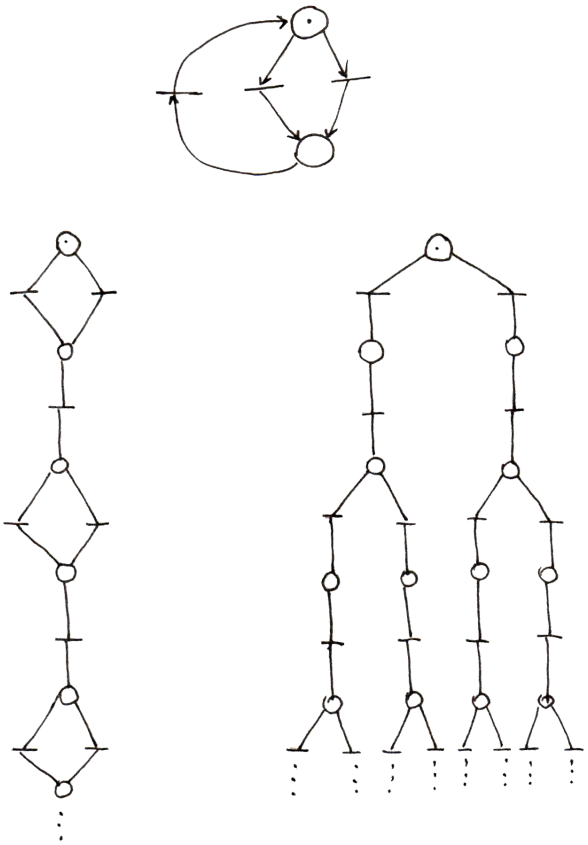

Figure 3.6: A safe Petri net $\mathcal{G}_2$, its unfolding$^-$ $\mathcal{U}_2^-$ (left) and its unfolding $\mathcal{U}$ (right)

to P as the set of places and T as the set of transitions. This is to simplify distinguishing the original net from its unfolding or unfolding$^-$.

We start with the concept of *mergeability* where merging two conditions b, b' means replacing b, b' by a single condition b'' whose flow relation is the sum of those of b and b'.

Definition 3.2.24. Conditions b and b' of an occurrence$^-$net O^- with a folding $f = (\eta, \beta)$ to a general Petri net are *mergeable* iff

1. $\beta(b) = \beta(b')$
2. $\forall e, e' \in {}^\bullet b \cup {}^\bullet b'.\ e \#^- e'$
3. merging b_1 and b_2 retains the occurrence$^-$net (i.e. does not create cycles).

Definition 3.2.25. An occurrence$^-$net is *compact* iff it does not have any mergeable conditions.

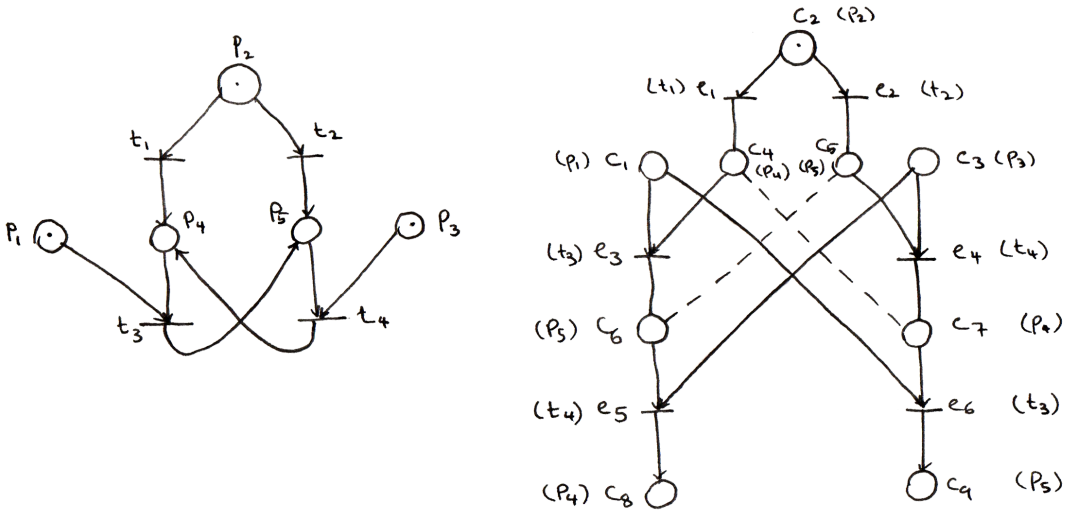

Figure 3.7: A safe Petri net $\mathcal{G}_3$ and its unfolding $\mathcal{U}_3$ (right)

Example 3.2.26. Consider the unfolding of the net in example 3.2.22. As explained in the example, conditions c_5 and c_6 and conditions c_4 and c_7 are mergeable, however, in the unfolding$^-$, c_4 and c_7 are not mergeable (anymore) as they do not satisfy conditions 2 and 3 of definition 3.2.24.

Proposition 3.2.27. Unfolding$^-$s of Petri nets are compact.

Proof. Follows from condition 3 of the definition of unfolding$^-$s, namely, that for every condition $b = (X, p)$, X is maximal. □

The following theorem establishes a property of compact unfoldings corresponding to that of traditional unfolding of Petri nets in theorem 3.1.7. Additionally, it captures that any compact occurrence net with a folding to a Petri net is isomorphic to an unfolding-$^-$of that net.

Theorem 3.2.28. Consider a general net $\mathcal{G} = (P, T, F, M)$, any unfoldings$^-$ $\mathcal{U}_i^-(\mathcal{G}) = (B_i, E_i, F_i, M_i)$ and the folding $f_i = (\eta_i, \beta_i) : \mathcal{U}_i^-(\mathcal{G}) \to \mathcal{G}$. Then the following holds

$$co^- \; S \;\&\; \beta_i^-.\, S = {}^\bullet t \Rightarrow \exists! e_i.\, S = {}^\bullet e_i \;\&\; \eta_i^-(e_i) = t \quad (*)$$

where $t \in T$, $e_i \in E_i$ and $S \subseteq B_i$.
Furthermore, for any occurrence$^-$ net $O^- = (B_0, E_0, F_0, M_0)$ and folding $f_0 = (\eta_0, \beta_0) : O^- \to \mathcal{G}$ where O^- is a compact occurrence$^-$ net satisfying the above property, there

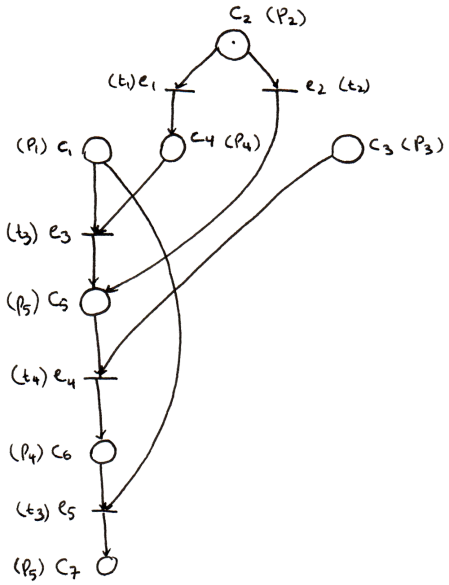

Figure 3.8: An unfolding$^-$ of $\mathcal{G}_3$, $\mathcal{U}_3^-$

is an isomorphic unfolding$^-$ $\mathcal{U}_k^-$.

Proof. For convenience we omit the $^-$ in this proof.

The first part (property *) follows easily from the definition of the unfolding$^-$. Given $O = (B_0, E_0, F_0, M_0)$ and the folding $f_0 : O \to \mathcal{G}$, we define a labelling function l such that $l(O)$ is an unfolding$^-$.

Let l be defined as follows (note the reuse of the same notation for a set of elements, i.e. $l(S) = \{l(a) \mid a \in S\}$):

- $\forall b_0 \in M_0.\ l(b_0) = (\bot, \beta_0(b_0))$
- $\forall b_0 \in B_0.\ l(b_0) = (l({}^\bullet b_0), \beta_0(b_0))$
- $\forall e_0 \in E_0.\ l(e_0) = (l({}^\bullet e_0), \eta_0(e_0))$

Since O is compact, it is easy to verify that $l(O)$ as defined above is an unfolding$^-$.

□

The following theorem describes the relation between occurrence nets with a morphism to a safe Petri net and unfolding$^-$s of that net.

Theorem 3.2.29. For any occurrence net $O = (B, E, F, M_0)$ with a morphism $f : (\pi, \gamma)$ to a general net $\mathcal{G} = (P, T, F, M)$, there is a unique morphism, $g_i : (\theta, \alpha)$, for each of its unfolding$^-$s $\mathcal{U}_i^-(\mathcal{G}) = (B_i, E_i, F_i, M_i)$, such that the following diagram commutes.

$$\begin{array}{ccc} \mathcal{U}_i & \xrightarrow{f_i} & \mathcal{G} \\ {\scriptstyle g_i}\uparrow & \nearrow_{f} & \\ O & & \end{array}$$

Proof. The proof follows a similar reasoning to that of Theorem 7.15 in [30]. We define g_i by defining $g_i^{(n)}$ at each depth.

For depth 0 of the elements in O, let $\theta^{(0)}$ be the empty function and

$$\alpha^{(0)}(b, b_i) \Leftrightarrow \exists p \in P.\ p = \gamma(b)\ \&\ b_i = (\gamma.M_0, p)$$

where $b \in B^{(0)}$ and $b_i \in B_i^{(0)}$.

For any element in O with depth less than n we define $g_i^{(n)}(x) = g_i^{(n-1)}(x)$. Then for elements of depth n we define:

$$\theta^{(n)}(e) = \begin{cases} *, & \text{if } \pi(e) = * \\ \text{(A,t)} & \text{if } \pi(e) = t\ \&\ \alpha^{(n-1)}.{}^\bullet e = A \end{cases} \tag{3.1}$$

$$\alpha^{(n)}(b, b_i) \Leftrightarrow \exists p \in P.\ p = \gamma(b)\ \ \&\ \ b_i = (X, p)$$

where $d(e) = n$, $d(b) = n$ and $n \in D(b_i)$.

It can then be shown that $g_i : (\theta, \alpha)$ defined as $g_i(x) = g^{(n)}(x)$ if $d(x) \leq n$ is a morphism (c.f. theorems 7.15 and 7.11 in [30]). Finally, it is straightforward to verify that g_i is the unique morphism for which the above diagram commutes.

□

Applying the above theorem to the traditional unfolding of a Petri net yields a stronger morphism between the traditional unfolding of a net and its unfolding$^-$, as the following theorem shows.

Proposition 3.2.30. Consider a safe Petri net $\mathcal{G}$ and its unfolding $\mathcal{U}$ with the folding $f_0 : \mathcal{U} \to \mathcal{G}$, and let $\mathcal{U}^-$ be an unfolding$^-$of $\mathcal{G}$ with the folding $f_1 : \mathcal{U}^- \to \mathcal{G}$. Then

applying theorem 3.2.29 to $\mathcal{U}$ (treated as O in the theorem) yields a total and onto morphism $g : \mathcal{U} \to \mathcal{U}^-$.

Proof. It is clear that g as described above is total, since $\mathcal{U}$ has a folding to $\mathcal{G}$ and therefore, g is defined for all the events and conditions in $\mathcal{U}$.

We show that g is onto by induction on the depth of the elements in $\mathcal{U}$ and $\mathcal{U}^-$. We focus on θ being onto, from which it follows that α is onto (proposition 2.4.9).

Let $\mathcal{U} = (B_0, E_0, F_0, M_0)$ and $\mathcal{U}^- = (B_1, E_1, F_1, M_1)$.
Base Case: It is clear from the definition of morphism that there is a bijection between $B_0^{(0)}$ and $B_1^{(0)}$. Then it follows that $E_0^{(1)}$ and $E_1^{(1)}$ are isomorphic as well, since for $\mathcal{U}$ we have $co\ A_0\ \&\ \beta_0.A_0 =^\bullet t \Rightarrow \exists! e_0.\ A_0 =^\bullet e_0.\ \eta(e_0) = t$ and for $\mathcal{U}^-$ we have $co^-\ A_1\ \&\ \beta_1.A_1 =^\bullet t \Rightarrow \exists! e_1.\ A_1 =^\bullet e_1.\ \eta_1(e_1) = t$. As initial conditions are concurrent for both occurrence nets and occurrence$^-$nets, each set of initial conditions enabling an event has a matching set of initial conditions in the other structure, enabling a unique corresponding event, which is captured in the definition of $\theta^{(0)}$.

Induction Hypothesis: $g^{(n)} : (\theta^{(n)}, \alpha^{(n)})$ is onto.

To show that $\theta^{(n+1)}$ is onto, suppose (A_1, e) is an event of $\mathcal{U}^-$ which has a depth of $n+1$. Then by definition of unfolding$^-$, $co^- A_1\ \&\ \beta_1.A_1 = {}^\bullet e$. Now by induction hypothesis, $\exists A_0.\ \alpha.A_0 = A_1$ and $\beta_0.A_0 = {}^\bullet e$. Thus, if we show that $co\ A_0$, by theorem 3.1.7 it follows that (A_0, e) is an event of $\mathcal{U}$ and $\theta^{(n+1)}(A_0, e) = (A_1, e)$, and therefore, $\theta^{(n+1)}$ is onto.

To prove that if $co^- A_1$ then $co\ A_0$, where A_0 and A_1 are as described above, note that morphisms preserve the flow relation in nets. Therefore, if $\exists x_0, x_0' \in E_0 \cup B_0.\ x_0 F_0 x_0'$ and $g(x_0) = x_1, g(x_0') = x_1'$, then $x_1 F_1 x_1'$. Now since $co^- A_1$, then $\nexists x_0, x_0' \in A_0.\ x_0 F_0^+ x_0'$ as otherwise $\exists x_1, x_1' \in A_1.\ x_1 F_1^+ x_1'$ which contradicts $co^- A$.

Moreover, from $co^- A_1$ it follows that A_1 is consistent and therefore, must have at least a history $H_1 \in^* A_1$. By induction hypothesis, $\exists H_0 \in E_0 \cup B_0.\ g(H_0) = H_1$. Then H_0 is #-free, since otherwise: $\exists e_0, e_0' \in H_0. e_0 \#_m e_0'$ which implies $\exists b_0 \in H_0. b_0 \in {}^\bullet e_0 \cap {}^\bullet e_0'$; and since $g^{(n)}$ is total and morphisms preserve the flow relation, then $\exists e_1, e_1', b_1 \in H_1.\ b_1 \in {}^\bullet e_1 \cap {}^\bullet e_1'$. However, this contradicts H_1 being #$^-$-free and therefore a history and therefore, H_0 is #-free.

Thus, we have shown that A_0 is #-free and that $\nexists x_0, x_0' \in A_0.\ x_0 F_0^+ x_0'$ and therefore $co\ A$.

□

Example 3.2.31. Figure 3.9 depicts the morphism from $\mathcal{U}_3$ to $\mathcal{U}_3^-$.

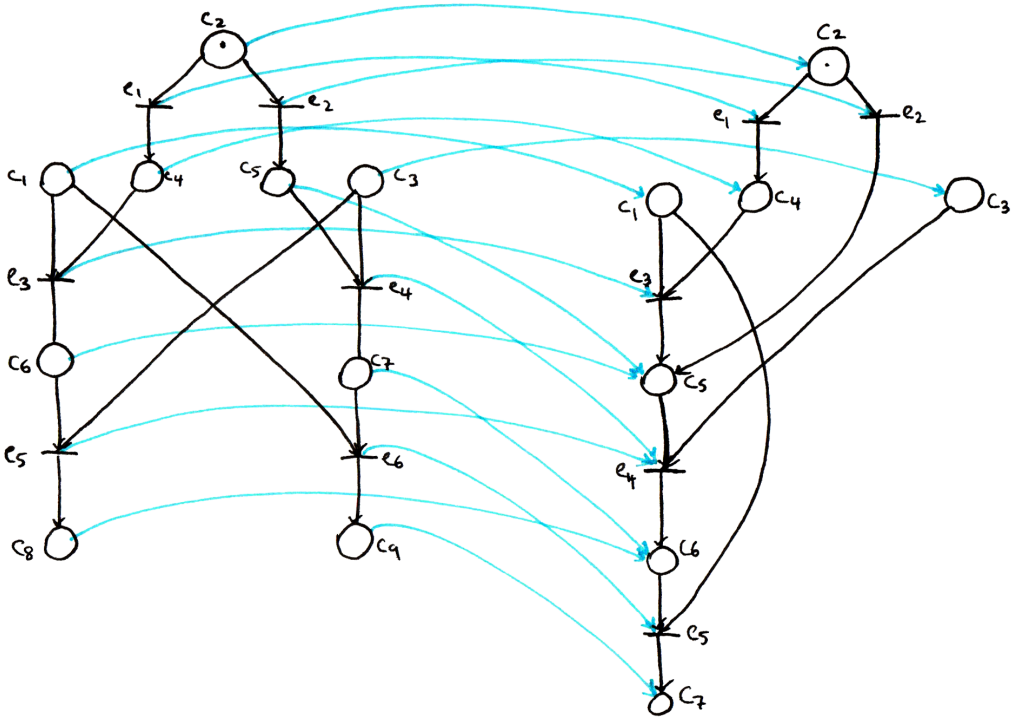

Figure 3.9: Morphism $g : \mathcal{U}_3 \to \mathcal{U}_3^-$ as in theorem 3.2.29 and proposition 3.2.30

We now define the configurations of an occurrence net and occurrence$^-$net. Roughly speaking, the configurations of these structures correspond to a possible history of a set of their elements.

Definition 3.2.32. A prefix $w = (B_w, E_w, F_w, M_w)$ of an unfolding or unfolding$^-$ $\mathcal{U}(^-)(\mathcal{G}) = (B, E, F, M_0)$ is a subnet of $\mathcal{U}(^-)$ for which the followings hold.

1. $\forall b \in B_w.\ {}^\bullet b \neq \emptyset \Rightarrow \exists e \in {}^\bullet b.\ e \in E_w$
2. $\forall e \in E_w.\ e^\bullet \cup {}^\bullet e \in B_w$
3. $\forall b \in B_w.{}^\bullet b \neq \emptyset \Rightarrow \exists e \in E_w.\ eFb.$

Definition 3.2.33. A configuration v of $\mathcal{U}$ is defined as any conflict-free prefix w of $\mathcal{U}$. The configuration v is maximal iff $\nexists x \in B \cup E.\ x \notin v$ & '$v \cup \{x\}$ is a configuration'.

Definition 3.2.34. A configuration v^- of $\mathcal{U}^-$ is defined as any consistent prefix w^-

of $\mathcal{U}^-$. The configuration v^- is maximal iff $\nexists x \in B^- \cup E^-.\ x \notin v^-$ & '$v^- \cup \{x\}$ is a configuration'.

It is important to note that when considering a history of a set of elements, $\#^-$-freeness coincides with consistency:

Lemma 3.2.35. A prefix v^- of an unfolding$^-$ $\mathcal{U}^-$ is $\#^-$-free iff it is consistent.

Proof. It is clear that if v^- is consistent then it has at least one history and therefore it is $\#^-$-free (lemma 3.2.7). The opposite follows from the definition of consistency. Suppose v^- is $\#^-$-free but not consistent. Then ${}^*v^- = \emptyset$ and $v^- \neq \emptyset$ (as the empty set is consistent, i.e. if $v^- = \emptyset$ then ${}^*v^- = \{\emptyset\}$). In other words, the set of histories of v^- cannot be empty unless there is a conflict in the union of histories of the elements in v^-, which is contained in v^- by definition of prefix. Therefore, v^- is not conflict-free which is a contradiction. (For more clarity, consider figure 3.3 of example 3.2.15, where $A = \{c_1, c_2, c_3\}$ is a $\#^-$-free and yet inconsistent set. Any attempt to expand A into a prefix results in including two events which are in $\#^-$). □

In the following theorem we draw a bijection among the configurations of an unfolding and unfolding$^-$of a safe Petri net. This further implies that every configuration (and hence marking) corresponding to a Petri net is captured by the unfolding$^-$s.

Theorem 3.2.36. The folding $g = (\theta, \alpha)$ as in proposition 3.2.30 induces a bijection between the configurations of $\mathcal{U}$ and $\mathcal{U}^-$.

Proof. We prove this by first showing that the morphism g defined in proposition 3.2.30 applied to a configuration v of the unfolding $\mathcal{U} = (B_0, E_0, F_0, M_0)$ yields a configuration v^- of $\mathcal{U}^- = (B_1, E_1, F_1, M_1)$ and that it is injective. Then we define a morphism from configurations of $\mathcal{U}^-$ to $\mathcal{U}$, showing it yields a configuration in $\mathcal{U}$.

This is straightforward to verify, as firstly, g is a morphism and preserves F. Therefore, a prefix in $\mathcal{U}$ is mapped to a prefix in $\mathcal{U}^-$. Secondly, since v is #-free, $\nexists e_0, e_0' \in v.\ {}^\bullet e_0 \cap {}^\bullet e_0' \neq \emptyset$. Therefore, g being a morphism, $\nexists e_1, e_1' \in g(v).\ {}^\bullet e_1 \cap {}^\bullet e_1' \neq \emptyset$ which implies $g(v)$ is $\#^-$-free. By lemma 3.2.35 it follows that $g(v)$ is consistent and therefore $g(v)$ is a configuration of $\mathcal{U}^-$. We refer to $g(v)$ by v^- in the rest of this proof.

We now show that g applied to configurations of $\mathcal{U}$ is injective. Suppose we have configurations v_1 and v_2 of $\mathcal{U}$ s.t. $v_1 \neq v_2$ and $g(v_1) = g(v_2) = v^-$ where v^- is a configuration of $\mathcal{U}^-$. Note that since $v_1 \neq v_2$, then $\exists e_1 \in v_1, e_2 \in v_2.\ e_1 \neq e_2$ & $\theta(e_1) = \theta(e_2)$,

which implies for $b_1 = e_1^\bullet$ and $b_2 = e_2^\bullet$, $b_1 \neq b_2$ & $\alpha(b_1) = \alpha(b_2)$ (by rule 2 of definition 3.2.32 of prefix and the fact that in $\mathcal{U}$, $|{}^\bullet b| \leq 1$). Thus for $b_1 = (\{e_1\}, p)$ and $b_2 = (\{e_2\}, p)$ let $b = (X, p) = \alpha(b_1) = \alpha(b_2)$. Clearly, $\theta(e_1) \in X$ and $\theta(e_2) \in X$, however, all the events in X are pairwise in conflict, thus, $\theta(e_1)\#^-\theta(e_2)$. However, that contradicts v^- being a configuration as such v^- cannot be consistent.

Since g applied to configurations of $\mathcal{U}$ yields configurations of $\mathcal{U}^-$ in an injective manner, consider the inverse of g, denoted by $g_{v^-}^{-1} = (\theta', \alpha')$, where:

$$\theta'(e', e) \Leftrightarrow \theta(e, e') \;\&\; \alpha.\, {}^\bullet e \in v^-$$

$$\alpha'(b', b) \Leftrightarrow \alpha(b, b') \;\&\; \theta.\, {}^\bullet b \in v^-$$

Note that g^{-1} is defined for any configuration v^- as g applied to an unfolding is total and onto (proposition 3.2.30). Then it is easy to verify that $g_{v^-}^{-1}$ is a morphism and that $g_{v^-}^{-1}$ yields a configuration in $\mathcal{U}$, thus, a bijection is constructed.

□

3.3 Complete Finite Prefix

Apart from defining the semantics of Petri nets, unfoldings of Petri nets are extensively used for model checking of certain properties for Petri nets, such as detection of deadlocks, etc. For the most common of such approaches, the notion of a complete finite prefix is crucial, which was first introduced in [51], improved in [22] and further formalised in [39].

In this section we briefly define the complete finite prefix for unfolding$^-$. Similar to merged processes, unfolding$^-$ sufficiently resemble the usual unfoldings with respect to verification. Therefore, the most common model checking approaches based on unfoldings can be adjusted and applied to unfolding$^-$.

We start by recalling some of the relevant definitions which are mostly taken or inspired from [22, 36].

Definition 3.3.1. Given a finite configuration v of an unfolding or unfolding$^-$, we define the corresponding marking of v as follows

$$Mark(v) = (\bigcup_{e \in v} e^\bullet) \setminus (\bigcup_{e \in v} {}^\bullet e)$$

Definition 3.3.2. For an event e_0 of an unfolding $\mathcal{U} = (B_0, E_0, F_0, M_0)$, let $[e_0] =_{\text{def}} \{x_0 \in B_0 \cup E_0 \mid x_0 F_0^* e_0\}$. For an event e_1 of an unfolding$^-$ $\mathcal{U}^- = (B_1, E_1, F_1, M_1)$, let $[e_1]_t =_{\text{def}} t \cup \{e_1\}$ where $t \in {}^*e_1$.

Lemma 3.3.3. For an event e_0 of an unfolding $\mathcal{U}$, $[e_0]$ is a configuration of $\mathcal{U}$ and for an event e_1 of an unfolding$^-$ $\mathcal{U}^-$ and $t \in {}^*e_1$, $[e_1]_t$ is a configuration of $\mathcal{U}$.

Proof. The lemma follows immediately from the definitions of configurations and history of events in occurrence$^-$net. □

The notions of a partial order relation and cut-off events, fundamental to defining a complete prefix, are defined below.

Definition 3.3.4. A partial order $\prec$ on the finite configurations of the unfolding of a Petri net is an *adequate order* if is well-founded, refines $\subset$ and is preserved by finite extensions.

Definition 3.3.5. Given a prefix ω of the unfolding $\mathcal{U}$, an event e of the prefix is a *cut-off* event with respect to an adequate order $\prec$ iff ω contains an event e' such that $Mark([e]) = Mark([e'])$ and $[e'] \prec [e]$. We denote the set of cut-off events by E_{cut}.

Definition 3.3.6. A prefix ω of $\mathcal{U}$ is *marking-complete* iff for every reachable marking M of $\mathcal{U}$, there exists a configuration v of ω such that $v \cap E_{cut} = \emptyset$ and $Mark(v) = M$.

Definition 3.3.7. A marking-complete prefix ω of $\mathcal{U}$ is *complete*, if it preserves the firings, i.e. if for each configuration v of ω such that $v \cap E_{cut} = \emptyset$ and for all events e of $\mathcal{U}$ such that $e \notin v$ and $v' = v \cup \{e\}$ is a configuration of $\mathcal{U}$, v' is a configuration of ω.

The new definition of cut-off events for unfolding$^-$s is as follows.

Definition 3.3.8. Given a prefix ω^- of the unfolding $\mathcal{U}^-$, an event e of the prefix is a *cut-off* event with respect to an adequate order $\prec$ iff for all $t \in {}^*\{e\}$, ω^- contains an event e' such that $Mark([e]_t) = Mark([e']_{t'})$ and $[e']_t \prec [e]_{t'}$ for some $t' \in {}^*\{e'\}$.

Given any algorithm for computing an unfolding$^-$of a safe Petri net with finitely many reachable markings, which adds events and conditions in a procedural manner, we define an algorithm for computing a complete finite prefix the unfolding$^-$. Our algorithm is based on that of [36], given for traditional unfoldings. More concretely, we have adjusted the condition for which an event is considered for being added to the prefix (algorithm 1, line 7). In our algorithm, let *P.E.* denote *possible extensions* representing

the events that can be added in the next stage by the algorithm which computes an unfolding^{-}.

Algorithm 1 Complete Finite Prefix

▷ Given a Petri net $\mathcal{G}$ (with finitely many initial and reachable markings), computes ω^{-}, the complete finite prefix of $\mathcal{U}^{-}(\mathcal{G})$, where P.E. are the possible extensions

$\omega^{-} \leftarrow M_0$

$P.E \leftarrow$possible extensions of ω^{-}

$E_{cut} \leftarrow \emptyset$

while $P.E \neq \emptyset$ **do**

  choose $e \in \min^{\prec} P.E$

  if $\exists t \in {}^{*}\{e\}.\ [e]_t \cap E_{cut} = \emptyset$ **then**

    $\omega^{-} \leftarrow \omega^{-} \cup \{e\}$

    $P.E \leftarrow$possible exntensions of ω^{-}

    if e is a cut-off event of ω^{-} **then**

      $E_{cut} \leftarrow E_{cut} \cup \{e\}$

    end if

  else

    $P.E \leftarrow P.E \setminus \{e\}$

  end if

end while

$\omega^{-} \leftarrow \omega^{-} \cup E_{cut}$

The following results capture some properties of complete finite prefix for unfolding$^-$ and prove the correctness of the above algorithm.

Lemma 3.3.9. If an event e' is a cut-off event in $\mathcal{U}^-$ of a net, then $\forall e \in \mathcal{U}.\ g(e) = e' \Rightarrow$ e is a cut-off event, where $g = (\alpha, \theta) : \mathcal{U} \to \mathcal{U}^-$ is the folding morphism as defined in theorem 3.2.29.

Proof. Follows immediately from the definition of cut-off events for unfolding$^-$ and the fact that the morphism g preserves the structure, and as such the history of the events of unfolding are mapped. □

Proposition 3.3.10. The algorithm in [36] (§2.7.1) terminates for the new cut-off definition (definition 3.3.8).

Proof. As there is a bijection among the configuration of an unfolding and unfolding$^-$ of a net, for any event e_i of an unfolding of a net, there is an event e in its unfolding$^-$ and a history $t \in {}^*\{e\}$ such that $[e_i]$ and $[e]_t$ are mapped together. Then the event e is a cut-off point of $\mathcal{U}^-$ as long as all such e_i of the $\mathcal{U}$ are cut-offs. Now suppose there is such an e_i which is not a cut-off in the $\mathcal{U}$. Either $[e_i]$ can enable some event e_i' such that $e_i \leq e_i'$ and e_i' is a cut-off or it never yields a cut-off event which means the path following it will eventually stop at e_d (because there are finitely many markings, so either some markings should be repeated or it should stop after finitely many steps).

