Algebraic Verification of Probabilistic and Concurrent Systems

Mananjanahary Tahiry Rabehaja

Supervisors

A/Prof. Annabelle McIver

Dr. Georg Struth

Department of Computing

Macquarie University

Australia

Department of Computer Science

The University of Sheffield

United Kingdom

A Thesis Submitted in Partial Fulfilment of the Requirements for the Joint Degree of Doctor of Philosophy in Computer Science

March 2014

Abstract

This thesis provides an algebraic modelling and verification of probabilistic concurrent systems in the style of Kleene algebra. Without concurrency, it is shown that the equational theory of continuous probabilistic Kleene algebra is complete with respect to an automata model under standard simulation equivalence. This yields a minimisation-based decision procedure for the algebra. Without probability, an event structure model of Hoare et al.'s concurrent Kleene algebra is constructed. These two algebras are then "merged" to provide probabilistic concurrent Kleene algebra which is used to discover and prove development rules for probabilistic concurrent systems (e.g. rely/guarantee calculus). Soundness of the new algebra is ensured by models based on probabilistic automata (interleaving) and probabilistic bundle event structures (true concurrency) quotiented with the respective simulation equivalences. Lastly, event structures with implicit probabilities are constructed to provide a state based model for the soundness of the probabilistic rely/guarantee rules.

Statement

The research presented in this thesis is my original work, except where otherwise indicated. Some parts of the thesis include revised versions of published papers. This work has not been submitted for a higher degree to any other University or Institution.

Tahiry Rabehaja

Signed :

Date :

Acknowledgements

First and foremost, I thank God for giving me the opportunity to pursue such an exciting life as a researcher, enabling me to test and expand the boundaries of our knowledge. I am grateful to my parents and family for their support through the ups and downs during this journey. My sincere thanks and appreciation to my supervisors, Annabelle McIver and Georg Struth, who have never turned their mind away from my babbling and numerous mistakes. They have always been happy to correct, share their experiences and give me more to read so as to increase my limited understanding. Their help was not just constrained to the academic area but extended to the social aspect of my life as well. I am more than thankful to the iMQRES support from Macquarie University and the EPRSC from The University of Sheffield. Without these funds, I would not have been able to support myself and my family during my PhD enrolment. I am grateful to Jeffrey Sanders for being the first person who introduced me to the world of research and co-authored my first paper, to Gerard Razafinamantsoa and Alain Ralambo who made me understand the power of rigorous abstract mathematical thinking, to AIMS for giving me the opportunity to invest myself in Theoretical Computer Science — a discipline that does not exist (yet) at my home university —, to UNU-IIST Macao for supporting me while I applied for a PhD position and to all the staff at the Department of Computing at Macquarie University and the University of Sheffield. They were always keen to help with any administrative issues. I am so grateful to Hazel Baker for helping me on ensuring the literary quality of this document. She has found all of my hidden commas. My sincere apologies to anyone I have not explicitly mentioned, including friends, colleagues and the anonymous reviewers of my publications who have contributed widely to the technicalities of this thesis.

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

C. A. R. Hoare (Source: Wikipedia).

Contents

Chapter 1

Introduction

Formal Methods is amongst the branches of science that aim to make the art of programming, software development, system design and verification a meticulous mathematical subject. It offers an impartial view of the studied or developed system whose crucial properties are to be proven using a clear and rigorous form of mathematical reasoning. A very detailed proof is usually hard to achieve, especially when the underlying system is complex but, once established, it removes any doubt about its correctness, including aspects related to safety, security and prediction regarding performance. Formal Methods contains a large number of concepts and techniques that range from the denotation of programs using mathematical objects, the development of refinement rules that allow the stepwise construction of a system and the use of mechanised tools towards an automated proof of correctness, to the more recent application of algebraic techniques that provide further abstractions to denotational and logical semantics. More recently, the algebraic tools became more popular because they bring simplicity, expressiveness and proof mechanisation within their elegant formalism, sometimes at the cost of losing low-level details.

Successes have been observed on the application of algebras to the formal treatment of probabilistic systems [32,46,53,76]. In such a system, the probability is either required for the correctness of the protocol (e.g. for symmetry breaking as

used in Herman's solution to the Leader Election Problem on the ring network [26, 34, 47, 48]), or it is the result of a measurable fault or error (e.g. noisy channels and faulty systems [48]). Algebras have also been applied to the specification and verification of concurrent systems [3, 28, 56, 81]. Many physical systems are naturally modelled using concurrency due to collaborations and the distribution of resources (e.g. controllers of a railway network [59]). This thesis aims to provide a unified algebraic setting for the treatment of probabilistic concurrent programs.

Background

The use of algebras in computing was firstly motivated by the correspondence between logic, semigroup structures and automata theory [62, 72]. An algebra is usually obtained by a layer of abstraction on a given structure such as the set of recognizable languages or automata endowed with the regular operations. Salomaa developed the first known complete axiomatisations of language equivalence between automata using a finite number of equations and equational implications [70]. However, Salomaa's *empty word property* is not algebraic in the sense that it is not preserved by substitution. So, the search for a fully algebraic axiomatisation of regular languages remained unanswered. A decade later, Conway worked extensively on regular algebras, more precisely on idempotent semirings [12], which provided more insights and understanding about the power of these algebras. Moreover, he conjectured some complete axiomatisations of the equality of regular languages that does not make use of Salomaa's empty word property. Many of Conway's conjectures were verified in the early 1990s by Krob [41] and Buffa [4]. However, the proofs were very complicated and long. In 1994, Kozen provided an alternative proof of completeness for a simplified axiomatisation using Conway's large collection of results and the minimisation technique from automata theory [38]. Since Conway's influential monograph, Kleene algebras have become a fundamental tool with application ranging from the development and verification of programs to the theory of finite machines [9, 16, 28, 39, 40, 49, 52, 53, 66, 67, 86].

Kleene algebras have been extended into various forms to increase their expressiveness. A specific extension I studied in this thesis is the *probabilistic Kleene algebra* of McIver and Weber [53] which allows us to consider the presence of probabilistic behaviours implicitly. This variant of Kleene algebras was designed primarily to provide a concise and compact algebraic framework for probabilistic programs and has been used for verification purposes [46, 48]. Concrete models of probabilistic Kleene algebra include aspects of the probabilistic powerdomain [30, 48, 80], continuous expectation transformers [48], sets of up-closed multirelations [21] which are isomorphic to monotone predicate transformers [60], and automata modulo simulation [10, 49]. Jones' powerdomain [30] was refined and generalised to correspond to a Dijkstra-like calculus by McIver and Morgan to account for a successful interaction between nondeterminism (seen as a worst-case scenario) and probability [48]. This extension resulted in a powerful specification language for every probabilistic sequential program. Moreover, McIver and Morgan established that most of the fundamental mathematical tools, such as invariant and variant techniques, from the non-probabilistic models of programs are preserved by the probabilistic extension. In multiple cases, these tools have algebraic translation [53].

Another variant of Kleene algebra has emerged more recently to deal with concurrency. The *concurrent Kleene algebra* of Hoare et al. [28] is an elegant framework to reason about grainless concurrency. It is an expansion of Gischer's concurrent semiring that underlies the interaction between a concurrency operation and the other regular operations of Kleene algebra using an interchange law that defines Gischer's subsumption property [23], i.e. the refinement of a truly concurrent composition with a partially interleaved implementation. Fundamental examples of concurrent systems include communicating processes, distributed and embedded systems where smaller components evolve dynamically within a host system or an environment. More recently, tremendous efforts have been observed to improve the reliability of multi-threaded programs due to the fast evolution of multi-cored microprocessors for multitasking as well as to improve efficiency. It

has been noted early in the development of a theory for concurrency that *compositionality* is the key to an efficient and rigorous development and verification of concurrent programs. Compositionality ensures that the desired property of a complex system can be deduced logically from the properties of its components, therefore, offering smaller and simpler chunks of formal verification that, usually, can be automated efficiently. Jones' rely/guarantee calculus is an example of such techniques and it is sound with respect to the execution traces semantics [11, 31] (see [17,24,35,43] for other related approaches). Rely/guarantee rules are particularly important when reasoning about *interference* in shared variable concurrency to circumvent the problems associated to *locking* mechanisms. They also offer a refinement calculus for the stepwise development of concurrent programs from a formal specification [17, 24]. Moreover, most of these rules can be derived algebraically within the context of concurrent Kleene algebra [28]. Hence, every concrete model of concurrent Kleene algebra possesses a rely/guarantee framework.

Many practical problems lie within the intersection of probabilistic and concurrent systems, that is, the implementation of a solution requires programs that involve quantitative information as well as concurrent execution. Primary examples include Rabin's choice coordination which is an important probabilistic resource management algorithm [68]. It should be noted that powerful techniques such as model checking have been successfully extended to handle probabilistic behaviours using quantitative extensions of the underlying logics [2,13,42]. However, they have limited expressiveness and usually suffer from the state space explosion problem as the size of the system increases exponentially with respect to the number of components due to non-compositionality. At the time of writing this thesis, many lines of research are being investigated to achieve compositionality in the setting of model checking [35, 43]. Therefore, a unified compositional framework is the bridge towards a successful development of robust and reliable quantitative concurrent solutions. Moreover, since these kinds of problems are usually highly complex, it is legitimate to seek for a formal technique with high levels of simplic-

ity and abstraction while maintaining its full power to reason about quantitative concurrency. Hence, one asks the question:

> *Can we verify algebraically and compositionally probabilistic programs in the presence of interference?*

Contributions

To answer that question, this thesis contributes to the problem of analysing large, complex probabilistic and concurrent systems by proposing a unifying algebraic technique with high level of abstraction. Its main contributions are:

1. a new completeness result for continuous probabilistic Kleene algebra enabling a decision procedure based on minimisation modulo simulation [49],
2. a new model for concurrent Kleene algebra and a novel perspective about the partial order approach to the theory of concurrency,
3. a novel extension of Kleene algebra that is suitable for reasoning about quantitative programs with interferences. The algebra captures the interleaving approach as well as true concurrency [50, 52],
4. the development of the first extension of Jones' rely/guarantee calculus to probabilistic programs.

The first contribution is the product of our attempt to fill the gap in the completeness result conjectured by Takai and Furusawa in [79]. We show that Kozen's correspondence between equations in Kleene algebras and the equality of regular languages can be translated to continuous probabilistic Kleene algebra and regular "tree languages", which unsurprisingly characterise the simulation order. From this completeness result, we construct a decision procedure for continuous probabilistic Kleene algebra by minimising automata while preserving simulation equivalence. Hence, provable equality in continuous probabilistic Kleene algebra corresponds to isomorphism on minimal automata.

Secondly, I provide an alternative model for concurrent Kleene algebra using Langerak's bundle event structure [44] and a variant of Gischer's pomset equivalence. Pomsets are isomorphism classes of labelled partially ordered sets and can be compared with each other using label preserving bijections whose inverses are monotonic (this is called subsumption by Gischer [23]). Similar to the original non-dependence model of concurrent Kleene algebra [28], the new model provides a true concurrent interpretation and can be alternatively characterised using *schedulers* and *finishers*. Note that interleaving models exist and can be obtained using a specific class of finishers.

Probabilistic concurrent Kleene algebra then emerges from these two kinds of algebraic structures. We furthermore incorporate an explicit probabilistic choice, constrained by its own axioms, to achieve a direct control on probabilistic information. Concrete models of probabilistic concurrent Kleene algebra include an interleaving interpretation of concurrency based on probabilistic automata and Segala's probabilistic forward simulation [73] as well as a true concurrent model using probabilistic bundle event structures [33]. In the interleaving case, our approach is closely related to the work of Segala and Prima [64] as well as Deng et al. [15]. The main difference is our focus on a Kleene algebraic approach, which provide a grainless treatment of concurrency, rather than using process algebras in the style of Milner [18, 56, 57]. This thesis includes small examples that show the use of the algebra in proving properties of quantitative systems. But more importantly, it also contains a probabilistic extension of the standard rely/guarantee calculus to treat concurrency from a compositional and algebraic point of view.

In the standard formalisation of rely/guarantee calculus, a component is specified by the usual Hoare triple specification augmented with a *rely* condition that specifies the impact of the environment in which the component runs, and a *guarantee* condition that constrains the effect of the component on the environment. An environment can be thought of as a "background" program that has access to the global or shared variables. In the algebraic formalisation, rely and guarantee conditions have a very specific closure property that is expressed using the type

of concurrency considered [27]. More precisely, a rely condition contains all of the behaviours found in its duplicated concurrent execution. That closure property is preserved by special forms of probabilistic rely conditions and this thesis presents the first extension of rely/guarantee reasoning to the study of state-based quantitative concurrent programs. Notice that other researchers have also applied and extended such techniques to action-based systems [35, 43]. However, since the set of probabilistic automata under the simulation equivalence described in this thesis forms a model of probabilistic concurrent Kleene algebra, the rely/guarantee rules can directly be used in that model.

Synopsis

The structure of this thesis adheres tightly to the sequence of contributions. In Chapter 2, all necessary background from Kleene algebra, automata theory and simulation are revised. Moreover, it contains sections on the soundness as well as completeness of the automata/simulation model with respect to continuous probabilistic Kleene algebra. Section 2.4 describes the minimisation technique that is a translation of Bustan and Grumberg's work on Kripke structures [6]. Finally, a decision procedure is described.

Chapter 3 provides the necessary background about concurrent Kleene algebra as well a new model based on bundle event structures. The refinement order of that model is then characterised using the notion of resolution which is obtained from the interaction of a scheduler and a finisher on a given event structure.

The axiomatisation of probabilistic concurrent Kleene algebra is given in Chapter 4 where its soundness is established with respect to an interleaving model: probabilistic automata modulo simulation equivalence.

Chapters 5 and 6 contain the development of true concurrent models of probabilistic concurrent Kleene algebra with respectively explicit and implicit probability. The first truly concurrent model offers a true-concurrent definition of probabilistic simulation while the second model is fundamental for the state-based approach to the probabilistic rely/guarantee of Chapter 7.