In the former case, e_i' will be mapped to an event e' enabled by e and the algorithm stops in this path as e' is now a cut-off (recall that the extensions of a cut-off event are cut-offs). In the latter case, e_d is mapped to some event e_d'. If e_d' does not enable any event, then the algorithm stops in this path, correctly showing e_d' as a non-cut-off event. If e_d' does enable other events, they can be mapped to from other events of e_i and as such they are cut-offs so the algorithm stops. □

Proposition 3.3.11. Given ω the complete prefix of an unfolding $\mathcal{U}$, $g(\omega)$ is a subnet of ω^-, where ω^- is a complete finite prefix of the unfolding$^-$ $\mathcal{U}^-$ corresponding to $\mathcal{U}$ and $g = (\theta, \alpha)$ is the folding morphism from $\mathcal{U} \to \mathcal{U}^-$ as defined in theorem 3.2.29.

Proof. Suppose $e \in \omega$, then either e is a cut-off event or not. If it is not a cut-off event then $g(e)$ cannot be a cut-off either, and therefore, is added to the prefix by the algorithm. If e is a cut-off event, then either $g(e)$ is also a cut-off or not. In the former

case $g(e)$ is added to the prefix by the last line of the algorithm and in the latter case it is added as usual. □

Proposition 3.3.12. The prefix constructed by the algorithm 1 is complete.

Proof. Let ω^- denote the prefix. Then from proposition 3.3.11 it follows that ω^- is marking-complete. To show that it is also complete, let v be a configuration of ω^- such that $v \cap E_{cut} = \emptyset$. Now suppose $\exists e \in \mathcal{U}^-.\ e \notin v\ \&\ v \cup \{e\}$ is a configuration of $\mathcal{U}^-$. Then e is a possible extension of v which is either a cut-off event or not. If it is, then it is added to ω^- by the last line of the algorithm and if not, it is added to ω^- as a non-cut-off event in the possible extensions of v. Therefore, ω^- preserves firings. □

3.4 An Algorithm for Computing Compact Unfoldings

The definition 3.2.19 of compact unfoldings implies a non-deterministic approach for finding unfolding$^-$s of a safe Petri net. In this section we present a deterministic algorithm for computing a compact unfolding of a safe Petri net. The algorithm we provide starts with the initial markings of the net and gradually adds conditions and events related to the existing ones by the flow relation.

Unfolding into occurrence$^-$nets, we are allowing backwards conflict and as per the definition of compact unfoldings, our main goal is to maximise grouping of events in conflict which can fire to a condition. In order to achieve this, before creating a condition corresponding to a place in the original net, we make sure that all the events corresponding to the transitions firing to that place are created first. In other words, we *wait* for all such events to be created.

Of course it may happen that a certain transition is never enabled in a net or there may be deadlocks, when there is a cyclic dependency between the waiting conditions; that is when two conditions are waiting for events which can only be created after those conditions are added.

Therefore, in our algorithm, we first add conditions which are *ready*, i.e. conditions for which all the possible preceding events are present. As it may be the case that there are infinitely many ready conditions to be added, we need to stop adding such conditions (and the in between events) at some point, e.g. at the cut-off points, to check and resolve the waiting conditions. Therefore, we assume that the original Petri

net has finitely many reachable markings. Now if the waiting conditions are waiting for an event and the preceding conditions for that event are not waiting to be enabled, then that event is impossible, therefore, we will not wait for it any longer. Otherwise, once all such conditions are removed from the waiting list, there must be a cyclic dependency between the remaining conditions, in which case we can define a function (deterministic or otherwise) to choose one of them.

The following pseudo algorithm gives the high level view of the algorithm. Later in this section we also present an example in order to further clarify the algorithm.

Pseudo - Algorithm

Algorithm 2 Pseudo - $Unfold^{-}(\mathcal{G})$

Calculate initial conditions and events of depth 1
Add conditions and events as follows:
repeat
  $P_{next} = \emptyset$ ▷ Next round of conditions
  repeat
    $P_{ready} =$ Enabled conditions not waiting for any other transitions
    Add conditions corresponding to P_{ready}
    Add $E_{next} =$ non-cutoff events newly enabled
  until $E_{next} = \emptyset$
  $P_{wait} = P_{next}$ ▷ stating that P_{next} only has waiting conditions now
  $W =$ waiting conditions tagged with transitions they await
  $P_{resolved} = Resolve(W)$ ▷ Resolve which waiting places should be added
  Add conditions corresponding to $P_{resolved}$
  $E_{next} =$ events newly enabled
  Optional: Remove cut-off events from E_{next}
  Add E_{next}
until $E_{next} = \emptyset$

The complete algorithm for computing an unfolding$^-$ of a safe Petri (with finitely many reachable markings) is given below.

Algorithm 3 *Unfold*$^-(\mathcal{G})$

$W = \emptyset$ ▷ Waiting list
$M_0 = B = \{(\emptyset, p) \mid p \in M\}$ ▷ Conditions of the unfolding$^-$
$E = E_{next} = Next\_Events(B, B)$
for $(A,t) \in E$, let $\eta^-(A,t) = t$ and for $(X,p) \in B$, let $\beta^-(X,p) = p$
repeat
  $P_{next} = \emptyset$
  repeat
    $P_{next} = P_{next} \cup \{p \in P \mid \eta^-(e) \in {}^\bullet p \text{ where } e \in E_{next}\}$ ▷ Enabled places
      ▷ Places ready to be added (places not waiting for any other transition)
    $P_{ready} = \{p \in P_{next} \mid \forall t \in {}^\bullet p.\ \exists e \in E.\ \eta^-(e) = t \ \&\ \nexists b \in B.\ eFb \ \&\ \beta^-(b) = p\}$
    $P_{next} = P_{next} \setminus P_{ready}$
    $B = B \cup Next\_Conditions(P_{ready}, E, B)$
    $E_{next} = Next\_Events(B, B_n) \setminus Cut\text{-}offs(E, B, E_{next})$
    $E = E \cup E_{next}$
  until $E_{next} = \emptyset$
  $P_{wait} = P_{next}$ ▷ emphasising that P_{next} only has waiting places now
    ▷ Tag waiting places with transitions they await
  $W = W \cup \{(p,T) \mid p \in P_{wait} \ \&\ T = \{t \in {}^\bullet p \mid \nexists e \in E.\ (\eta^-(e) = t \ \&\ \nexists b \in B.\ eFb \ \&\ \beta^-(b) = p)\}\}$
  $P_{resolved} = Resolve(W)$ ▷ Resolve which waiting places should be added
  $B = B \cup Next\_Conditions(P_{resolved}, E, B)$
  $E_{next} = Next\_Events(B, B_n)$
  $E = E \cup E_{next}$
until $E_{next} = \emptyset$
$eF(X,p) \Leftrightarrow e \in X$
$bF(A,t) \Leftrightarrow b \in A$

Algorithm 4 *Next_Events*(B, B_n)

▷ Find new events (corresponding to new places in B_n)
return $\{(A,t) \mid A \subseteq B \ \&\ A \cap B_n \neq \emptyset \ \&\ t \in T \ \&\ co^-\ A \ \&\ \beta^-.A = {}^\bullet t\}$

Algorithm 5 *Next_Conditions*(P,E,B)

$list = \emptyset$

for all $p \in P$ **do**

$E_p = \{e \in E \mid \eta^-(e) \in {}^\bullet p \;\&\; \nexists b \in B.(\beta(b) = p \;\&\; eFb\,)\}$

$B_p = (E_p, p)$

$list = list \cup \{B_p\}$

end for

return *list*

Algorithm 6 *Resolve*(W)

$list = \emptyset$

for all $w = (p,T) \in W$ **do**

▷ Check if w depends on something in waiting list

if ¬*Depends(w,W)* **then**

$list = list \cup \{p\}$ ▷ If not, it is ready to be added

$W = W \setminus \{w\}$

end if

end for

if $list = \emptyset \;\&\; W \neq \emptyset$ **then**

▷ There is a cyclic dependency between places

$list = \{w\}$ where w is any waiting place in W (which can be determinised by any deterministic function of choice)

end if

return *list*

Algorithm 7 *Depends*(w,W)

$D_w = \emptyset$

▷ where $w = (p,T)$

for $t \in T$ **do**

if $\exists (p',T') \in W.\; p'Ft$(i.e. there is an acylic path from p' to t) **then**

return *true*

end if

end for

return *false*

The following proposition proves the correctness of the algorithm.

Proposition 3.4.1. Given a safe Petri net $\mathcal{G}$, algorithm 3 computes an unfolding$^-$of $\mathcal{G}$.

Proof. Let $\mathcal{N}$ be the net obtained by the algorithm. First observe that $f = (\eta^-, \beta^-)$ forms a folding $\mathcal{N} \to \mathcal{G}$, which can be easily verified. It then follows that in sub-algorithm 5, $\forall e, e' \in E_p.\ e\#^- e'$, as otherwise $B_p = (E_p, p)$ can have a marking bigger than one, which is not possible since $\mathcal{G}$ is safe and markings are preserved by morphisms (theorem 2.4.8). Thus, it is clear that $\mathcal{N}$ is an occurrence$^-$net. By theorem 3.2.28, all we need to show is that $\mathcal{N}$ is compact.

Now consider two conditions b, b' in $\mathcal{N}$ where $b = (X, p)$ & $b' = (X', p)$. Note that if they can be merged then $\forall e, e' \in X \cup X'.\ e\#^- e'$. However, if $\exists e' \in X'.\forall e \in X.e\#^- e'$ then based on the algorithm, e' should have been included in X, unless there was a cyclic dependency between b and b'. Now suppose condition b is chosen to be added first. Since condition b' was dependent on b and morphisms preserve the flow relation, then once condition b' is added, we have bF^*b'. Therefore, merging b and b' does not preserve the occurrence$^-$net as it would introduce a cycle. Moreover, since bF^*b', then it is no longer the case that $\forall e, e' \in X \cup X'.\ e\#^- e'$. □

Example 3.4.2. In this example, we describe how the algorithm 3 can be applied to $\mathcal{G}_3$ to construct $\mathcal{U}_3^-$ (example 3.2.22).

At line 5 of the algorithm we have:
$W = \emptyset$
$M_0 = B = \{(\emptyset, p_1) = c_1, (\emptyset, p_2) = c_2, (\emptyset, p_3) = c_3\}$
$E = \{(\{c_1\}, t_1) = e_1, (\{c_2\}, t_2) = e_2\}$

In the first round in inner loop (lines 8 to 14) we have:
$P_{next} = \{p_4, p_5\}, P_{ready} = \{\}$
$B = B \cup \{\}, E_{next} = \{\}, E = E \cup \{\}$

After the first run of lines 16 to 18 we have:
$P_{wait} = \{p_4, p_5\}\ W = \{(p_4, \{t_4\}) = w_1, (p_5, \{t_3\}) = w_2\}$

Running $Resolve(W)$, we see that w_1 depends on w_2 and w_2 depends on w_1. Assuming

w_1 is chosen, we get: $P_{resolved} = \{w_1\}$. Going through lines 19 to 22 we have:
$B = B \cup (\{e_1\}, p_4)$
$E_{next} = \{(\{c_1, c_4\}, t_3) = e_3\}, E = E \cup E_{next}$

The rest of the conditions and places are added by running through the inner loop.

3.4.1 Complete Finite Prefix of Algorithm 3

As mentioned previously, algorithm 1 can be adjusted to fit any algorithm which calculates unfolding$^-$s of a net by adding events and conditions in a procedural manner. Here we describe how algorithm 1 can be used in conjunction with algorithm 3.

Observing that in algorithm 3 new events are added in two locations, namely, lines 14 and 22, the conjuncted algorithm for computing a complete finite prefix is as follows.

Algorithm 8 *Complete Finite Prefix for algorithm 3*

$W = \emptyset$ ▷ Waiting list

$M_0 = B = \{(\emptyset, p) \mid p \in M\}$ ▷ Initial Marking and Conditions of the unfolding$^-$

$E = E_{next} = Next\_Events(\emptyset, B, B)$

$E_{cut} = \emptyset$

repeat

  $P_{next} = \emptyset$

  repeat

    $P_{next} = \ldots$

    $\vdots$

    $E_{next} = Next\_Events(B, B_n)$

    while $E_{next} \neq \emptyset$ **do**

      select $e \in \min^{\prec} E_{next}$

      if $\nexists h \in {}^*e.[e]_h \cap E_{cut} = \emptyset$ **then**

        $E_{next} = E_{next} \setminus \{e\}$

      else

        if e is a cut-off event of (B, E, F, M_0) **then**

          $E_{cut} = E_{cut} \cup \{e\}$

        end if

      end if

    end while

    $E = E \cup E_{next}$

  until $E_{next} = \emptyset$

  $P_{wait} = \ldots$

  $\vdots$

  $E_{next} = Next\_Events(B, B_n)$

  while $E_{next} \neq \emptyset$ **do**

    select $e \in \min^{\prec} E_{next}$

    if $\nexists h \in {}^*e.[e]_h \cap E_{cut} = \emptyset$ **then**

      $E_{next} = E_{next} \setminus \{e\}$

    else

      if e is a cut-off event of (B, E, F, M_0) **then**

        $E_{cut} = E_{cut} \cup \{e\}$

      end if

    end if

  end while

  $E = E \cup E_{next}$

until $E_{next} = \emptyset$

3.5 Petri nets to (Stable) Event Structures

Given an occurrence net $O = (B_0, E_0, F_0, \mathbb{M}_0)$, one can derive its corresponding event structure $\mathcal{E}_0(O) = (E_0, F_0^* \upharpoonright E_0, \#)$. For an occurrence$^-$ net $O^- = (B_1, E_1, F_1, \mathbb{M}_1)$ deriving the corresponding stable event structure $\mathcal{E}_1 = (E_1, \vdash, Con)$ is similar.

The consistency relation corresponds well to that defined for occurrence$^-$ nets, thus, we define:

$$\forall X \subseteq E_1.\ X \in Con \Leftrightarrow_{def} X \text{ is consistent.}$$

Then the enabling relation $\vdash$ can be defined as follows for each event e in E_1.

$$X \vdash_{min} e \Leftrightarrow_{def} X = T \cap E_0\ s.t.\ T \in {}^*\{e\}$$
$$X \vdash e \Leftrightarrow X \cup \{e\} \in Con\ \&\ \exists X' \subseteq X.\ X' \vdash_{min} e$$

It is straightforward to verify that the event structure defined above is indeed a stable event structure.

Example 3.5.1. The corresponding stable event structure of unfolding$^-$ $\mathcal{U}_3^- = (E, \vdash, Con)$ of $\mathcal{G}_3$ in example 3.2.22 is as below.

$E = \{e_1, e_2, e_3, e_4, e_5\}$
$Con = \{X \subseteq E \mid e_2 \in X \Rightarrow e_1 \notin X\ \&\ e_3 \notin X\}$
$\{\} \vdash_{min} e_1, \{\} \vdash_{min} e_2$
$\{e_1\} \vdash_{min} e_3$
$\{e_1, e_3\} \vdash_{min} e_4, \{e_2\} \vdash_{min} e_4$
$\{e_2, e_4\} \vdash_{min} e_5$

3.6 Other approaches

As mentioned before, there are different approaches for defining the semantics of a net and for forming a suitable structure for its model checking. Most of such approaches emerge from the notion of unfolding and some of the well-known ones are Trellis Processes [23], Merged Processes [37, 40] and R_i Unfoldings [58].

The key point to observe is that the root of the complexity of the problem of unfolding or unravelling a net remains the same. In other words, in any approach the behaviour of the net in terms of the relation between configurations (or markings) has to be decoded from the original net. Therefore, in any approach the core issues to be addressed are the same, arising from the nature of the problem. Nevertheless, each approach can achieve different objectives while addressing these core issues.

As mentioned before, Trellis processes and R_i unfoldings are both defined for safe nets. Merged processes can handle unsafeness, by allowing unsafeness in their final outcome. In this thesis, unfolding$^-$s are defined for safe Petri nets, however, our approach can be extended to handle unsafeness as well.

Among all these approaches, the merged processes are elegantly defined as the most compact one. However, this is at the cost of the configurations being more complex to define and deal with. They are also not necessarily acyclic and which makes verification approaches based on marking equations less straightforward. Merged processes were initially defined through merging the traditional unfoldings. Later an algorithm for building the merged processes of safe Petri nets was introduced [40], although there are still no algorithms for directly building the merged processes of the unsafe nets.

Trellis processes focus on having the factorisation property. They unfold time but not conflicts. Due to using a height function, Trellis processes are perhaps more flexible than the other approaches. Though, this requires complementing every net to a multi-clock net and the outcome may be different from the original Trellis net and contain non-executable cycles.

In [58] unfoldings are introduced that forget the history step by step. The R_i-unfolding is more forgetful than the unfoldings, and less forgetful than unfolding$^-$s. They are also defined for safe nets only.

To sum up, in comparison to the above approaches, unfolding$^-$s are defined for safe Petri nets, although they can be extended to deal with unsafeness directly. Moreover, they unfold cycles while keeping the conflicts as folded as possible. Finally, their complete finite prefixes can be used for verification purposes.

3.7 Conclusion

Traditional unfoldings of Petri nets, while being widely known and used as semantics of Petri nets, have the disadvantage of growing exponentially with the choices of a net as they create an event for every single possible occurrence of a transition in the original net. This also implies that little can be inferred immediately about the markings of the original net, without further computations.

In this chapter we presented compact unfoldings of safe Petri nets. Similar to the traditional unfoldings, they capture all the reachable markings of a general Petri nets, with the difference that they do not create a new event for every single occurrence of a transition, but rather group the conflicting events together. In other words, conflicts remain folded as far as possible. This reduces the size of the new unfoldings significantly compared to their corresponding traditional unfolding.

In addition to describing the behaviour of nets, traditional unfoldings of Petri nets are used extensively for verification of reachability-like problems. The majority of the existing algorithms use a finite prefix of the unfoldings, namely, the complete finite prefix of a Petri net. The notion of a complete finite prefix, also presented in this chapter, can be directly defined for the compact unfoldings and is presented, making them suitable candidates for a large body of existing model checking algorithms. We have also presented a deterministic algorithms for constructing compact unfoldings and their complete finite prefix in a more direct manner.

Chapter 4

Probabilistic Event Structures

As with the non-probabilistic systems, the two approaches of interleaving and partial-order can be taken when considering probabilistic concurrent systems. While probabilistic systems with the interleaving approach have been studied intensively (examples include [73, 18, 11, 27] among many more), applying the true concurrency approach has gained interest over the last decade or so. Similar to the non-probabilistic case where the true concurrency semantics does not distinguish between the behaviours which are equivalent up to reordering of concurrent events, in the probabilistic framework such behaviours should be assigned the same probability.

Taking event structures as simple yet capable models of true concurrency, the most prominent works in this area include randomised non-sequential processes of Volzer [74] and probabilistic event structures of [34, 72, 1, 77]. This thesis focuses on probabilistic event structures as defined in [1] and in this chapter we introduce and justify this choice.

To our best knowledge, the first probabilistic model for event structures in the literature was given in [34], where the author defines probabilistic extended bundle event structures. Similar to stable event structures, bundle event structures allow different possible causes for an event, only one of which can be the cause in a run. The concept of choice is captured by groups of events which are mutually in conflict and enabled at the same time, called *clusters*. Clusters can be determined statically and maintain the concept of choice internal to the system.

In [72] a domain theoretic view, closely related to the probabilistic powerdomains of [32, 68], was taken by Varacca, Volzer and Winskel [72], where continuous valuations

are defined on the domain of congurations. The notion of *non-leaking valuations* are defined and constructed on confusion-free event structures and valuations with independence are described. It turns out that for confusion-free event structures, the so called non-leaking valuations with independence coincide with distributed probabilities as defined by [1].

Randomised Petri nets along with their corresponding probabilistic branching processes defined in [74] focus on free-choice Petri nets only, namely, Petri nets in which the choice between two transitions is not influenced by the behavior of the rest of the system. Therefore, confusion is not dealt with. Finally, the most recent probabilistic event structures defined by Winskel [77] extend existing notions of probabilistic event structures in order to make them suitable for dealing with certain interactions between strategies.

Since we are more interested in probabilistic event structures arising from Petri nets with probabilistic behaviours (rather than being concerned with strategies), and all other such approaches do not address the notion of confusion, in this thesis we work with probabilistic event structures proposed by [1].

As mentioned above, the main objective of a probabilistic concurrent system with partial order semantics consists of finding units of choice in such a way that concurrent units are probabilistically independent. Abbes and Benveniste in [1] define distributed probabilities taking *branching cells* as units of choice and show that in probabilistic event structures not only does concurrency match probabilistic independence, but also that this cannot be achieved at a grain finer than that of branching cells. Furthermore, they show how finite configurations can be decomposed into branching cells in a dynamic way, where maximal configurations of the branching cells enforce all the conflicts within the cell to be resolved. *Local probabilities* are assigned to each branching cell and it is shown that this can be extended to a *limiting probability* measure on the space of maximal configurations. The only constraint required is that of *local finiteness* which can be viewed as bounded confusion and which is defined later in this chapter.

In this chapter, we first define *net-driven* stable event structures, by describing characteristics of the stable event structures arising from safe Petri nets (i.e. event structures corresponding to the compact unfoldings, defined previously in chapter 3). We then present the definition of probabilistic event structures and the related concepts as defined by [1]. In order to define probabilistic stable event structures in a similar manner,

we need to confine the stable event structures, and consequently the original Petri net. Thus, we define a certain class of event structures, namely, *jump-free* (stable) event structures and show that probabilistic jump-free stable event structures can be defined analogously to probabilistic event structures.

4.1 Net-driven (Stable or Prime) Event Structures

In the previous chapter we introduced compact unfoldings of safe Petri nets and saw how they could be translated into stable event structures. In this section we define certain characteristics of the stable event structures derived from compact unfoldings. We start by defining the notion of morphism between stable event structures.

Definition 4.1.1. [76] Let $\mathcal{E}_0 = (E_0, Con_0, \vdash_0)$ and $\mathcal{E}_1 = (E_1, Con_1, \vdash_1)$ be two stable event structures. A *morphism* from $\mathcal{E}_0$ to $\mathcal{E}_1$ is a partial function $\theta : E_0 \to E_1$ on events satisfying:

1. $X \in Con_0 \Rightarrow \theta.X \in Con_1$
2. $\{e, e'\} \in Con_0 \;\&\; \theta(e) = \theta(e') \Rightarrow e = e'$
3. $X \vdash_0 e \;\&\; \theta(e)$ is defined $\Rightarrow \theta.X \vdash_1 \theta(e)$

A morphism is *synchronous* if it is a total function.

Lemma 4.1.2. Let $\mathcal{E}_1 = (E, Con, \leq)$ be a prime event structure. There is a evident natural and obvious map I mapping $\mathcal{E}_1$ to a stable event structure $\mathcal{E}_0 = (E, Con, \vdash)$, where $X \vdash_{min} e \Leftrightarrow X = \lceil e \rceil \setminus \{e\}$. While we do not need it, I extends to a functor: a morphism of prime event structures is a partial function on events which satisfies certain conditions, and these conditions also make it a morphism of (stable) event structures [75]. (Note: [75] uses 'event structures' for what he now and we here call 'stable event structures', with a different but equivalent notation.)

Remark 4.1.3. In this chapter we identify a prime event structure $\mathcal{E}_1$ with $I(\mathcal{E}_1)$, when required.

Recall the translation from stable event structures into prime event structures in definition 2.3.20: Given a stable event structure $\mathcal{E}_0 = (E, Con, \vdash)$, let $\Theta(\mathcal{E}_0)$ be the prime event structure $\mathcal{E}_1 = (P, Con_P, \leq)$, with isomorphic domain of configurations, defined as follows.

- $P = \{\lceil e \rceil_x \mid e \in x \in \mathcal{V}(\mathcal{E})\}$.
- $p' \leq p \Leftrightarrow p' \subseteq p$.
- $X \in Con_P \Leftrightarrow X \subseteq_{\text{fin}} P \;\&\; X \uparrow$.

Then Θ extends to a functor: a morphism $\mathcal{E} \to \mathcal{E}'$ given by $\theta : E \to E'$ maps to the morphism given by $\{\lceil e \rceil_x \mid e \in x \in \mathcal{V}(\mathcal{E})\} \mapsto \{\lceil \theta(e) \rceil_{x'} \mid \theta(e) \in x' \in \mathcal{V}(\mathcal{E}')\}$. Moreover, Θ is right adjoint to the inclusion function I defined above [75] (theorem 4.3).

For a stable event structure $\mathcal{E}_0$ we refer to $\Theta(\mathcal{E}_0)$ as its associated prime event structure.

Let SES be the category of stable event structures.

Proposition 4.1.4. Let $\mathcal{E}_0 = (E, Con, \vdash)$ be a stable event structure and $\mathcal{E}_1 = (P, Con_P, \leq) = \Theta(\mathcal{E}_0)$ be its associated prime event structure, which can also be viewed as a stable event structure via the inclusion I. Let $\theta_{\mathcal{E}_0} : \mathcal{E}_1 \to \mathcal{E}_0$ be the (SES) morphism given by $\theta_{\mathcal{E}_0}(p) = e$ for $p = \lceil e \rceil_x \in P$, $e \in E \;\&\; x \in \mathcal{V}(\mathcal{E}_0)$. Then $\theta_{\mathcal{E}_0}$ is a synchronous morphism in SES. Indeed, though we do not use this, $\theta : \mathcal{E}_0 \mapsto \theta_{\mathcal{E}_0}$ is a natural transformation from $I\Theta$ to Id_{SES}, and is the counit of the adjunction between I and Θ from [75] mentioned above.

Proof. The proof follows trivially from the definition of morphism and translation of stable event structures into prime event structures. □

In the definition of stable event structures, the consistency predicate Con is required to satisfy only one condition. Namely, $Y \subseteq X \;\&\; X \in Con \Rightarrow Y \in Con$. This does not necessarily have to fit with the configurations of the stable event structures. Consider the following example.

Example 4.1.5. Let $\mathcal{E} = (E, Con, \vdash)$ be a stable event structure, where $E = \{e_1, e_2, e_3, e_4\}$, $\{e_1\} \vdash e_2, \{e_3\} \vdash e_4$, $\{e_1, e_3\} \notin Con$ and $\{e_2, e_4\} \in Con$. It is easy to see that even though e_2 and e_4 are consistent, they can never appear together in a configuration. Therefore, the consistency predicate is not sensible with respect to the configurations.

We therefore define *sensible* event structures as follows.

Definition 4.1.6. Let $\mathcal{E}$ be a stable or prime event structure with the consistency relation Con. We say $\mathcal{E}$ is *sensible* iff $\forall X \in Con \Leftrightarrow \exists v \in \mathcal{V}(\mathcal{E}).\, X \subseteq v$.

Lemma 4.1.7. Prime event structures are sensible.

Proof. Follows trivially by the definition of prime event structures, in particular property 4 in definition 2.3.15. □

The following lemma further describes the relation between a stable event structure and its associated prime event structure.

Lemma 4.1.8. Let $\mathcal{E}_0$ be a sensible stable event structure and let $\mathcal{E}_1 = \Theta(\mathcal{E})$ be its associated prime event structure. Then we have the following.

$$e \leq_x e' \Leftrightarrow \lceil e \rceil_x \subseteq \lceil e' \rceil_x$$

$$X \in Con \Leftrightarrow \forall v \in \mathcal{V}(\mathcal{E}_0)\ s.t.\ X \subseteq v.\ \{\lceil e \rceil_v \mid e \in X\} \in Con_P$$

Proof. Follows trivially from the relevant definitions. □

So far we have described how the consistency predicate in sensible event structures relates to the configurations of that structure. We now define the notions of conflict and immediate conflict, and show later that they are the source of inconsistency in stable event structures driven from compact unfoldings.

Definition 4.1.9. Two events e and e' of a stable event structure are in *conflict* under a finite configuration v, represented by $e\#_v e'$ iff $\{e\} \cup \{e'\} \cup v \notin Con$. Then two events are in conflict, represented by $e\#e'$, iff $\forall v \in \mathcal{V}.\ e\#_v e'$.

Note that this is in line with the definition of conflict for unfolding$^-$s. The notion of immediate conflicts in the context of a configuration v is then defined as follows.

Definition 4.1.10. *Immediate conflict* between two events e and e' of a stable event structure under the configuration v is defined as follows.

$$e\#_{\mu,v} e' \Leftrightarrow_{def} v \vdash e\ \&\ v \vdash e'\ \&\ \#_v \cap (\lceil e \rceil_v \times \lceil e' \rceil_v) = \{(e, e')\}.$$

We define $e\#_\mu e'$ iff $\forall v, v' \in \mathcal{V}.\ v \vdash e\ \&\ v' \vdash e' \Rightarrow \exists v'' \subseteq v \cup v'.\ e\#_{\mu,v''} e'$.

Definition 4.1.11. In the following few results, for a set $X \subseteq_{\text{fin}} E$, let ${}_*X$ be the set of the sets consisting of exactly one history $\lceil e \rceil_v$ for each event e and configuration v depending on e.

In the following definition we specify a special class of stable event structures, namely, net-driven stable event structures. They are called net-driven stable event structures,

because we will show later that they describe the stable event structures arising from safe Petri nets.

Definition 4.1.12. A stable event structure $\mathcal{E} = (E, Con, \vdash)$ is called *net-driven* iff it satisfies the following.

1. $\mathcal{E}$ is sensible.
2. $\forall X \subseteq_{fin} E.\, X \notin Con \Rightarrow \forall T \in {}_*X.\, \exists e_1, e_2 \in \bigcup T.\, e_1 \#_\mu e_2$
3. $\forall e, e' \in E, v \in \mathcal{V}.\, e \#_{\mu,v} e' \Rightarrow e \# e'$

Note that from 2 and 3 it follows that for net-driven stable event structures, $\forall X \subseteq E.\, X \notin Con \Rightarrow \forall T \in {}_*X.\, \exists e_1, e_2 \in \bigcup T.\, e_1 \# e_2$

As mentioned before, the first characteristic describes that the consistency predicate is in line with configurations and the second one implies that the source of inconsistency is a conflict in the past . The last constraint describes that an immediate conflict derived from a net persists (this is because the source of the conflict of two events in the net is a common preceding condition).

The notion of conflict for prime event structures is defined as follows.

Definition 4.1.13. Let $\mathcal{E}_1 = (E, Con, \leq)$ be a prime event structure. Define the conflict relation # between two events by

$$e \# e' \Leftrightarrow \{e, e'\} \notin Con$$

In the following theorem we show that for the associated prime event structures of a net-driven stable event structure, the consistency predicate corresponds to a binary conflict relation as defined above.

Theorem 4.1.14. Let $\mathcal{E}_0 = (E, Con, \vdash)$ be a net-driven stable event structure and let $\mathcal{E}_1 = \Theta(\mathcal{E}_0) = (P, Con_P, \vdash)$ be its associated prime event structure. Then,we have:

$$X \in Con_P \Leftrightarrow X \subseteq_{\text{fin}} P \;\&\; \forall p, p' \in X.\, \neg(p \# p')$$

Proof. ($\Rightarrow$) follows from the definition of consistency relation. More precisely, if $\exists p, p' \in X.\, p \# p'$ then it follows that p and p' are not consistent (by definition of #) and therefore, they are not compatible as configurations of $\mathcal{E}_0$. Thus, $X \notin Con_P$ which is a contradiction.

($\Leftarrow$) follows from the definition of net-driven stable event structures. More specifically, suppose by contradiction that there is a finite set of events X s.t. $\forall p, p' \in X.\ \neg(p \# p')$ and $X \notin Con_P$. Let $X = \{p_i \mid p_i = \lceil e_i \rceil_{v_i} \ \&\ v_i \in \mathcal{V}(\mathcal{E}_0)\}$. Since $X \notin Con_P$ this implies that p_i as configurations of $\mathcal{E}_0$ are not compatible, in other words, if we let $\bar{X} = \cup \lceil e_i \rceil_{v_i}$, then $\bar{X} \notin Con$. Now since $\mathcal{E}_0$ is net-driven, then for any $T \in {}_*\bar{X}.\ \exists e_1, e_2 \in \bigcup T.\ e_1 \# e_2$. For such e_1, e_2, suppose $p_1, p_2 \in X.\ p_1 = \lceil e \rceil_{v_1}\ \&\ e_1 \in p_1\ \&\ p_2 = \lceil e' \rceil_{v_2}\ \&\ e_2 \in p_2$. Then p_1 and p_2 are not compatible are configurations of $\mathcal{E}_0$, therefore $\{p_1, p_2\} \notin Con_P$, which contradicts $\forall p, p' \in X.\ \neg(p \# p')$. Therefore, X must be consistent, i.e. $X \in Con_P$.

□

Lemma 4.1.15. Let $\mathcal{E}_1 = (E, Con, \leq)$ be a prime event structure such that there exists a binary relation # such that $e \# e' \Leftrightarrow \{e, e'\} \notin Con$. Then $\mathcal{E}_1$ can also be seen as an event structure $(E, \leq, \#)$. This embedding extends to a functor I' (leaving morphisms $f : E \to E'$ unchanged) from the subcategory of prime event structures generated by binary conflict relation to the category of event structures.

Remark 4.1.16. In this chapter we identify prime event structures whose consistency predicate corresponds to a binary conflict relation with their natural embedding I' into event structures, when required.

From the above theorem it follows that a net-driven stable event structure can be associated to an event structure.

Definition 4.1.17. Given a net-driven stable event structure $\mathcal{E}_0$, we denote by $\Theta'(\mathcal{E}_0) = (E, \leq, \#)$ the event structure $\mathcal{E}_2 = I'(\Theta(\mathcal{E}_0))$, and we refer to $\mathcal{E}_2$ as the associated event structure of $\mathcal{E}_0$.

In the following lemma we show that the $\#_m^-$ relation in occurrence$^-$nets corresponds to the $\#_\mu$ relation for stable event structures.

Lemma 4.1.18. Let $\mathcal{U}^- = (E, B, F, M_0)$ be an unfolding$^-$of a safe Petri net and $\mathcal{E} = (E, Con, \vdash)$ the corresponding stable event structure as defined in section 3.5. If $e_0 \#_m^- e_1$, then for $\mathcal{E}$ we have:

$$\forall T \in {}_*\{e_0, e_1\}.\ \exists e_0', e_1' \in \bigcup T.\ e_0' \#_\mu e_1'$$

Proof. First observe that if $e_0 \#_m^- e_1$ as events of $\mathcal{U}^-$, then $e_0 \# e_1$ as events of $\mathcal{E}$. That is because e_0 and e_1 have a common preceding condition which causes a conflict under any configuration. Clearly, if $e_0 \#_\mu e_1$ then the condition in lemma is satisfied. Now

if $\neg(e_0 \#_\mu e_1)$, since we know that $e_0 \# e_1$, that is $\forall v \in \mathcal{V}(\mathcal{E}).\ e_0 \#_v e_1$, then from the definition of $\#_\mu$ it follows that $\exists v_0, v_1.\ v_0 \vdash e_0$ & $v_1 \vdash e_1$ & $\nexists v \subseteq v_0 \cup v_1.\ e_0 \#_{\mu,v} e_1$. Now since $\forall v \in \mathcal{V}(\mathcal{E}).\ e_0 \#_v e_1$, then it must be that $\nexists v \subseteq v_0 \cup v_1 . v \vdash e_0$ & $v \vdash e_1$ or if such v exists, then $\#_v \cap (\lceil e \rceil_v \times \lceil e' \rceil_v) \neq \{(e,e')\}$. In either of the cases, it follows that $\exists e'_0 \in \lceil e_0 \rceil_{v_0}, e'_1 \in \lceil e_1 \rceil_{v_1}.\ e'_0 \# e'_1$.

Now either $e'_0 \#_\mu e'_1$ or the exact reasoning applies to deduce that $\exists e''_0 \in \lceil e'_0 \rceil_{v_0}, e''_1 \in \lceil e'_1 \rceil_{v_1}.\ e''_0 \# e''_1$. As the events of $\mathcal{E}$ have finite histories, then we must reach a point where $e_0^{(n)} \#_\mu e_1^{(n)}$, thus, the condition in the lemma is satisfied. □

Finally, we show that the stable event structures driven from compact unfoldings are net-driven.

Proposition 4.1.19. Let $\mathcal{U}^- = (E, B, F, M_0)$ be an unfolding$^-$of a safe Petri net and $\mathcal{E} = (E, Con, \vdash)$ the corresponding stable event structure as defined in section 3.5. Then $\mathcal{E}$ is net-driven.

Proof. (1) Follows from the translation in section 3.5. More concretely, $X \in Con \Leftrightarrow X$ (as a set of events in $\mathcal{U}^-$) is consistent, which is true iff X has a history h. It is now very easy to verify that $h \cup X$ is a configuration in $\mathcal{U}^-$ and that $(h \cup X) \cap E$ is a configuration in $\mathcal{E}$.

(2) Suppose $X \subseteq E$ and $X \notin Con$. Then from the definition of Con (section 3.5) it follows that X (as a set of events in $\mathcal{U}^-$) is not consistent. In other words, X has no history and therefore, any set consisting of a history for each element in X, is not $\#^-$-free, i.e. for any such set $\bar{X}$, $\exists e, e' \in \bar{X}.\ e \#^- e'$. Then by lemma 3.2.7 in any history of e, H, and any history of e', H', $\exists e_0 \in H \cup \{e\}, e_1 \in H' \cup \{e'\}.\ e_0 \#_m^- e_1$. Finally, by applying lemma 4.1.18 the proposition is proved.

(3) Follows from the fact that conflicts are first introduced in nets when two events share a common preceding condition. More concretely, observe that if $e \#_{\mu,v} e'$, then $\exists b \in {}^\bullet e \cap {}^\bullet e'$, as events in $\mathcal{U}^-$. Thus, $e \#_m e'$, which implies $\forall v \in \mathcal{V}(\mathcal{E}).e \#_v e'$ as events of $\mathcal{E}$, therefore, $e \# e'$.

□

4.2 Probabilistic Event Structures

In this section we define probabilistic event structures as introduced by [1]. We start by presenting the definitions for the relevant concepts.

Remark 4.2.1. A substantial majority of the definitions in this section are taken from [1].

Definition 4.2.2. A subset $P \subseteq E$ is called a *prefix* of an event structure $\mathcal{E} = (E, \leq, \#)$, if $P = \lceil P \rceil$.

Definition 4.2.3. Let $\mathcal{E} = (E, \leq, \#)$ be an event structure and let F be a subset of E. Then $(F, \leq \restriction F, \# \restriction (F \times F))$ is an event structure (which is a sub-event structure of $\mathcal{E}$).

Remark 4.2.4. In this chapter, we use F to also refer to the sub-event structure induced by F as above, when no confusion arises.

Configurations can then be viewed as conflict-free prefixes, representing the set of events which can occur in a specific run of an event structure and in that way they can capture the global state of the system. As before, the set of configurations of an event structure $\mathcal{E}$ is represented by $\mathcal{V}(\mathcal{E})$ or $\mathcal{V}$ if no confusion arises.

Definition 4.2.5. A configuration v is *maximal* iff $\forall v'.\ v \subseteq v' \Rightarrow v = v'$. We denote the maximal configurations of event structure $\mathcal{E}$ by $\Omega(\mathcal{E})$ or Ω if no confusion arises.

Definition 4.2.6. The *future* of a configuration v of event structure $\mathcal{E}$ is defined as $\mathcal{E}^v =_{\text{def}} (E^v, \leq^v, \#^v)$ where,

- $E^v = \{e \in E \mid \lceil e \rceil \text{ is compatible with } v \text{ and } e \notin v\}$
- $\leq^v = \leq \cap\, E^v \times E^v$
- $\#^v = \# \cap E^v \times E^v$

Definition 4.2.7. Given configurations $u \in \mathcal{V}$ and $v \in \mathcal{V}^u$, the *concatenation* of u and v is defined as $u \oplus v =_{\text{def}} u \cup v$.

Definition 4.2.8. We define the *subtraction* of two configurations $u, v \in \mathcal{V}$ s.t. $v \subseteq u$ as $u \ominus v =_{\text{def}} u \setminus v$

In a probabilistic event structure, the probability reflects the notion of choice which arises whenever a conflict is encountered for the first time. Thus, we consider the

'first-hand' conflicts, i.e. conflicts which are not inherited, as constituents of units of choice. The notion of immediate conflict for event structures is formally defined as follows.

Definition 4.2.9. Events e and e' are in *immediate conflict* iff

$$e\#_\mu e' \Leftrightarrow_{\text{def}} \# \cap (\lceil e \rceil \times \lceil e' \rceil) = \{(e, e')\}.$$

As we shall see later in this chapter, in probabilistic event structures the configurations are decomposed into units of choices, which are probabilistically independent. To define these units of choice, called *branching cells*, we need to define a few other concepts, including *stopping prefixes, stopped configurations* and *recursively stopped configurations*.

Definition 4.2.10. A prefix B is called a *stopping prefix* if it is $\#_\mu$-closed.

Proposition 4.2.11. Given a set $X \subseteq E$ of an event structure, there is a minimal stopping prefix containing X and denoted by X^*.

Proof. We define X^* as the closure of X under the following conditions.

1. $e \in X^* \;\&\; e' \in E \;\&\; e\#_\mu e' \Rightarrow e' \in X^*$
2. $e \in X^* \Rightarrow \lceil e \rceil \subseteq X^*$

□

Definition 4.2.12. A configuration v of $\mathcal{E}$ is called *B-stopped* if v is a maximal configuration of B; a configuration v of $\mathcal{E}$ is called *stopped* if v is a maximal configuration of v^*.

The main idea behind stopping prefixes is that they keep conflicts and thus choices internal. Stopped configurations then refer to the maximal configurations of such prefixes, representing the maximal possible set of events contained in a stopping prefix.

Unfortunately, it can be shown that the class of stopped configurations is not closed under concatenation (see example 4.2.14). This is undesirable since we want to decompose a configuration into probabilistically independent units. Therefore, a more relaxed notion of stopped configurations is introduced, namely, the *recursively stopped configurations* or *R-stopped* configurations.

Definition 4.2.13. A configuration v of event structure $\mathcal{E}$ is *R-stopped* if there is a sequence of configurations $v_0, \ldots, v_{n-1} \subseteq v_n, \ldots$, for $0 \leq n < N \leq \infty$ such that:

- $v_0 = \emptyset$ and $v = \bigcup_{0 \leq n < N} v_n$, and
- $\forall n \geq 0,\ n+1 < N \Rightarrow v_{n+1} \ominus v_n$ is finite and stopped in $\mathcal{E}^{v_n}$.

The sequence is called a *valid decomposition* of v and if $N < \infty$ then v is said to be *finite R-stopped*. Finally, we refer to the set of R-stopped configurations of an event structure $\mathcal{E}$ by $\mathcal{W}(\mathcal{E})$.

R-stopped configurations are in a sense *weaker* than the stopped configurations. They can be decomposed into steps which are finite and stopped, i.e. steps which keep the choices internal while being finite. It can be shown that R-stopped configurations are closed under concatenation. Moreover, it is obvious that finite stopped configurations are R-stopped.

Example 4.2.14. The event structure in figure 4.1 (taken from [1]) clarifies the difference between stopped and R-stopped configurations. Consider configuration $v = \{e_1, e_3\}$. Then v is not stopped (or e_5 should be added) but it is R-stopped as e_3 is finite stopped in $\mathcal{E}^{\{e_1\}}$.

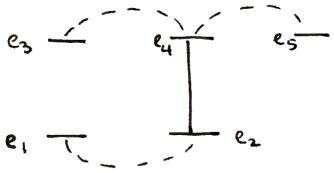

Figure 4.1: An event structure, $\mathcal{E}_1$

Example 4.2.15. In the event structure in figure 4.2, configuration $v = \{e_1, e_3\}$ is not R-stopped as $\mathcal{E}^{\{e_1\}}$ includes e_2, e_3 and e_4, thus e_2 needs to be added to v in order to make it R-stopped .

Therefore, R-stopped configurations serve as good candidates for our purpose. What remains is to find a canonical way of decomposing a configuration. Thus, we define *branching cells* and *coverings* through branching cells. We first define a few more concepts including that of *local finiteness*.

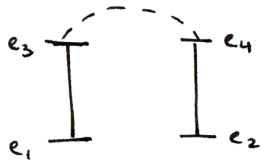

Figure 4.2: An event structure, $\mathcal{E}_2$

Definition 4.2.16. An event structure $\mathcal{E}$ is *pre-regular*, if for every finite configuration v of $\mathcal{E}$, the set of initial events in future of v, i.e. $\{e \in E \mid v \oplus \{e\} \text{ is a configuration}\}$, is finite.

More informally, in pre-regular event structures, for every configuration the next causal step is finite. In other words, at every point, there are finitely many events which are enabled by their causal predecessors. Pre-regular event structures are tightly connected to unfoldings of Petri nets.

Definition 4.2.17. An event structure $\mathcal{E}$ is *locally finite* if for every event $e \in E$, there is a finite stopping prefix of $\mathcal{E}$ containing e.

Locally finite event structures are also connected to unfoldings of Petri nets, although unlike pre-regular event structures, the unfolding of a safe finite Petri net is not necessarily locally finite. However, if an event structure is pre-regular and *confusion-free*, then it is locally finite [1]. Therefore, the unfoldings of confusion-free Petri nets are locally finite. An important property implied by local finiteness is that maximal configurations of a locally finite event structure are R-stopped (theorem 3.12 in [1]). This is desirable as we shall see when we have defined branching cells.

Remark 4.2.18. From this point onwards we assume that all event structures are locally finite, unless specified otherwise.

Definition 4.2.19. An initial stopping prefix is a non-empty stopping prefix for which the only other initial stopping prefix included strictly in it is $\emptyset$.

Note that initial stopping prefixes may also contain events which are not minimal and also it is not necessarily the case that every minimal event is in an initial stopping prefix.

Remark 4.2.20. For a set of configurations $\mathcal{X}$, we denote the subset of its finite con-

figurations by $\overline{X}$.

Definition 4.2.21. A *branching cell* of an event structure $\mathcal{E}$ and configuration v (ranging over finite R-stopped configurations $\overline{\mathcal{W}}(\mathcal{E})$) is an *initial stopping prefix* of $\mathcal{E}^v$. The set of all branching cells of $\mathcal{E}$ is denoted by $\mathcal{C}(\mathcal{E})$ and the set of maximal configurations of a branching cell c is denoted by Ω_c.

Definition 4.2.22. The branching cells which are initial stopping prefixes of $\mathcal{E}^v$ for $v \in \overline{\mathcal{W}}(\mathcal{E})$ are called the branching cells *enabled* by v and their set is denoted by $\delta_{\mathcal{E}}(v)$ of $\delta(v)$ if no confusion arises.

Example 4.2.23. The event structure depicted in figure 4.3 has initial stopping prefixes c_1 and c_2, which do not consists of initial events only, nor do they cover all initial events.

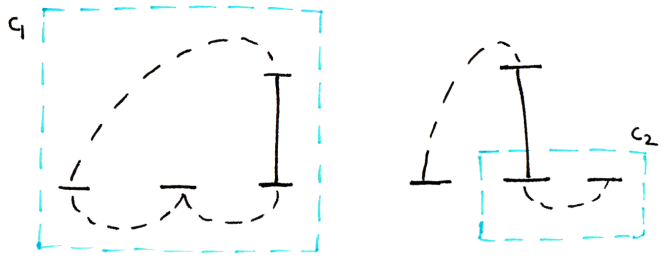

Figure 4.3: An event structure, $\mathcal{E}_3$, and its initial stopping prefixes

The relation between branching cells and R-stopped configurations is closer than appears from the definition. They go hand in hand, describing each other, as can be seen below.

Lemma 4.2.24. [1] Let $v \in \mathcal{W}(\mathcal{E})$ be an R-stopped configuration of some event structure $\mathcal{E}$. Then there is a valid decomposition (v_n) for $0 \leq n < N \leq \infty$ and a sequence of branching cells (c_n) for $0 < n < N \leq \infty$ such that for $0 \leq n < N-1$:

- c_{n+1} is a branching cell enabled by v_n, and
- $v_{n+1} \ominus v_n$ is maximal in c_{n+1}.

Moreover, the c_n are pairwise disjoint. If (v'_n) for $0 \leq n < N'$ and (c'_n) for $0 < n < N'$ is another pair of such sequences, then we have the following equality of sets (where

in particular $N = N'$).

$$\{c_n \mid 0 < n < N\} = \{c'_n \mid 0 < n < N'\}$$

The above lemma enables us to define the covering of R-stopped configurations as follows.

Definition 4.2.25. Let (v_n) for $0 \leq n < N$ and (c_n) for $0 < n < N$ be two sequences of an R-stopped v of an event structure $\mathcal{E}$ as described in lemma 4.2.24. Then the *covering* of v is defined as $\Delta_{\mathcal{E}}(v) = \{c_n \mid 0 < n < N\}$.

It is important to note that while branching cells decomposing a configuration are disjoint, in general, different branching cells of an event structure may overlap. This is because not all of the events that can potentially constitute a branching cell are enabled at every configuration. Thus, configurations determine their corresponding branching cells and in this way we say that decomposition through branching cells is dynamic.

4.2.1 Distributed Probabilities and Probabilistic Event Structures

We are now prepared to define probabilistic event structures. However, the full framework of [1] requires several pages of technical definitions and theorems using sophisticated probability theory. As we are taking the framework as read, and extending it, we will here give only the definitions we need to use explicitly. For the rest of the framework, we sketch the development, and refer the reader to [1] for the details.

Given a space Ω, a *probability measure* on Ω is a function from a suitable collection (technically a σ-algebra) of subsets of Ω to the unit interval $[0,1]$, satisfying the appropriate behaviour for probabilities (the measure of a union of disjoint sets is the sum of the measures).

Now consider an event structure, and let Ω be the space of its maximal configurations (which may be finite or infinite, perhaps uncountably infinite). There is a σ-algebra on Ω comprising the sets $\{\omega \in \Omega : \omega \supseteq v\}$ for each configuration v – that is, each set is a full subtree of the configuration tree rooted at v. Call this set $S(v)$ (the *shadow* of v). The intuition will be that the probability of v should be equal to the sum of the probabilities of all the maximal configurations ω reachable from v – except that

not all such configurations are probabilistically independent, so the usual probability calculations for non-independent events have to be made.

Definition 4.2.26. A *probabilistic event structure* is a pair $(\mathcal{E}, \mathbb{P})$, where $\mathbb{P}$ is a *probability measure* on the space of maximal configurations of $\mathcal{E}$ with the σ-algebra of shadows.

The *likelihood function* on configurations of $\mathcal{E}$ is the function $p(v) = \mathbb{P}(S(v))$.

Equipping each branching cell with a probability for its maximal configurations, we can construct a specific class of probabilistic event structures, where the corresponding probabilities are called *distributed*, in the following manner.

Now a theorem of probability known as Prokhorov's extension theorem is used to allow the abstract definition of probabilistic event structures to be related to a more operational definition where choice probabilities are attached to the possible next events: the probabilities are given locally, instead of in terms of the shadow in the (infinite) future.

The branching cells of the event structure provide the connection between causal independence and probabilistic independence.

Definition 4.2.27. A locally finite event structure $\mathcal{E}$ is *locally randomised* if each branching cell $c \in \mathcal{C}$ is equipped with a *local transition probability* q_c on the set Ω_C of configurations that can be chosen within the cell.

It can be shown that such a locally randomised event structure generates a probabilistic event structure in the previous sense.

Definition 4.2.28. For a locally randomised event structure $(\mathcal{E}, (q_c)_{c \in \mathcal{C}})$, the likelihood function $p : \overline{\mathcal{W}} \to [0,1]$ is defined as:

$$\forall v \in \overline{\mathcal{W}},\ p(v) = \prod_{c \in \Delta(v)} q_c(v \cap c)$$

where $\Delta(v)$ denotes the covering of v in $\mathcal{E}$.

p is indeed the likelihood function of the generated abstract probabilistic event structure.

This likelihood function in turn induces a probability measure $\mathbb{P}_B$ on the (countable) space B of stopping prefixes of $\mathcal{E}$; and the full probability measure $\mathbb{P}$ can be derived from $\mathbb{P}_B$ via a construction called the *distributed product* and the Prokhorov exten-

sion theorem. Hence the measure $\mathbb{P}$ is called the distributed product of the branching probabilities $\{q_c : c \in C\}$.

Finally, it can be shown that the above definition of likelihood function can also be applied to the space of R-stopped configurations of $\mathcal{E}$.

Thus [1] obtains a definition of probabilistic event structures in which local choice probabilities are attached either to branching cells or to R-stopped configurations.

4.3 Probabilistic Jump-free Stable Event Structures

In this section we consider adjoining probabilities to stable event structures. We start by presenting the definition of concepts analogous to those of probabilistic event structures. Our aim is to derive an isomorphism between the events of branching cells of net-driven stable event structures and their associated event structure. It turns out that such isomorphism exists if the stable event structures are *jump-free*, as we shall define later in this section.

Remark 4.3.1. We assume that the stable event structures in this section are net-driven unless stated otherwise.

4.3.1 Branching Cells on Stable Event Structures

We now define branching cells for stable event structures, in a similar manner and show that in general, unlike the branching cells of event structures, they do not form the units of choice.

Definition 4.3.2. A subset $P \subseteq E$ is called a *prefix* of a stable event structure $\mathcal{E}^-$ iff $\forall e \in P.\ \exists X \subseteq P.\ X \vdash e$.

Definition 4.3.3. Let $\mathcal{E}^- = (E, Con, \vdash)$ be a stable event structure and let F be a subset of E. Then $(F, Con \restriction \wp(F), \vdash \restriction (\wp(F) \times F))$ is a stable event structure (which is a sub-event structure of $\mathcal{E}$).

Remark 4.3.4. In this chapter, we use F to also refer to the sub-event structure induced by F as above, when no confusion arises.

Configurations can then be viewed as consistent prefixes and the set of configurations of an event structure $\mathcal{E}^-$ is represented by $\mathcal{V}^-(\mathcal{E}^-)$ or $\mathcal{V}^-$ if no confusion arises. Concepts of compatibility of configuration and maximal configurations are defined as for event structures and we represent the set of maximal configurations of event structure $\mathcal{E}^-$ by $\Omega^-(\mathcal{E}^-)$.

Definition 4.3.5. The *future* of a configuration v of event structure $\mathcal{E}^-$ is defined as $\mathcal{E}^{-v} = (E^v, Con^v, \vdash^v)$ where,

- $E^v = \{e \in E \mid \{e\} \cup v \in Con \text{ and } e \notin v\}$
- $Con^v = Con \restriction E^v$
- $\vdash^v = \vdash \restriction E^v$

Recalling definitions of $\#_{\mu,v}$ (definition 4.1.10), the stopping prefix$^-$ for stable event structures is defined as below.

Definition 4.3.6. A prefix B^- is called a *stopping prefix*$^-$ if it is $\#_{\mu,v}$-closed in the following sense:

$$\forall v \subseteq B^-.\, e \in v \;\&\; \exists e' \in E.\, e \#_{\mu,v} e' \Rightarrow e' \in B$$

The notions of B^--stopped and stopped configurations for stable event structures are similar to those of event structures. However, unlike event structures, given X a subset of events, a canonical stopping prefix including X cannot be derived. This is because in the definition of prefix for stable event structures an event can have different sets of events enabling it. Thus, these notions are defined as follows.

Definition 4.3.7. A configuration v of $\mathcal{E}^-$ is called B^-*-stopped* if v is a maximal configuration of B^-; a configuration v of $\mathcal{E}^-$ is called *stopped* if there is a stopping prefix B^- such that v is B^--stopped.

Stopping prefix$^-$s of a net-driven stable event structures have a close relation with the stopping prefixes of their associated event structure. To expand this further, we first observe the relation between $\#_{\mu,v}$ of a stable event structure and $\#_\mu$ of its corresponding event structure.

Lemma 4.3.8. Consider $\mathcal{E}^-$ and $\mathcal{E} = \Theta'(\mathcal{E}^-)$. Then we have $e \#_{\mu,v} e' \Leftrightarrow \lceil e \rceil_v \#_\mu \lceil e' \rceil_v$

Proof. Follows trivially from the definitions of $\#_{\mu,v}$, $\#_\mu$ and Θ'. □

Using the above lemma we can describe the relation between stopping prefix$^-$ and stopping prefixes as follows.

Proposition 4.3.9. Consider $\mathcal{E}^-$ and $\mathcal{E} = \Theta'(\mathcal{E}^-)$. Then we have: B^- is a stopping prefix$^-$ of $\mathcal{E}^-$ iff $\Theta'(B^-)$ is a stopping prefix of $\mathcal{E}$.

Proof. It is easy to verify that B^- is a prefix$^-$ iff $B = \Theta'(B^-)$ is a prefix as well (follows from the definition of Θ' and θ being a morphism). Also, from lemma 4.3.8 it follows that B^- is $\#_{\mu,\nu}$-closed (for $\nu \subseteq B^-$) iff B is $\#_\mu$-closed. □

Definition 4.3.10. A configuration ν of event structure $\mathcal{E}^-$ is *R-stopped* if there is a non-decreasing sequence of configurations (ν_n) for $0 \leq n < N \leq \infty$ such that:

- $\nu_0 = \emptyset$ and $\nu = \bigcup_{0 \leq n < N} \nu_n$, and
- $\forall n \geq 0,\ n+1 < N \Rightarrow \nu_{n+1} \ominus \nu_n$ is finite stopped in $\mathcal{E}^{-\nu_n}$.

The sequence is called a *valid decomposition* of ν and if $N < \infty$ then ν is said to be *finite R-stopped*. Finally, we refer to the set of *R*-stopped configurations of an event structure $\mathcal{E}^-$ by $\mathcal{W}(\mathcal{E}^-)$.

Definition 4.3.11. A stable event structure $\mathcal{E}$ is *pre-regular*, if for every finite configuration ν of $\mathcal{E}$, the set $\{e \in E \mid \nu \oplus \{e\}\}$ is finite.

Definition 4.3.12. A stable event structure $\mathcal{E}$ is *locally finite* if for every event $e \in E$, there is a finite stopping prefix of $\mathcal{E}$ containing e.

Definition 4.3.13. As before, an *initial stopping prefix* is a non-empty stopping prefix which the only other initial stopping prefix it includes strictly is $\emptyset$.

Definition 4.3.14. A *branching cell* of a stable event structure $\mathcal{E}^-$ and configurations ν ranging over $\overline{\mathcal{W}}(\mathcal{E}^-)$ is an initial stopping prefix of $\mathcal{E}^\nu$. The set of all branching cells of $\mathcal{E}^-$ is denoted by $\mathcal{C}(\mathcal{E}^-)$ and the maximal configurations of a branching cell c is denoted by Ω_c^-.

Definition 4.3.15. The branching cells which are initial stopping prefixes of $\mathcal{E}^{-\nu}$ for $\nu \in \overline{\mathcal{W}}(\mathcal{E}^-)$, are called the branching cells *enabled* by ν and their set is denoted by $\delta^-_{\mathcal{E}^-}(\nu)$ or $\delta^-(\nu)$ if no confusion arises.