Publications

The materials presented in Chapters 2, 4 and 5 (with the soundness result of Chapter 3) have been published respectively in:

[49] A. K. McIver, T. M. Rabehaja, and G. Struth. On probabilistic Kleene algebras, automata and simulations. In Proceedings of RAMICS11, pages 264-279, 2011.

[50] A. K. McIver, T. M. Rabehaja, and G. Struth. An event structure model for probabilistic concurrent Kleene algebra. In Proceedings of LPAR19, pages 653-667, 2013.

[52] A. K. McIver, T. M. Rabehaja, and G. Struth. Probabilistic concurrent Kleene algebra. In Proceedings of QAPL11, volume 117 of EPTCS, pages 97-115, 2013.

Chapter 2

Continuity in Probabilistic Kleene Algebra

Kleene algebras are a family of mathematical structures that are fundamental to many computing applications. Variants for specific models and tasks include processes [57, 70], probabilistic analysis [46, 53], program refinement [54, 55, 67, 86] or grainless concurrency [28]. The best studied variant, whose equational theory is completely characterised, has been introduced by Kozen [38]. A classical result relates Kozen's Kleene algebras to regular languages and the regular expressions that represent them. In other words, regular languages are models of this algebra and every valid identity between regular expressions can be derived from its axioms. However, much less is known about other variants of Kleene algebras where completeness results and decision procedures would be of comparable interest. In this chapter, we show that a completeness result similar to Kozen's can be achieved for probabilistic Kleene algebra.

The axioms of probabilistic Kleene algebra have been developed by McIver and Weber [53] to study probabilistic programs in the style of Conway [12]. The axiomatisation admits many concrete interpretations, ranging from probabilistic powerdomains [48] to the set of up-closed multirelations [20, 21]. It also possesses

transition-based models such as the set of automata modulo simulation equivalence [10, 49]. In particular, this chapter shows that continuous probabilistic Kleene algebra completely axiomatises simulation equivalence between automata, hence providing a solution to the same completeness property conjectured by Takai and Furusawa [79]. Continuity is the special ingredient here and we will establish it for the sequential composition of automata in Proposition 2.2.10. Equivalently, an equation $u = v$ is provable in continuous probabilistic Kleene algebra iff the automata associated to u and v are simulation equivalent. Hence, the existence/absence of a simulation between two automata can be used to decide the provability of the equality of two algebraic terms (without free variables). In Section 2.4, we refine the checking of a simulation by minimising the automata while preserving simulation equivalence. It is shown that minimal automata are simulation equivalent iff they are *isomorphic*. Hence a decision procedure for continuous probabilistic Kleene algebra is achieved by isomorphism checking on the corresponding minimal automata.

2.1 Probabilistic Kleene algebra

Probabilistic Kleene algebras have been introduced for resolving nondeterministic choice as they occur, for instance, in probabilistic protocols that involve adversarial scheduling [46, 53]. They are very similar to process algebras like CCS or ACP, but do not consider parallelism, communication or the notion of atomic action. In probabilistic Kleene algebra, simulation equivalence instead of bisimilarity is the underlying notion of equivalence which ensures that all defined algebraic operations are monotone.

2.1.1 Axiomatisation

Two axiomatisations of Kleene algebras were firstly suggested by Salomaa in his quest for a complete algebraic characterisation of the equality of regular languages [70]. Salomaa's original aximatisations were later refined by Conway [12],

then by Kozen, who gave a simplified proof of completeness with respect to regular language equivalence [38]. Many other variants emerged out of Kozen's axiomatisation, including *probabilistic concurrent Kleene algebras*.

Formally, a probabilistic Kleene algebra is a structure $(K, +, \cdot, 0, 1, *)$ where

- $(K, +, 0)$ is a commutative idempotent monoid (axioms 2.1-2.4),
- $(K, \cdot, 1)$ is a monoid (axioms 2.5-2.7),
- 0 is a left and right anihilator (axioms 2.8-2.9),
- the sequential composition $(\cdot)$ right-distributes and left-subdistributes through addition (axioms 2.10-2.11),
- the Kleene star $(*)$ satisfies the unfold (2.12), the left induction (2.13) and the right induction (2.14) axioms.

$$
\begin{aligned}
X + X &= X && (2.1)\\
X + Y &= Y + X && (2.2)\\
X + (Y + Z) &= (X + Y) + Z && (2.3)\\
X + 0 &= X && (2.4)\\
X \cdot 1 &= X && (2.5)\\
1 \cdot X &= X && (2.6)\\
X \cdot (Y \cdot Z) &= (X \cdot Y) \cdot Z && (2.7)\\
0 \cdot X &= 0 && (2.8)\\
X \cdot 0 &= 0 && (2.9)\\
(X + Y) \cdot Z &= X \cdot Z + Y \cdot Z && (2.10)\\
X \cdot Y + X \cdot Z &\leq X \cdot (Y + Z) && (2.11)\\
X^* &= 1 + X \cdot X^* && (2.12)\\
X \cdot Y \leq Y &\Rightarrow X^* \cdot Y \leq Y && (2.13)\\
Y \cdot (X + 1) \leq Y &\Rightarrow Y \cdot X^* \leq Y && (2.14)
\end{aligned}
$$

We assume that $(*)$ has priority over $(\cdot)$ which in turn has priority over $(+)$.

Table 2.1: Axioms of probabilistic Kleene algebra.

Notice that the unfold axiom (2.12) postulates the existence of a fixed point for the function $f(Y) = 1 + X \cdot Y$ and the left induction axiom (2.13) ensures that X^*

is the least fixed point of that function with respect to the semilattice order $\leq$ such that $X \leq Y$ iff $X + Y = Y$. In particular, the function $Y \mapsto X + Y$ is monotonic because if $Y \leq Z$ then $(X+Y)+(X+Z) = X+(Y+Z)$ by the associativity (2.3) and idempotence (2.1) axioms. Therefore $(X + Y) + (X + Z) = (X + Z)$, i.e., $X + Y \leq X + Z$.

2.2 Nondeterministic automata and simulation

Each variant of Kleene algebras usually axiomatises a different form of equivalence on automata. In Boffa [4], Krob [41] and Kozen's [38] completeness theorems, the set of automata modulo the equality of recognisable languages is the main axiomatised model. In Kozen's proof, the distributivity law

$$X \cdot Y + X \cdot Z = X \cdot (Y + Z) \tag{2.15}$$

plays a primary role in lifting the Kleene algebraic structure to the set of matrices over the algebra. Moreover, that distributivity law is necessary to ensure that a nondeterministic automaton and its deterministic version are representing the same Kleene algebra term.

However, only a subdistributivity law (Equation 2.11) holds in probabilistic Kleene algebra, because if X has a probabilistic outcome, then its output can be additionally used in the resolution of the nondeterministic choice $Y + Z$ in the expression $X \cdot (Y + Z)$. Such a resolution is impossible in the distributed expression $X \cdot Y + X \cdot Z$ because the choice is resolved before the execution of X. The subdistributivity axiom ensures that sequential composition is monotonic. In fact, they are equivalent because if $Y \leq Z$, i.e. $Y + Z = Z$, then $X \cdot Y + X \cdot Z \leq X \cdot (Y + Z) = X \cdot Z$, i.e. $X \cdot Y \leq X \cdot Z$. Conversely, if the multiplication by X from the left is monotonic, then $X \cdot Y \leq X \cdot (Y + Z)$ because $Y \leq Y + Z$. Similarly, $X \cdot Z \leq X \cdot (Y + Z)$ and thus $X \cdot Y + X \cdot Z \leq X \cdot (Y + Z) + X \cdot (Y + Z) = X \cdot (Y + Z)$, by the idempotence axiom 2.1.

Therefore, a stronger form of equivalence is needed for the soundness with respect to an automata model. This section presents a Kleene-style construction of an automata-theoretic model for probabilistic Kleene algebras, hence a soundness result. It makes use of Cohen's coalgebraic construction [10] using a variant of Brzozowski derivatives [5], which is perhaps more elegant, more explicit. The main difference between our approach and Cohen's is that we show and rely heavily on the continuity of sequential composition (Proposition 2.2.10).

If the left distributivity is altogether missing, then the resulting Kleene algebraic structure can be adapted to axiomatise bisimulation equivalence [18, 58, 69, 81, 82].

Definition 2.2.1. *A* nondeterministic finite automaton *is a tuple* $(G, \to, x_G, F)$ *where* G *is a finite set of states and* $\to \subseteq G \times (\Sigma \cup \{\varepsilon\}) \times G$ *is a transition relation. The set* Σ *is a fixed finite alphabet which is the same for all automata. The symbol* ε *denotes the empty word,* $x_G \in G$ *the initial state and* $F \subseteq G$ *the set of final states, which are underlined when drawing the automaton.*

In this thesis, an automaton is usually identified by its set of states (i.e. we simply denote an automaton by G instead of the tuple in Definition 2.2.1). The transition relation is indexed with G when confusions may arise.

An automaton is *accessible* if every state is reachable from the initial state and every other state reaches some final state. We will only consider accessible automata. We follow the standard construction in Kleene's theorem and inductively interpret terms of probablisitic Kleene algebras by nondeterministic finite automata.

2.2.1 Inductive construction of automata

Basic automata are defined as follows:

- The constant 0 corresponds to $(\{x\}, \emptyset, x, \emptyset)$. Diagrammatically, we have a single initial state x and no final state.

- The constant 1 corresponds to $(\{x\}, \emptyset, x, \{x\})$. Diagrammatically, we have a single initial and final state $\underline{x}$.

- The *basic automaton* a is drawn as $(\{x, x'\}, \{x \xrightarrow{a} x'\}, x, \{x'\})$. Diagrammatically, it corresponds to

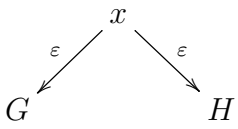
$$x \xrightarrow{a} \underline{x'} .$$

In the reminder of this section, we fix two automata $(G, \to_G, x_G, F_G)$ and $(H, \to_H, x_H, F_H)$. We also assume that the state spaces of G and H are disjoint. We now give an automata construction for each of the operations in $(+, \cdot, *)$.

- The *nondeterministic choice* between G and H is defined by:

$$G + H = (G \cup H \cup \{x\}, \to_G \cup \to_H \cup \{x \xrightarrow{\varepsilon} x_G, z \xrightarrow{\varepsilon} x_H\}, x, F_G \cup F_H)$$

where $x \notin G \cup H$. That is, we obtain the following diagram (where initial states of G and H have been abstracted away):



- The *sequential composition* of G followed by H is defined by:

$$G \cdot H = (G \cup H, \to_{G \cdot H}, x_G, F_H)$$

where

$$\to_{G \cdot H} = \to_G \cup \to_H \cup \{x \xrightarrow{\varepsilon} x_H \mid x \in F_G\}.$$

That is, we obtain the following diagram

$$G \xrightarrow{\varepsilon} H$$

- The *tail iteration* or *Kleene star* of G is $G^* = (G \cup \{x\}, \to_{G^*}, x, \{x\})$ where

$$\to_{G^*} = \to_G \cup \{x \xrightarrow{\varepsilon} x_G, y \xrightarrow{\varepsilon} x \mid y \in F_G\}$$

and $x \notin G$. That is, we obtain the following diagram: $\underline{x} \overset{\varepsilon}{\underset{\varepsilon}{\rightleftarrows}} G$

To obtain all axioms of probabilistic Kleene algebra on the set of automata endowed with these operations, it is sometimes necessary to reduce the resulting automaton to its accessible part after the application of any of these constructions. This is, for instance, required in the product $0 \cdot G$ or $G \cdot 0$, where every state of G becomes inaccessible.

2.2.2 Simulation and ε-closure

Given an automaton G and a state x, the *ε-closure* $\varepsilon(x)$ of x is the set of states which are reachable by ε-transitions only from x (including itself). The ε-extension of a transition $x \xrightarrow{a} y$ is obtained by performing a finite number of ε-transitions before the execution of the action a. That is, we write $x \overset{a}{\Longrightarrow} y$ (more precisely $x \overset{a}{\Longrightarrow}_G y$) iff there exists $x' \in \varepsilon(x)$ such that $x' \xrightarrow{a} y$. Similarly, we define $\overline{F} = \{x \mid \varepsilon(x) \cap F \neq \emptyset\}$.

Definition 2.2.2. *Let $(G, \to_G, x_H, F_G)$ and $(H, \to_H, x_H, F_H)$ be two automata. A relation $R \subseteq G \times H$ is a* simulation *from G to H if*

- $(x_G, x_H) \in R$,
- *for all $a \in \Sigma$, if $(x, y) \in R$ and $x \overset{a}{\Longrightarrow}_G x'$ then $(x', y') \in R$ for some y' such that $y \overset{a}{\Longrightarrow}_H y'$,*
- *if $(x, y) \in R$ and $x \in \overline{F_G}$, then $y \in \overline{F_H}$.*

We write $G \preceq H$ and say that H *simulates* G whenever there is a simulation $R \subseteq G \times H$. Simulations need not be total. There can be $x \in G$ on which a simulation is undefined (see Figure 2.1). The reason is that in the second

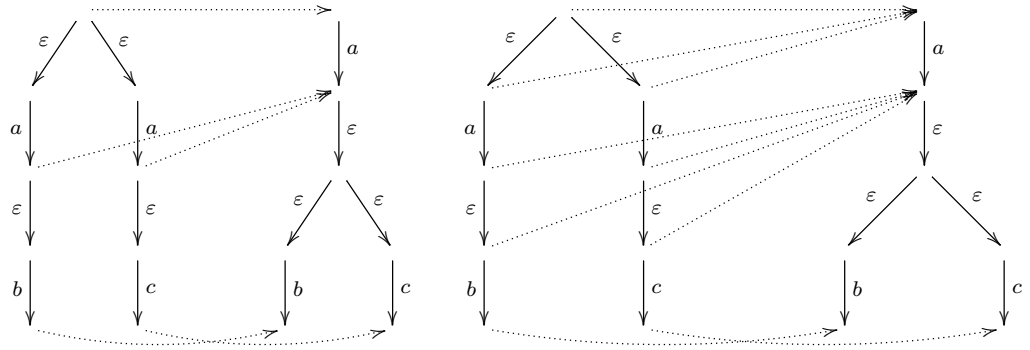


Figure 2.1: Simulations from $a \cdot b + a \cdot c$ to $a \cdot (b + c)$.

condition above, ε-transitions have not been considered. But every simulation R can be totalised by setting $R' = R \cup \{(x, y) \mid R.x = \emptyset \wedge \exists x' \cdot (x \in \varepsilon(x') \wedge (x', y) \in R)\}$.

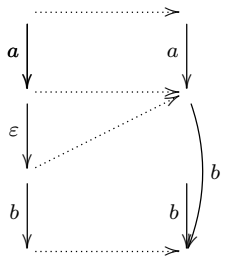
Example 2.2.3. Figure 2.1 (a) provides an example of simulation from $a \cdot b + a \cdot c$ to $a \cdot (b + c)$. The total version is drawn in Figure 2.1 (b). Notice that the ε-transitions were introduced by the definition of $(\cdot)$ and $(+)$ and they can be removed "safely" using Proposition 2.2.5. ■

It is well known that simulations on $G \times H$ are closed under union and relational composition. It follows that all simulations can be extended to maximal ones. It is also well known that simulations induce preorders and equivalences. Two automata G and H are *simulation equivalent*, written $G \cong H$, if $G \preceq H$ and $H \preceq G$. The simulation equivalence and simulation order corresponds to the algebraic constructs $=$ and $\leq$.

Lemma 2.2.4. *Given to automata G, H, we have $G \preceq H$ iff $G + H \cong H$.*

Proof. If $G \preceq H$ then $G + H \preceq H + H \preceq H$ (the last simulation is essentially the identity on H). Since $H \preceq G + H$, we have $H \cong G + H$. Conversely, if $G + H \preceq H$ then the restriction of the simulation on the states of G will generate a simulation from G to H. ■

We close this subsection by showing that ε-elimination is possible in our automata model, using the standard technique. The general idea is shown in the following diagram:



The ε-transition on the left-hand automaton is remove and a new transition labelled by b is introduced. Indeed, the state to which the discarded ε-transition pointed to becomes unreachable, but the right-hand automata is made accessible by removing all unreachable states.

Proposition 2.2.5. *Each accessible automaton is simulation equivalent to an accessible ε-free automaton.*

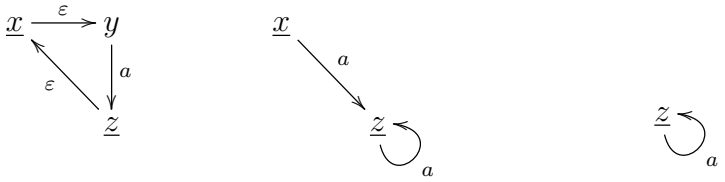
Proof. Let G be an (accessible) automaton and let G_ε be an automaton such that $x \xrightarrow{a}_{G_\varepsilon} x'$ iff there exists $y \in \varepsilon(x)$ such that $y \xrightarrow{a}_G x'$. The automaton G_ε has the same initial state x_G as G and its set of final states is:

$$F_{G_\varepsilon} = \{y \in \overline{F_G} \mid y \text{ accessible in } G_\varepsilon\}.$$

Notice G_ε has no transition labelled with ε and that it is further reduced to its accessible part if required.

First, we show that $G \preceq G_\varepsilon$. Consider the relation $R \subseteq G \times G_\varepsilon$ such that $(x, y) \in R$ iff $x \in \varepsilon(y)$ in G. We must show that R is indeed a simulation. We have $(x_G, x_G) \in R$ because $x_G \in \varepsilon(x_G)$.

- Assume $(x, y) \in R$ and $x \in \overline{F_G}$, by definition $\varepsilon(x) \cap F_G \neq \emptyset$ but $\varepsilon(x) \subseteq \varepsilon(y)$ so $y \in F_{G_\varepsilon}$.
- Let $a \in \Sigma$, $x \xrightarrow{a} x'$ be a transition of G and $(x, y) \in R$. Since $x \in \varepsilon(y)$, we deduce that $y \xrightarrow{a} x'$ is a transition of G_ε by definition of its set of transition.



The ε-free automaton can further be reduced to the right-most automaton while preserving simulation equivalence. Hence counterexamples will be expressed using the compact transition system.

Figure 2.2: Removing ε-transitions in a^*.

With similar argument, we can further show that R^{-1} is also a simulation. Hence $G_\varepsilon \preceq G$ and we deduce $G \cong G_\varepsilon$. ■

The reduction of an automaton to an ε-free version highly simplifies the interpretation of a pKA term with an automaton. For example, given an action a, the automaton associated to a^* is reduced to a self loop (Figure 2.2).

2.2.3 Soundness of probabilistic Kleene algebra

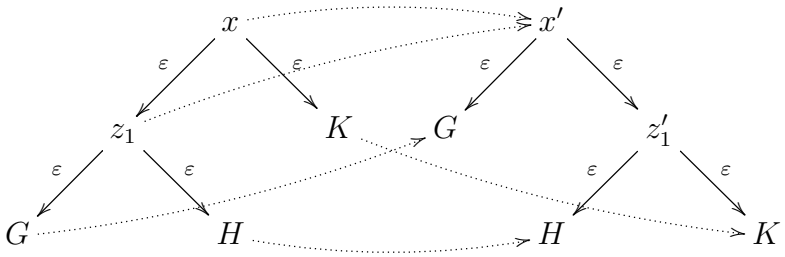
The following soundness theorem shows that the set **Aut** of accessible automata ordered by simulation is "almost" a probabilistic Kleene algebra. In fact, **Aut** satisfies all axioms of Figure 2.1 but we only establish all axioms other than the equational implication (2.14) in Theorem 2.2.6. The unfold (2.12) and the left induction law (2.13) ensure that the construction of Kleene star G^* in Subsection (2.2.1) coincides (up to simulation equivalence) with the supremum of the sequence $f^n(0)$ where $f(X) = 1 + G \cdot X$. Notice that G^* is a finite automaton while computing the limit of $f^n(0)$ directly may result in an automaton with infinitely many states. The implication 2.14 is then a consequence of continuity and Lemma 2.2.8.

Theorem 2.2.6. *The structure* $(\mathbf{Aut}, +, \cdot, ^*, 0, 1)$ *modulo simulation equivalence satisfies Axioms (2.1-2.13).*

Proof. Let G, H, K be automata. The following proofs show sketches of the simulations between the expressions involved in each of the axioms.

- Axioms (2.1-2.2): the simulations for $G + H \cong H + G$ and $G + G \cong G$ are trivial.

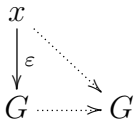
- Axiom (2.3): the relation $R \subseteq ((G + H) + K) \times (G + (H + K))$ defined by the following diagram is a simulation.



  We can use a similar construction for the converse direction.

- Axiom (2.4): we can use the identity simulation because $0+G$ is transformed into G by making the automaton accessible.

- Axioms (2.5-2.6). For $1 \cdot G \cong G$, we use the simulation

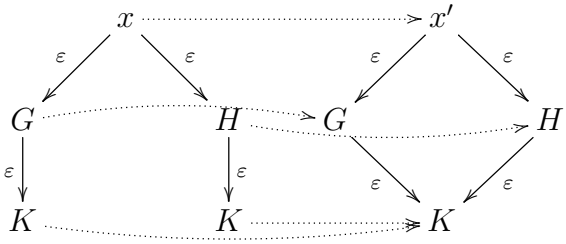


  and its converse. The simulation for $G \cdot 1 \cong G$ is similar.

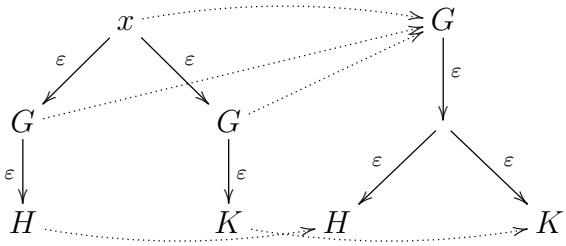
- Axiom (2.7): the automata corresponding to the left-hand and the right-hand side expressions are identical by construction.

- Axiom (2.8-2.9): the automata $G \cdot 0$ and 0 have no final state. By making the left-hand side accessible, it becomes identical to the right-hand side. Similarly for $0 \cdot G \cong 0$. In the left-hand side, G is not reachable and the automaton becomes 0 by making it accessible.

- Axiom (2.10): the following figure gives a simulation that works both ways.

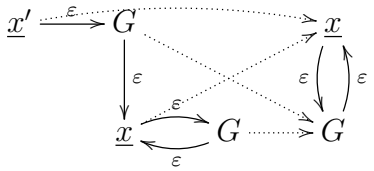


- Axiom (2.11): the simulation is shown in the following diagram.



The initial state x is mapped to the initial state of G and the dotted arrows from G to G (resp. H to H, resp. K to K) are the identity relations on G (resp. H, resp. K).

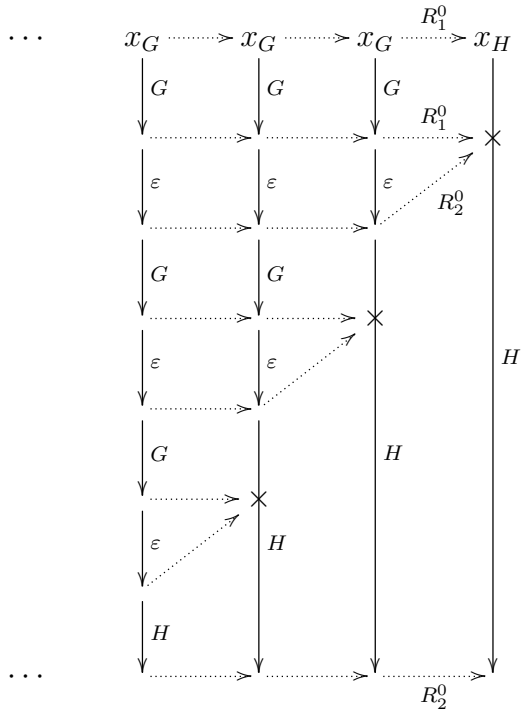
- Axioms (2.12): the following construction yields a simulation $1+G\cdot G^* \preceq G^*$ for one unfold of G^*.



where $\underline{x}'$ is the initial state of $1 + G \cdot G^*$. Moreover, the presented relation is a bisimulation (i.e. its inverse is also a simulation). Hence we have both $1 + G \cdot G^* \preceq G^*$ and $G^* \preceq 1 + G \cdot G^*$.

- Axioms (2.13): let $R \subseteq (G \cdot H) \times H$ be the maximal simulation. We write $G^n = G_n \cdots G_1$, where G_n is the n-th copy of the automaton G. We write

x_n for the copy of state x in G_n. The notation in the following diagram has been changed to show the construction more clearly.



We define $R = R^0 = R_1^0 \cup R_2^0$ where $R_1^0 \subseteq G \cdot \varepsilon \times H$ and $R_2^0 \subseteq H \times H$ as in the diagram, so that R^0 is a simulation. We inductively define $R^n = R_1^n \cup R_2^n$ where $R_1^n = R_1^0$ and $R_2^n = R^{n-1} \circ \cdots \circ R^0 \circ R_2^0$. It follows by induction that each R^n is a simulation for $G^n \cdot H \preceq H$.

We now define $R^* \subseteq G^* \cdot H \times H$ and show that it is a simulation for $G^* \cdot H \preceq H$. For $x, y \in G \cup H$ we define

$$(x,y) \in R^* \text{ iff } \begin{cases} (x_j, y) \in R^n \text{ for some } j, n \text{ with } j \leq n, & \text{if } x \in G, \\ (x,y) \in R, & \text{if } x \in H. \end{cases}$$

The initial state x_{G^*} of G^* is mapped to every state of H in the image of a copy of the initial state x_G of G under R^n. The final states of $G^* \cdot H$ are related to those of H. We now prove that R^* is a simulation by inspecting

transitions in the automata.

First, $(x_{G^*}, x_H) \in R^*$ since $(x_G, x_H) \in R^0$.

Next, suppose $x \in \overline{F_{G^* \cdot H}}$ and $(x, y) \in R^*$. There are two cases: (i) If $x \in H$, then (x, y) is already in R. (ii) If $x \in G \cup \{x_{G^*}\}$ then $1 \preceq H$, so $x \in \overline{F_G} \cup \{x_{G^*}\}$. By definition $(x_i, y) \in R^n$ for some i and n. Therefore $x_i \in \overline{F_{G^n \cdot H}}$ and consequently $y \in \overline{F_H}$ because R^n is a simulation (consider the diagram).

Next, suppose $(x, y) \in R^*$ and $x \xrightarrow{a} x'$ is the result of a transition in the automaton $G^* \cdot H$, that is, it is either a transition in H or a transition in G^* or a transition from some final state of G^* to some state in H. We distinguish three cases. (i) If $x \in H$, then we are done since the simulation used for the step is R by definition. Otherwise, let us assume that x is not in H. There exists i, n such that $(x_i, y) \in R^n$. (ii) $x \xrightarrow{a} x'$ is obtained by a transition in G. Then, since R^n is a simulation, there exists $y \xrightarrow{a} y'$ such that $(x'_i, y') \in R^n$. Hence $(x', y') \in R^*$. (iii) $y \xrightarrow{a} y'$ is obtained by a transition of the form $x \xrightarrow{\varepsilon} x_{G^*} \xrightarrow{\varepsilon a} x'$. In the diagram, by ε-closure, there will therefore be additional edges that can either loop back into G or lead into H. That is, $x_i \xrightarrow{a} x'_{i-1}$ or $x' \in H$ and $i = 1$. In the first case, when we loop back into G, there exists a state y' such that $(x'_{i-1}, y') \in R^n$. Therefore, by definition, $(x', y') \in R^*$. In the second case, when the transition leads into H, there exists a state y' such that $(x', y') \in R$, by definition. Again, $(x', y') \in R^*$. ■

The following examples show that the axioms for automata under simulation equivalence can neither be weakened nor strengthened.