4.3.2 Probabilistic Event Structures and Stable Event Structures

Probabilistic event structures as defined by [1] and presented in the previous sections are powerful structures, coping beautifully with the notion of conflicts by viewing them as probabilistically independent units of choices. Therefore, in this section we study if similar structures can be defined for (net-driven) stable event structures. More concretely, as branching cells are at the core of probabilistic event structures, it would be interesting if the branching cells of net-driven stable event structures and their associated event structures were isomorphic. However, that is not always the case as the following example shows.

Example 4.3.16. Consider the stable event structure $\mathcal{E}^-$ in figure 4.4 and its associated event structure $\mathcal{E}$ in figure 4.5, where the dashed curved lines represent immediate conflicts. More concretely, for $\mathcal{E}$, let $e_1\#_\mu e_2, e_2\#_\mu e_3, e_3\#_\mu e_4, e_4\#_\mu e_5, e_5\#_\mu e_7$. The dotted curved line shows immediate conflict under a particular configuration; in this case $e_6\#_{\mu,\{e_1\}}e_7$. For $\mathcal{E}'$ we have $e_1\#_\mu e_2, \ldots e_4\#_\mu e_5, e_5\#_\mu e_7, e_7\#_\mu e_6$.

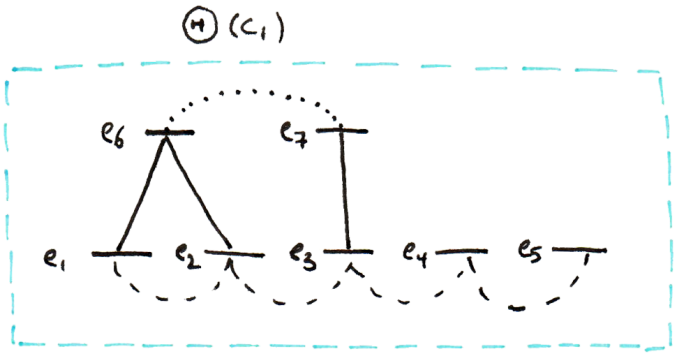

Figure 4.4: A (net-driven) stable event structre $\mathcal{E}^-$

As it can be seen from the figures, $\mathcal{E}$ has two branching cells (c_1 and c_2), while $\theta(\mathcal{E})$ has only one. To expand this, suppose we want to construct the branching cells of $\mathcal{E}'$. Consider the configuration $v_1 = \{e_2, e_4, ...\}$. Having event e_2 implies that events e_1, e_3, e_4, e_5, e_7 and e_6 must be added to the branching cell to comply to $\#_\mu$-closure. Thus, e'_6 is not in this branching cell, but in the next branching cell, consisting of e'_6 only. However, in the stable event structure, both e_6 and e'_6 of $\mathcal{E}'$ are represented by event e_6 of $\mathcal{E}$.Therefore, it is not possible to cover $\mathcal{E}$ in any manner that is consistent with the covering for $\mathcal{E}'$.

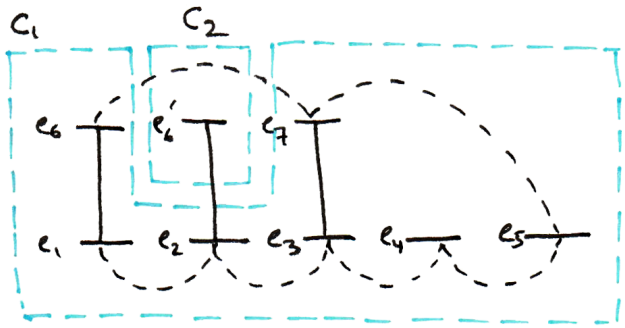

Figure 4.5: The corresponding event structure $\mathcal{E} = \Theta'(\mathcal{E}^{-})$.

Remark 4.3.17. Note that although in this example c_2 does not reflect any choice being made, more complex examples exist where all branching cells have a choice to make. Therefore, it is not possible to resolve this at the probabilistic level, e.g. by trying to combine a number of branching cells with respect to their probabilities.

Analysing this example (and concept) further, it can be seen that this problem arises due to an event e of the stable event structure representing two events of its mapping event structure (e.g. e_1 and e_2) which belong to different branching cells of the event structures (e.g. $e_1 \in c_1$ and $e_2 \notin c_1$). That is when we will encounter a problem when mapping the branching cells of one structure to another. Note that for all such events, $e_1 \# e_2$ & $\neg e_1 \#_\mu e_2$. Therefore, they cannot be the initial events of a configuration, as the conflict in their past must be resolved before any of them occurs. Therefore, the reason they are grouped in the same branching cell is that at least one of them is connected to the source of conflict in their past via a chain of events in immediate conflict.

Thus, to achieve isomorphic branching cells and avoid the above situation, we consider the stable event structures (and their associated event structures) which do not allow for these cases, namely, those structures which are *jump-free*.

Definition 4.3.18. An event structure is *jump-free* iff $\forall e, e'.\ e < e' \Rightarrow \nexists e_1, \ldots, e_k.\ k > 1$ & $e \#_\mu e_1, e_i \#_\mu e_{i+1}$ & $e_k \#_\mu e'$, for $1 \leq i \leq k$.

Similarly, a jump-free stable event structure is defined as below.

Definition 4.3.19. A stable event structure is jump-free iff $\forall e, e'.\ e <_v e' \Rightarrow \nexists e_1, \ldots, e_k.\ k >$

$1 \;\&\; e\#_{\mu,v_0}e_1, e_i\#_{\mu,v_i}e_{i+1}, e_k\#_{\mu,v_k}e',$ for $1 \leq i \leq k-1 \;\&\; v_i \subseteq v$.

Example 4.3.20. The stable event structure in figure 4.5 is not jump-free as either of the chains of events e_3, e_4, e_5, e_7 and $e_1, \ldots, e_7$ break jump-freeness.

Jump-free event structures are simpler to deal with, as, unlike event structures, they are *flat* in the following sense.

Proposition 4.3.21. The branching cells of jump-free event structures (as initial stopping prefixes) consist of initial events only.

Proof. Suppose c has non-initial events and let $e \in c$ be such event s.t. $\exists e_0 \in c.\ e_0 < e \;\&\; \nexists e_1 \in c.\ e < e_1$. Then there exists an initial event e' s.t. $\nexists e_0 \in c.\ e_0 < e' \;\&\; e' < e$. Noting that c is an initial stopping prefix and therefore, $\nexists c'.\ c' \subset c$, then in the formation of $\{e'\}^*$, e can only be added to achieve the closure of $\#_\mu$. Therefore, there must be a chain of events $e_1, \ldots, e_k$ s.t. $e'\#_\mu e_1, e_i\#_\mu e_{i+1} \;\&\; e_k\#_\mu e$. Observe that $k > 1$ as otherwise $\neg(e\#_\mu e_1)$. Therefore, above chain forms a jump which is a contradiction and as such c only consists of non-initial events.

□

The same holds for stable event structures as expressed in the following proposition.

Proposition 4.3.22. The branching cells of jump-free stable event structures (as initial stopping prefixes) consist of initial events only.

Proof. The proof follows a similar reasoning to that of event structures, noting that for all the initial events in $\mathcal{E}^v$, $\#_\mu$ is resolved meaning $\forall v' \in \mathcal{V}(\mathcal{E}^v).\ e\#_{\mu,v'}e' \Rightarrow e\#_\mu e'$. □

Lemma 4.3.23. Let $\mathcal{E}^-$ be a net-driven stable event structure and let $\mathcal{E} = \Theta'(\mathcal{E}^-)$ be its associated event structure. Then we have

$$e\#_{\mu,v}e' \Leftrightarrow \lceil e \rceil_v \#_\mu \lceil e' \rceil_v.$$

Proof. Follows trivially from lemma 4.1.8 and definitions of $\#_{\mu,v}$ and $\#_\mu$. □

Lemma 4.3.24. Let $\mathcal{E}$ be a jump-free net-driven stable event structure. Then $\Theta'(\mathcal{E})$ is jump-free.

Proof. Follows trivially from lemmas 4.1.8 and 4.3.23. □

We now show that the branching cells covering the configurations of a stable event structure are isomorphic to those of the corresponding configurations of its associated event structure. We start by proving the following lemma.

Lemma 4.3.25. Let $\mathcal{E}^-$ be a net-driven stable event structure and let $\mathcal{E} = \Theta'(\mathcal{E}^-)$ be its associated event structure. Then, for $p \neq p'$ if $\theta(p) = \theta(p') \Rightarrow p\#p'$ & $\neg(p\#_\mu p')$.

Proof. Suppose $\theta(p) = \theta(p') = e$, then $p = \lceil e \rceil_v$, $p' = \lceil e \rceil_{v'}$. Let v_0 be the subset of v s.t. $v_0 \vdash_{min} e$ and similarly, let v_1 be the subset of v' s.t. $v_1 \vdash_{min} e'$. By the stability axiom it follows that $v_0 \cup v_1 \notin Con$. It is then obvious that $p\#p'$. Since $\mathcal{E}^-$ is net-driven, it follows that $\exists e_0 \in v_0, e_1 \in v_1.\ e_0\#e_1$, and therefore, $\lceil e_0 \rceil_v \# \lceil e_1 \rceil_{v'}$ and since $\lceil e_0 \rceil_v < \lceil e \rceil_v$ and $\lceil e_0 \rceil_{v'} < \lceil e \rceil_{v'}$ it follows that $\neg(p\#_\mu p')$. □

Recalling that the configurations of a stable event structure and its associated event structure are isomorphic, we show in the following theorem that the branching cells of a jump-free net-driven stable event structure and its mapping event structure are isomorphic.

Theorem 4.3.26. Given a jump-free net-driven stable event structure $\mathcal{E}^-$ and its mapping event structure $\mathcal{E} = \Theta'(\mathcal{E}^-)$, $\mathcal{C}$, the set of branching cells of $\mathcal{E}$, is isomorphic to $\mathcal{C}^-$, the set of branching cells of $\mathcal{E}^-$.

Proof. We use the morphism θ as in proposition 4.1.4 to prove this theorem. First consider configuration v of $\mathcal{E}$ and $v' = \theta(v)$ of $\mathcal{E}^-$. Since configurations of $\mathcal{E}$ and $\mathcal{E}^-$ are isomorphic, it is clear that $\exists e \in E \setminus v.\ v \cup \{e\} \in \mathcal{V}(\mathcal{E}) \Leftrightarrow \exists e' \in E^- \setminus v'.\ v' \cup \{e'\} \in \mathcal{V}(\mathcal{E}^-)$. In other words, every initial event in future of v has an associated initial event in future of v' and vice versa. Let $\mathcal{E}_0^v$ and $\mathcal{E}_0^{-v'}$ represent the initial events of each structure, respectively. Then we show that θ yields a bijection between $\mathcal{E}_0^v$ and $\mathcal{E}_0^{-v'}$.

Suppose $e, e' \in \mathcal{E}_0^v$ and $\theta(e) = \theta(e') = e''$. By lemma 4.3.25 it follows that $e\#e'$ and $\neg e\#_\mu e'$, i.e. there is a conflict in their past. But this is a contradiction as they are both initial events in future of a configuration which is conflict-free. As shown above, every initial event in future of v' has an associated event in future of v, therefore, θ (applied to initial events $\mathcal{E}^v$) is onto (for initial events of $\mathcal{E}^{-v}$. It then follows that θ yields a bijection between the events of $\mathcal{E}_0^v$ and $\mathcal{E}_0^{-v'}$, making them isomorphic.

Furthermore, as we are dealing with the initial events that can occur in future of v' i.e. immediately after v', the immediate conflict relation among the events of $\mathcal{E}_0^{-v'}$

is resolved, in the sense that it does not depend on any configuration in $\mathcal{E}_0^{-v'}$, and therefore, is obviously compatible with the immediate conflict relation of $\mathcal{E}_0^v$. Thus, the branching cells of $\mathcal{E}_0^v$ and $\mathcal{E}_0^{-v'}$ are isomorphic, which implies the branching cells of $\mathcal{E}$ and those of $\mathcal{E}^-$ are isomorphic. □

Recalling that the configurations of a net-driven stable event structure and its associated event structure are isomorphic, the most important result of theorem 4.3.26 is that the covering of any configuration in a stable event structure is exactly the same as that of its corresponding configuration in its associated event structure. That is because the configurations in the future of two isomorphic configurations are also isomorphic and therefore, two isomorphic configurations have isomorphic coverings. Therefore, all the probabilistic properties of branching cells of event structures are applicable to those of stable event structures, and as such, all the probabilistic machinery described in section 4.2.1 for event structures can be applied to net-driven jump-free stable event structures. Thus, for example, the likelihood function for a stable event structure $\mathcal{E}^-$, $p^- : \overline{\mathcal{W}^-} \to \mathbb{R}$ is defined as:

$$\forall v \in \overline{\mathcal{W}^-},\ p^-(v) = \prod_{c \in \Delta^-(v)} q_c(v \cap c)$$

4.4 Conclusion

In this chapter we have introduced a new class of stable event structure called net-driven stable event structures, which describe major properties of stable event structures driven from general Petri nets. We have then presented the definition of probabilistic event structures as defined by [1]. Required concepts of stopping prefix, stopped configurations, *R*-stopped configurations, locally finiteness, branching cells and local transition probabilities have been defined, which help in assigning probabilities to finite *R*-stopped configurations. Since for local finite event structures maximal configurations are also *R*-stopped [1], then branching cells decompose maximal configurations in an elegant, recursive and dynamic manner. They can be considered as units of choice constituting a configuration, which resolve choices and confusions internally. The most important result is that branching cells are the finest grain units for which choices are resolved independently from each other, therefore, "concurrency matches probabilistic independence" [1].

Our aim was to prove that the solid results of [1] can also be applied to stable event structures. We therefore studied if configurations of stable event structures can be decomposed in the same manner, and proved that although this is not the case in general, by avoiding certain structures, this can be achieved. More concretely, we defined the new concept of jump-free event structures and stable event structures for which the branching cells consist of events which are not causally related. We then proved that for such stable event structures and their associated event structures, the branching cells are isomorphic. Thus, probabilistic jump-free stable event structures were defined in a similar manner to probabilistic event structures of [1].

Chapter 5

Probabilistic Event Structure Logic

Temporal logics and their verification over concurrent and reactive systems have been studied intensely over the last decades. The most well-known logics implementing the interleaving approach include LTL [60], CTL[13], CTL*[13], L_μ[43] (which are usually defined on labeled transition systems)and POTL[59], ISTL[35] and TLC[5] as partial order logics. Logics which implement the true concurrency approach are fairly recent, and there is space for their further exploration. Main examples of these logics include SFL[25] (inspired by [31]) and TCL[8].

The same division can be found for probabilistic logics over concurrent systems. Major work has been achieved on studying and defining probabilistic logics with the interleaving approach. Probabilities were foremost adjoined to linear temporal logics, with the interpretation that a property is (almost) surely satisfied (i.e. with probability 1). Such logics were defined both on finite Markov chains [46, 61, 29, 4, 62] and extended Markov models which can capture both notions of nondeterminism and probabilistic choice [73, 17, 62].

Later, the logics PCTL and PCTL* [28, 6] were developed over discrete Markov chains which could directly express quantitative properties of such systems. Further studies introduced interpretations of PCTL and PCTL* on Markov models incorporating nondeterministic and probabilistic behaviour [11, 12, 44, 7, 45]. The interpretations of these logics over such systems requires a notion of strategy or policy and furthermore, due to existence of nondeterminism, only maximal/minimal probabilities can be defined on each scenario.

As mentioned before, to our best knowledge, there are no probabilistic logics with the

true concurrency semantics. Therefore, in order to define such a logic on probabilistic event structures, new relations need to be defined between configurations. These relations should be able to capture different types of interactions between groups of events such as R-stopped configurations. For example, when dealing with R-stopped configurations, one cannot immediately apply the non-probabilistic logics such as SFL or POTL to, for example, describe an action (event) that can causally follow a configuration. Consider the following example.

Example 5.0.1. Consider $\mathcal{E}_1$ depicted in figure 5.1. Then existence of event e_1 implies occurrence of e_3 in an R-stopped configuration $v = \{e_1, e_3\}$. Now let us consider a configuration which is caused by v. As an example, event e_4 can causally follow v, however, its occurrence in a configuration implies existence of e_6. Therefore, v can progress to $v' = \{e_1, e_3, e_4, e_6\}$ and the events in v and v' are related both causally and concurrently.

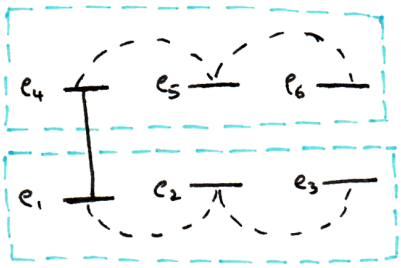

Figure 5.1: Event structure $\mathcal{E}_1$

Thus, the relation between configurations need not be purely causal or concurrent and as such it encompasses causality and concurrency by implying a new notion of progression.

In this chapter we introduce the logic PESL which is defined on probabilistic event structures in a number of layers, describing properties related to events, configuration and the interactions between R-stopped configurations. In this framework, PESL can describe properties related to scenarios, such as 'it is unlikely that a configuration (scenario) can *lead* to an undesirable one'. We start by the definition of its syntax and semantics, followed by a description of its expressivity and conclude by a model checking algorithm for finite systems.

Remark 5.0.2. Note that although logics such as ISTL can differentiate concurrency from non-determinism, they achieve this through grouping of possible interleavings

into different sets. Therefore, it can be argued that they do not truly have the true concurrency semantics.

5.1 Definitions and Concepts

We define the logic PESL on R-stopped configurations of probabilistic event structures as defined in the previous chapter. The logic is structured over a number of layers, describing configurations in terms of their constituting events and the relation between R-stopped configurations. We start by introducing definitions of different relations between R-stopped configurations.

Definition 5.1.1. A configuration v enables an event e denoted by $v \vdash e$ iff $e \notin v$ & $\lceil e \rceil - \{e\} \subseteq v$ and $v \cup \{e\}$ is conflict-free.

Definition 5.1.2. We denote immediate causality of events by $e \rightarrow e'$ iff $e < e'$ & $\nexists e''.\ e < e''$ & $e'' < e$.

The following definition defines a notion related to causality between R-stopped configurations. Compatibility of two configurations is defined as below.

Definition 5.1.3. Two configurations v and v' are *compatible* if $v \cup v'$ is a configuration. The union of compatible configurations is then represented by $v \circledast v' =_{\text{def}} v \cup v'$ and we also use $v \circledast v'$ to denote that v and v' are compatible.

Recalling definition 4.2.7 of concatenation of configurations ($v \oplus v'$), we define a progression step of a configuration as follows.

Definition 5.1.4. For configurations v and v' we write $v \rightarrow v'$ iff $v' = v \oplus (\circledast v_i)$ where v_i are maximal configurations of branching cells $c_i \in \delta(v)$ s.t. $\exists e \in v, e' \in c_i.\ e \rightarrow e'$.

Note that if $v \rightarrow v'$ then v' is clearly R-stopped as the c_is added to the covering for v form a covering for v'.

Example 5.1.5. Consider the event structure $\mathcal{E}_2$ as in figure 5.2 and its configuration $v = \{e_1, e_3, e_8\}$. Then v can lead to v' (i.e. $v \rightarrow v'$) where $v' = v \cup \{e_4, e_6, e_9\}$. Note that c_1 is not enabled unless e_{13} is added to v.

The following definition relates configurations which are completely concurrent.

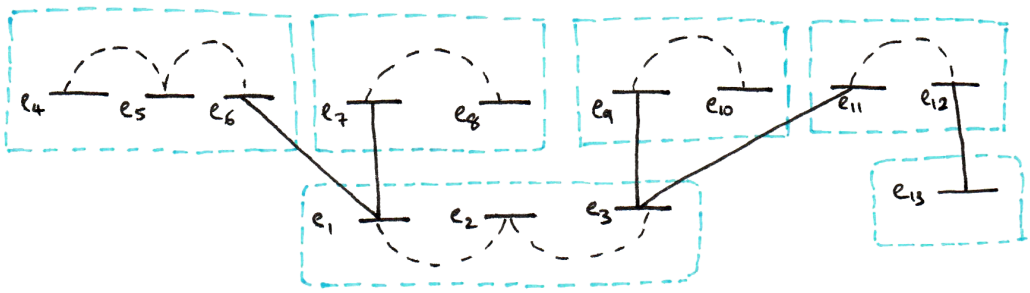

Figure 5.2: Event structure $\mathcal{E}_2$

Definition 5.1.6. For configurations v and v' we write $v \parallel v'$ iff $\forall e \in v, e' \in v'.\ e\ co\ e'$.

In the following definition, we define synchronisation of two R-stopped configurations.

Definition 5.1.7. For configurations v, v' and v'' we write $(v, v') \curlywedge v''$ iff v'' is a minimal R-stopped configuration such that $v \circledast v' \circledast E_{\curlywedge} \subseteq v''$ where $E_{\curlywedge} \subseteq \{e'' \in E \setminus (v \cup v') \mid \exists e \in v, e' \in v'.\ e \neq e' \ \&\ e \rightarrow e'' \ \&\ e' \rightarrow e'' \ \&\ v \circledast v' \vdash e''\}$ and is non-empty.

Example 5.1.8. Consider the event structure $\mathcal{E}_3$ as in figure 5.3. Configuration $v = \{e_1, e_2\}$ can synchronise with $v' = \{e_3, e_4\}$ into $v'' = \{e_5, e_8\}$, i.e. $(v, v') \curlywedge v''$.

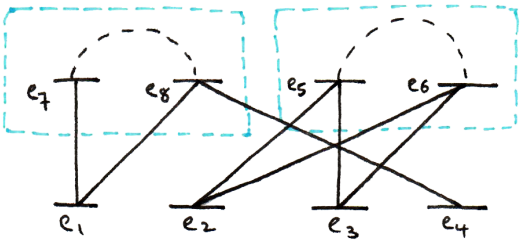

Figure 5.3: Event structure $\mathcal{E}_3$

In order to define the probabilistic operator for PESL, we first define the notion of conditional probability for two configurations.

Definition 5.1.9. Given a probabilistic event structure $(\mathcal{E}, \mathbb{P})$, the probability of a configuration v occurring given configuration v' has occurred is defined in the usual way, as follows.

$$\mathbb{P}(v \mid v') =_{\text{def}} \begin{cases} 0 & \text{if } v \cup v' \text{ is not a configuration} \\ \frac{\mathbb{P}(v \circledast v')}{\mathbb{P}(v)} & \text{otherwise} \end{cases}$$

Note that $\mathbb{P}(v \circledast v')$ represents the probability of the event that v and v' occur, in other words, the intersection of the events that v occurs and v' occurs as well.

Next, we define the conditional probability of any configuration from a set of R-stopped configurations V occurring, given that an R-stopped configuration v has occurred. Since the branching cells of R-stopped configurations decompose a configuration into probabilistically independent units, the probability of any configuration of a set of configurations occurring corresponds to the probability of the union of those configurations.

Definition 5.1.10. Given a probabilistic event structure $(\mathcal{E}, \mathbb{P})$ and a finite set $V = \{v_1, \ldots, v_n\}$ of R-stopped configurations of $\mathcal{E}$, the conditional probability of any of the configurations in V occurring, if v has occurred is defined as:

$$\mathbb{P}(V \mid v) =_{\text{def}} \sum_{i=1}^{n} \mathbb{P}(v_i|v) - \sum_{i \neq j} \mathbb{P}(v_i \cup v_j|v) + \ldots + (-1)^{n+1} \mathbb{P}(v_1 \cup \ldots \cup v_n|v)$$

5.2 Syntax and Semantics

We can now define the syntax and semantics for PESL, by describing each level of logic. The event-level formulae are represented by η formulae, while the configuration-level formulae are represented by α formulae. Probabilistic formulae over R-stopped configurations are then defined by logical formulae ϕ which in turn incorporate ψ formulae describing the relation between R-stopped configurations through the operators defined previously.

5.2.1 Event-Level formula

Given a set of events E, the basic properties of an event $e \in E$ are described by formulae in the form η according to the following syntax:

$$\eta := tt \mid a \mid \neg\eta \mid \eta \wedge \eta' \mid \Diamond\!\!\!\!-\, \eta \mid \leq \eta \mid \geq \eta$$

where a is an atomic proposition, $\barwedge$ is the concurrency operator and $\leq, \geq$ are the causality operators.

The formulae over events are interpreted in the context of a set of events ($X \subseteq E$). The semantics for η formulae are given below, where $Q : E \to \wp(AP)$ is a validator function for the set of atomic propositions AP.

5.2.1. $\forall e \in E.\ e \vDash_X tt$

5.2.2. $e \vDash_X a$ iff $a \in Q(e)$

5.2.3. $e \vDash_X \neg\eta$ iff $e \nvDash_X \eta$

5.2.4. $e \vDash_X \eta \wedge \eta'$ iff $e \vDash_X \eta$ and $e \vDash_X \eta'$

5.2.5. $e \vDash_X \barwedge\eta$ iff $\exists e' \in X.\ e' \vDash_X \eta \ \&\ e\ co\ e'$
In other words, an event e satisfies a formula $\barwedge\ \eta$ if η is satisfied concurrently by another event.

5.2.6. $e \vDash_X \leq\eta$ iff $\exists e' \in X.\ e' \vDash_X \eta \ \&\ e \leq e'$
In other words, an event e satisfies a formula $\leq \eta$ if η is satisfied by an event in the potential causal future of e.

5.2.7. $e \vDash_X \geq\eta$ iff $\exists e' \in X.\ e' \vDash_X \eta \ \&\ e \geq e'$
In other words, an event e satisfies a formula $\leq \eta$ if η is satisfied by an event in the causal past of e.

5.2.2 Configuration-Level Formulae

Let $\mathcal{E}$ be an event structure. The properties of configurations of $\mathcal{E}$ are described by α formulae according to the following syntax.

$$\alpha := \neg\alpha \mid \alpha \wedge \alpha' \mid ?\eta \mid ?\overline{\eta} \mid ?\underline{\eta}$$

The semantics for the configuration-level formulae are interpreted under the context of a set of events ($X \subseteq E$), as follows.

5.2.8. $v \vDash^X \neg\alpha$ iff $v \nvDash^X \alpha$

5.2.9. $v \vDash^X \alpha \wedge \alpha'$ iff $v \vDash^X \alpha$ and $v \vDash^X \alpha'$

5.2.10. $v \vDash^X ?\eta$ iff $\exists e \in v.\ e \vDash_v \eta$

5.2.11. $v \vDash^X ?\overline{\eta}$ iff $\exists e \in v \setminus X.\ e \vDash_v \eta$

5.2.12. $v \vDash^X ?\underline{\eta}$ iff $\exists e \in X.\ e \vDash_v \eta$

5.2.3 PESL

Let $(\mathcal{E}, \mathbb{P})$ be a probabilistic event structure. A PESL formula capturing properties of R-stopped configurations of $\mathcal{E}$ is defined as:

$$\phi := \alpha \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathcal{P}_{\sim\lambda}\psi$$

where $\mathcal{P}$ is the probabilistic operator, $\sim$ is one of the comparators $<, >, \leq, \geq, =$, λ is a real number and ψ describes a set of R-stopped configurations according to the following syntax.

$$\psi :=\rightarrow \phi \mid \phi_1 U \phi_2 \mid \ \| \phi \mid \phi_1 \curlywedge \phi_2$$

The $\rightarrow$ operator corresponds to the *next* configuration as in definition 5.1.4, the U operator is the *until* operator on a path of $\rightarrow$-related configurations, the $\|$ describes concurrent configurations as in definition 5.1.6 and finally the $\curlywedge$ operator captures synchronisation of configurations as in definition 5.1.7.

We are now ready to present the semantics of PESL over R-stopped configurations which assure maximal progression with respect to executable immediate conflict. In other words, for R-stopped configurations the (probabilistically) dependent choices are resolved. The semantics for PESL formulae are as follows.

5.2.13. $v \vDash^X \alpha$ as defined above (§5.2.2).

5.2.14. $v \vDash^X \neg\phi$ iff $v \nvDash^X \phi$

5.2.15. $v \vDash^X \phi_1 \wedge \phi_2$ iff $v \vDash^X \phi_1$ and $v \vDash^X \phi_2$

5.2.16. $v \vDash^X \mathcal{P}_{\sim\lambda}\psi$ iff $\mathbb{P}(V_{\psi,v} \mid v) \sim \lambda$

where $\mathbb{P}(V_{\psi,v} \mid v)$ refers to the probability of occurrence of any of the configurations in $V_{\psi,v}$ as in definition 5.1.10 and $V_{\psi,v}$ is defined in the following.

Remark 5.2.17. If the context X is not explicitly mentioned, then we are referring to the empty set. In other words, let $v \vDash \phi$ iff $v \vDash^{\{\}} \phi$.

Given a formula ψ and an R-stopped configuration v, we define the $V_{\psi,v}$, the set of R-stopped configurations captured by ψ as follows.