Example 2.2.7.

1. It is clear by considering the diagram for subdistributivity in the proof of Theorem 2.2.6 that a simulation from the right-hand automaton to the left-hand automaton is impossible. This refutes left distributivity for our model.

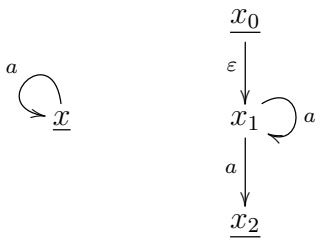


Figure 2.3: There is no simulation from a^* to $1 + a^* \cdot a$.

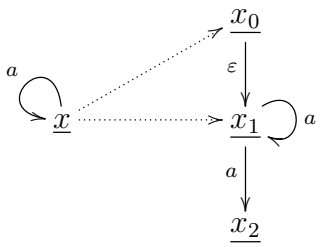
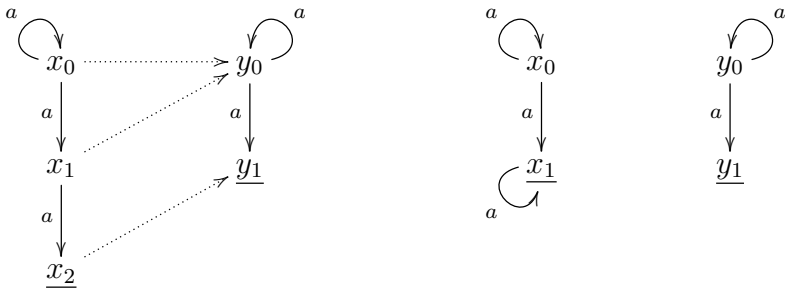


Figure 2.4: A simulation between compact interpretations of a^* and $1+a^* \cdot (a+1)$.

2. The left star unfold axiom can be strengthened to $1 + X \cdot X^* = X^*$, but the inequality $X^* \leq 1 + X^* \cdot X$ is not valid.

   It is clear that there cannot be a simulation between the automata of Figure 2.3 because x_0, x_2 and x are the final states (underlined) and hence x cannot be simulated by x_1. But Figure 2.4 shows the existence of a simulation from a^* to $1 + a^*(a + 1)$, because x_1 is now a final state.

3. A right star induction law $Y \cdot X \leq Y \Rightarrow Y \cdot X^* \leq Y$ does not hold.



   The left diagram shows a simulation from $a^* \cdot a \cdot a$ to $a^* \cdot a$ though there

is no simulation between the automata in the right-hand side i.e. refuting $a^* \cdot a \cdot a^* \leq a^* \cdot a$.

4. Kozen's counter-example [36] on Kleene algebras possessing a least fixpoint for $1 + a \cdot X$ but not for $1 + X \cdot a$ still holds in our setting (i.e. for $1 + X \cdot (a + 1)$). Therefore the right induction axiom of probabilistic Kleene algebras is independent.

■

2.2.4 Continuity of sequential composition

The star unfold (Axiom 2.12) and left induction law (Axiom 2.13) postulate that the least fixed point of $f(Y) = 1 + X \cdot Y$ exists. The standard computation of the least fixed point can be achieved by iterating the monotonic function f from the bottom 0 on the considered mathematical model. In general, the least fixed point cannot be obtained by accumulating finite behaviours only. However, with a stronger hypothesis, such as continuity, it can be expressed as the aggregation of all finite iterations of f, i.e., $X^* = \sup_{n \in \mathbb{N}} f^n(0)$. In the work of Kozen [37], the sequence involved in the computation of the Kleene star is described in his *star-continuity* property. That is, $Y \cdot X^* \cdot Y' = \sup_n (Y \cdot f^n(0) \cdot Y')$.

In this subsection, we present a more general form of continuity using *directed sets*. A set $\mathcal{A}$ is directed if for every $X, Y \in \mathcal{A}$, there exists $Z \in \mathcal{A}$ such that $X \leq Z$ and $Y \leq Z$. A probabilistic Kleene algebra K is *continuous* if the sequential composition is continuous from the left and the right, that is, if it distributes over left and right directed joins:

$$X \cdot (\sup \mathcal{A}) = \sup\{X \cdot Y \mid Y \in \mathcal{A}\} \quad \text{and} \quad (\sup \mathcal{A}) \cdot X = \sup\{Y \cdot X \mid Y \in \mathcal{A}\}$$

hold for all elements X and directed subsets $\mathcal{A}$ of the carrier set K with supremum $\sup \mathcal{A}$. It should be noted that we only need *conditional continuity*, that is, only existing suprema of directed sets need to be preserved. In particular, if

the supremum of the increasing powers of the function f above exists then it is preserved by sequential composition from the left and the right. This corresponds to Kozen's star-continuity.

Lemma 2.2.8. *In every continuous probabilistic Kleene algebra*

$$X^* = \sup_{n\in\mathbb{N}}(1+X)^n.$$

Proof. The continuity hypothesis implies that $X^* = \sup_n f^n(0)$. It then suffices to prove by induction that $f^{n+1}(0) = (1+X)^n$. For $n = 0$, $f^1(0) = 1 + X \cdot 0 = 1$ and $(1+X)^0$ by convention. Assume that $f^n(0) = (1+X)^n$. On the one hand, we have $(1+X)^{n+1} = (1+X)\cdot(1+X)^n = (1+X)^n + X\cdot(1+X)^n \geq 1 + X\cdot f^n(0)$ because of the induction hypothesis and the fact that $(1+X)^n \geq 1$. On the other hand, since f is monotonic, we have $f^{n+1}(0) \geq f^n(0)$. But $f^{n+1}(0) \geq X \cdot f^n(0)$, therefore, $f^{n+1}(0) = f^n(0) + X \cdot f^n(0) = (1+X)^n$. ■

Lemma 2.2.9. *The operation + distributes through arbitrary suprema.*

Proof. Consider a family Y_i such that $\sup_i Y_i = Y$. Then $\sup_i(X+Y_i) \leq X+Y$ by monotonicity. Conversely, if $Z \geq X+Y_i$ for all i, then $Z+X+Y \geq X+Y_i+X+Y = X+Y$, hence $\sup_i(X+Y_i) \geq X+Y$. ■

We now establish the continuity of the sequential composition in the set of accessible automata modulo simulation equivalence.

Proposition 2.2.10. *The sequential composition of automata is conditionally continuous i.e. multiplication from left and right preserves the supremum of every directed set (if it exists).*

Proof. We first define a notion of residuation on automata. We then establish that the residuation $(/)$ and sequential composition $(\cdot)$ form a Galois connection, that is, for every automata G, H and K: $K \cdot H \preceq G$ iff $K \preceq G/H$. Right continuity follows from this property because if a family $(K_i)_i$ with supremum K satisfies $K_i \cdot H \leq G$ for every i, then $K_i \preceq G/H$ for every i. Hence we deduce, by definition of suprema, that $K \preceq G/H$ i.e. $K \cdot H \preceq G$.

Let us construct the residuation. For automata G and $H \neq 0$ we define the automaton G/H with initial state $x_{G/H} = x_G$ (the initial state of G), final states $F_{G/H} = \{x \in G \mid H \preceq G_x\}$, where G_x is constructed from G by translating its initial state into x. We make the resulting automaton accessible by discarding all states and edges that do not lead to a final state.

We now show that $K \cdot H \preceq G$ iff $K \preceq G/H$. Assume R is a simulation from $K \cdot H$ to G. That means R is in particular a simulation from H to G_x for some state x of G. By definition of G/H, therefore, R is a simulation from K to G/H because the state x of G becomes final state of G/H and they are images of the final states of K under R.

For the converse direction, suppose that R is a simulation from K to G/H. By axioms 2.11 and 2.10, multiplication is monotonic, hence $K \cdot H \preceq (G/H) \cdot H$, and it remains to show that $(G/H) \cdot H \preceq G$.

First, if $F_{G/H}$ is empty then $G/H = 0$ (after making it accessible) and the result follows.

Otherwise, assume that R' is the simulation from $K \cdot H$ to $(G/H) \cdot H$. By construction of G/H, we also know that there exists a simulation S_x from H to G_x for all final state x of G/H and that there is a simulation (except for the final state property) between G/H and G, namely the identity relation id. Hence $S' = (\cup_x S_x) \cup id$ is indeed a simulation from $(G/H) \cdot H$ to G and $R' \circ S'$ is a simulation from $K \cdot H$ to G.

It then follows from general properties of Galois connections [1] that $L \mapsto L \cdot H$ is (conditionally) completely additive, hence right continuous.

It remains to show left continuity. Let $(G_i)_i$ be a directed set of automata such that $\sup_i G_i = G$ and let H be any automaton. Then $\sup_i(H \cdot G_i) \preceq H \cdot G$ because multiplication is monotone and it remains to show $H \cdot G \preceq \sup_i(H \cdot G_i)$. Let us assume that $\sup_i(H \cdot G_i) \preceq K$. We will show that $H \cdot G \preceq K$.

By definition of supremum, $H \cdot G_i \preceq K$ for all i, hence there is a set of states $X_i = \{x \in K \mid G_i \preceq K_x\}$, that is, the set of all those states in K from which G_i is simulated. Obviously, $X_i \subseteq X_j$ if $G_j \preceq G_i$ in the directed set. But since K has

only finitely many states, there must be a minimal non-empty set X in the directed set $(X_i)_i$ such that all G_i are simulated by K_x for $x \in X$. By definition, therefore, $G = \sup_i G_i \preceq K_x$ for all $x \in X$. There exists a simulation $S_X \subseteq (H \cdot G_i) \times K$ for some i such that the residual automaton K/G_i has precisely X as its set of final states. We can thus take the union of S_X restricted to H and all simulations yielding $G \preceq K_x$ for all $x \in X$ and verify that this is indeed a simulation of $H \cdot G$ to K. ■

2.2.5 Constructing automata from pKA terms

Let Σ be fixed alphabet. It is standard knowledge that regular expressions are mainly constructed inductively from Σ using the operations $(+, \cdot, ^*)$ and constants $0, 1$. We write T_Σ for the set of such terms and equate them modulo provability using the axioms of Table 2.1. Each regular expression can then be transformed into an automaton by directly translating each term and sub-terms with the operations of Section 2.2.1 followed by ε-removal (Proposition 2.2.5). Concretely, if we denote this correspondence by G, then

- $G(0) = 0, G(1) = 1$ and $G(a)$ is the basic automaton created from a, whose initial state is labelled by a and the final state is labelled by 1.
- $G(u + v)$ is the ε-free version of $G(u) + G(v)$. The initial state is labelled by $u + v$.
- $G(u \cdot v)$ is the ε-free version of $G(u) \cdot G(v)$ where a state of $G(u)$ labelled by w is relabelled with $w \cdot v$ in the automaton $G(u \cdot v)$. The initial state is $u \cdot v$.
- $G(u^*)$ is the ε-free version of $G(u)^*$. The initial state is labelled by u^* and all other state of $G(u)$ labelled by w is relabelled to $w \cdot u^*$ in the automaton $G(u^*)$.

It is clear that G is a homomorphism, i.e. $G(u + v) \cong G(u) + G(v), G(u \cdot v) \cong G(u) \cdot G(v)$ and $G(u^*) \cong G(u)^*$ for all terms u, v. In other words, each state of $G(u)$ is again labelled by a term, say v, and $G(u)$ translated to that state is $G(v)$.

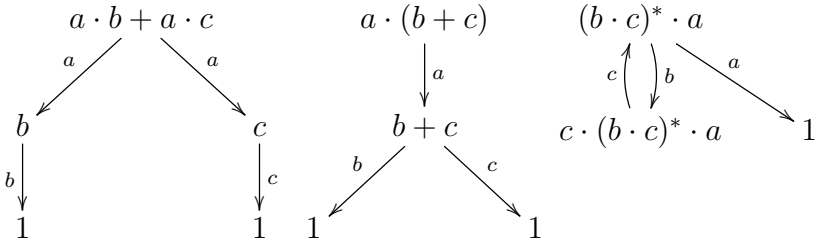


Figure 2.5: The automata $G(a \cdot b + a \cdot c)$, $G(a \cdot (b + c))$ and $G((b \cdot c)^* \cdot a)$.

Hence, we will denote each state of $G(u)$ by the term associated with it. Notice that this labelling is a straightforward adaptation of Brzozowski derivatives [5] and a similar technique was used by Cohen [10].

Example 2.2.11. $G(a \cdot b + a \cdot c)$, $G(a \cdot (b + c))$ and $G((b \cdot c)^* \cdot a)$ are depicted in Figure 2.5. Since the ε-removal technique of Proposition 2.2.5 preserves simulation equivalence, the first two pictured automata are respectively simulation equivalent to the ones detailed in Figure 2.1. ■

Final states of $G(u)$ are characterised by the following endomorphism [5]:

- $o(1) = 1$, $o(0) = 0$, and $o(a) = 0$ for each action $a \in \Sigma$.
- $o(u + v) = o(u) + o(v)$, $o(u \cdot v) = o(u) \cdot o(v)$ and $o(u^*) = 1$.

The following proposition says that each regular expression is equivalent (up to provability by the axioms of probabilistic Kleene algebra) to a decomposition produced by the transition relation of the associated automaton $G(u)$. It adapts a similar statement and proof by Milner [56] and Cohen [10].

Proposition 2.2.12. *For every term u, it can be proved in probabilistic Kleene algebra that*

$$u = \sum_{a \in \Sigma} \sum_{u \xrightarrow{a}_{G(u)} u'} a \cdot u' + o(u)$$

where $\longrightarrow_{G(u)}$ is the of the transition relation of $G(u)$.

Proof. By structural induction. We only consider the induction steps.

- If the term is of the form $u + v$, then

$$u = \sum_{a\in\Sigma} \sum_{u \xrightarrow[G(u)]{a} u'} a \cdot u' + o(u) \qquad \text{and} \qquad v = \sum_{a\in\Sigma} \sum_{v \xrightarrow[G(v)]{a} v'} a \cdot v' + o(v).$$

  By definition of ε-removal, $u + v \xrightarrow{a}_{G(u+v)} w$ iff either $u \xrightarrow{a}_{G(u)} w$ or $v \xrightarrow{a}_{G(v)} w$ holds. It then follows that

$$\sum_{a\in\Sigma} \sum_{u+v \xrightarrow[G(u+v)]{a} w} a{\cdot}w{+}o(u{+}v) = \sum_{a\in\Sigma} \sum_{u \xrightarrow[G(u)]{a} w} a{\cdot}w{+}o(u){+}\sum_{a\in\Sigma} \sum_{v \xrightarrow[G(v)]{a} w} a{\cdot}w{+}o(v).$$

  The right hand side evaluates to $u + v$.

- In the product case $u \cdot v$, we assume the same sums for u and v as before. We have

$$\begin{aligned} u \cdot v &= \left(\sum_{a\in\Sigma} \sum_{u \xrightarrow[G(u)]{a} u'} a \cdot u' + o(u) \right) \cdot v \\ &= \sum_{a\in\Sigma} \sum_{u \xrightarrow[G(u)]{a} u'} a \cdot u' \cdot v + o(u) \cdot \left(\sum_{a\in\Sigma} \sum_{v \xrightarrow[G(v)]{a} v'} a \cdot v' + o(v) \right) \end{aligned}$$

  Since, $o(u) \in \{0, 1\}$, it can be distributed through the bracketed sum. Moreover, $u \cdot v \xrightarrow{a}_{G(u\cdot v)} w$ holds iff

  - $u \xrightarrow{a}_{G(u)} u'$ and $u' \cdot v = w$ or
  - $o(u) = 1$ and $v \xrightarrow{a}_{G(v)} w$ holds.

  Hence,

$$u \cdot v = \sum_{a\in\Sigma} \sum_{u\cdot v \xrightarrow[G(u\cdot v)]{a} w} a \cdot w + o(u \cdot v)$$

- Finally, for the case of $*$, we can assume without loss of generality that

$o(u) = 0$ because $(u+1)^* = u^*$ in probabilistic Kleene algebras. Therefore

$$u^* = u \cdot u^* + 1 = \left(\sum_{a \in \Sigma} \sum_{u \xrightarrow{a}_{G(u)} u'} a \cdot u' \right) \cdot u^* + 1 = \sum_{a} \sum_{u \xrightarrow{a}_{G(u^*)} w} a \cdot w + o(u^*).$$

The second step uses the induction hypothesis. The third step follows from the construction of transitions in $G(u^*)$ and the fact that $o(u^*) = 1$. ■

2.3 A completeness result for continuous pKA

In this section we simply call a *tree* an automaton whose graph is a directed acyclic graph[1]. In the case of accessible automata, all leaves (nodes without any child) are final states, but there may also be some internal final states. In contrast to Furusawa and Takai's approach [79], where the soundness of the axioms of probabilistic automata is proven against a particular set of tree-automata, we are dealing with standard automata. Our approach is therefore more similar to Cohen's [10].

It is clear from our inductive construction of automata that each tree is the interpretation of some $*$-free term in probabilistic Kleene algebra. If T is a tree then u_T denotes a $*$-free term such that $T \cong G(u_T)$.

Proposition 2.3.1. *Let T and T' be trees and v be a term.*

1. *If $T \preceq G(v)$, then $u_T \leq v$ is provable in probabilistic Kleene algebra.*
2. *If $T \preceq T'$, then $u_T \leq u_{T'}$ is provable in probabilistic Kleene algebra.*

Proof. Let $T \preceq G(v)$, without loss of generality we assume that the automaton T is also ε-free.

Assume that $T \cong G(u_T)$ for some $*$-free term u_T and consider a leaf in $G(u_T)$. Let x be a state of $G(u_T)$ and y be a state of $G(v)$ such that $G(u_T)_x \preceq G(v)_y$. We

[1] Notice that $G((a+b) \cdot c)$ is not a tree but a dag. However, it is simulation equivalent to a tree.

reason by induction on the maximal distance between x and all leaves accessible from x.

- if x is a leaf then $x = 1$ and y is a final state of $G(v)$. Hence $x = 1 \leq y$.
- Otherwise, by Proposition 2.2.12,

$$x = \sum_a \sum_{x \xrightarrow{a}_{G(u_T)} x'} a \cdot x' + o(x) \tag{2.16}$$

  where x' is strictly closer to a leaf than x. Similarly $y = \sum_a \sum_{y \xrightarrow{a}_{G(v)} y'} a \cdot y' + o(y)$ and by definition of simulation and the induction hypothesis, for every x' in Equation (2.16), there exists a corresponding y' such that $x' \leq y'$ is provable in probabilistic Kleene algebra. Hence $x \leq y$ is also provable by monotonicity of $(+)$ and $(\cdot)$.

The case where both automata are trees is an instance. ■

As a consequence of this proposition, we will denote a $*$-free term and a tree by the same notation, usually t. For each automaton G, consider the set of trees

$$\tau(G) = \{t \mid t \preceq G \wedge t \text{ is a tree}\}.$$

This set is stable under addition and is down-closed by definition. We define the operations

$$\begin{aligned} \tau + \tau' &= \{t + t' \mid t \in \tau \wedge t' \in \tau'\}, \\ \tau\tau' &= \downarrow\{t \cdot t' \mid t \in \tau \wedge t' \in \tau'\}, \\ \tau^* &= \downarrow\{(t+1)^n \mid t \in \tau \wedge n \in \mathbb{N}\}, \end{aligned}$$

where $\downarrow\tau$ denotes the down-closure of τ. All these sets are again stable under addition and are down-closed.

The previous proposition implies that $\tau(G(u)) = \{G(t) \mid t \leq u \wedge t \text{ is } *\text{-free}\}$. We denote $\tau(u) = \{t \mid t \leq u \wedge t \text{ is } *\text{-free}\}$.

Proposition 2.3.2. *τ is a homomorphism to sets of $*$-free terms.*

Proof. It is clear that $\tau(u)+\tau(v) \subseteq \tau(u+v)$. For the converse inclusion suppose $t \in \tau(u+v)$, that is, $t \leq u+v$ so $G(t) \preceq G(u+v) \cong G(u)+G(v)$. Since the automata are disjoint, we can decompose t into t_u+t_v such that $t_u \leq u$ and $t_v \leq v$ (this is possible because t is a tree). Therefore $t \in \tau(u)+\tau(v)$.

We have $\tau(u)\tau(v) \subseteq \tau(u \cdot v)$ by monotonicity. If $t \in \tau(u \cdot v)$ then $G(t) \preceq G(u \cdot v) \cong G(u)G(v)$. Then $t = t_u \cdot (t_{v_1}, \ldots, t_{v_n})$ for some $t_u \in \tau(u)$ and $t_{v_i} \in \tau(v)$. So $t \leq t_u \cdot (\sum_i t_{v_i}) \in \tau(u)\tau(v)$.

By monotonicity of $(*)$, $\tau(u)^* \subseteq \tau(u^*)$. Conversely, let $t \in \tau(u^*)$, then $G(t) \preceq G(u)^*$ so $t \leq (t'+1)^n$ for some $t' \in \tau(u)$ and $n \in \mathbb{N}$. In fact, since t has finite depth, we may unfold u^* finitely many times and reason as in the case of sequential composition to construct t'. ■

The following theorem shows that the simulation order $\preceq$ corresponds to tree language inclusion.

Theorem 2.3.3. *$G \preceq H$ iff $\tau(G) \subseteq \tau(H)$.*

Proof. The forward implication is clear by transitivity of $\preceq$. For the converse implication, suppose $\tau(G) \subseteq \tau(H)$ and consider the relation $R \subseteq G \times H$ such that $(x, y) \in R$ iff $\tau(G_x) \subseteq \tau(H_y)$. We show that R is a simulation (the maximal simulation, in fact). We must check the three defining conditions. (i) The initial states are indeed in R. (ii) If $(x, y) \in R$ and $x \in F_G$, then $1 \in \tau(G_x) \subseteq \tau(H_y)$, so $y \in F_H$. (iii) Assume, by contradiction, that $(x, y) \in R$, $x \xrightarrow{a} x'$ and for every y'_i such that $y \xrightarrow{a} y'_i$ holds in H, there exists $t_i \in \tau(x')$ such that $t_i \notin \tau(y'_i)$. Since there are only finitely many such y'_i, we define $t = \sum_i t_i \in \tau(x')$ and therefore, $a \cdot t \in \tau(x) \subseteq \tau(y)$. Thus, there exists y'_{i_0} such that $y \xrightarrow{a} y'_{i_0}$ and $t \preceq G_{y'_{i_0}}$. Since $\tau(G_{y'_{i_0}})$ is down-close and $t_{i_0} \leq t$, one obtains $t_{i_0} \in \tau(G_{y'_{i_0}})$ which is a contradiction. ■

We now characterise terms in continuous probabilistic Kleene algebras by the set of ($*$-free) terms or trees that approximate them from below.

Proposition 2.3.4. *Every term u of a continuous probabilistic Kleene algebra satisfies $u = \sup \tau(u)$.*

Proof. By structural induction. We already know that $\sup \tau(u) \leq u$ and that $\tau(u)$ is directed.

- For the base case, if u is a tree then $u \in \tau(u)$ and we have $\sup \tau(u) = u$.
- If $u = u_1 + u_2$ then by Proposition 6.4.3, $\tau(u) = \tau(u_1) + \tau(u_2)$ so $\sup \tau(u) = \sup\{t_1 + t_2 \mid t_1 \in \tau(u_1) \wedge t_2 \in \tau(u_2)\}$. Let $t_1 \leq u_1$, by continuity and induction hypothesis $t_1 + u_2 = \sup\{t_1 + t \mid t \leq u_2\} \leq \sup \tau(u)$. Therefore, by continuity again $u \leq \sup\{t_1 + u_2 \mid t_1 \leq u_1\} \leq \sup \tau(u)$. Hence $u = \sup \tau(u)$.
- Let $u = u_1 \cdot u_2$, we have $\tau(u) = \tau(u_1)\tau(u_2)$ and we use the same reasoning as before. Let $t_1 \in \tau(u_1)$, then by continuity and induction hypothesis, $t_1 \cdot u_2 = \sup\{t_1 \cdot t \mid t \leq u_2\} \leq \sup \tau(u)$. By continuity again, $u_1 u_2 = \sup\{t_1 \cdot u_2 \mid t_1 \leq u_1\} \leq \sup \tau(u)$. we conclude $u \leq \sup \tau(u)$. Hence $u = \sup \tau(u)$.
- Let $u = v^*$. Then by Proposition 6.4.3, $\tau(u) = \tau(v)^*$ and we have to show $u \leq \sup\{(t+1)^n \mid t \leq v \wedge n \in \mathbb{N}\}$ by definition of τ^*. But $\sup\{(t+1)^n \mid t \leq v\} = (v+1)^n \leq \sup \tau(u)$ (induction on n and using the case of multiplication). So, $\sup\{(v+1)^n \mid n \in \mathbb{N}\} \leq \tau(u)$ and therefore, by continuity (existence of $\sup_n (v+1)^n$), $v^* \leq \tau(u)$. ■

Finally, we can prove our completeness theorem; the main result of this chapter.

Theorem 2.3.5. *If $G(u) \preceq G(v)$, then $u \leq v$ is derivable in continuous probabilistic Kleene algebra.*

Proof. If $G(u) \preceq G(v)$, then $\tau(u) \subseteq \tau(v)$ by Theorem 2.3.3. It follows from Proposition 2.3.4 that $u = \sup \tau(u) \leq \sup \tau(v) = v$. ■

Notice that G is a continuous mapping (i.e. it preserves directed limits) due to Proposition 2.3.4 and Theorem 2.3.3.

Theorem 2.3.6. *An equation $u = v$ is derivable in continuous probabilistic Kleene algebra iff $G(u) \cong G(v)$.*

Proof. Theorem 2.2.6 and Theorem 2.3.5. ■

In other words, proving an equation in continuous probabilistic Kleene algebra is exactly the same as checking for simulation equivalence in **Aut**. Since, by definition, we consider finite automata only, Theorem 2.3.6 implies that the equational theory of continuous probabilistic Kleene algebra is decidable. In the following section, we reduce such a decision using a minimisation of automata modulo simulation. That is, proving an equation in continuous probabilistic Kleene algebra is equivalent to comparing minimal automata which, as it turns out, is the same as isomorphism checking.

2.4 Minimisation and decision procedure

In Figure 2.2, it was shown that the automaton corresponding to a^* is simulation equivalent to a self loop labelled by a. In fact, the self loop is the *minimal automaton* simulation equivalent to $G(a^*)$ and such a result holds in general. Moreover, we show that minimal automata are unique up to isomorphism. Therefore, deciding whether an equation $u = v$ holds or not in continuous probabilistic Kleene algebra can be reduced to checking the existence or absence of isomorphisms between minimal automata. This section presents a minimisation algorithm modulo simulation equivalence for automata. The algorithm has three steps.

The first step deals with simulation equivalent states which are "merged" via *quotient*. This is related to the reduction of automata under bisimulation equivalence. However, there are two crucial differences: (a) two simulation equivalent states may not be bisimilar, so a quotient with respect to simulation equivalence produces an automaton with a smaller number of states (see Figure 2.6). (b) Simulation equivalence is not a congruence. That is, given an automaton G, if two states $x_1, x_2 \in G$ are simulation equivalent and that $x_1 \xrightarrow{a} x_1'$ is a transition of G, then it is possible that there is no transition labelled by a from x_2 to any state in the equivalence class of x_1'. This makes the definition of the quotient structure for simulation equivalence much more difficult.