5.2.18. $V_{\rightarrow\phi,v} =_{\text{def}} \{v_i \mid v_i \vDash^v \phi \;\&\; v \rightarrow v_i\}$

5.2.19. $V_{\phi U\phi',v} =_{\text{def}} \{v_n \mid$ there is a path $\pi : v \rightarrow v_1 \rightarrow^* v_n$ s.t. $v_n \vDash^{v_{n-1}} \phi' \;\&$
for every other configuration v_i in the path $v_i \vDash^{v_{i-1}} \phi$, where $0 \leq i < n$ and $v_0 = v\}$

5.2.20. $V_{\|\phi,v} =_{\text{def}} \{v_i \mid v_i \vDash^v \phi \;\&\; v \parallel v_i\}$

5.2.21. $V_{\phi\curlywedge\phi',v} =_{\text{def}} \{v_i \mid \exists v'.\ v'$ is a R-stopped and minimal$^{\subseteq}$ configuration s.t.
$(v,v') \curlywedge v_i \;\&\; v' \vDash^v \phi \;\&\; v_i \vDash^{(v \circledast v')} \phi'\}$

5.3 Expressivity

PESL is in many ways incomparable to any of the other existing logics in the related areas. On the one hand, as mentioned before, PESL is the first probabilistic logic with truly concurrent semantics defined for a model of true concurrency in the literature. On the other hand, because it is defined over R-stopped configurations, a notion of progression step is introduced which does not have a parallel in other logics (probabilistic or non-probabilistic). Therefore, in this section we attempt to describe a relatively full picture of its expressivity, comparing it to other logics such as pCTL and SFL for comparable properties.

Generally, the major advantage of probabilistic logics is that they achieve a more realistic and detailed expressivity over the behaviour of systems. As an example, a system which may fail with a very small probability will be labelled by a non-probabilistic logic as a failing system without any indication of how unlikely it is to fail, whereas a probabilistic logic assigns a likelihood to each desirable or undesirable scenario, describing the behaviour of a system in a more detailed and quantitative manner.

As mentioned above, a crucial difference between PESL and other existing logics for event structures (both with interleaving and true concurrency semantics) is that given

the definition of probabilistic event structures using branching cells, an atomic action between configurations is not necessarily a single event but a particular set of events. This follows from the fact that in probabilistic event structures probabilities are assigned to R-stopped configurations covered by branching cells. Therefore, it is both interesting and important to note the new notion of progression defined by the $\rightarrow$ operator used by the logic. As example 5.1.5 shows, the idea is that given an R-stopped configuration, there are a number of units of choices enabled to be resolved. These choices between events may be grouped together in a branching cell, or may be concurrently enabled in separate branching cells. Thus, $\rightarrow$ defines in some way a notion of maximal progression in terms of resolving all the enabled units of choices possible. To clarify, by *being enabled* here we mean that the given configuration is causally required to enable those choices (branching cells). Calling this progression a step, then since steps are defined in between groups of events, they are neither purely causal nor concurrent but possibly a mixture of these. Therefore, it must be emphasised that PESL describes the flow of scenarios described by events grouped together and in that sense it is different from other existing logics for concurrent systems. The examples in this section are aimed to clarify this difference.

We believe that the notion of step and this kind of view of the system fits the concept of true concurrency in concurrent systems, as no order is enforced on a set of concurrent events. Therefore, defining the interleaving space created by such events is neither required nor justified. Instead, we view configurations as scenarios which can *lead* to other scenarios in a progressive, synchronising or concurrent manner (or a combination of them).

We also show that as a probabilistic logic, PESL is as expressive as pCTL over paths of configurations defined by steps of progression ($\rightarrow$). In terms of its qualitative expressivity, PESL can express certain properties as other true concurrency logics such as SFL and LTC. The main differences lie in the expressive power resulting from using fixed points in those logics expressivity of the notion of causality between events. On the other hand, PESL directly expresses properties related to synchronisation of configurations, which is not possible in general by SFL. This also appears to be the case for LTC, as it is not immediately clear if the logic can express synchronisation properties at least in a direct manner.

We now explain the above facts in more details along some examples. PESL, in addition to its propositional fragment, consists of three major fragments, namely, those of progression, concurrency and synchronisation. We present examples of expressivity of PESL for each of these fragments.

Remark 5.3.1. It should be noted that the configuration-level formulae (α) assigned to the configurations play an important role in the expressive power of PESL. This is because the expressive power of α formulae directly affects the expressivity of the logic. For example, another logic describing configurations such as ESL can be used instead of the configuration-level formulae. Clearly if the underlying configuration-level logic is more powerful then we obtain a much more expressive logic. However, it will be very difficult to distinguish between the expressivity achieved by the probabilistic layer and the lower layer and to determine the possible overlapping of the two. We therefore have selected a very basic logic for the lower layer formulae on configurations and focus more on the expressivity achieved in between configurations rather than that of the structure of a configuration.

5.3.1 Progression

The operators $\rightarrow$ and U are used for expressing properties on paths formed by progression of configurations with respect to the choices enabled at each step. We show that the progression fragment of PESL can be interpreted as encoding the logic pCTL over a tree-structured Markov chain.

Recall that a Markov chain is defined as below.

Definition 5.3.2. A (discrete time) Markov chain is a tuple $\mathcal{M} = (S, \mathbb{P}_{\mathcal{M}}, AP, L)$ where S is a countable set of states, AP is a set of atomic propositions, $L : S \rightarrow 2^{AP}$ is a labelling function and $\mathbb{P}_{\mathcal{M}} : S \times S \rightarrow [0,1]$ is the transition probability function such that for all states s, $\sum_{s' \in S} \mathbb{P}_{\mathcal{M}}(s,s') = 1$.

Recall the definition of pCTL over Markov chains:

$$\phi := a \mid \phi \wedge \phi' \mid \neg\phi \mid \mathcal{P}_{\sim\lambda}\psi$$

where

$$\psi ::= \rightarrow \psi \mid \psi_1 U \psi_2.$$

The semantics for the propositional fragment of pCTL is as expected, and the semantics of the probabilistic operators over a Markov chain $\mathcal{M}$ is defined as below, with $\pi[i]$ representing the ith state on the path π.

5.3.3. $s \vDash \mathbb{P}_{\sim\lambda}\psi$ iff $\mathbb{P}_{\mathcal{M}}(\{\pi \in \mathit{Paths}(s) \mid \pi \vDash \psi\}) \sim \lambda$.

5.3.4. $\pi \vDash \phi$ iff $\pi[0] \vDash \phi$.

5.3.5. $\pi \vDash \rightarrow \psi$ iff $\pi[1] \vDash \psi$.

5.3.6. $\pi \vDash \psi_1 U \psi_2$ iff $\exists j \geq 0.\ (\pi[j] \vDash \psi_2)\ \&\ \forall k.\ 0 \leq k < j\ \&\ \pi[k] \vDash \psi_1$.

Note that for both pCTL and PESL, the operator U can be used in the usual way to capture that a formula *eventually* (or dually *always*) holds, along a $\rightarrow^*$ path, as defined below.

$$\Diamond\psi =_{\text{def}} tt\ U\ \psi,\ \Box\psi =_{\text{def}} \neg\Diamond\neg\psi$$

Theorem 5.3.7. The logic PESL encodes pCTL interpreted over progressive paths (formed by $\rightarrow^*$) of configurations.

Proof. We first build the Markov chain as required by pCTL and then show PESL can encode all operators of pCTL.

Let $\mathcal{S}_{\mathcal{E}}$ be the R-stopped configurations of the event structure $\mathcal{E}$. Then the Markov chain whose states are such configurations can be constructed as $\mathcal{M}_{\mathcal{S}_{\mathcal{E}}} = (\mathcal{S}_{\mathcal{E}}, \mathbb{P}_{\mathcal{S}}, AP, L)$. The probability of a transition is then defined as $\mathbb{P}_{\mathcal{S}}(v, v') = \mathbb{P}(v'|v)$ where $v \rightarrow v'$. Since for any configuration v, all the configurations v' such that $v \rightarrow v'$ are pairwise in conflict, then the progressive paths correspond to paths in pCTL and furthermore, $\mathbb{P}(\{v' \mid v \rightarrow v'\}) = \sum \mathbb{P}(v')$. Consider the sub-event structure $\mathcal{E}' = \bigcup c_i$ where c_is are the branching cells covering each $v' \setminus v$ (and $v \rightarrow v'$). Then from the definition of $\rightarrow$ it follows that $v' \setminus v$ form the maximal configurations $\mathcal{E}'$. Since $\mathbb{P}(v'|v) = \mathbb{P}'(v' \setminus v)$ (c.f. lemma 5.5. and proposition 5.6. of [1]) where $\mathbb{P}'$ is the probability measure for $\mathcal{E}'$, then $\sum_{v \rightarrow v'} \mathbb{P}(v'|v) = \sum_{v \rightarrow v'} \mathbb{P}'(v' \setminus v) = \sum_{v_i \in \Omega(\mathcal{E}')} \mathbb{P}'(v_i) = 1$ (note that all configuration of different c_is are compatible with each other). Furthermore, it is clear that the above

structure has the Markov property, although this is only because each state has all the past information within itself.

As for the operators, first observe that in $\mathcal{M}_{\mathcal{S}_\mathcal{E}}$ a path of configurations is in fact a configuration itself. Therefore, it is easy to see that if $\pi(v) \vDash\!\!\rightarrow \phi$ then $v \rightarrow \pi[1] \,\&\, \pi[1] \vDash \phi$. Similarly, noting the only difference between the U operators in pCTL and PESL is that in the latter the path has the length at least 1, the U operator of pCTL can be simulated by $\pi(v) \vDash \phi_2 \vee (\phi_1 \wedge \phi_1 U \phi_2)$. Thus, computing the probability a set of paths is the same as computing the probability of a set of configurations. It therefore follows that the progressive fragment PESL encodes the behaviour of pCTL and as such all the related expressivity results follow. □

Immediate results of the above theorem include expressivity of almost sure repeated reachability ($\mathcal{P}_{=1}(\Box\mathcal{P}_{=1}(\Diamond ?a))$), persistence probabilities ($\mathcal{P}_{\sim\lambda}(\Diamond\mathcal{P}_{=1}(\Box ?a))$) and repeated reachability probabilities ($\mathcal{P}_{\sim\lambda}(\Diamond\mathcal{P}_{=1}(\Box\mathcal{P}_{=1}(\Diamond ?a)))$). It is known that pCTL can express properties that CTL fails to do, such as $\mathbb{P}_{=1}(\Diamond ?a)$. On the other hand, it fails to express some CTL properties such as $\forall\Diamond a$ (for infinite paths). A similar argument to the proof of theorem 5.3.7 can be used to show that the same applies to PESL. Thus, in PESL we cannot describe the property that for all paths (i.e. configurations) v' starting in a state (e.g. configuration v s.t. $v \rightarrow^* v'$) it holds that $v' \vDash \Diamond\alpha$. Dually, it cannot express that there is no path v' such that $v' \vDash \Box\alpha$. Therefore, in the examples below we some times refer to almost sure probabilities. For example, $\mathcal{P}_{=1} \parallel \phi$ expresses that almost surely property ϕ can happen in parallel. Note that similar to pCTL, this only happens when we have infinite paths (configurations).

Example 5.3.8. Consider the event structure $\mathcal{E}_4$ depicted in figure 5.4. Then for $v = \{e\}$, $v \vDash \mathcal{P}_{=1} \parallel ?b$ even though there is a configuration $v' = \{a, a, a, \ldots\}$ such that $v' \nvDash ?b$ and $v \parallel v'$.

Another important concept to be clarified is that of progressive steps versus causality. For example, the notion of reachability should not be confused with causality of events. Consider the event structure in the following example.

Example 5.3.9. Consider the event structure $\mathcal{E}_5$ depicted in figure 5.5. Then configuration $v = \{e_1\}$ leads to $v' = \{e_3\}$ even though there is no causal relation between e_1 and e_3. Thus, $v \rightarrow v'$ implies that in the next step following e_1 one of the choices enabled is e_3.

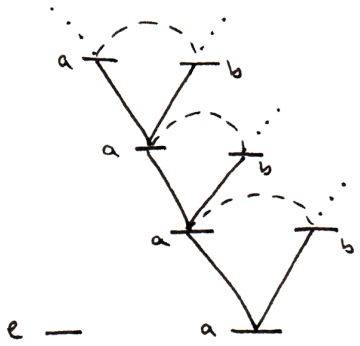

Figure 5.4: Event structure $\mathcal{E}_4$

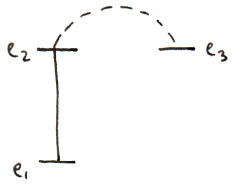

Figure 5.5: Event structure $\mathcal{E}_5$

Therefore, it could be said that PESL is not directly designed for expressing causality. In other words, dealing with confusion in event structures, requires us to consider the notion of progress rather than causality.

This however does not imply that causality is completely indescribable, as even the current very basic underlying configuration-level logic facilitates expressing certain causal properties by PESL, as we shall see below. Clearly even a slightly more powerful logic can expand the expressive power with respect to causality.

Reachability In general reachability can be described by $\mathcal{P}_{>0}\phi$.

The property that whenever an event labelled by a occurs it can lead to an event b (if b has not already occurred) is expressed by $?a \wedge \neg ?b \Rightarrow \mathbb{P}_{>0}\Diamond ?\overline{b}$, noting again that this does not necessarily imply that b is causally related to a. We can express similar causality related properties as below.

Request Granted We can describe that every request (event a) almost surely leads

(causally) to a grant (event b) by

$$\neg?(a \wedge \neg \leq b) \vee \mathcal{P}_{=1} \Diamond \neg?(a \wedge \neg \leq b).$$

It is interesting to note that PESL can only capture the qualitative form of the above property, i.e. if all or some requests are granted. It cannot describe for example the percentage of the requests being granted in the system as a whole as this notion cannot be defined probabilistically. Consider the following example.

Example 5.3.10. Consider the event structure $\mathcal{E}_6$ as in figure 5.6. Then the R-stopped configuration $\{a, a\}$ does not satisfy the above formula, and we cannot assign a likelihood measuring requests being granted.

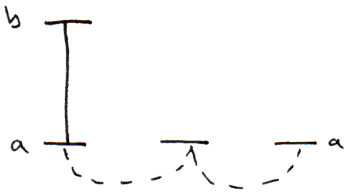

Figure 5.6: Event structure $\mathcal{E}_6$

Thus, $\mathcal{P}_{\sim\lambda} \Diamond \neg?(a \wedge \neg \leq b)$ describes the likelihood that all the requests in a configuration are granted. Clearly, this limitation is due to the concepts of branching cells and maximal progression and cannot be lifted by a more expressive underlying logic.

We have mentioned the crucial difference in the concept of progression versus causality. The above property can be a good example to portray the benefit of this view point. Consider the SFL formula $\phi_1 = \nu Z.[a](\mu Y.\langle b \rangle_c \, tt \wedge \langle - \rangle_c Y) \wedge [-]Z)$ describing that every request (a) is granted (b) on some of its causal paths. The PESL formula is different as the following example shows.

Example 5.3.11. Consider the event structures $\mathcal{E}_7, \mathcal{E}_7'$ as in figure 5.7. The PESL formula above is only satisfied by the $\mathcal{E}_7'$ while the SFL formula holds for both.

That is because SFL can only specify all or some of the causal paths following the requests having a grant whereas PESL can specify that in all *scenarios* or progressions, there is a causal path with a grant. This is indeed interesting as we can express properties related to scenarios which group the events that can occur concurrently together.

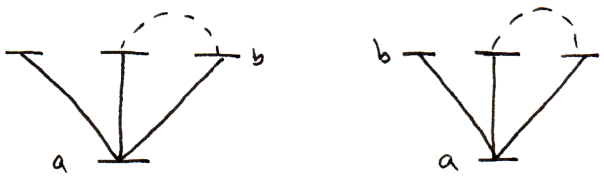

Figure 5.7: Event structures $\mathcal{E}_7$ (left) and $\mathcal{E}_7'$ (right)

Furthermore, note that the $\leq$ operator of the underlying formula here implies that there is a causal path from a which eventually leads to b. Thus, more complex properties such as those obtained by the until operator over causal paths cannot be expressed in the usual sense. For example, it cannot be expressed that every causal path followed a request has a grant without expanding the expressivity of the underlying logic.

Safety (Unreachability) Safety properties can be expressed by $\mathbb{P}_{=0}\Diamond\phi$ where ϕ can either be $?error$ or a more complex formula describing an undesirable property (e.g. refer to mutual exclusion below). More generally, the formula $\mathbb{P}_{\leq\varepsilon}\Diamond\phi$ can describe that the likelihood of a system (whose initial configurations satisfy the above property) reaching the undesirable configuration is very low.

Repeated Reachability
The formula $\mathcal{P}_{=1}(\Box\mathcal{P}_{=1}(\Diamond?\overline{a}))$ describes (almost sure) repeated reachability. Note that the similar formula $\mathcal{P}_{=1}(\Box\mathcal{P}_{=1}(\Diamond?a))$ cannot describe repeated reachability since once an event a is reached, it is included in all the configurations it leads to. Therefore, it is important to emphasise a *new* a event occurs repeatedly.

5.3.2 Concurrency

As mentioned previously, concurrency is usually handled in two ways, namely, by interleaving and true concurrency semantics. In the interleaving semantics, a sequence is considered for every possible ordering of concurrent events which quickly leads to the state explosion problem. For example, in the general case when there are two concurrent events e and e', either e can be observed next or e'. Then every event following e (and not caused by e') is also concurrent with e' and all orderings possible

between these events is considered which is the cause of state explosion. In the true semantics this is avoided and the ordering between events is only considered when they are causally related. PESL handles true concurrency in different manners as we shall see below.

First observe that two events of an event structure can be concurrent and have a common cause in their history or be totally separate. Additionally, concurrent events can be linked by a chain of $\#_\mu$, as it can occur in a branching cell. We now define these concepts more formally.

Definition 5.3.12. Two events e and e' are co-concurrent iff $e \; co \; e'$ and $\exists e''.e'' < e \;\&\; e'' < e'$.

Definition 5.3.13. Two events e and e' are parallel iff $e \; co \; e'$ and $\nexists e''.\; e'' < e \;\&\; e'' < e'$.

Definition 5.3.14. Two events e and e' are cell-concurrent iff $e \; co \; e'$ and $\exists e_1, \ldots e_n.\; e \#_\mu e_1 \;\&\; e_i \#_\mu e_{i+1} \;\&\; e_n \#_\mu e'$ where $1 \leq i < n$.

In PESL parallel and cell-concurrent events are captured by the $\|$ operator and the underlying logic $\overline{\forall}$ operator, respectively. Similarly, any concurrent events if already in the configuration, are covered by the $\overline{\forall}$ operator. However, for co-concurrent events it may be required to carry out at least one progressive step ($\rightarrow$) so that all such events are included to be covered by the $\overline{\forall}$ operator.

We now give some examples to clarify the above.

True concurrency

The $\|$ and in particular $\overline{\forall}$ operators can be used to describe true concurrency. The classical example of true concurrency versus interleaving semantics is given below.

Example 5.3.15. Consider the event structures $\mathcal{E}_8$ and $\mathcal{E}_8'$ depicted in figure 5.8. The systems can by differentiated by $?a \wedge \mathcal{P}_{>0} \,\|?b$ satisfied only by the left hand event structure or by $?a \wedge \mathcal{P}_{>0} \rightarrow ?b$ satisfied only by the right hand one.

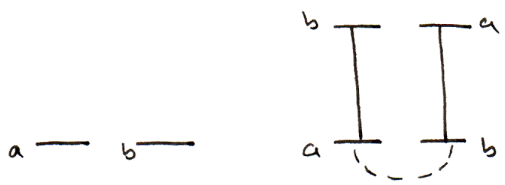

Figure 5.8: Event structures $\mathcal{E}_8$ (left) and $\mathcal{E}_8'$ (right)

Safety - Mutual Exclusion

The safety property that two processors cannot be in the critical section at the same time can be captured by $\phi_{safe} \wedge \mathcal{P}_{=1} \Box \phi_{safe} \wedge (?c \Rightarrow \mathbb{P}_{=0} \,||?c)$ where $\phi_{safe} = \neg?(c \wedge \overset{\vee}{\gamma} c)$.

Apart from the qualitative properties mentioned above, PESL can describe probabilities of parallel configurations as well. Thus, one can specify the probability of a desirable or undesirable scenario. For example, $?c \Rightarrow \mathbb{P}_{\leq 1\%} \,||\, c$ expresses that the probability of an error in parallel mutual exclusion is less than 1%. Similarly, the probability of ending up in an unsafe configuration from a safe one can be expressed by $\phi_{safe} \Rightarrow \mathcal{P}_{<1\%} \Diamond \neg \phi_{safe}$. Note that probabilities cannot be assigned to mutual exclusion being violated by cell-concurrent events.

5.3.3 Synchronisation

PESL can directly describe certain aspects of synchronisation in and between configurations. Consider the formula $?a \Rightarrow \mathbb{P}_{>0} ?b \curlywedge ?\overline{c}$ which describes a configuration with an a event can synchronise with one with a b event into a configuration which has a c event. Note that this does not indicate the exact causal relation between synchronising events and in order to achieve that a stronger underlying logic is required. In the examples below, we only express the general form of synchronisation. We give examples of how a stronger underlying logic can help expressing exact causal relation between events. Furthermore, we gives examples of situations where we need to assume that event structures are jump-free to avoid certain complications.

Can Synchronise - Exists

Formula $\psi_1 = ?a \wedge \mathbb{P}_{>0} ?b \curlywedge ?c$ holds for configurations which have an a event and can synchronise with a configuration having a b event to enable a c event. In its general form, $\phi_1 \wedge \mathbb{P}_{>0} \phi_2 \curlywedge \phi_3$ can be interpreted as: given a configuration ϕ_1, there exists a configuration ϕ_3 caused by a synchronisation of ϕ_1 and ϕ_2.

Synchronisation occurs surely

Formula $\psi_2 = ?a \Rightarrow \mathbb{P}_{=1} ?b \curlywedge ?c$ states that given a configuration with an a event, it will surely synchronise with a b event to enable a c.

For all synchronisation

Formula $\psi_3 = ?a \Rightarrow \mathbb{P}_{=0}?b \between \neg?c$ states that any synchronisation between a and b leads to c only.

Synchronisation can occur for all parties

Extending the event-level formulae α, let $e \vDash_X \rightarrowtail \eta$ iff $\exists e' \in X.\ (e' \vDash_X \eta \ \&\ e \leq e' \ \&\ \nexists e''.\ e'' \leq e' \ \&\ e \leq e'')$. Suppose a *read* event a synchronises with a *write* b to form a *communication* event c. Then the formula $\psi_4 = ?(a \wedge \neg \rightarrowtail c) \Rightarrow \mathbb{P}_{>0}?b \between (\neg?(a \wedge \neg \rightarrowtail c))$ states that every read event can engage with a write event in a communication.

Example 5.3.16. Consider event structure $\mathcal{E}_9$ depicted in figure 5.9 (left). Then for configuration $v = \{a\}$ we have $v \vDash \psi_1 \wedge \psi_4$, $v \nvDash \psi_2 \vee \psi_3$. For $\mathcal{E}_{10}$ (right) and $v' = \{a, a\}$, $v' \vDash \psi_1 \wedge \psi_2 \wedge \psi_3$ and $v' \nvDash \psi_4$.

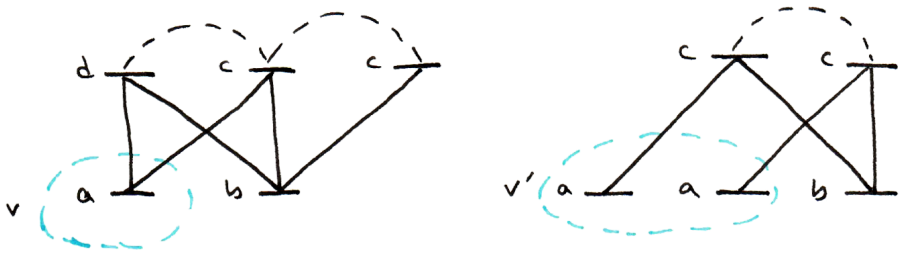

Figure 5.9: $\mathcal{E}_9$ (left) and $\mathcal{E}_{10}$ (right)

Noting that $\mathcal{P}_{\sim\lambda}\phi \between \phi'$ calculates the probability of configurations satisfying ϕ' and occurring via ϕ, the probability includes both the likelihood of the synchronising events (ϕ) occurring and it resulting in ϕ'. This can be used in other ways as well. For example the formula $?a \wedge \neg \overline{\forall}\, b \Rightarrow \mathbb{P}_{\geq\lambda}?b \between ?c$ computes how likely is it for the synching pair of a, namely b, to occur and further synchronise into a c. On the other hand, $?(a \wedge \overline{\forall}\ (b \wedge \neg \rightarrowtail c) \Rightarrow \mathbb{P}_{\geq\lambda}?b \between ?c$ computes the least probability of a synchronisation happening if the synchronising elements are already present.

We now briefly justify the jump-free assumption for event structures. Consider the following example.

Example 5.3.17. Consider the event structure $\mathcal{E}_{11}$ as in figure 5.10. Then e_2 and e_4

always appear together in a R-stopped configuration.

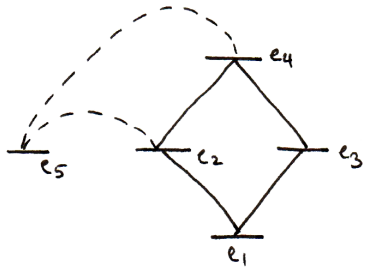

Figure 5.10: Event structure $\mathcal{E}11$

Thus, in event structures which are not jump-free, for some cases synchronisation properties have to be expressed using the $\leq$ and $\geq$ operators of the underlying logic. Therefore, one has to take into account the possibility of such synchronisation in the probabilities computed.

5.4 Model Checking

The model checking problem for PESL is defined as determining if a given configuration v of a probabilistic event structure $\mathcal{E}$ satisfies formula ϕ. We assume that either $\mathcal{E}$ is finite or that we are considering a finite prefix of an infinite event structure. The algorithm below sketches the model checking procedure, except for the obvious algorithms for verifying α and η formulae.

Algorithm 9 $MC(v, v', \phi)$

▷ The model checking algorithm; returns true if $v \vDash \phi$ where v' defines previous context if required.
if $\phi = \alpha$ **then return** $v \vDash^{v'} \alpha$
else if $\phi = \neg\phi$ **then return** $v \nvDash^{v'} \alpha$
else if $\phi = \phi_1 \wedge \phi_2$ **then return** $MC(v, v', \phi_1)$ & $MC(v, v', \phi_2)$
else if $\phi = \mathcal{P}_{\sim\lambda}\psi$ **then return** $MC'(v, \psi)$
end if

For clarity we have broken the algorithms for $V(v, \psi)$ into the following parts.

Algorithm 10 $MC'(v,\psi)$

▷ The model checking algorithm; returns true if $v \vDash \psi$

if $\psi = \rightarrow \phi$ **then return** $\sum_{v_i \in V(v,\psi)} \mathbb{P}(v_i|v) \sim \lambda$
else if $\psi = \phi_1 U \phi_2$ **then return** $\sum_{v_i \in V(v,\psi)} \mathbb{P}(v_i|v) \sim \lambda$
else if $\psi = \phi_1 \not\upharpoonright \phi_2$ **then return** $\mathbb{P}(V(v,\psi) \mid v) \sim \lambda$
else if $\psi = \| \phi$ **then return** $\mathbb{P}(V(v,\psi) \mid v) \sim \lambda$
end if

▷ Where $V(v,\psi)$ is defined in algorithms 11-14.

Algorithm 11 $V(v, \rightarrow \phi)$

▷ Computes $V_{\rightarrow\phi,v}$ as in the semantics of PESL

$C := \delta(v)$
$C = C \setminus \{c_i \in C \mid \forall e \in c_i.\ \nexists e' \in v.\ e' \leq e\}$ ▷ Remove non causal cells
$V = Maxim(\bigcup C)$ ▷ Find maximal configurations in C
for all $v_i \in V$ **do**
  if $\neg MC(v_i, v, \phi)$ **then**
    remove v_i
  end if
end for
return V

Algorithm 12 $V(v, \phi_1 U \phi_2)$

▷ Computes $V_{\phi_1 U \phi_2, v}$ as in the semantics of PESL

$V = V(v, \rightarrow \phi_2)$
$V' = (V(v, \rightarrow \phi_1) \setminus V)$
while $V' \neq \emptyset$ **do**
  take any $v' \in V$
  $V = V \cup V(v', \rightarrow \phi_2)$
  $V' = V' \cup (V(v', \rightarrow \phi_1) \setminus V) \setminus v'$
  return V
end while

Algorithm 13 $V(v, \phi_1 \nwarrow\!\!\!\nearrow \phi_2)$

▷ Computes $V_{\phi_1 \nwarrow\!\!\!\nearrow \phi_2, v}$ as in the semantics of PESL

$E_{\nwarrow\!\!\!\nearrow} = \{e'' \in E \setminus v \mid \exists e \in v, e' \in E.\ e \neq e' \ \&\ e \to e'' \ \&\ e' \to e''\}$
$\rho = \{X \mid X \subseteq E_{\nwarrow\!\!\!\nearrow} \ \&\ X \neq \emptyset \ \&\ X \text{ is conflict-free}\}$
for $X \in \rho$ **do**
  $V_x = \{e' \mid \exists e \in X.\ e' < e\}$
  if $V_x \setminus v = \emptyset$ **then**
    $v' = v$
    $V' = \{v'\}$
  else
    $v' = V_x \setminus v \cup \{e' \mid \exists e \in V_x.\ e' < e\ \}$
    $V' = RS(v')$
  end if
  for $v' \in V'$ **do**
    if $MC(v', v, \phi_1)$ **then**
      $V_{v''} = RS(v \cup v' \cup X)$
      for $v'' \in V_{v''}$ **do**
        if $MC(v'', v \cup v', \phi_2)$ **then**
          $V'' = V'' \cup \{v''\}$
        end if
      end for
    end if
  end for
end for
return V''

Algorithm 14 $V(v, \| \, \phi)$

▷ Computes $V_{\|\phi,v}$ as in the semantics of PESL

$E_{\|} = \{e \in E \setminus v \mid \forall e' \in v.\ e \ co \ e'\}$
$V_{max} = Maxim(E_{\|})$
for $v' \in \rho(V_{max}) \setminus \{\}$ **do**
  if $MC(v', v, \phi)$ **then**
    optimise:
    if $\nexists v'' \in V_{\|}.\ v'' \subset v'$ **then**
      $V_{\|} = V_{\|} \cup v'$
    end if
  end if
end for
return $V_{\|}$

Algorithm 15 $RS(v)$

▷ Makes v R-stopped in all the ways possible

Let V be the set of minimal R-stopped configuration v_i s.t. $v \subseteq v_i$
return V

Algorithm 16 $Maxim(X)$

▷ Computes the maximal configurations of a given set X

$\rho = \{x \mid x \subseteq X\}$
$\rho = \rho \setminus \{x \in \rho \mid \exists e, e' \in x.\ e \# e'\}$ ▷ Remove inconsistent subsets
$\rho = \rho \setminus \{x \in \rho \mid \exists x' \in \rho.\ x \subset x'$ ▷ Remove non-maximal subsets
return ρ

5.5 Conclusion

In this chapter we defined PESL interpreted on probabilistic event structures, which to our best knowledge, is the first probabilistic logic with true concurrency semantics. The main challenge in defining such logic was defining operators which could capture the new concept of progressions of configurations, arising from dealing with R-stopped configurations. In other words, typically in other existing logics, from partial order logics such as POTL to true concurrency logics such as SFL, actions or steps are at the granularity of events. However, this is no more the case in the framework of probabilistic event structures. Thus, a new notion of progress was defined which can be viewed as a maximal advancement in terms of resolving all the choices casually following a configuration.

We then described the expressivity of PESL, by first proving that PESL encodes pCTL with respect to the new concept of progress. Therefore, we showed that expressivity results of pCTL follows in the new framework. We further explained how PESL is not directly designed for expression of causality, even though some interesting properties are still expressible, such as responsive of the system (every request begin granted). We showed that PESL can capture different kinds of concurrency via different operators and more interestingly, that it can directly express synchronisation of configurations. Examples of the properties expressible by PESL include mutual exclusion, synchronisation occurs surely, for all synchronisations (some property holds), etc. Finally, a model checking algorithm for finite event structures was presented.

Chapter 6

Stable Event Structure Logic

Logics for event structures can in general be viewed from different perspectives. As a partial order model, one can consider partial order logics, such as the classic logics POTL [59] and ISTL[35]. These logics attribute properties to configurations, and configurations are partially ordered by inclusion. Therefore, causality and concurrency are not expressible directly. From a structural point of view, logics are specifically defined on event structures, such as ESL [54] and its extensions [54, 55, 52]. They attribute properties to events and describe the relation between events. Causality and concurrency between two events are therefore expressible, however, in general configurations and their interactions cannot be identified.

From the implementation of concurrency point of view, partial order logics POTL and ISTL defined over configurations of event structures follow the interleaving approach. The concurrent system is viewed as a collection of possible runs, where the formulae describe the properties of a run. Even though ISTL can distinguish between nondeterminism and concurrency, it achieves this through different groupings of possible interleavings. Thus, the well-known verification problem of state explosion ([16, 70]) persists. On the other hand, POTL can also be interpreted over event structures, however, it still lacks the power to explicitly express concurrency and conflict between events.

The other group of logics, namely, ESL and its extensions describe structural properties of event structures. Since they can describe the causal, conflict and concurrency relations between events, we find it useful as a starting point to define a logic with a similar point of view for stable event structures.

In this chapter, we introduce Stable Event Structure Logic (SEL). The aim is to better understand the temporal behaviour of stable event structures. We show that SEL is as expressive as ESL-based logics interpreted over event structures which are encoded as stable event structures. We also point out some of the conceptual differences among SEL and ESL-based logics, which arise from the differences between event structures and stable event structures plus the fact that SEL can describe properties about certain sets of events.

We start by defining some relations and predicates which are later used in the semantics of SEL.

Remark 6.0.1. We assume in this chapter that all the stable event structures are sensible (as per definition 4.1.6). However, they are not necessarily net-driven or jump-free.

6.1 Relations and Predicates

In this section a number of relations and predicates are defined which are used later in describing the semantics of SEL.

6.1.1 Relations and Predicates on Stable Event Structures

For clarity, given a set X of sets x, let $\Cup X =_{\text{def}} \bigcup_{x \in X} x$.

History

Definition 6.1.1. The *predecessors* of a consistent set of events, y, is defined as:

$$pre(y) =_{\text{def}} \{x \mid \exists e \in y.\, x \vdash_{\text{min}} e\}$$

As the definition describes, the set of sets $pre(y)$ consists of sets of events which are essential to enable the events in y. Thus, for example for a singleton $y = \{e\}$, $pre(y)$ would consist of all the sets that can enable e in a minimal way note that according to the stability condition, all these sets are in conflict with each other). Since we are dealing with sensible stable event structures and y is consistent, it follows that y can be enabled and therefore, there is a subset of $pre(y)$, for which the union of the events involved is consistent and can enable every event in y.

Definition 6.1.2. The *enabling* set of a set of events $y \in Con$ is defined as follows.

$$en(y) =_{\text{def}} \{\Cup X \mid X \subseteq pre(y) \text{ and } \forall e \in y.\ \Cup X \vdash e\}$$

Note that $\Cup X \vdash e$ implies that $\Cup X \in Con$.

The set $en(y)$ exactly refers to sets of events which are required and sufficient to enable y. The events are minimal, in the sense that all of them are required to enable something in y. This can also be seen as a collection of causal histories for each event of y.

Example 6.1.3. Consider the stable event structure depicted in figure 6.1, where $\mathcal{E} = (\{e_1,\ldots,e_{11}\},\vdash,\wp(E)\setminus\{x \mid \{\{e_1,e_2\}\subseteq x \text{ or } \{e_5,e_6\}\subseteq x \text{ or } \{e_6,e_7\}\subseteq x \text{ or } \{e_9,e_{10}\}\subseteq x\})$ and $\emptyset \vdash e_1$, $\emptyset \vdash e_2$, $\emptyset \vdash e_3$, $\emptyset \vdash e_4$, $\{e_1\} \vdash e_5$, $\{e_2\} \vdash e_5$, $\{e_3\} \vdash e_6$, $\{e_4\} \vdash e_7$ and so on.

We then have:
$pre(\{e_5\}) = \{\{e_1\},\{e_2\}\}$
$pre(\{e_9\}) = \{\{e_1,e_5\},\{e_2,e_5\},\{e_3,e_6\}\}$
$pre(\{e_9,e_10\}) = pre(\{e_8,e_{10}\}) = \{\{e_1,e_5\},\{e_2,e_5\},\{e_3,e_6\},\{e_4,e_7\}\}$
$en(\{e_5\}) = pre(\{e_5\})$ (as is the case for any singleton)
$en(\{e_9,e_{10}\}) = \{\}$
$en(\{e_8,e_{10}\}) = en(\{e_{11}\}) = \{\{e_1,e_5,e_4,e_7\},\{e_2,e_5,e_4,e_7\}\}$

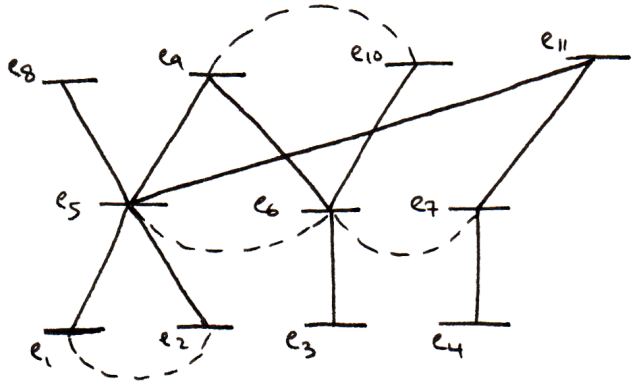

Figure 6.1: Stable Event structure $\mathcal{E}_1$

Enabling - Next

Definition 6.1.4. A set $y \in Con$ *cooperatively and immediately enables* a set $y' \in Con$ represented by $y \to y'$ iff

$$y \neq \emptyset \text{ and } \nexists e \in y'.\ \emptyset \vdash e \text{ and}$$

$$\exists x \in en(y).\ y \cup x \in en(y') \text{ and } y' \cap (y \cup x) = \emptyset$$

This relation captures the notion of events taking part in enabling another event, in the next step, as roughly depicted in figure 6.2.

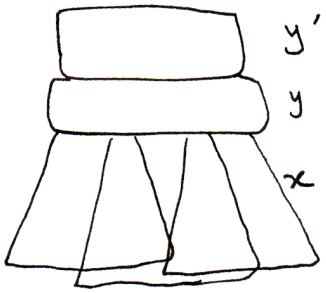

Figure 6.2: A rough sketch of $y \rightarrow y'$, where $x \in en(y)$

The definition implies that every event in y, if enabled, takes part in enabling an event in y' (as $y \cup x \in en(y')$). If y' is a singleton and y has concurrent events, then this also captures the concurrent events in y synchronising together to cause the single event in y'. The condition $y' \cap (y \cup x) = \emptyset$ ensures that y' is in the next step in future (of the most recent events).The first condition is to separate the concurrent initial events from those which are caused by other events. It is also important to emphasise that every event in y' is fully enabled by some subset of $y \cup x$.

Example 6.1.5. For $\mathcal{E}_1$ as in figure 6.1 we have:
$\{e_5, e_7\} \rightarrow \{e_8, e_{10}\}$
$\{e_5, e_7\} \rightarrow \{e_{11}\}$ (capturing synchronisation in this case)

Definition 6.1.6. A set y' is *potentially enabled* by set y, represented by $y \rhd y'$ iff:

$$y \neq \emptyset \ \&\ \exists x \in en(y).\ \exists x' \in en(y').\ x \cup y \subseteq x' \ \&$$

$$\nexists e \in (x' \cup y') \setminus (x \cup y).\ \emptyset \vdash e \ \ \&\ y' \cap (x \cup y) = \emptyset$$

This relation simply captures the fact that if y is enabled, then y' can be enabled without requiring any other event which is not in the past or future of events of y. In other

words, the occurrence of y *can be sufficient* for enabling y'. Also, all events of y should have a role in enabling y'. This is roughly depicted in figure 6.3.

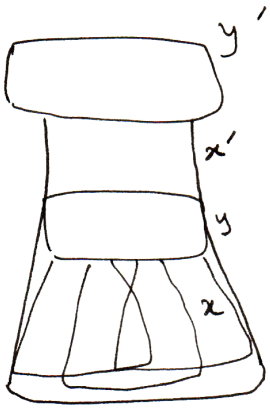

Figure 6.3: A rough sketch of $y \rhd y'$, where $x \in en(y), x' \in en(y')$

Enabling-partial

Definition 6.1.7. A set (sometimes) *partially enables* another, represented by $y \leq y'$ iff

1. $y \cup y' \in Con$
2. $\forall e \in y, \exists e' \in y'.\ e \in \Cup pre(e')$
3. $\forall e' \in y', \exists e \in y.\ e \in \Cup pre(e')$

This means that there is *some* configuration where y is required, though not necessarily sufficient, for enabling y'. The first condition guarantees that y and y' are consistent.

Definition 6.1.8. A set (sometimes) *partially enables* another in the *next* step, represented by $y \mapsto y'$ iff

1. $y \cup y' \in Con$
2. $\forall e \in y, \exists e' \in y'$ s.t. $e \rightarrow e'$
3. $\forall e' \in y', \exists e \in y$ s.t. $e \rightarrow e'$

where $e \rightarrow e'$ iff $e \in \Cup pre(e')$ and $\nexists e''.\ (e \in \Cup pre(e'')$ and $e'' \in \Cup pre(e'))$.

This means that there is *some* configuration where y is required as the immediate history of y', though not necessarily sufficient, for enabling it. The first condition guarantees that events of y are not from different possible causes of any event of y' and also that events of y and y' are consistent.

Example 6.1.9. Consider the stable event structure in figure 6.4. We have:

$y_1 \leq y_4$

$y_1 \rhd y_4$

$y_2 \Mapsto y_4$

$y_2 \leq y_4$

$y_3 \rightarrow y_4$

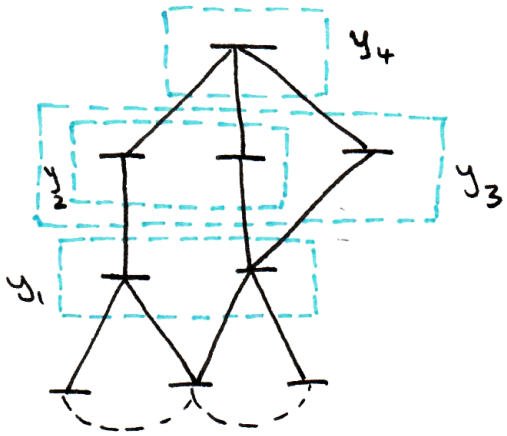

Figure 6.4: Stable Event Structure $\mathcal{E}_2$

Conflict

Definition 6.1.10. The *consistency* and *conflict* relations between two finite sets of events are defined as follows.

$$y \text{ Con } y' \text{ iff } y \cup y' \in Con$$

$$y \# y' \text{ iff } y \cup y' \notin Con$$

Concurrency

Definition 6.1.11. Two events are defined to be *independent* of each other when:

$$e \; indep \; e' \text{ iff } e \notin \Cup pre(e') \;\&\; e' \notin \Cup pre(e)$$

Definition 6.1.12. The *concurrency* relation between two sets of events should capture the fact that the events of the sets are independent and can occur together, therefore,

$$y \parallel y' \text{ iff } (\forall e \in y, e' \in y' .\; e \; indep \; e') \;\&\; y \cup y' \in Con$$

6.1.2 Relations on Event Structures

Next - Partial Cause

Recalling definition 5.1.2, a *next* event of an event e is defined as follows.

$$e \rightarrow e' \text{ iff } e \leq e' \;\&\; \nexists e''.\, e \leq e'' \;\&\; e'' \leq e'$$

Concurrency

Definition 6.1.13. Events e and e' are concurrent, denoted by $e \; co \; e'$ iff $e \not\leq e' \;\&\; e' \not\leq e \;\&\; \neg e \# e'$.

6.2 Syntax and Semantics

In this section we present the syntax and semantics for SEL. The formulae of SEL are interpreted over special subsets of events, namely, the *infixes* of stable event structures as defined below.

Definition 6.2.1. An *infix* y of an event structure $\mathcal{E}$, is a subset of a configuration $v \in \mathcal{V}(\mathcal{E})$ such that $\forall e, e' \in y.\, \forall e'' \in E.\, e \in \Cup pre(e'') \;\&\; e'' \in \Cup pre(e') \Rightarrow e'' \in y$.

Infixes can be seen as cuts of configurations, or configurations in the future of configurations. Interpreting SEL over infixes provides us with the flexibility to capture properties of configurations or sets of events which can occur together or even single events. This is achieved using two types of limit (maximal/minimal) operators, as we shall define in the semantics of SEL.

The syntax of SEL is defined as follows,

$$\phi := p \mid \forall\phi \mid \neg\phi \mid \phi \wedge \phi \mid \lceil\!\lceil \phi \rceil\!\rceil \mid \lfloor\!\lfloor \phi \rfloor\!\rfloor \mid \lceil \Box_R \phi \rceil \mid \lfloor \Box_R \phi \rfloor$$

where p is an atomic proposition, R denotes a relation in the set of relations $\{\rhd, \lhd, \rightarrow, \leftarrow, \leq, \geq, \mapsto, \leftmapsto, \#, \|\}$ and $\lceil\!\lceil \; \rceil\!\rceil / \lfloor\!\lfloor \; \rfloor\!\rfloor$ and $\lceil \; \rceil / \lfloor \; \rfloor$ capture two different notions of maximality and minimality for ϕ and $\Box_R\phi$ formulae, respectively.

We now define the semantics of SEL. In the following, let $\mathcal{Q} : \wp(E) \rightarrow \wp(AP)$ be a validator function for the set of atomic propositions AP.

– **Basics**

6.2.2. $y \vDash p$ iff $p \in Q(y)$

6.2.3. $y \vDash \forall\phi$ iff $\forall e \in y.\ \{e\} \vDash \phi$

6.2.4. $y \vDash \neg\phi$ iff $y \nvDash \phi$

6.2.5. $y \vDash \phi \wedge \phi'$ iff $y \vDash \phi$ and $y \vDash \phi'$

– **Limits**

6.2.6. $y \vDash \lceil\!\lceil \phi \rceil\!\rceil$ iff $y \vDash \phi$ & $\nexists y'.\ y' \vDash \phi$ & $y \subset y'$
In other words, y is a maximal set satisfying ϕ.

6.2.7. $y \vDash \lfloor\!\lfloor \phi \rfloor\!\rfloor$ iff $y \vDash \phi$ & $\nexists y'.\ y' \vDash \phi$ & $y' \subset y$
In other words, y is a minimal set satisfying ϕ.

6.2.8. $y \vDash \lceil \Box_R \phi \rceil$ iff $\forall y' \in Ry.\ y'$ is maximal in $Ry \Rightarrow y' \vDash \phi$
In other words, all maximal sets y' which are in R relation with y satisfy ϕ.

6.2.9. $y \vDash \lfloor \Box_R \phi \rfloor$ iff $\forall y' \in Ry.\ y'$ is minimal in $Ry \Rightarrow y' \vDash \phi$
In other words, all minimal sets y' which are in R relation with y satisfy ϕ.

Remark 6.2.10. For convenience, in the following let $\Box\phi$ denote $\Box_{\triangleright}\phi$ and let $\overline{\Box}_R\phi$ denote the formulae for the reverse of R. For example, $\overline{\Box}\phi$ denotes $\Box_{\triangleleft}\phi$.

The intuition behind defining the above operators is clarified in the rest of this chapter, by giving a comparison to existing ESL-based logics and giving further examples of SEL formulae.

6.3 Logics on Event Structures

In this section we briefly introduce some of the other logics which can be interpreted over event structures. In particular ESL-based logics are presented and later compared with SEL.

The first temporal logic specifically defined on event structures is Sequential Event Structure Logic (SESL) defined on the subclass of n-agent event structures by Lodaya and Thiagarajan [47]. An n-agent event structure consists of n sequential agents, meaning that each agent is an event structure whose events are either related by causality or conflict, but are not concurrent. Therefore, such event structures can capture

sequentiality and non-determinism. Concurrency is then captured via communication between the agents. Thus, while each agent has its local state, communication between the agents defines a global partial order. While SESL has true concurrency semantics and can express properties such as safety and possibility properties (both at the system or agent level), it cannot describe eventuality.

Event Structure Logic (ESL) was introduced by Penczek [54] (around the same time that a similar logic was defined by Mukund and Thiagarajan [48]), where a modality for conflict was introduced. ESL has the power to express properties such as conflict freeness and inevitability, but it cannot describe eventuality and properties related to immediate successors/predecessors. Thus, Discrete Event Structure Logic (DESL) was defined to extend the expressivity of ESL by describing properties of successors/predecessors [54]. Finally, ESL[c] [52] extends ESL by adding a concurrency operator while ESL[δ] [55] allows for marking a particular run of interest.

We now present the syntax and semantics of ESL-based logics.

6.3.1 Syntax of ESL-based logics [56]

The syntax of a formula φ of ESL is as follows, where AP is the set of atomic propositions and $p \in AP$.

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \Box\varphi \mid \overline{\Box}\varphi \mid \Box_{\#}\varphi$$

The dual formulas $\Diamond\varphi$, $\overline{\Diamond}\varphi$ and $\Diamond_{\#}\varphi$ are defined in the usual way. A *model* for ESL formulas is simply $\mathcal{M} = (\mathcal{E}, Q)$ where $\mathcal{E} = (E, \leq, \#)$ is an event structure and $Q : E \to \wp AP$ is a valuation function. Then for such a model the semantics of ESL is as follows.

- $e \Vdash p$ iff $p \in Q(e)$, where $p \in AP$.
- $e \Vdash \neg\varphi$ iff $e \nVdash \varphi$.
- $e \Vdash \varphi \wedge \psi$ iff $e \Vdash \varphi$ and $e \Vdash \psi$.
- $e \Vdash \Box\varphi$ iff $\forall e' \in E.\ e \preceq e' \Rightarrow e' \Vdash \varphi$.
- $e \Vdash \overline{\Box}\varphi$ iff $\forall e' \in E.\ e' \preceq e \Rightarrow e' \Vdash \varphi$.
- $e \Vdash \Box_{\#}\varphi$ iff $\forall e' \in E.\ e' \# e \Rightarrow e' \Vdash \varphi$.

The logic ESL[c] extends ESL by a concurrency operator, $\Box_c$, with the following semantics.

- $e \Vdash \Box_c \varphi$ iff $\forall e' \in E.\ e' co\ e \Rightarrow e' \Vdash \varphi$.

Finally, the logic DESL extends ESL by forward and backward next operators, $\otimes$ and $\overline{\otimes}$ with the following semantics.

- $e \Vdash \otimes$ iff $\forall e' \in E.\ e \rightarrow e' \Rightarrow e' \Vdash \varphi$.
- $e \Vdash \overline{\otimes}$ iff $\forall e' \in E.\ e' \rightarrow e \Rightarrow e' \Vdash \varphi$.

6.4 Comparison and Expressivity

In this section we present a full comparison between SEL and ESL-based logics, namely, ESL, DESL and ESL[c]. The comparison describes the expressive power of ESL and we explain this further by giving some examples.

The comparison is carried out, in two parts. The first part shows that SEL can express properties defined by ESL-based logics. Recall that since the event structures on which ESL-based logics are defined are a simpler subclass of stable event structures (where each event has only one possible set of events causing it), there is a natural mapping from event structures to stable event structures . Therefore, for each ESL formula, we present an SEL formula, proving that they are satisfied by the same events. All the proofs are presented at the end of this chapter.

The second part focuses more on the concepts captured by SEL and ESL-based formulae. Namely, it studies the main difference caused by working with sets of events as denotations of formulae and the capability of the logics to express synchronisation of events and certain configuration-related properties.

6.4.1 From ESL-based logics to SEL

Here, for each ESL-based formula, we present an SEL formula with the same satisfying set of events. For some cases, extra operators need to be defined as explained below.

ESL to SEL

Proposition 6.4.1. A formula φ of ESL is translated to $f(\varphi)$ in the following manner, where $e \Vdash_{ESL} \varphi$ iff $\{e\} \Vdash_{SEL} f(\varphi)$.

- $p : f(p) = \forall p$
- $\neg : f(\neg\varphi) = \neg f(\varphi)$
- $\wedge : f(\varphi \wedge \varphi') = f(\varphi) \wedge f(\varphi')$
- $\Box : f(\Box\varphi) = \lceil \Box_{\leq} \forall f(\varphi) \rceil$ (Note that this is not directly translatable with causality operators 6.1.4 and 6.1.6 (conjecture 6.6.4))
- $\overline{\Box} : f(\overline{\Box}\varphi) = \lceil \overline{\Box} \forall f(\varphi) \rceil$
- $\Box_{\#} : f(\Box_{\#}\varphi) = \lfloor \Box_{\#} \exists f(\varphi) \rfloor$

Proof. Refer to proof 6.6.1. □

Thus, in summary, SEL can be as expressive as ESL. It is important to note that, although SEL can express ESL formulae, the formulae given above do not represent the same notion for stable event structures, simply because stable event structures allow for disjunction of causes. For example, $\overline{\Diamond}\varphi$ in ESL, denotes that there is an event with φ property in the past, thus such event is always needed. However, its equivalent in SEL would only imply that there is some configuration for which such event exists and is needed, but this is not necessarily the case for all configurations. This property (necessity of an event φ in past) is expressible in SEL by a different formula, namely, $\lceil \overline{\Box} \exists f(\varphi) \rceil$.

ESL[c] to SEL

Proposition 6.4.2. The formula $\Box_c\varphi$ of ESL[c] is can be captured by ESL in the following manner, where $f(\varphi)$ is defined in proposition 6.4.1.

$$e \Vdash_{ESL} \Box_c\varphi \text{ iff } \{e\} \Vdash_{SEL} \lceil \Box_{\Rightarrow} \forall f(\varphi) \rceil$$

Proof. Refer to proof 6.6.2. □

DESL to ESL

Proposition 6.4.3. DESL can be captured by ESL in the following manner, where $f(\varphi)$ is defined in proposition 6.4.1.

$$e \Vdash_{ESL} \otimes\varphi \text{ iff } \{e\} \Vdash_{SEL} \lceil \Box_{\Mapsto} \forall f(\varphi) \rceil$$

$$e \Vdash_{ESL} \overline{\otimes}\varphi \text{ iff } \{e\} \Vdash_{SEL} \lfloor \overline{\Box}_{\rightarrow} \forall f(\varphi) \rfloor$$

Proof. Refer to proof 6.6.3. □

6.4.1.1 SEL to ESL-based logics

There is an obvious difference between SEL and ESL-based logics, namely, SEL is defined on infixes, a particular subsets of events, whereas ESL is defined on events. This difference is enough to say that ESL cannot express SEL formulae. The main advantage is that working with sets of events gives the power of grouping events into different sets, which is highly desirable when working with stable event structures and not possible with ESL-based logics. This grouping effect is what causes the major difference in expressivity of SEL and ESL-based logics. Here we explain different aspects of this difference.

Synchronising of events

Consider stable event structure $\mathcal{E}_0'$ and its associated event structure $\mathcal{E}_0$ depicted in figure 6.5.

$$\mathcal{E}_0 = (E = \{e_1, \ldots, e_6\}, \{e_1 \leq e_5, e_2 \leq e_5, e_3 \leq e_6, e_4 \leq e_6\}, \emptyset),$$
$$\mathcal{E}_0' = (E, \{\emptyset \vdash e_1, e_2, e_3, e_4, \{e_1, e_2\} \vdash e_5, \{e_3, e_4\} \vdash e_6\}, \wp(E)),$$

where $Q(e_5) = Q(e_6) = \{p\}$.

As can be seen, events e_1 and e_2 synchronise to enable e_5 and e_3 and e_4 synchronise to enable e_6. Now suppose we want to find the events that synchronise to achieve an event with property p. This can easily be expressed in SEL by $\varphi = \lfloor \Box_{\rightarrow} [\![\forall p]\!] \rfloor$ for which $\{e_1, e_2\} \vDash \varphi$ and $\{e_3, e_4\} \vDash \varphi$.However, any formula of ESL-based logics, will either be true for all events e_1, e_2, e_3 and e_4 (such as $\otimes p$) or for neither. This is due to the symmetry of the given event structure which is maintained in the translation. This

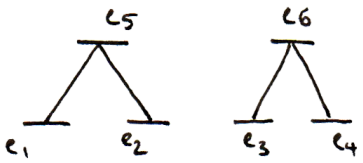

Figure 6.5: $\mathcal{E}_0$

example clearly demonstrates the grouping effect mentioned above, in addition to the expressibility of synchronisation properties by SEL.

The same concept is captured by the fact that ESL-based logic cannot distinguish between the following event structures, represented in figure 6.6.

$$\mathcal{E}_1 = (E, \{e_1 \leq e_5, e_2 \leq e_5, e_3 \leq e_6, e_4 \leq e_6\}, \emptyset),$$

$$\mathcal{E}_2 = (E, \{e_1 \leq e_5, e_2 \leq e_5, e_3 \leq e_5, e_4 \leq e_6\}, \emptyset)$$

where $E = \{e_1, \ldots, e_6\}$ and $Q(e_5) = Q(e_6) = \{p\}$. This can be proved by exhaustive inspection of formulae of ESL-based formulae of depth 2 or 3 at most.

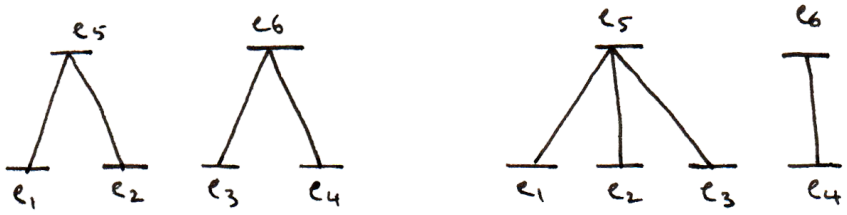

Figure 6.6: $\mathcal{E}_1$ (left) and $\mathcal{E}_2$ (right)

Reachability via Cooperation

Another major difference between SEL and ESL-based logics is that SEL can express reachability properties. For example, given a set of events, SEL can express what other events can be reached which require all the events of that set. The formula of the form $\lfloor \overline{\Diamond}(\exists p_1 \wedge \exists p_2 \wedge \ldots) \wedge \neg\exists(\neg p_1 \vee \neg p_2 \vee \ldots)) \rfloor$ describes what minimal infixes can be reached with events p_i. Alternatively, the maximal of the same formula yields the

biggest reachable infixes. However, even for a stable event structure mapped from an event structure, $e_i \models_{ESL} \overline{\Diamond} p_i$, $\cup\{e_i\}$ does not express the same, as it may not include other events possibly required to enable e_i. In other words, ESL-based logics can only express partial causality and not full causality, i.e. cooperation of events, as illustrated previously for synchronisation of events.