The second step consists of removing any *irrelevant* transition. Removing such a transition is equivalent to using the equation

$$X \cdot Y + X \cdot (Y + Z) = X \cdot (Y + Z) \tag{2.17}$$

as a rewrite rule in the algebraic proof, where the sub-graph containing $X \cdot Y$ is pruned out by monotonicity. Since the left distributivity law (2.15) does not hold in probabilistic Kleene algebra, the determinisation of automata cannot be used. We can however use the subdistributivity axiom (2.11) to prove Equation (2.17) and remove the redundant branches.

The last step makes the automaton accessible which is a standard normalisation technique that has been used quite widely in this chapter.

2.4.1 Minimal automata

In this subsection, we explore in more details the simulation order and equivalence on the set of automata. Since we may remove ε-transitions using Proposition 2.2.5, we will only deal with ε-free automata so that the definition of simulation relation is simpler. Notice that a simulation between two ε-free automata is always total.

We denote $\mathcal{S}_{GH}$ the maximal simulation (with respect to the set inclusion $\subseteq$) from G to H which exists whenever $G \preceq H$. In particular, since the identity relation on G is a simulation, the maximal simulation of G, denoted by $\mathcal{S}_G$ or simply $\mathcal{S}$, always exists. Simulation equivalence of states is then given by the set $\mathcal{B} = \mathcal{S} \cap \mathcal{S}^{-1}$, that is, two states $x, y \in G$ are *simulation equivalent* iff (x, y) and (y, x) are both in $\mathcal{S}$. We write $[x]$ to represent the set of states that are simulation equivalent to x. If x and y are bisimilar states then they belong to $\mathcal{B}$. However, the relation $\mathcal{B}$ is in general not a bisimulation, as illustrated by the following example:

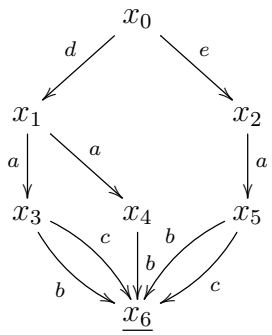


Figure 2.6: An automaton where the set of pairs of simulation equivalent states is strictly larger than the maximal bisimulation.

Example 2.4.1. Consider the automaton of Figure 2.6. We have

$$\mathcal{S} = id \cup \{(x_1, x_2), (x_2, x_1), (x_3, x_5), (x_5, x_3), (x_4, x_3), (x_4, x_5)\}$$

and

$$\mathcal{B} = id \cup \{(x_1, x_2), (x_2, x_1), (x_3, x_5), (x_5, x_3)\}$$

though the maximal bisimulation does not contain (x_1, x_2) nor (x_2, x_1). ■

Since $\mathcal{S}$ is a preorder, we say that an element $x \in A \subseteq G$ is maximal in A iff

$$\forall y \in A : (x, y) \in \mathcal{S} \Rightarrow (x, y) \in \mathcal{B}.$$

The relation $\mathcal{B}$ is an equivalence relation but not usually a congruence. Therefore, we have to define the quotient structure $(G/\mathcal{B}, \rightarrow_{G/\mathcal{B}}, [x_G], \{[x] \mid x \in F\})$ explicitly where

$$[x] \xrightarrow{a} [x'] \quad \text{iff} \quad \forall y \in [x] \exists y' \in [x'] : y \xrightarrow{a} y'.$$

Notice that the set of final states of $G/\mathcal{B}$ satisfies $[x] \in F_{G/\mathcal{B}}$ iff $[x] \subseteq F$ iff $x \in F$. Proving that this quotient structure produces an automaton simulation equivalent to G is not as straightforward as in the quotient with respect to bisimulation.

The following proposition is an adaptation of Bustan and Grumberg's on the

quotient of Kripke structures with respect to simulation equivalence to finite automata [6].

Proposition 2.4.2. *Let G be an automaton, then $G/\mathcal{B} \cong G$.*

Proof. We prove this proposition by constructing two simulations, namely, $G/\mathcal{B} \preceq G$ and $G \preceq G/\mathcal{B}$.

To prove $G/\mathcal{B} \preceq G$, we show that the inverse of the map $\pi : G \to G/\mathcal{B}$ such that $\pi(x) = [x]$ is a simulation. We have $([x_G], x_G) \in \pi^{-1}$, that is, the initial states of $G/\mathcal{B}$ and G belong to π^{-1}. Let $x, x', y \in G$ such that $[x] \xrightarrow{a} [x']$ and $([x], y) \in \pi^{-1}$ i.e. $y \in [x]$. By definition of $\to_{G/\mathcal{B}}$ and the fact that $y \in [x]$, there exists $y' \in [x']$ such that $y \xrightarrow{a} y'$. This establishes the second property in the definition of simulation because $y' \in [x']$ is the equivalent to $([x'], y') \in \pi^{-1}$. The property about final states is clear because $[x] \in F_{G/\mathcal{B}}$ iff $x \in F_G$. Hence, π^{-1} is a simulation from $G/\mathcal{B}$ to G.

To prove that $G \preceq G/\mathcal{B}$, let $\mathcal{S}$ be the maximal simulation of G and consider the relation

$$R = \{(x, [y]) \mid \exists z \in [y] : (x, z) \in \mathcal{S}\} \subseteq G \times G/\mathcal{B}.$$

It is clear that $(x_G, [x_G]) \in R$ because $(x_G, x_G) \in \mathcal{S}$. Let $(x, [y]) \in R$ and $x \xrightarrow{a} x'$ be a fixed transition of G. We need to find a state $[m] \in G/\mathcal{B}$ such that $[y] \xrightarrow{a} [m]$ is a transition of $G/\mathcal{B}$ and $(x', [m]) \in R$. We consider two cases:

1. if $x \in [y]$, then the set

$$g_a(x', [y]) = \{z' \in G \mid \exists z \in [y] : z \xrightarrow{a} z' \wedge (x', z') \in \mathcal{S}\}$$

   is not empty since it contains x'. Let m be a maximal element of $g_a(x', [y])$ (maximality is taken with respect to the simulation $\mathcal{S}$). Let us show that $[y] \xrightarrow{a} [m]$ is the transition of $G/\mathcal{B}$ corresponding to $x \xrightarrow{a} x'$ by R. We know that $m \in g_a(x', [y])$ (because it is a finite set) so there exists $z \in [y]$ such that $z \xrightarrow{a} m$ and $(x', m) \in \mathcal{S}$. Let $y' \in [y]$, then $(z, y') \in \mathcal{B}$ because $[z] = [y] = [y']$. By definition of simulation $\mathcal{S}$, since $(z, y') \in \mathcal{B} \subseteq \mathcal{S}$ and

$z \xrightarrow{a} m$, there exists m' such that $y' \xrightarrow{a} m'$ and $(m, m') \in \mathcal{S}$. By maximality of m, $(m, m') \in \mathcal{B}$ i.e. $m' \in [m]$. Therefore, by definition of the quotient, $[y] \xrightarrow{a} [m]$ is a transition of $G/\mathcal{B}$. Moreover, since $m \in g_a(x', [y])$, we deduce that $(x', m) \in \mathcal{S}$ and hence $(x', [m]) \in R$.

2. If $x \notin [y]$, there exists $z \in [y]$ such that $(x, z) \in \mathcal{S}$. By hypothesis $x \xrightarrow{a} x'$ is a transition of G, then there exists z' such that $z \xrightarrow{a} z'$ and $(x', z') \in \mathcal{S}$. Since $z \in [y]$, Case 1 ensures that there exists m such that $[y] \xrightarrow{a} [m]$ and $(z', [m]) \in R$. By definition of R, $(z', [m]) \in R$ implies the existence of $z'' \in [m]$ such that $(z', z'') \in \mathcal{S}$. Thus, $(x', z'') \in \mathcal{S}$ by transitivity. Hence $(x', [m]) \in R$ by definition of R.

The conservation of final states by R is again clear. In fact, if $(x, [y]) \in R$ and $x \in F_G$ then there exists $z \in [y]$ such that $(x, z) \in \mathcal{S}$ which implies that $z \in F_G$. Hence, $[y] = [z] \in F_{G/\mathcal{B}}$. ■

The next proposition says that there are no more simulation equivalent states in the quotient structure $G/\mathcal{B}$.

Proposition 2.4.3. *Let G be an automaton, then $\mathcal{B}_{G/\mathcal{B}}$ is the identity of $G/\mathcal{B}$.*

Proof. It suffices to show that the relation

$$R = \{(x, y) \mid \exists m : (x, z) \in \mathcal{S}_G \wedge ([z], [y]) \in \mathcal{S}_{G/\mathcal{B}}\} \subseteq G \times G$$

is a simulation on G. In fact, if R is a simulation, then it is included in the maximal simulation $\mathcal{S}_G$. Therefore, if $([x], [y]) \in \mathcal{B}_{G/\mathcal{B}}$, then $(x, y), (y, x) \in R \subseteq \mathcal{S}_G$ because $\mathcal{S}_G$ contains the identity relation of G. That is, $(x, y) \in \mathcal{B}_G$ i.e. $[x] = [y]$.

Let us then show that R satisfies the three properties of a simulation:

- It is clear that $(x_G, x_G) \in R$ because $(x_G, x_G) \in \mathcal{S}_G$ (the maximal simulation of G) and $([x_G], [x_G]) \in \mathcal{S}_{G/\mathcal{B}}$ (the maximal simulation of $G/\mathcal{B}$).

- Let $(x, y) \in R$ and $x \xrightarrow{a} x'$ be a transition of G. By definition of R, let z be a state of G such that $(x, z) \in \mathcal{S}_G$ and $([z], [y]) \in \mathcal{S}_{G/\mathcal{B}}$. From the fact

that $x \xrightarrow{a} x'$ is a transition of G and $(x, z) \in \mathcal{S}_G$, we obtain a transition $z \xrightarrow{a} z'$ of G for some $z' \in G$ such that $(x', z') \in \mathcal{S}_G$. Using the set $g_a(z', [z])$ defined in the proof of the previous proposition and the fact that $z \in [z]$, we have a transition $[z] \xrightarrow{a} [m]$ of the quotient $G/\mathcal{B}$ for some maximal state $m \in g_a(z', [z])$. Moreover, since $z' \in g_a(z', [z])$, m can be chosen such that $(z', m) \in \mathcal{S}_G$. Therefore, we have $([z], [y]) \in \mathcal{S}_{G/\mathcal{B}}$ and $[z] \xrightarrow{a} [m]$ is a transition of $G/\mathcal{B}$. By definition of the simulation $\mathcal{S}_{G/\mathcal{B}}$, there exists a state $y' \in G$ such that $[y] \xrightarrow{a} [y']$ and $([m], [y']) \in \mathcal{S}_{G/\mathcal{B}}$. By definition of the transitions of $G/\mathcal{B}$, there exists $y'' \in [y']$ such that $y \xrightarrow{a} y''$ is a transition of G. From $(z', m) \in \mathcal{S}_G$, $([m], [y']) \in \mathcal{S}_{G/\mathcal{B}}$ and $[y''] = [y']$, we deduce that $(z', y'') \in R$. Hence, by transitivity of $\mathcal{S}_G$ and the fact that $(x', z') \in \mathcal{S}_G$, we have found a state $y'' \in G$ such that $y \xrightarrow{a} y''$ is a transition of G and $(x', y'') \in R$.

- Let $(x, y) \in R$ and x be a final state of G. Then there exists a state $m \in G$ such that $(x, m) \in \mathcal{S}_G$ and $([m], [y]) \in \mathcal{S}_{G/\mathcal{B}}$. Since $\mathcal{S}_G$ is a simulation, m is also a final state of G and hence $[m]$, and consequently $[y]$, is a final state of $G/\mathcal{B}$.

Therefore, R is indeed a simulation. ■

Example 2.4.4. The quotient of the automaton of Figure 2.6 by its simulation equivalence is illustrated in Figure 2.7. Notice that the transition $x_1 \xrightarrow{a} x_4$ has been "broken" in the quotient structure because there is no transition labelled by a from $x_2 \in [x_1]$ to any state in $[x_4] = \{x_4\}$. A different sort of quotient based on an existential-existential rather than a universal-existential is described in Bustan and Grumberg's work [6]. That is, a transition $[x] \xrightarrow{a} [x']$ holds in $G/\mathcal{B}$ iff there exists a pair of states $(x_1, x_1') \in [x] \times [x']$ such that $x_1 \xrightarrow{a} x_1'$ is a transition of G. In the existential-existential version, the resulting quotient will have a transition $[x_1] \xrightarrow{a} [x_4]$. However, both types of quotient result in automata simulation equivalent to the original automaton. Hence, we use the universal-existential because it has fewer transitions. ■

We now develop the second step of our minimisation process. We define a

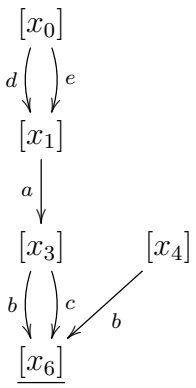


Figure 2.7: The quotient of an automaton by simulation equivalence.

---

preorder on the set of transitions using the maximal simulation $\mathcal{S}$ of G. Given two transitions of G, we define

$$x \xrightarrow{a} y \leq_G x' \xrightarrow{a'} y' \text{ iff } x = x' \wedge a = a' \wedge (y, y') \in \mathcal{S}$$

In general $\leq_G$ is a preorder but we can check easily that it is a partial order iff $\mathcal{B} = id$. A *minimal* automaton is then designed to have transitions which are pairwise incomparable.