Steps

SEL can also express what events form a step in enabling another set of events. Consider stable event structure $\mathcal{E}_3'$ mapped from the event structure $\mathcal{E}_3$ as below, depicted in figure 6.7.

$$\mathcal{E}_3 = (E = \{e_1, \ldots, e_5\}, \{e_1 \leq e_3, e_2 \leq e_4, e_3 \leq e_5, e_4 \leq e_5\}, \emptyset),$$
$$\mathcal{E}_3' = (E, \{\emptyset \vdash e_1, e_2, \{e_1\} \vdash e_3, \{e_2\} \vdash e_4, \{e_3, e_4\} \vdash e_5\}, \wp(E)).$$

where $Q(e_5) = p$. Then, the SEL formula $\lfloor \Diamond \forall p \rfloor$, yields all the sets which are sufficient as steps towards enabling e_5, such as $\{e_3, e_4\}$. However, ESL-based logics cannot identify this but only all the events that are required, i.e. e_1, e_2, e_3 and e_4, for all of which $\Diamond p$ holds.

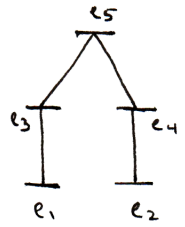

Figure 6.7: $\mathcal{E}_3'$

Conflict

Another obvious difference between expressivity of SEL and ESL results form the difference between the models they are defined on, in terms of consistency and conflict. Suppose we are dealing with a stable event structure with more than 2 events being inconsistent with each other. A simple example would be $\mathcal{E}_4 = (E = \{e_1, e_2, e_3\}, \emptyset \vdash e_1, e_2, e_3, Con)$ where $Con = \wp(E) \setminus \{e_1, e_2, e_3\}$ (figure 6.8). Such stable event structure and its isomorphic prime event structure cannot be mapped to an event structure, as the inconsistencies here cannot be mapped into a binary conflict relation. Thus, SEL can express conflict properties related to sets being in conflict, whereas ESL-based logics

only describe binary conflict relations.

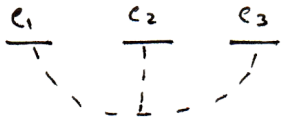

Figure 6.8: $\mathcal{E}_4$

Disjunctive Causes and Constant Requirement

The grouping capability of SEL becomes more evident when it comes to expressing properties related to disjunctive causes. Consider the following stable event structure, $\mathcal{E}_5 = (E = \{e_1, e_2, e_3, e_4\}, \{\emptyset \vdash e_1, e_2, e_3, \{e_1, e_3\} \vdash e_4, \{e_2, e_3\} \vdash e_4\}, Con)$, where $Con = \wp(E) \setminus \{x \mid \{e_1, e_2\} \subseteq x\}$ and $Q(e_4) = p$ (figure 6.9).

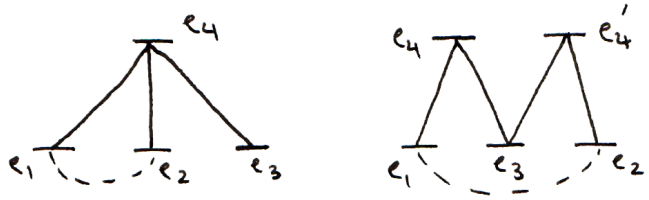

Figure 6.9: $\mathcal{E}_5$

Now suppose we want to know what is required to enable event 4. In SEL this is given by $\lceil \Diamond [\![\forall p]\!] \rceil$, holding for $\{1,3\}$ and $\{2,3\}$. Moreover, with SEL one can distinguish that some events are *always* required to enable another, i.e. they are in every possible history of that event. For example, in the previous case, suppose $Q(3) = q$, then $\lceil \Diamond [\![\forall p]\!] \rceil \rightarrow \exists q$ indicates that an event q is always required to enable an event p.

Remark 6.4.4. The stable event structure $\mathcal{E}_5$ is translated into the event structure $\mathcal{E}'_5 = (E' = \{e_1, e_2, e_3, e_4, e'_4\}, \{e_1 \leq e_4, e_2 \leq e'_4, e_3 \leq e_4, e_3 \leq e'_4\}, \{e_1 \# e_2\})$ and it follows that $Q(e_4) = Q(e'_4)$. Note that in $\mathcal{E}'$ $e_1 \# e_2$ and $e_1 \leq 4$ imply $e_1 \# e_4$ and similarly $e_2 \# e_4$. Thus, as an event structure, $\mathcal{E}'$ cannot capture the complete relation between events of $\mathcal{E}$. Moreover, grouping of events is not expressible by ESL-base logics.

6.5 Model Checking

The model checking problem for SEL is defined as the process of determining if an infix u of a stable event structure $\mathcal{E}$ satisfies a formula ϕ of SEL. In this section we present the model checking algorithm, under the assumption that $\mathcal{E}$ is finite or that we are considering a finite prefix of an infinite stable event structure.

The model checking process presented by algorithm 19, called $MC(\phi)$, returns a list of infixes for which ϕ holds and for this calculation we need to populate the list of all possible infixes, as presented by algorithm 17. The model checking algorithm is given for relations $\leq, \rhd, \rightarrow, \leftarrow, \#$ and $\|$ and the algorithm can be extended to include $\leq, \geq, \mapsto, \dashv$ in a similar manner. All these algorithms are for a given stable event structure, $\mathcal{E} = (E, \vdash, Con)$.

Algorithm 17 *Inf*

▷ Creates all the infixes of a stable event structure and expands Con to exclude implied conficlits

$Infixes = \emptyset$
$subs = \wp(E)$
for all $s \in subs$ **do**
  $flag = \text{true}$
  if $s \in Con$ **then**
    if $En?(s) \neq \emptyset$ **then**
      for all $e \in s$ **do**
        for all $e' \in s.\ e \neq e'$ **do**
          for all $e'' \in E$ **do**
            if $e \in \Cup Pre(e'') \wedge e'' \in \Cup Pre(e') \wedge e'' \notin s$ **then**
              $flag = \text{false}$
              break
            end if
          end for
          if $flag =_? \text{false}$ **then**
            break
          end if
        end for
        if $flag =_? \text{false}$ **then**
          break
        end if
      end for
      if $flag =_? \text{true}$ **then**
        $Infixes = Infixes \cup \{s\}$
      end if
    else
      $Con = Con \setminus s$
    end if
  end if
end for

Algorithm 18 $En?(s)$

▷ Given a set, returns empty if it cannot be an infix due to inconsistencies or otherwise its enabling set.

for $i := 1, |s|$ **do**
  $Pres[i] = Pre(\{e_i\})$, where $e_i \in s$
  $Size[i] = |Pres[i]|$
end for
while $\exists i.\ Size[i] = 1$ **do**
  $Fixed = Fixed \cup x$, where $x \in pre \in Pres[i]$
  if $Fixed \notin Con$ **then**
    return $\emptyset$
  end if
  for all $j \neq i \wedge Size[j] > 1$ **do**
    for all $u \in Pres[j]$ **do**
      if $Conf(u,x)$ **then**
        $Pres[j] = Pres[j] - u$
        $Size[j] = Size[j] - 1;$
      end if
    end for
  end for
end while
for all $i.\ Size[i] > 1$ **do**
  $Var[i] = Pres[i]$
  $Enums = Enum(Var, 1)$
end for
if $Enums = \emptyset$ **then**
  return $\emptyset$
else
  for all $en \in Enums$ **do**
    $result = result \cup \{en \cup Fixed\}$
  end for
  return $result$
end if

Algorithm 19 $MC(\phi)$ - Part 1

▷ The model checking algorithm, returns the set of all the infixes for which ϕ holds.

$result = \emptyset$

$flag =$ true

if $\phi = p$ **then**
  for all u **do**
    if $p \in Q(u)$ **then**
      $result = result \cup \{u\}$
    end if
  end for
else if $\phi = \forall\phi'$ **then**
  for all u **do**
    for all $e \in u$ **do**
      if $\{e\} \notin MC(\phi')$ **then**
        $flag =$ false
        break
      end if
    end for
    if $flag =_?$ true **then**
      $result = result \cup \{u\}$
    end if
  end for
else if $\phi = \neg\phi'$ **then**
  for all u **do**
    if $u \notin MC(\phi')$ **then**
      $result = result \cup \{u\}$
    end if
  end for
else if $\phi = \phi' \wedge \phi''$ **then**
  for all u **do**
    if $u \in MC(\phi') \wedge u \in MC(\phi'')$ **then**
      $result = result \cup \{u\}$
    end if
  end for

Algorithm 20 $MC(\phi)$ - Part 2

else if $\phi = \lceil\!\lceil \phi' \rceil\!\rceil \vee \phi = \lfloor\!\lfloor \phi' \rfloor\!\rfloor$ **then**
  for all u **do**
    if $u \in MC(\phi')$ **then**
      $result = result \cup \{u\}$
    end if
  end for
  if $\phi = \lceil\!\lceil \phi' \rceil\!\rceil$ **then**
    $result = Max(result)$
  else
    $result = Min(result)$
  end if
else if $\phi = \lceil \Box_R \phi' \rceil \vee \phi = \lfloor \Box_R \phi' \rfloor$ **then**
  for all u **do**
    if $\phi = \lceil \Box_R \phi' \rceil$ **then**
      $result = Max\ (Find(R,u))$
    else
      $result = Min\ (Find(R,u))$
    end if
    for all $u' \in result$ **do**
      if $u' \notin MC(\phi')$ **then**
        $flag = \text{false}$
        break
      end if
    end for
    if $flag =_? \text{true}$ **then**
      $result = result \cup \{u\}$
    end if
  end for
end if
return *result*

Algorithm 21 $Find(R,u)$

▷ Given a causal operator and an infix u, returns the set of all the infixes in relation with u

$result = \emptyset$
for all u' **do**
  if $R = \triangleright$ & $Future(u,u')$ **then**
    $result = result \cup \{u'\}$
  else if $R = \triangleleft$ & $Future(u',u)$ **then**
    $result = result \cup \{u'\}$
  else if $R = \rightarrow$ & $Next(u,u')$ **then**
    $result = result \cup \{u'\}$
  else if $R = \leftarrow$ & $Next(u',u)$ **then**
    $result = result \cup \{u'\}$
  else if $R = \#$ & $Conf(u,u')$ **then**
    $result = result \cup \{u'\}$
  else if $R = \|$ & $Par(u,u')$ **then**
    $result = result \cup \{u'\}$
  end if
end for
return $result$

Algorithm 22 $Future(u,u')$

▷ Determines if $u \triangleright u'$

if $u \neq \emptyset$ **then**
  for all $x \in En?(u)$ **do**
    for all $x' \in En?(u')$ **do**
      if $x \cup u \subseteq x' \wedge \nexists e \in (x' \cup u') - (x \cup u).\ \emptyset \vdash e \wedge y' \cap (x \cup y) = \emptyset$ **then**
        return $true$
      end if
    end for
  end for
end if
return $false$

Algorithm 23 $Next(u,u')$

▷ Determines if $u \rightarrow u'$

if $u \neq \emptyset \wedge \nexists e \in y'.\emptyset \vdash e$ **then**
  for all $x \in En?(u)$ **do**
    if $x \cup u \in En?(u') \wedge u' \cap (u \cup x) = \emptyset$ **then**
      return true
    end if
  end for
end if
return false

Algorithm 24 $Conf(u,u')$

▷ Determines if $u\#u'$

return $u \cup u' \notin Con?$

Algorithm 25 $Par(u,u')$

▷ Determines if $u \parallel u'$

if $\neg Conf(u,u')$ **then**
  for all $e \in u$ **do**
    for all $e' \in u'$ **do**
      if $e \in \Cup Pre(e') \vee e' \in \Cup Pre(e)$ **then**
        return false
      end if
    end for
  end for
  return true
else
  return false
end if

Algorithm 26 $Pre(u)$

▷ Given an infix u, returns $Pre(u)$

$result = \emptyset$
for all $e \in u$ **do**
  for all $x.\ x \vdash_{\min} e$ **do**
    $result = result \cup \{x\}$
  end for
end for
return *result*

Algorithm 27 $Enum(X, i)$

▷ Given an array of *Pre* of some events, returns all possible combinations of their *Pre*'s.

$result = \emptyset$
$enumRest = \emptyset$
$flag = \text{true}$
if $i \leq |X|$ **then**
  $enumRest = Enum(X, i+1)$
  if $enumRest = \emptyset \wedge i < |X|$ **then**
    return $\emptyset$
  end if
else
  return $\emptyset$
end if
for all $pre \in X[i]$ **do**
  if $enumRest = \emptyset$ **then**
    $result = result \cup \{pre\}$
  else
    $flag = \text{false}$
    for all $en \in enumRest$ **do**
      if $\neg Conf(pre, en)$ **then**
        $result = result \cup \{pre \cup en\}$
        $flag = \text{true}$
      end if
    end for
  end if
end for
if $flag =_? \text{false}$ **then**
  return $\emptyset$
else
  return *result*
end if

Algorithm 28 *Max(list)*

▷ Given a set of infixes, returns those who are maximal

for all $u \in list$ **do**
  for all $u' \in list.\ u \neq u'$ **do**
    if $u \subset u'$ **then**
      $list = list \setminus \{u\}$
    end if
  end for
end for
return *list*

Algorithm 29 *Min(list)*

▷ Given a set of infixes, returns those who are minimal

for all $u \in list$ **do**
  for all $u' \in list.\ u \neq u'$ **do**
    if $u' \subset u$ **then**
      $list = list \setminus \{u\}$
    end if
  end for
end for
return *list*

6.6 Proofs

Proof 6.6.1. Prove that a formula φ of ESL is translated to $f(\varphi)$ as in §6.4.1 , where $e \Vdash_{ESL} \varphi$ iff $\{e\} \Vdash_{SEL} f(\varphi)$. This can be proved by induction on the formulae.

- Base case $f(p) = \forall p$. Clearly, if $e \vDash_{ESL} p$, then for the set $\{e\}$, it holds that all of its events having property p. Thus, $\{e\} \vDash_{SEL} \forall p$. The backwards is also as trivial.

- $f(\neg\varphi) = \neg f(\varphi)$.
  $e \vDash_{ESL} \neg\varphi \quad \leftrightarrow e \nvDash_{ESL} \varphi$
  $\leftrightarrow \{e\} \nvDash_{SEL} f(\varphi)$ by I.H.
  $\leftrightarrow \{e\} \vDash_{SEL} \neg f(\varphi)$.

- $f(\varphi \wedge \varphi') = f(\varphi) \wedge f(\varphi')$
  $e \vDash_{ESL} \varphi \wedge \varphi' \quad \leftrightarrow e \vDash_{ESL} \varphi$ and $e \vDash_{ESL} \varphi'$
  $\leftrightarrow \{e\} \vDash_{SEL} f(\varphi)$ and $\{e\} \vDash_{SEL} f(\varphi')$ by I.H.
  $\leftrightarrow \{e\} \vDash_{SEL} f(\varphi) \wedge f(\varphi')$.

- $f(\Box\varphi) = \lceil \Box_{\leq} \forall f(\varphi) \rceil$. Proved in Induction Step 6.6.1.

- $f(\overline{\Box}\varphi) = \lceil \overline{\Box} \forall f(\varphi) \rceil$. Proved in Induction Step 6.6.2.

- $f(\Box_{\#}\varphi) = \lfloor \Box_{\#} \exists \rightarrow \exists f(\varphi) \rfloor$. Proved in Induction Step 6.6.3.

Proof 6.6.2. Prove that a formula φ of ESL[c] is translated to $f(\varphi)$ as in §6.4.1 and §6.4.1 , where $e \Vdash_{ESL[c]} \varphi$ iff $\{e\} \Vdash_{SEL} f(\varphi)$. This can be proved by induction on the formulae.

As ESL[c] extends ESL by $\Box_c$ operator, the proof is the same as proof 6.4.1, with one added case, as follows.

- $f(\Box_c)\varphi = \lfloor \Box_{\|} \forall f(\varphi) \rfloor$. Proved in Induction Step 6.6.4.

Proof 6.6.3. Prove that a formula φ of DESL is translated to $f(\varphi)$ as in §6.4.1 and §6.4.1 , where $e \Vdash_{DESL} \varphi$ iff $\{e\} \Vdash_{SEL} f(\varphi)$. This can be proved by induction on the formulae.

As DESL extends ESL by $\otimes$ and $\overline{\otimes}$ operators, the proof is the same as proof 6.4.1, with two added cases, as follows.

- $f(\otimes)\varphi = \lceil \Box_{\Rightarrow} \forall f(\varphi) \rceil$. Proved in Induction Step 6.6.5.

- $f(\overline{\otimes}\varphi) = \lfloor \overline{\Box}_{\rightarrow} \forall f(\varphi) \rfloor$. Proved in Induction Step 6.6.6

Induction Step 6.6.1. $e \vDash_{ESL} \Box\varphi \leftrightarrow \{e\} \vDash_{SEL} \lceil \Box_{\leq} \forall f(\varphi) \rceil$
R.H.S means: $\forall y.\ \{e\} \leq y \Rightarrow \forall e' \in y \{e'\} \vDash f(\varphi)$, where by applying the definition of $\leq$ for sets of events, $\{e\} \leq y$ iff:

1. $\{e\} \cup y \in Con$
2. $\exists e' \in y$ s.t. $e \in \Cup pre(e')$
3. $\forall e' \in y.\ e \in \Cup pre(e')$

Note that y cannot be an empty set, as otherwise $\{e\} \not\leq y$. Therefore, if 3 holds, then so does 2.
($\Rightarrow$) $e \vDash_{ESL} \Box\varphi \Rightarrow \{e\} \vDash_{SEL} \lceil \Box_{\leq} \forall f(\varphi) \rceil$
L.H.S unfolds to: $\forall e'.\ e \leq e' \Rightarrow e' \vDash \varphi$ by semantic rules of ESL.
Based on the translation of relations between event structureswe have:

$$\forall e'.\ e \in \Cup pre(e') \Rightarrow e' \vDash \varphi.$$

Thus, for any set y consisting of such e' events, it holds that $\forall e' \in y.\ e' \vDash \varphi$, or by I.H. $\forall e' \in y.\ \{e'\} \vDash f(\varphi)$, which is denoted by $y \vDash \forall f(\varphi)$.
Also, such y sets are exactly those for which $\{e\} \leq y$ as they satisfy conditions 1 and 3, and thus 2. Therefore, for all sets y such that $\{e\} \leq y$, it holds that $\forall f(\varphi)$. Thus, the same holds for all such y sets which are maximal and by semantic rules of SEL for the $\lceil \Box_{\leq} \rceil$ operator:

$$\{e\} \vDash_{SEL} \lceil \Box_{\leq} \forall f(\varphi) \rceil.$$

($\Leftarrow$) $\{e\} \vDash_{SEL} \lceil \Box_{\leq} \forall f(\varphi) \rceil \Rightarrow e \vDash \Box\varphi$
L.H.S unfolds to: for all maximal y. $\{e\} \leq y \Rightarrow \forall e' \in y.\ \{e'\} \vDash f(\varphi)$.
Then, the definition of $\leq$ and transformation of event structures give us:

$$\text{for all maximal } y \text{ s.t. } \forall e' \in y.\ e \in \Cup pre(e') \Rightarrow \{e'\} \vDash f(\varphi).$$

Clearly for any event e' such that $e \in \Cup pre(e')$, there is a maximal set y such that $e' \in y$ and y consists of only such events. Therefore,

$$\forall e'.\ e \in \Cup pre(e') \Rightarrow \{e'\} \vDash f(\varphi),$$

or equivalently by I.H,

$$\forall e'.\ e \leq e' \Rightarrow e' \vDash \varphi.$$

Induction Step 6.6.2. $e \vDash_{ESL} \overline{\Box}\varphi \leftrightarrow \{e\} \vDash_{SEL} \lceil \overline{\Box}\forall f(\varphi) \rceil$

($\Rightarrow$) $e \vDash_{ESL} \overline{\Box}\varphi \Rightarrow \{e\} \vDash_{SEL} \lceil \overline{\Box}\forall f(\varphi) \rceil$
L.H.S unfolds to: $\forall e'.\ e' \leq e \Rightarrow e' \vDash \varphi$ by semantic rules of ESL. Based on the translation of relation between event structures we have

$$\forall e'.\ (\exists x \in en(\{e\}).\ e' \in x) \Rightarrow e' \vDash \varphi.$$

Thus, for any set y consisting of such e' events, it holds that $\forall e' \in y.\ e' \vDash \varphi$, or equivalently by I.H. $\forall e' \in y.\ \{e'\} \vDash f(\varphi)$, which is denoted by $y \vDash \forall f(\varphi)$.

On the other hand, the R.H.S. concerns sets $y \rhd \{e\}$ and based on the definition of $\rhd$ we have $\exists x \in en(\{e\}).\ y \subseteq x$. Therefore, for any y, $y \rhd \{e\} \Rightarrow \forall e' \in y.\ \exists x \in en(\{e\}).\ e' \in x$ and as shown above $y \vDash \forall f(\varphi)$. Since this is true for any $y \rhd \{e\}$, it holds also for all such maximal sets, therefore, by semantics of $\overline{\Box}\varphi$ we have

$$\{e\} \vDash_{SEL} \lceil \overline{\Box}\forall f(\varphi) \rceil.$$

($\Leftarrow$) $\{e\} \vDash_{SEL} \lceil \overline{\Box}\forall f(\varphi) \rceil \Rightarrow e \vDash \overline{\Box}\varphi$
L.H.S unfolds to: for all maximal y. $y \rhd \{e\} \Rightarrow \forall e' \in y.\ \{e'\} \vDash f(\varphi)$.

Thus, as proved in proof 6.6.5, $\forall x \in en(\{e\}) \Rightarrow \forall e' \in x.\ \{e'\} \vDash f(\varphi)$. Based on the translation of operators, $\forall e'.\ e' \in x \Leftrightarrow e' \leq e$. Therefore, $\forall e' \leq e \Rightarrow\vDash f(\varphi)$ or equivalently, by I.H., $\forall e' \leq e \Rightarrow\vDash \varphi$.

Induction Step 6.6.3. $e \vDash_{ESL} \Box_{\#}\varphi \leftrightarrow \{e\} \vDash_{SEL} \lfloor \Box_{\#}\exists f(\varphi) \rfloor$.

($\Rightarrow$) $e \vDash_{ESL} \Box_{\#}\varphi \Rightarrow \{e\} \vDash_{SEL} \lfloor \Box_{\#}\exists f(\varphi) \rfloor$.
L.H.S unfolds to: $\forall e'.\ e\#e' \Rightarrow e' \vDash \varphi$, or equivalently by I.H. $\{e'\} \vDash f(\varphi)$. Also, for any set y, $\{e\}\#y$ iff y has an event e' s.t. $e\#e'$. Thus, for any set y in conflict with e, there is an event $e' \vDash \varphi$. Equivalently, $\forall y.\ \{e\}\#y \Rightarrow \exists f(\varphi)$, thus, the same holds for all minimal ys. This is exactly expressed by the semantics of the $\lfloor \Box_{\#} \rfloor$ operator in SEL, i.e. $\{e\} \vDash \lfloor \Box_{\#}\exists f(\varphi) \rfloor$.

($\Leftarrow$) $\{e\} \vDash_{SEL} \lfloor \Box_{\#} \exists f(\varphi) \rfloor \Rightarrow e \vDash_{ESL} \Box_{\#}\varphi$.
L.H.S unfolds to: for all minimal y. $\{e\}\#y \Rightarrow y \vDash \exists(f\varphi)$. And as we are dealing with event structures where the conflict relation is a binary relation on events, $\{e\}\#y$ iff $\exists e' \in y$ s.t. $e\#e'$. Thus, the minimal y is a singleton $\{e'\}$ such that $e\#e'$. Therefore, from L.H.S. $\forall e'.\, e\#e' \Rightarrow \{e'\} \vDash f(\varphi)$ or equivalently by I.H. $\forall e'.e\#e' \Rightarrow \{e'\} \vDash \varphi$ which is exactly expressed by the semantics of the $\Box_{\#}$ operator in ESL, i.e. $e \Vdash \Box_{\#}\varphi$.

Induction Step 6.6.4. Prove that $e \vDash_{ESL} \Box_c\varphi$ iff $\{e\} \vDash_{SEL} \lfloor \Box_{\|} \forall f(\varphi) \rfloor$.

($\Rightarrow$) $e \vDash_{ESL} \Box_c\varphi \Rightarrow \{e\} \vDash_{SEL} \lfloor \Box_{\|} \forall f(\varphi) \rfloor$.
Consider $y \parallel \{e\}$. Applying the definition of concurrent sets yields $y \cup \{e\} \in Con$ and $\forall e' \in y.\, e\ indep\ e'$. Therefore, $\forall e' \in y.\, \neg e\#e'$ as otherwise $y \cup \{e\} \notin Con$. Moreover, by translation of event structures and definition of $indep$, $\forall e' \in y.\, e \not\leq e'$ and $e' \not\leq e$. Thus, $\forall y.\, y \parallel \{e\} \Rightarrow \forall e' \in y.\, e\ co\ e'$.

Now by semantic rules of ESL the L.H.S. expands to $\forall e'.\, e\ co\ e' \Rightarrow e' \vDash \varphi$. Therefore, for any y if $y \parallel \{e\}$ then $\forall e' \in y.\, e' \vDash \varphi$ or equivalently by I.H. $\forall e' \in y.\, \{e'\} \vDash f(\varphi)$. Clearly, if this is true for any y it is also true for all minimal ys and this is exactly given by $\{e\} \vDash \lfloor \Box_{\|} \forall f(\varphi) \rfloor$.

($\Leftarrow$) $\{e\} \vDash_{SEL} \lfloor \Box_{\|} \forall f(\varphi) \rfloor \Rightarrow e \vDash_{ESL} \Box_c\varphi$.

By semantic rules of SEL the L.H.S. expands to for all minimal y. $y \parallel \{e\} \Rightarrow \forall e' \in y.\{e\} \vDash f(\varphi)$, or equivalently by I.H., $\forall e' \in y.e' \vDash \varphi$.

As we are dealing with event structures, the concurrency relation is a binary relation defined on events. Therefore, the minimal set $y \parallel \{e\}$ is a singleton $\{e'\}$ such that $\{e'\} \parallel \{e\}$. Moreover, if $\{e'\} \parallel \{e\}$ then applying the definition of $\parallel$ and using translation of event structures we obtain $\{e, e'\} \in Con$ and $e \not\leq e'$ and $e' \not\leq e$, and thus, $e\ co\ e'$. Thus, for the singleton $y = \{e'\}$ we have $y \parallel \{e\}$. Therefore, for any $e'\ co\ e$ there is a minimal set $y' = \{e'\}$ s.t. $\{e'\} \parallel \{e\}$. Additionally, from above we have $\forall e'\ co\ e \Rightarrow e' \vDash \varphi$ which is exactly given by the ESL formula $\Box_c\varphi$.

Induction Step 6.6.5. Prove that $e \vDash_{ESL} \otimes\varphi$ iff $\{e\} \vDash_{SEL} \lceil \Box_{\Mapsto} \forall f(\varphi) \rceil$.

($\Rightarrow$)$e \vDash_{ESL} \otimes\varphi \Rightarrow \{e\} \vDash_{SEL} \lceil \Box_{\Mapsto} \forall f(\varphi) \rceil$
R.H.S expands to: for all maximal y. $\{e\} \Mapsto y \ \Rightarrow \forall e' \in y.\, \{e'\} \vDash f(\varphi)$, where by applying the definition of $\Mapsto$ for sets of events, $\{e\} \Mapsto y$ iff:

1. $\{e\} \cup y \in Con$
2. $\exists e' \in y$ s.t. $e \to e'$
3. $\forall e' \in y.\ e \to e'$

Note that y cannot be an empty set, as otherwise $\{e\} \mapsto y$ does not hold. Therefore, if 3 holds, then so does 2.

($\Rightarrow$) $e \vDash_{ESL} \otimes\varphi \Rightarrow \{e\} \vDash_{SEL} \lceil \Box_{\mapsto} \forall f(\varphi) \rceil$
L.H.S unfolds to: $\forall e'.\ e \to e' \ \Rightarrow e' \vDash \varphi$ by semantic rules of ESL. Thus, for any set y consisting of such e' events, it holds that $\forall e' \in y.\ e' \vDash \varphi$, or equivalently by I.H., $\forall e' \in y.\ \{e'\} \vDash f(\varphi)$, which is denoted by $y \vDash \forall f(\varphi)$.
Also, such y sets are exactly those for which $\{e\} \mapsto y$ as they satisfy conditions 1 and 3, and thus 2. Since this holds for any such y, clearly it also holds for all maximal y as well. Therefore, by semantic rules of SEL for the $\lceil \Box_{\mapsto} \rceil$ operator:

$$\{e\} \vDash \lceil \Box_{\mapsto} \forall f(\varphi) \rceil.$$

($\Leftarrow$) $\{e\} \vDash \lceil \Box_{\mapsto} \forall f(\varphi) \rceil \Rightarrow e \vDash \otimes\varphi$
L.H.S unfolds to: for all maximal y. $\{e\} \mapsto y \ \Rightarrow \forall e' \in y.\ \{e'\} \vDash f(\varphi)$.
The definition of $\mapsto$ for sets imposes that $\forall e' \in y.\ e \to e'$ and as such:

$$\text{for all maximal } y \text{ s.t. } \forall e' \in y.\ e \to e' \Rightarrow \{e'\} \vDash f(\varphi).$$

Now for any event e' such that $e \to e'$, consider the singleton $y = \{e'\}$. Either y is maximal or $\exists y'.\ y \subset y'$ & $\{e\} \Rightarrow y'$ & y' is maximal. Thus, any such e' is part of a maximal set y' such that $\forall e'' \in y'.e \to e''$. Thus,

$$\forall e'.\ e \to e' \Rightarrow \exists \text{maximal } y'.\ \{e\} \Rightarrow y' \ \&\ e' \in y'$$

Therefore, from above, $\forall e'.\ e \to e' \Rightarrow \{e'\} \vDash f(\varphi)$ or equivalently, by I.H.,

$$\forall e'.\ e \to e' \Rightarrow e' \vDash \varphi.$$

This is exactly captured by the ESL formula $e \vDash \otimes\varphi$

Induction Step 6.6.6. Prove that $e \vDash_{ESL} \overline{\otimes}\varphi$ iff $\{e\} \vDash_{SEL} \lfloor \overline{\Box}_{\rightarrow} \forall f(\varphi) \rfloor$.
This is proved using proof 6.6.6, as follows.