Definition 2.4.5. *G is* minimal *if $\leq_G$ is the discrete order (i.e. $x \xrightarrow{a} y \leq_G x' \xrightarrow{a'} y'$ implies $x = x' \wedge a = a' \wedge y = y'$) and $\mathcal{B}_G = id$.*

By Proposition 2.4.2, we work with (ε-free) automata satisfying $\mathcal{B}_G = id$ only.

Definition 2.4.6. *We say that two automata $(G, \to_G, x_H, F_G)$ and $(H, \to_H, x_H, F_H)$ are* isomorphic *if there exists a bijection $f : G \to H$ such that*

1. *$f(x_G) = x_H$,*
2. *for all $a \in \Sigma$, a transition $x \xrightarrow{a}_G x'$ holds iff the transition $f(x) \xrightarrow{a}_H f(x')$ holds,*
3. *$x \in F_G$ iff $f(x) \in F_H$.*

Proposition 2.4.7. *If G, H are minimal accessible automata and $G \cong H$, then they are isomorphic.*

Proof. We will show that the relation $f = \mathcal{S}_{GH} \cap \mathcal{S}_{HG}{}^{-1} \subseteq G \times H$ is an isomorphism i.e. it is a bijective function from G to H such that both f and f^{-1} preserve the initial state, all transitions and every final states.

Let us firstly show that f is indeed a function by showing that f^{-1} is an injective relation. Let (y, x) and (y, x') be related by $f^{-1} \subseteq H \times G$ i.e. (x, y) and (x', y) are related by f. From the definition of f, $(x, y) \in \mathcal{S}_{GH}$ and $(x', y) \in \mathcal{S}_{HG}^{-1}$. Therefore, (x, x') and (x', x) are elements of the composition $\mathcal{S}_{GH}\mathcal{S}_{HG} \subseteq \mathcal{S}_G$, which implies that $(x, x') \in \mathcal{B}_G$. Thus, $x = x'$ because $\mathcal{B}_G = id_G$ and f^{-1} is injective. Symmetrically, we show that f is an injective function.

Secondly, let us show that f is a simulation, from which the totality of f follows (because we restrict to ε-free accessible automata). It is clear that $(x_G, x_F) \in f$ and since $f \subseteq \mathcal{S}_{GH}$, f preserves final states. Let $(x, y) \in f$ and $x \xrightarrow{a}_G x'$ for some $x' \in G$. We need to find a state $y' \in H$ such that $y \xrightarrow{a}_H y'$ and $(x', y') \in f$. On the one hand, since $f \subseteq \mathcal{S}_{GH}$ and $\mathcal{S}_{GH}$ is a simulation, there is some $y' \in H$ such that $y \xrightarrow{a}_H y'$ and $(x', y') \in \mathcal{S}_{GH}$. On the other hand, $(y, x) \in \mathcal{S}_{HG}$ and $y \xrightarrow{a}_H y'$ implies that $x \xrightarrow{a}_G x''$ and $(y', x'') \in \mathcal{S}_{HG}$ for some $x'' \in G$. Thus, $(x', x'') \in \mathcal{S}_G$ follows from $(x', y') \in \mathcal{S}_{GH}$ and $(y', x'') \in \mathcal{S}_{HG}$. Therefore $x \xrightarrow{a}_G x' \leq_G x \xrightarrow{a}_G x''$ and since G is reduced, $x' = x''$. Hence, we found a $y' \in H$ such that $y \xrightarrow{a}_H y'$, $(y', x') \in \mathcal{S}_{HG}$ and $(x', y') \in \mathcal{S}_{HG}$ i.e. $(x', y') \in f$.

Symmetrically, we show that f^{-1} is a simulation and then total. Therefore, f is a surjective function and we conclude that f is indeed an isomorphism. ■

Notice that the isomorphism between two minimal automata is unique, if it exists. In fact, if f, g are isomorphisms between the minimal automata G and H then $f \circ g^{-1}$ is a bisimulation on H. Therefore it is necessarily the identity function of H. Hence $f = g$.

2.4.2 Minimisation of automata modulo simulation

It now remains to transform each automaton into a minimal one. The minimisation algorithm we present here is an immediate application of the previous results and is a translation of Bustan and Grumberg's minimisation of Kripke structures to accessible nondeterministic finite automata [6].

- input: an accessible automaton G.

- output: a minimal accessible automaton G' simulation equivalent to G.

Step 1. Quotient by $\mathcal{B}$, so that $\leq_G$ becomes a partial order:
This step follows from Proposition 2.4.2.

Step 2. Reduction of $\leq_G$ to make it trivial (flat ordering):
In this step, we remove any transition $x \xrightarrow{a}_G y'$ where $x \xrightarrow{a}_G y' \leq x \xrightarrow{a}_G y$ for some state y such that $(y, y') \notin \mathcal{S}_G$.

Step 3. Elimination of unreachable non-terminating states:
This step consists of making the automaton obtained from 2 accessible as required.

Theorem 2.4.8. *Let G be an accessible automaton, a unique (up to isomorphism) minimal automaton G' can be constructed such that $G \cong G'$.*

Proof. We can assume that $\mathcal{B}_G = id$ so it suffices to show that the application of Steps 2 and 3 result in a minimal automaton simulation equivalent to G.

We reason step by step for Step 2, that is, we remove one by one each transition e_j which is strictly less than some other transition. Let G_j, for some $j \in \mathbb{N}$, be the transition system obtained by removing $e_j = x_j \xrightarrow{a_j} y_j$, we prove that $G_j \cong G$ for any j. Let $x_j \xrightarrow{a_j} z_j$ be a maximal element of $\{e \mid e_j <_G e\}$. Consider the relation $id_j \cup \{(y_j, z_j)\} \subseteq G \times G_j$ where

$$id_j = \begin{cases} id_G & \text{if } y_j \text{ is accessible in } G_j \\ id_{G \setminus \{y_j\}} & \text{otherwise} \end{cases}$$

and since $(y_j, z_j) \in \mathcal{S}_G$, it generates a simulation S_j from G to G_j. That is, $id_j \cup \{(y_j, z_j)\}$ is a subset of a simulation which yields $G \preceq G_j$.

On the other hand, the injection of G_j in G is a simulation (G_j is just a subgraph of G), so $G \cong G_j$.

Now, we notice that if $e <_G e'$ in G_j, then $e <_G e'$ in G. So we apply the above construction again on G_j. That is, we choose a transition e_k such that $e_k <_G e'$ for some transition e' of G and remove it to obtain an automaton G_{jk}. The removal of e_k from G_j is associated with a simulation $S_{jk} \subseteq G_j \times G_{jk}$. Therefore we can iteratively construct a sequence of transition systems $G_1, G_{12}, G_{123}, \ldots$. Since there are only finitely many such transitions, say r, we consider the range G' of the composition of simulations $S_1 S_{12} S_{123} \cdots S_{1\ldots r}$ which is a sub-structure of G. The transition system G' is of course an accessible automaton (by definition of simulation). It is minimal since its transitions are pairwise incomparable by $\leq'_G$, and $G \cong G'$. ■

2.4.3 Decision procedure

Deciding the existence of a simulation between two automata G and H can be achieved by computing the maximal simulation $\mathcal{S}$ of $G + H$ using techniques such as partition refinement [6, 25] and then checking that the restriction $\mathcal{S} \cap G \times H$ is indeed a simulation from G to H. Notice however that the size of the automata G and H, and thus $G+H$, may be very big. Hence minimisation enables the computation of the maximal simulation of $G' + H'$, the sum of the minimal automata computed from G and H, which are usually considerably smaller.

The steps involved in a decision procedure for the equality $u = v$ in continuous probabilistic Kleene algebra are described as follows. Firstly, we convert the terms u and v into the automata $G(u)$ and $G(v)$ using the correspondence of Section 2.2.5 or Cohen's variation of Brzozowski derivatives [10] (which is more efficient because no ε-elimination is needed). Secondly, the automata $G(u)$ and $G(v)$ are minimised to $G(u)'$ and $G(v)'$ using the construction of Section 2.4. This step will reduce the size of the automata considerably as it will factor simu-

lation equivalent states as well as remove "redundant" transitions and inaccessible states. Lastly, it suffices to check the existence or absence of an isomorphism between the minimal accessible automata $G(u)'$ and $G(v)'$. The construction of the isomorphism f in the proof of Proposition 2.4.7 can be used to achieve this step. In fact, computing the maximal simulation $\mathcal{S}_{G(v)'G(u)'}$ (resp. $\mathcal{S}_{G(u)'G(v)'}$) amounts to finding the maximal simulation $\mathcal{S}$ of $G(v)' + G(u)'$ and observing that $\mathcal{S}_{G(v)'G(u)'} = \mathcal{S} \cap [G(v)' \times G(u)']$ (resp. $\mathcal{S}_{G(u)'G(v)'} = \mathcal{S} \cap [G(u)' \times G(v)']$). Hence, it remains to check if $f = \mathcal{S}_{G(u)'G(v)'} \cap \mathcal{S}_{G(v)'G(u)'}$ is indeed an isomorphism. If it is, then the equation $u = v$ is provable in continuous probabilistic Kleene algebra. Otherwise, $u = v$ is not provable and a counter-example is given by the automata $G(u)$ and $G(v)$.

For the details about the time and space complexity of the implementation of these procedures, we refer the reader to the related works on Kripke structures [6, 25].

2.5 Discussion

The main result of this chapter is the Completeness Theorem 2.3.5 which uses the fundamental properties and consequences of the continuity of sequential composition. Notice that other axiomatisations of simulation order exist in the literature. Frendrup and Jensen [19] have given an alternative complete set of axioms using the recursive version of equational implication

$$o(X) = 0 \wedge Y \leq 1 + X \cdot Y \Rightarrow Y \leq X^* \tag{2.18}$$

where o is the $0,1$-valued endomorphism defined in Section 2.2.5. However, the use of the condition $o(X) = 0$ is equivalent to Salomaa's notion of "X does not have the empty word property". Together with the left induction (2.13), the implication (2.18) ensures that the function $f(Y) = 1 + X \cdot Y$ has a unique fixed point, namely X^*, when $o(X) = 0$. Hence, the characterisation presented in this

thesis rather uses the continuity condition to obtain a well-behaved Kleene star.

Chapter 3

Event Structures and Concurrent Kleene Algebra

This chapter provides a review of concurrent Kleene algebra (CKA) that will handle the concurrency in our proposed probabilistic concurrent extension. The soundness of concurrent Kleene algebra with respect to an event structure model and a detailed study of that model are provided from Section 3.3. In particular, we develop a new concept called *finisher*, which formalises the intuitive notion of when an event has terminated. We then show that every labelled partially ordered set representing a possible run of an event structure can be obtained from a scheduler and finisher.

3.1 Concurrent Kleene algebra

Concurrent Kleene algebra was introduced in [28] to provide an algebraic approach to the theory of concurrency. The algebra is an expansion of Gischer's work on concurrent semirings [23]. A concrete example of a concurrent semiring is the set of regular languages endowed with a *shuffle* operation [22] and the usual union $(+)$ and concatenation $(\cdot)$. The shuffle product $X \| Y$ of two regular languages X

and Y is formed of all possible interleavings of every pair of words from X and Y. It was shown by Gischer that the shuffle operation possesses many natural properties such as commutativity (3.15), associativity (3.18), distributivity with respect to $(+)$ (3.19) and the interchange law (3.20). The concurrent semiring structure was extended by Hoare et al. to account for the Kleene star operation [28].

Definition 3.1.1 ([28])**.** *A* concurrent Kleene algebra *is an algebraic structure* $(K, +, \cdot, \|, *, 0, 1)$ *such that* $(K, +, \cdot, *, 0, 1)$ *is a Kleene algebra (i.e. satisfies all axioms in Table 3.1) and where all equations from Table 3.2 hold.*

All the axioms of concurrent Kleene algebra are standard except, perhaps, for the interchange law (3.20). The essence of this axiom can be explained in terms of concretely using the regular languages with shuffle operation outlined before. In the expression $(X\|Y) \cdot (X'\|Y')$, the position of the operation $(\cdot)$ induces a sequential ordering between words from X and Y', i.e. they are only concatenated but not shuffled. However, the expression $(X \cdot X')\|(Y \cdot Y')$ allows words from X and Y' to be shuffled. Hence, the inequality says that all words from $(X\|Y) \cdot (X'\|Y')$ are present in $(X \cdot X')\|(Y \cdot Y')$. This can be seen as Gischer's subsumption property that highlights the implementation of concurrency with a partially interleaved behaviour.

A particularly important example of concurrent Kleene algebra is given by a concurrent quantale [28]. A quantale is an idempotent semiring which is a complete lattice under the natural order $X \leq Y$ iff $X + Y = Y$ and the multiplication distributes over arbitrary infima and suprema. In particular, a quantale satisfies the distributivity law (2.15) which, recall from Chapter 2, is not valid in probabilistic Kleene algebra. Hoare et al. [28] define a concurrent quantale as follows:

Definition 3.1.2. *A* concurrent quantale *is formed of a quantale* $(K, 0, 1, +, \cdot)$ *and a commutative quantale* $(K, 0, 1, +, \|)$ *linked by the interchange law (3.20).*

In concurrent quantales, the continuity of sequential composition which follows from the distributivity over arbitrary suprema ensures that the Kleene star X^* can be defined as the least fixed point of the function $f(Y) = 1 + X \cdot Y$.

$$X + X = X \tag{3.1}$$
$$X + Y = Y + X \tag{3.2}$$
$$X + (Y + Z) = (X + Y) + Z \tag{3.3}$$
$$X + 0 = X \tag{3.4}$$
$$X \cdot 1 = X \tag{3.5}$$
$$1 \cdot X = X \tag{3.6}$$
$$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z \tag{3.7}$$
$$0 \cdot X = 0 \tag{3.8}$$
$$X \cdot 0 = 0 \tag{3.9}$$
$$(X + Y) \cdot Z = X \cdot Z + Y \cdot Z \tag{3.10}$$
$$X \cdot Y + X \cdot Z = X \cdot (Y + Z) \tag{3.11}$$
$$X^* = 1 + X \cdot X^* \tag{3.12}$$
$$X \cdot Y \leq Y \Rightarrow X^* \cdot Y \leq Y \tag{3.13}$$
$$Y \cdot X \leq Y \Rightarrow Y \cdot X^* \leq Y \tag{3.14}$$

Table 3.1: Kozen's axioms for Kleene algebras

$$X \| Y = Y \| X \tag{3.15}$$
$$1 \| X = X \tag{3.16}$$
$$0 \| X = 0 \tag{3.17}$$
$$X \| (Y \| Z) = (X \| Y) \| Z \tag{3.18}$$
$$X \| Y + X \| Z = X \| (Y + Z) \tag{3.19}$$
$$(X \| Y) \cdot (X' \| Y') \leq (X \cdot X') \| (Y \cdot Y') \tag{3.20}$$

Table 3.2: Basic axioms for $\|$ making $(K, +, \|, 0, 1)$ an idempotent commutative semiring and the interchange law (3.20) which links it to the original Kleene algebra.