($\Rightarrow$) $e \vDash_{ESL} \overline{\otimes}\varphi \Rightarrow \{e\} \vDash_{SEL} \lfloor \overline{\Box}_{\rightarrow} \forall f(\varphi) \rfloor$
The L.H.S. expands to $\forall e'.\ e' \leq e \Rightarrow e' \vDash \varphi$. Consider the set $y = \{e' \mid e' \leq e\}$, for which $\forall e' \in y.\ e' \vDash \varphi$. But $y \rightarrow \{e\}$ by theorem 6.6.6 and y is unique. Therefore, $\forall y.\ y \rightarrow \{e\} \Rightarrow \forall e' \in y.\ e' \vDash \varphi$ or equivalently, $\forall y.\ y \rightarrow \{e\} \Rightarrow \forall e' \in y.\ \{e'\} \vDash f(\varphi)$ (by induction hypothesis) . But this is exactly given by the semantics of $\overline{\Box}_{\rightarrow} \forall f(\varphi)$.

($\Leftarrow$) $\{e\} \vDash_{SEL} \lfloor \overline{\Box}_{\rightarrow} \forall f(\varphi) \rfloor \Rightarrow e \vDash_{ESL} \overline{\otimes}\varphi$.

The l.h.s. expands to $\forall y.\ y \rightarrow \{e\} \Rightarrow \forall e' \in y.\ \{e'\} \vDash f(\varphi)$, or equivalently, $\forall y.\ y \rightarrow \{e\} \Rightarrow \forall e' \in y.\ e' \vDash \varphi$. By proof 6.6.6 $y = \{e_i \mid e_i \rightarrow e\}$, therefore, $\forall e_i.\ e_i \rightarrow e \Rightarrow e_i \vDash \varphi$. But that is exactly given by the semantics of $\overline{\otimes}\varphi$.

Conjecture 6.6.4. SEL cannot describe the ESL formula $e \vDash \Box\varphi$ by using operators $\forall$, $\neg$, $\wedge$, $\Box$, $\overline{\Box}$, $\Box_{\rightarrow}$, $\overline{\Box}_{\rightarrow}$, $\Box_{\#}$ and $\Box_{\|}$.

Plausibility Argument: Consider the event structure $\mathcal{E}_6 = (\{e_1, e_2, e_3, e_4\}, \leq, \#)$ where $e_1 \leq e_3$, $e_2 \leq e_3$, $e_2 \leq e_4$ and the relation # is empty (figure 6.10). Suppose that $Q(e_3) = \{p\}$ and $\varphi = p$. Then, clearly $e_1 \vDash \Box p$. This would translate to the stable event structure $E' = (\{e_1, e_2, e_3, e_4\}, \vdash, Con)$ where $\{e_1, e_2\} \vdash e_3$, $\{e_2\} \vdash e_4$ and Con contains any subset of the events of E' . We will now show an exhaustive inspection of SEL formulae.

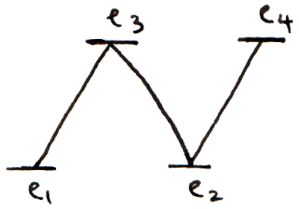

Figure 6.10: $\mathcal{E}_6$

To start with, it is clear that the operators $\Box_{\#}$ *and* $\Box_{\|}$ are of no avail here, as they only capture conflict and concurrency. Therefore, the main operators to consider are $\Box$, $\overline{\Box}$, $\Box_{\rightarrow}$ and $\overline{\Box}_{\rightarrow}$ in combination with the other operators. Let us first list all the sets

which are in $\rightarrow$ or $\triangleright$ relation with each other.

$\{\} \nrightarrow$ undefined	$\underline{\{e_1,e_2\} \rightarrow \{e_3\},\{e_3,e_4\}}$	$\{e_1,e_2,e_3\} \nrightarrow$
$\{e_1\} \nrightarrow$	$\{e_1,e_3\} \nrightarrow$	$\{e_1,e_2,e_4\} \nrightarrow$
$\underline{\{e_2\} \rightarrow \{e_4\}}$	$\{e_1,e_4\} \nrightarrow$	$\{e_1,e_3,e_4\} \nrightarrow$
$\{e_3\} \nrightarrow$	$\{e_2,e_3\} \nrightarrow$	$\{e_2,e_3,e_4\} \nrightarrow$
$\{e_4\} \nrightarrow$	$\{e_2,e_4\} \nrightarrow$	$\{e_1,e_2,e_3,e_4\} \nrightarrow$
	$\{e_3,e_4\} \nrightarrow$	

Similarly, for the sets in $\triangleright$ relation the following list is obtained.

$\{\} \triangleright$ undefined	$\{e_2,e_3\} \not\triangleright$
$\{e_1\} \not\triangleright$	$\{e_2,e_4\} \not\triangleright$
$\underline{\{e_2\} \triangleright \{e_4\}}$	$\{e_3,e_4\} \not\triangleright$
$\{e_3\} \not\triangleright$	$\{e_1,e_2,e_3\} \not\triangleright$
$\{e_4\} \not\triangleright$	$\{e_1,e_2,e_4\} \not\triangleright$
$\underline{\{e_1,e_2\} \triangleright \{e_3\},\{e_3,e_4\}}$	$\{e_1,e_3,e_4\} \not\triangleright$
$\{e_1,e_3\} \not\triangleright$	$\{e_2,e_3,e_4\} \not\triangleright$
$\{e_1,e_4\} \not\triangleright$	$\{e_1,e_2,e_3,e_4\} \not\triangleright$

From the lists above, it is clear that the nothing in the relations $\rightarrow$ and $\triangleright$ gives us the singleton $\{e_1\}$. Therefore, no matter how many levels of logic operators are used, forwards or backwards, and applying maximal or minimal operators, there is no way to obtain $\{e_1\}$ in a relevant manner. The reason is that the causality related operators in SEL only consider full causes as opposed to partial causes considered by ESL. Since the logic does not have any means of taking out a subset of a set, it is plausible that any formula of ESL relating to partial causes is not expressible by SEL.

Proof 6.6.5. Prove that for any nonempty set y and y', y is a maximal set such that $y \triangleright y'$ iff $y \in en(y')$.

($\Rightarrow$) Based on the definition of $\triangleright$, if $y \triangleright y'$, then $\exists x \in en(y').\ y \subseteq x$. Also, y is maximal iff $\nexists y''.\ y'' \triangleright y'$ & $y \subseteq y''$. Clearly, the maximal subset of $x \in en(y)$ is x itself. Thus, $y \in en(y')$.

($\Leftarrow$) Following the definition of $\triangleright$, for any nonempty set $x \in en(y')$, $x \triangleright y'$.

Proof 6.6.6. Prove that for an event structure $\mathcal{E}_6 = (E, \leq, \#)$ and the stable event structure $\mathcal{E}_6' = (E, \vdash, Con)$ mapped to it, for a nonempty minimal set y, $y \to \{e\}$ where $e \in E$ iff $y = \{e_i \mid e_i \to e\}$ (Recall that $e \to e'$ iff $e \leq e'$ and $\nexists e''.e \leq e'' \text{ and } e'' \leq e'$).

The first observation is that as for event structures there are no disjunctive causes, then each event or set of consistent events has a unique set of events causing it. Therefore, generally for a set y, $en(y) = \Cup pre(y)$ and for the case of a singleton such as here, $pre(\{e\}) = en(\{e\})$ is the singleton $\{x = \{e' \mid e' \leq e\}\}$.

($\Rightarrow$) $y \to \{e\} \Rightarrow y$ is the minimal set s.t. $y = \{e_i \mid e_i \to e\}$.
If $y \to \{e\}$, by definition of $\to$ and the minimality condition, we have the following.

1. $y \neq \emptyset$ and as such, $\emptyset \nvdash e$
2. $\exists x \in en(y)$ s.t. $x \cup y \in en(\{e\})$ and thus, $x \cup y = \{e' \mid e' \leq e\}$ and $e \notin x \cup y$
3. $\nexists y'$ s.t. $y' \subseteq y$ and $y' \to \{e\}$

From above, it is clear that $\forall e' \in E.e' \in y \Rightarrow e' \leq e$. Now we just need to show that $\forall e' \in y.\nexists e''$ s.t. $e' \leq e'' \leq e$. To prove by contradiction, suppose such e'' exists. Then either $e'' \in y$ or $e'' \notin y$.

If $e'' \notin y$, then this is in contradiction with $x \cup y = en(\{e\})$, as $en(\{e\}) = \{e_i \mid e_i \leq e\}$. Therefore, such e'' does not exist outside y. Now suppose $e'' \in y$ and consider $y' = y - z$ where $z = \{e_i \mid e_i \in y \text{ and } e_i \leq e''\}$. It can be shown that $x \cup y = x' \cup y'$ where $x' = en(y')$ and therefore, $\exists y' \subseteq y$ s.t. $y' \to \{e\}$ which is a contradiction.

Consider $x \cup y$ and $x' \cup y'$ as described above. Then $\forall e' \in x \cup y$, either $e' \in y$ or $e' \in x - y$. In the former case, then if $e' \notin z \Rightarrow e \in y'$, otherwise, as $x' = en(y') = \{e_j \mid e_j \leq e_i \text{ for some } e_i \in y\}$, then $e' \in x'$ as $e' \in z$ and as such $e' \leq e''$ (and $e'' \in y$). In the latter case, i.e. $e' \in x - y$ implies that $e' \leq e_i$ for some $e_i \in y$. If $e_i \notin z$, then $e_i \in y'$ and therefore, $e' \in x'$ as $e' \leq e_i$. Otherwise, if $e_i \in z$, again $e' \in x'$ as $e' \leq e_i$ and $e_i \leq e_i'$ for some $e_i' \in y'$ and by transitivity of $\leq$, $e' \leq e_i'$ and as such, $e' \in x'$. Therefore, $\forall e'.e' \in x \cup y \Rightarrow e' \in x' \cup y'$. Similarly, $\forall e' \in x' \cup y'$, either $e' \in y'$, and as $y' \subseteq y$, $e' \in y$ or $e' \in x' - y'$, which means $e' \leq e_i$ for some $e_i \in y'$. Then again as $y' \subseteq y$, $e' \leq e_i$ for some $e_i \in y$ and as such $e' \in x$. Therefore, $\forall e'.\ e' \in x' \cup y' \Rightarrow e' \in x \cup y$. Thus, $x \cup y = x' \cup y'$.

($\Leftarrow$)y is the minimal set s.t. $y = \{e_i \mid e_i \rightarrow e\} \Rightarrow y \rightarrow \{e\}$.
As y is not empty, $\emptyset \not\vdash e$ as $\exists e_i.\ e_i \rightarrow e$. Next we show that $\{x \cup y\} = en(\{e\})$. First observe that in a simple event structure, $\forall e_i \leq e$, either $e_i \rightarrow e$ or by finiteness of the history of each event, $e_i \leq e' \rightarrow e$. The L.H.S. expands to $y = \{e_i \mid e_i \leq e$ and $\nexists e_i'.\ e_i \leq e_i' \leq e\}$. Then $x = en(y) = \Cup pre(y) = \{e_j \mid e_j \leq e_i\}$. Therefore, $x \cup y = \{e_k \mid e_k \leq e_i$ and $\nexists e_i'.\ e_i \leq e_i' \leq e\}$. It can be shown that $x \cup y = en(\{e\})$. Clearly, $\forall e_k \in x \cup y$, $e_k \in en(\{e\})$ as by transitivity of $\leq$,$\forall e_k.\ e_k \leq e$. Similarly, as mentioned above $\forall e' \in en(\{e\})$, either $e' \in y$ or $e' \leq e_i$ for some $e_i \in y$, and therefore $e' \in x$. Therefore, $x \cup y = en(\{e\})$.

It now suffices to show that $\nexists y' \subseteq y.\ y' \rightarrow \{e\}$. Suppose $y' \subseteq \{e_i \mid e_i \rightarrow e\}$ and $y' \rightarrow \{e\}$. Then $x' \cup y' = en(\{e\}) = x \cup y$, where $x' = en(y')$. Equivalently, $x' \cup y' = \{e_j \mid e_j \leq e\} = x \cup y$. Now take $e_i \in y - y'$. Clearly $e_i \in x \cup y$ and therefore, $e_i \in x' \cup y'$. As $e_i \notin y'$ then $e_i \in x'$. Thus, as $x' = en(y')$, $e_i \leq e'$ for some $e' \in y'$ and $e' \leq e$. But that is in contradiction with $e_i \rightarrow e$. Therefore, such y' does not exist.

Conjecture 6.6.7. SEL cannot express the ESL formula $\otimes\varphi$ using operators $\forall$, $\neg$, $\wedge$, $\Box$ $\overline{\Box}$, $\Box_{\rightarrow}$, $\overline{\Box}_{\rightarrow}$, $\Box_{\#}$ and $\Box_{\|}$.

Plausibility Argument: The argument is very similar to that of conjecture 6.6.4, as the reason this formula cannot be expressed is the same. Namely, the ESL formula captures any event partially caused by an event, whereas in SEL, the causality operators mentioned above express events being fully caused or enabled by other events.

Consider the event structure in the argument for conjecture 6.6.4, ESL formula $\otimes p$ and the list of sets in $\rightarrow$ and $\triangleright$ relation. As before, there are no formulae with denotation $\{e_1\}$ or its negation $\{e_2, e_3, e_4\}$. Similarly, even though there are formulae with denotation $\{e_1, e_2\}$, there are no formulae with denotation $\{e_2\}$ or its negation $\{e_1, e_3, e_4\}$, so that $\{e_1\}$ can be extracted from it. Moreover, maximality or minimality constraints cannot change this. Therefore, it is plausible that no SEL formulae can express this operator of ESL.

6.7 Conclusion

In this chapter, SEL, the first logic specifically designed for stable event structures is presented. The aim was to better understand stable event structures, while being able to express properties of interest. We then proved that SEL can capture ESL-based logics over event structures encoded as stable event structures. Furthermore, the maximal/minimal operators help with expression of properties both at the granularity of events and sets of events. In addition to properties expressible by ESL-based logics, PESL can capture properties such as synchronisation, disjunctive causes and constant requirements. Finally, a model checking algorithm for finite stable event structures was presented.

Chapter 7

Conclusion and Future Works

In this thesis we have studied different aspects related to the verification of some of the well-known models of true concurrency, namely, Petri nets, event structures and stable event structures.

First we considered verification of Petri nets through unfoldings. As explained in this thesis, unfoldings of Petri nets are widely used for description and verification of the behaviour of Petri nets. However, they have the drawback of growing exponentially with the choices of a net, since every single concurrent or conflicting firing of a transition in the Petri net has a separate corresponding event in the unfolding. Thus, unfolding of a net describes its behaviour by describing the full state space, without assigning any order to concurrent events. The issue of unfolding a net in a more compact manner has recently been addressed in [37, 23, 58].

In this thesis, we added to these solutions by developing a new and more compact unfolding for safe Petri nets, namely, the unfolding^{-}s of a net. We showed that similar to unfoldings, they capture all the reachable markings of the net. However, by reusing the conflicting places or in other words, by not unfolding the conflicts as far as possible, the unfolding^{-}s constructed are more compact.

A prominent technique for verification of Petri nets is that of McMillan [51] which defines and uses complete finite prefix of Petri nets for verification of reachability-like properties. Majority of the existing algorithms use or improve the same notion. Therefore, by defining complete finite prefixes of unfolding^{-}s, we show that a large body of existing verification algorithms are applicable to them. We have also provided algorithms for constructing the compact unfoldings and their complete finite stopped

prefixes.

The other perspective we have investigated in this thesis is that of model checking the probabilistic behaviour of models of true concurrency with randomness. Probabilistic models of true concurrency have been studied only in the last decade or so, and among such models (such as [74, 72, 1]) we have focused on probabilistic event structures as described by [1] as the most general approach which, unlike the others, handles confusion. We presented its definition, with the most important concept being that of *branching cells*. Branching cells can be viewed as units of choice decomposing configurations while resolving conflict and confusion internally. Thus, they decompose configurations in a probabilistically independent manner and as such, it can be said that in this model, concurrency matches probabilistic independence. We showed in this thesis that the same approach can be used on stable event structures, as long as confusions have a simple structure, and do not involve events which are causally related to each other. We call such (stable) event structures *jump-free* and define probabilistic jump-free stable event structures similar to probabilistic event structures.

The model checking problem is the procedure of verifying if a model $\mathcal{M}$ has a property captured by formulae ϕ. Thus, having chosen a model, we required a logic which could capture properties of interest for the verification purpose. While there are few logics with the true concurrency semantics, we knew of no such probabilistic logic. Therefore, we have defined a new logic, PESL, which introduces a new concept of progress, since dealing with R-stopped configurations requires formalising the interaction between sets of events. We have shown that with respect to the new concept of progress, PESL captures pCTL. Thus, in terms of expressivity it subsumes pCTL while it also has the ability to express concurrency and more interestingly, synchronisation. A model checking algorithm for finite probabilistic event structures was then proposed.

Finally, we have developed a new logic, SEL, interpreted over stable event structures. The aim was to better understand stable event structures, while being able to express properties of interest related to event structures in general and more specifically, stable event structures. We showed that an extension of SEL can capture ESL-based logics and also presented some of the properties expressible by SEL specific to stable event structures.

7.1 Future Work

Some natural directions for continuation of this research include the following.

7.1.1 On Compact Unfoldings

The results and algorithms for compact unfoldings presented in this thesis are defined for safe Petri nets. However, this approach can be extended to be applicable to general Petri nets. The main complexity arises from the fact that in a general Petri net, multiple transitions can fire into a place. Therefore, to make sure a minimum number of conditions are created in the unfolding process, we can have several choices. This can be resolved using graph theory, and in particular the graph coloring problem fits this purpose beautifully. Furthermore, it would be reasonable to give some formal measurement of the extent of compactness of unfolding$^-$s. Finding the patterns which results in having cyclic dependencies in the algorithm can be the first step in that direction.

On another note, while we have provided the theoretical foundation related to these unfoldings, it would be interesting to investigate these from the implementation point of view. Seeking efficient computation algorithms can be fulfilled by taking advantage of the existing optimisation techniques for calculating the possible extensions in construction of the unfoldings (such as [36]), and in particular by investigating in efficient ways to compute and store the consistency relation. More importantly, optimising the 'wait' and 'resolve' processes would significantly improve the efficiency of the algorithm. Finally, computing the computational complexity for construction of unfoldings can help with determining the extent of its feasibility.

7.1.2 On PESL

As mentioned before, the main challenge in defining PESL was in definition of operators suitable for capturing the interaction between sets of events. It would be interesting to study the interaction between these sets, i.e. an R-stopped configuration and what it can develop in to, in a more systematic manner. One direction would be that of investigation ways to formalise the way in which R-stopped configurations communicate. Some insight may be taken from object Petri nets [69] and structured occurrence nets [42, 41], to evaluate if this idea has scope for further research.

The other important direction is that of model checking infinite systems. Based on theorem 5.3.7 it is reasonable to assume that existing techniques for model checking of infinite state systems with Markov chains and PCTL may be applicable to a major fragment of PESL [45, 3, 10]. If that is possible, a similar approach should be taken for the other operators. Since the synchronisation operator has a finite nature, it is likely that only the parallelism operator has to be tackled. Another approach is studying *recurrent nets* [2] and consider event structures arising from such nets.

7.1.3 On Probabilistic Stable Event Structures

We have given a definition of probabilistic jump-free stable event structures, based on probabilistic event structures of [1]. However, it would be interesting to investigate if other definitions or representation of the same definitions can be found which are more intrinsic to the concept of stable event structures. To expand this, consider the compact unfoldings developed in this thesis. The main idea is that we may not always require the knowledge of the exact history of how an event has occurred. Therefore, finding a class of stable event structures for which the history of an event does not affect its future (e.g. with respect to branching cells) may be a good starting point. If the history is important and therefore required, one can consider a different representation of probabilistic stable event structures where the sets enabling events are tagged with the probability of being the cause for the event. Either way, there is scope for speculation of different types of probabilistic stable event structures.

Bibliography

[1] S. Abbes and A. Benveniste. True-concurrency probabilistic models branching cells and distributed probabilities for event structures. *Inf. Comput.*, 204(2), 2006.

[2] S. Abbes and A. Benveniste. True-concurrency probabilistic models: Markov nets and a law of large numbers. *Theoretical Computer Science*, 390(2):129–170, 2008.

[3] P. A. Abdulla, N. Bertrand, A. Rabinovich, and P. Schnoebelen. Verification of probabilistic systems with faulty communication. *Information and Computation*, 202(2):141–165, 2005.

[4] R. Alur, C. Courcoubetis, and D. Dill. Verifying automata specifications of probabilistic real-time systems. In *Real-Time: Theory in Practice*, pages 28–44. Springer, 1992.

[5] R. Alur, D. Peled, and W. Penczek. Model-checking of causality properties. In *Logic in Computer Science, 1995. LICS'95. Proceedings., Tenth Annual IEEE Symposium on*, pages 90–100. IEEE, 1995.

[6] A. Aziz, V. Singhal, F. Balarin, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *Computer Aided Verification*, pages 155–165. Springer, 1995.

[7] C. Baier. On algorithmic verification methods for probabilistic systems. *Universität Mannheim*, 1998.

[8] P. Baldan and S. Crafa. A logic for true concurrency. In *CONCUR 2010-Concurrency Theory*, pages 147–161. Springer, 2010.

[9] A. Benveniste, E. Fabre, and S. Haar. Markov nets: probabilistic models for distributed and concurrent systems. *Automatic Control, IEEE Transactions on*,

48(11):1936–1950, 2003.

[10] N. Bertrand and P. Schnoebelen. Model checking lossy channels systems is probably decidable. In *Foundations of Software Science and Computation Structures*, pages 120–135. Springer, 2003.

[11] A. Bianco and L. De Alfaro. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*, pages 499–513. Springer, 1995.

[12] F. Ciesinski and M. Größer. On probabilistic computation tree logic. In *Validation of Stochastic Systems*, pages 147–188. Springer, 2004.

[13] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[14] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

[15] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT press, 1999.

[16] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani. Model checking and the state explosion problem. In *Tools for Practical Software Verification*, pages 1–30. Springer, 2012.

[17] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 338–345. IEEE, 1988.

[18] L. De Alfaro. *Formal verification of probabilistic systems*. PhD thesis, Standford University, 1997.

[19] L. De Alfaro. *Stochastic transition systems*. Springer, 1998.

[20] C. Derman. *Finite state Markovian decision processes*. Academic Press, Inc., 1970.

[21] E. A. Emerson. The beginning of model checking: A personal perspective. In *25 Years of Model Checking*, pages 27–45. Springer, 2008.

[22] J. Esparza, S. Römer, and W. Vogler. An improvement of mcmillan's unfolding algorithm. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 87–106. Springer, 1996.

[23] E. Fabre. Trellis processes: a compact representation for runs of concurrent systems. *Discrete Event Dynamic Systems*, 17(3):267–306, 2007.

[24] P. Godefroid, J. van Leeuwen, J. Hartmanis, G. Goos, and P. Wolper. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032. Springer Heidelberg, 1996.

[25] J. Gutierrez. Logics and bisimulation games for concurrency, causality and conflict. In *Foundations of Software Science and Computational Structures*, pages 48–62. Springer, 2009.

[26] S. Haar. Probabilistic cluster unfoldings for petri nets. 2002.

[27] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[28] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994.

[29] S. Hart et al. Probabilistic temporal logics for finite and bounded models. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 1–13. ACM, 1984.

[30] J. Hayman. *Petri net Semantics*. PhD thesis, Computer Laboratory, University of Cambridge., 2009.

[31] J. Hayman and G. Winskel. Independence and concurrent separation logic. In *Logic in Computer Science, 2006 21st Annual IEEE Symposium on*, pages 147–156. IEEE, 2006.

[32] C. Jones and G. D. Plotkin. A probabilistic powerdomain of evaluations. In *Logic in Computer Science, 1989. LICS'89, Proceedings., Fourth Annual Symposium on*, pages 186–195. IEEE, 1989.

[33] D. Kartson, G. Balbo, S. Donatelli, G. Franceschinis, and G. Conte. *Modelling with generalized stochastic Petri nets*. John Wiley & Sons, Inc., 1994.

[34] J.-P. Katoen. *Quantitative and qualitative extensions of event structures*. PhD thesis, University of Twente, 1996.

[35] S. Katz and D. Peled. Interleaving set temporal logic. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, 1987.

[36] V. Khomenko. *Model checking based on prefixes of petri net unfoldings*. PhD thesis, University of Newcastle upon Tyne, 2003.

[37] V. Khomenko, A. Kondratyev, M. Koutny, and W. Vogler. Merged processes: a new condensed representation of petri net behaviour. *Acta Informatica*, 43(5):307–330, 2006.

[38] V. Khomenko and M. Koutny. Towards an efficient algorithm for unfolding petri nets. In *CONCUR 2001Concurrency Theory*, pages 366–380. Springer, 2001.

[39] V. Khomenko, M. Koutny, and W. Vogler. Canonical prefixes of petri net unfoldings. *Acta Informatica*, 40(2):95–118, 2003.

[40] V. Khomenko and A. Mokhov. *An algorithm for direct construction of complete merged processes*. Springer, 2011.

[41] J. Kleijn and M. Koutny. *Causality in structured occurrence nets*. Springer, 2011.

[42] M. Koutny and B. Randell. Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques. *Fundamenta Informaticae*, 97(1):41–91, 2009.

[43] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333 – 354, 1983. Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982.

[44] M. Kwiatkowska, G. Norman, and D. Parker. Modelling and verification of probabilistic systems. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems. CRM Monograph Series*, 23:93–215, 2004.

[45] M. Kwiatkowska, G. Norman, and J. Sproston. *Pctl model checking of symbolic probabilistic systems*. University of Birmingham, School of Computer Science, 2003.

[46] D. Lehmann and S. Shelah. Reasoning with time and chance. *Information and Control*, 53(3):165–198, 1982.

[47] K. Lodaya and P. S. Thiagarajan. A modal logic for a subclass of event structures. In *14th International Colloquium on Automata, languages and programming*, 1987.

[48] M. Madhavan and P. S. Thiagarajan. A logical characterization of well branching event structures. *Theor. Comput. Sci.*, 96(1):35–72, 1992.

[49] A. Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 278–324. Springer, 1987.

[50] K. L. McMillan. *Symbolic model checking*. Springer, 1993.

[51] K. L. McMillan and D. Probst. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[52] M. Mukund and P. S. Thiagarajan. An axiomatization of event structures. In *Proceedings of the Ninth Conference on Foundations of Software Technology and Theoretical Computer Science*, 1989.

[53] M. Nielsen, G. Plotkin, and G. Winskel. *Petri nets, event structures and domains*. Springer, 1979.

[54] W. Penczek. A temporal logic for event structures. *Fundamenta Informaticae*, 11(3):297–362, 1988.

[55] W. Penczek. A temporal logic for the local specification of concurrent systems. In *IFIP Congress*, pages 857–862, 1989.

[56] W. Penczek. Branching time and partial order in temporal logics. 1995.

[57] J. L. Peterson. Petri nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252, 1977.

[58] G. M. Pinna. How much is worth to remember? a taxonomy based on petri nets unfoldings. In *Applications and Theory of Petri Nets*, pages 109–128. Springer, 2011.

[59] S. Pinter and P. Wolper. A temporal logic for reasoning about partially ordered computations (extended abstract). In *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing*, 1984.

[60] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[61] A. Pnueli. On the extremely fair treatment of probabilistic algorithms. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 278–290. ACM, 1983.

[62] A. Pnueli and L. D. Zuck. Probabilistic verification. *Information and computation*, 103(1):1–29, 1993.

[63] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*, volume 414. Wiley. com, 2009.

[64] M. O. Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963.

[65] R. Segala. Modeling and verification of randomized distributed real-time systems. 1996.

[66] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[67] E. Smith. On the border of causality: contact and confusion. *Theoretical computer science*, 153(1):245–270, 1996.

[68] R. Tix, K. Keimel, and G. Plotkin. Semantic domains for combining probability and non-determinism. *Electronic Notes in Theoretical Computer Science*, 222:3–99, 2009.

[69] R. Valk. Object petri nets. In *Lectures on Concurrency and Petri Nets*, pages 819–848. Springer, 2004.

[70] A. Valmari. The state explosion problem. In *Lectures on Petri nets I: Basic models*, pages 429–528. Springer, 1998.

[71] D. Varacca and M. Nielsen. Probabilistic petri nets and mazurkiewicz equivalence. 2003.

[72] D. Varacca, H. Völzer, and G. Winskel. Probabilistic event structures and domains. In *CONCUR 2004-Concurrency Theory*, pages 481–496. Springer, 2004.

[73] M. Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 327–338. IEEE, 1985.

[74] H. Völzer. Randomized non-sequential processes. In *CONCUR 2001Concurrency Theory*, pages 184–201. Springer, 2001.

[75] G. Winskel. Event structure semantics for ccs and related languages. DAIMI Research Report, University of Aarhus, April 1983.

[76] G. Winskel. Event structures. In *Advances in Petri Nets*, pages 325–392, 1986.

[77] G. Winskel. Distributed probabilistic strategies. In *29th Conference on the Mathematical Foundations of Programming Semantics*, 2013.

[78] G. Winskel and M. Nielsen. Models for concurrency. 1995.

[79] W. Yi and K. G. Larsen. Testing probabilistic and nondeterministic processes. In *PSTV*, volume 12, pages 47–61, 1992.