3.2 Bundle event structures

In this section, we construct a new mathematical model that satisfies the axioms of concurrent Kleene algebra. More precisely, the model will be composed of bundle event structures and will form a concurrent quantale.

The event structure semantics of concurrent systems were introduced in Winskel's thesis to obtain a truly concurrent abstraction of parallel execution without enforcing one particular interleaving [87]. It was later refined and extended by various authors [33, 83, 84, 85, 88], in particular by Langerak [44], to obtain the so-called *bundle event structures*, which offer a rich framework for the interpretation of concurrent programs without the limited expressiveness of prime event structures. Unlike the interleaving semantics where behaviours are specified by totally ordered sequence of actions, event structures are considered modulo partially ordered sets of events which reflects the causal dependencies between events as well as concurrent executions.

3.2.1 Basic definitions

The fundamental objects in event structures are the *events*. The following definitions extend Langerak's original operations on bundle event structure [44] with final events. The set of final events is mainly used to define the sequential composition as in the case of automata (Chapter 2 Section 2.2).

Definition 3.2.1. *A bundle event structure (BES) $\mathcal{E}$ is a tuple $(E, \#, \mapsto, \lambda, \mathbf{\Phi})$, such that E is a set of events,* the conflict relation $\# \subseteq E \times E$ *is an irreflexive and symmetric binary relation, $\mapsto \subseteq \mathcal{P}(E) \times E$ is called a bundle relation where*

$$\forall x \subseteq E \; \forall e \in E : x \mapsto e \Rightarrow x \# x, \tag{3.21}$$

where $x\#x$ holds iff for every $e, e' \in x$ such that $e \neq e'$, we have $e\#e'$. The map $\lambda : E \to \Sigma$ is a labelling (partial) function and $\mathbf{\Phi} \subseteq E$ is a set of events such that $\mathbf{\Phi}\#\mathbf{\Phi}$. Elements of $\mathbf{\Phi}$ are called final events and $\mathcal{P}(E)$ is the powerset of E.

The conflict relation # contains pairs of events that cannot occur simultaneously in a consistent behaviour of a system. That is, if $e\#e'$ holds and e has occurred, then e' has not happened nor will it ever occur at any time in the future. Conflicts then need to be resolved. The bundle relation $\mapsto$ specifies the causal dependency between events. If $x \mapsto e$ holds, then at least one event from x has occurred before e can happen. In fact, Property 3.21 ensures that exactly one event from x needs to have happened before the event e. The function λ labels events with actions that are executed with the occurrence of the event. Lastly, final events usually mark the boundary between two sequentially composed bundle event structures. That is, in the composition $\mathcal{E} \cdot \mathcal{E}'$ (Definition 3.3.2), all events of $\mathcal{E}'$ are causally dependent on all final events of $\mathcal{E}$. The set of labels or actions Σ is fixed and the collection of all bundle event structures is denoted by **BES**.

Example 3.2.2. Let us assume a register that can hold a single bit and has an initial value 0. Let us denote by $w{:}b$ and $r{:}b$ the respective action of writing and reading a bit $b \in \{0, 1\}$ on the register. In the event structure

$$(\{e_w, e_r\}, \emptyset, \{\{e_w\} \mapsto e_r\}, \{(e_w, w{:}1), (e_r, r{:}1)\}, \{e_r\}),$$

the event that occurs when reading 1 cannot happen before 1 has been written to the register. ■

Let $\mathcal{E}$ be a BES and $x \subseteq E$ be a set of events. We denote $\mathbf{cfl}(x)$ the set of events of $\mathcal{E}$ that are in conflict with some event in x

$$\mathbf{cfl}(x) = \{e \mid \exists e' \in x : e\#e'\}.$$

A set of events $x \subseteq E$ is conflict-free if $x \cap \mathbf{cfl}(x) = \emptyset$. We will use widely the set $\underline{\mathbf{cfl}}(x) = x \cup \mathbf{cfl}(x)$ instead of $\mathbf{cfl}(x)$ to obtain a simplified presentation of the results depending on the conflict relation. Given an event $e \in E$, we write $\mathbf{cfl}(e) = \mathbf{cfl}(\{e\})$. It follows immediately that $\mathbf{cfl}(x) = \cup_{e \in x} \mathbf{cfl}(e)$ (and similarly for $\underline{\mathbf{cfl}}$).

3.2.2 Trace, configuration and lposet

The semantics of bundle event structures can be expressed using three equivalent techniques, namely: event traces, configurations and labelled partially ordered sets (lposet) [44].

Definition 3.2.3 ([44]). *A (finite) sequence of events $e_1 e_2 \cdots e_n$ from E is called an* event trace *if for every $i \geq 1$ and every bundle relation $y \mapsto e_i$, there exists $j < i$ such that $e_j \in y$ and $e_i \notin \underline{\mathbf{cfl}}(\{e_1, \ldots, e_{i-1}\})$. The set of all traces of $\mathcal{E}$ is denoted by $\mathcal{T}(\mathcal{E})$.*

Given a trace $\alpha = e_1 e_2 \cdots e_n$, we write $\leq_\alpha$ the order such that $e_1 \leq_\alpha e_2 \leq_\alpha e_3 \cdots e_{n-1} \leq_\alpha e_n$. A configuration is obtained by forgetting the order of a trace.

Definition 3.2.4 ([44]). *A configuration is a subset $x \subseteq E$ such that $x = \{e_1, \ldots, e_n\}$ for some event trace $e_1 \cdots e_n$ referred to as a linearisation of x. The set of all configurations of $\mathcal{E}$ is denoted by $\mathcal{C}(\mathcal{E})$.*

Example 3.2.5. Assume again the register and the read and write operations defined in example 3.2.2. Consider the bundle event structure $\mathcal{E}$ defined by

$$(\{e_w, e'_w, e_r, e'_r, e\}, \#, \mapsto, \{(e'_w, w{:}0)(e_w, w{:}1), (e_r, r{:}1), (e'_r, r{:}1)\}, \{e, e'_w\})$$

where the bundles $\{e_w\} \mapsto e_r, \{e_w\} \mapsto e'_r, \{e_r\} \mapsto e$, $\{e'_r\} \mapsto e$ and the conflicts $e_w \# e'_w$ and $e'_w \# e$ hold. The reading of 1 associated to the events e_r, e'_r can occur concurrently (or even simultaneously if the hardware allows it). The configurations of $\mathcal{E}$ are $\emptyset$,$\{e_w\}$, $\{e_w, e_r\}$, $\{e_w, e'_r\}$, $\{e_w, e_r, e'_r\}$ and $\{e_w, e_r, e'_r, e\}$. Every trace that contains both e_r and e'_r will interleave the readings. The bundle event structure $\mathcal{E}$ specifies a (concurrent) program that writes once on the register and will read the value after writing 1. Notice that the final event e is unlabelled. ■

A *labelled partially ordered set* (lposet) is constructed from a configuration by recovering a minimal order on events. More precisely, an lposet is a tuple $(x, \leq, \lambda)$ such that $\leq$ is a partial order on x and $\lambda : x \to \Sigma$ is a labelling of

events in x. Recall, from order theory, that a partial order is the intersection of its linearisations. If x is a configuration of a bundle event structure $\mathcal{E}$, then the lposet generated by x is defined by $(x, \leq, \lambda)$, such that

$$\leq = \bigcap_{\alpha \text{ linearisation of } x} \leq_\alpha$$

and λ is the restriction of the labelling function of $\mathcal{E}$ to x. We refer to this order as the *canonical* order of x and we usually identify a configuration with the associated lposet. The set of lposets of $\mathcal{E}$ is denoted by $\mathcal{L}(\mathcal{E})$.

Unlabelled events of a lposet $u = (x, \leq, \lambda)$ can be removed to obtain the sub-lposet $\hat{u} = (\hat{x}, \leq_{\hat{x}}, \lambda_{\hat{x}})$, such that $\hat{x} = \{e \in x \mid \lambda(e) \text{ is defined}\}$ and where $\leq_{\hat{x}}$ and $\lambda_{\hat{x}}$ are the respective restrictions of $\leq$ and λ to the set $\hat{x}$. A lposet $u = (x, \leq_x, \lambda_x)$ *implements* another lposet $v = (y, \leq_y, \lambda_y)$ if there exists a label-preserving monotonic bijection $f : \hat{y} \to \hat{x}$. If u implements v then we write $u \sqsubseteq_s v$ or simply $x \sqsubseteq_s y$ if no confusion arises (s stands for subsumption [23]). Two lposets u, v are *s-equivalent* if $u \sqsubseteq_s v$ and $v \sqsubseteq_s u$. For finite lposets, which we assume unless otherwise specified, s-equivalence is the same as isomorphism. Two lposets $(x, \leq_x, \lambda_x)$ and $(y, \leq_y, \lambda_y)$ are isomorphic if there exists a label preserving bijective function $f : x \to y$ such that f and f^{-1} are monotonic.

3.3 Soundess of concurrent Kleene algebra

In this section, we provide the operations and comparison of bundle event structures based on *partially ordered multisets* (pomset) following the works of Pratt [65] and Gischer [23]. The main difference is that primary objects are composed of event structures rather than collections of pomsets.

3.3.1 Pomset language of a bundle event structure

A *pomset* is an isomorphism class of lposets, denoted by $[u]$ for some lposet u. The pomset language of a bundle event structure $\mathcal{E}$ is obtained by taking all

pomsets that implement some lposet in $\mathcal{L}(\mathcal{E})$, that is,

$$\mathcal{P}(\mathcal{E}) = \{[\hat{u}] \mid \exists v \in \mathcal{L}(\mathcal{E}) : u \sqsubseteq_s v\} \tag{3.22}$$

Notice that the language $\mathcal{P}(\mathcal{E})$ is down closed. It is then clear that the pomset-language of $\mathcal{E}$ is included in that of $\mathcal{F}$ iff for every $u \in \mathcal{L}(\mathcal{E})$ there exists $v \in \mathcal{L}(\mathcal{F})$ such that $u \sqsubseteq_s v$.

3.3.2 Operations on BES

We now provide the definition of the operations and constants of concurrent Kleene algebra on the set of bundle event structures. The constant 0 corresponds to an special object defined such that $0 + \mathcal{E} = \mathcal{E}$, $0 \| \mathcal{E} = 0$ and $0 \cdot \mathcal{E} = \mathcal{E} \cdot 0 = 0$ for every BES $\mathcal{E}$ (we convene that $\mathcal{P}(0) = \emptyset$). The constant 1 is interpreted as $(\{e\}, \emptyset, \emptyset, \emptyset, \{e\})$ (thus $\mathcal{P}(1) = \{\emptyset\}$). For each $a \in \Sigma$, a basic bundle event structure $(\{e\}, \emptyset, \emptyset, \{(e, a)\}, \{e\})$, which we usually denote again by a, is constructed.

We fix $\mathcal{E} = (E, \#_{\mathcal{E}}, \mapsto_{\mathcal{E}}, \lambda_{\mathcal{E}}, \mathbf{\Phi}_{\mathcal{E}})$ and $\mathcal{F} = (F, \#_{\mathcal{F}}, \mapsto_{\mathcal{F}}, \lambda_{\mathcal{F}}, \mathbf{\Phi}_{\mathcal{F}})$, such that their sets of events are disjoint. We define, $\mathbf{in}(\mathcal{E}) = \{e \mid \nexists x \subseteq E : x \mapsto e\}$ the set of initial events of $\mathcal{E}$. The following operations are akin to the definitions given by Langerak [44], where special care is taken for the newly introduced set of final events.

Definition 3.3.1. *The* nondeterministic *choice* $\mathcal{E} + \mathcal{F}$ *is the disjoint union*

$$(E \cup F, \#_{\mathcal{E}+\mathcal{F}}, \mapsto_{\mathcal{E}} \cup \mapsto_{\mathcal{F}}, \lambda_{\mathcal{E}} \cup \lambda_{\mathcal{F}}, \mathbf{\Phi}_{\mathcal{E}} \cup \mathbf{\Phi}_{\mathcal{F}})$$

where $\#_{\mathcal{E}+\mathcal{F}} = \#_{\mathcal{E}} \cup \#_{\mathcal{F}} \cup \mathrm{sym}(\mathbf{in}(\mathcal{E}) \times \mathbf{in}(\mathcal{F}) \cup \mathbf{\Phi}_{\mathcal{E}} \times \mathbf{\Phi}_{\mathcal{F}})$ *and* sym *is the symmetric closure.*

The construction of the nondeterministic choice between two bundle event structures outlined by the above definition is conceptually different from the corresponding construction on automata (Chapter 2 Definition 2.2.1) where ε transitions have been introduced to separate the two operands. Rather than using

such a device, Definition 3.3.1 defines the set of initial events of $\mathcal{E} + \mathcal{F}$ to be the disjoint union $\mathbf{in}(\mathcal{E}) \cup \mathbf{in}(\mathcal{F})$. The general effects of both constructions are similar, i.e., they both force a choice between the operands. However, the choice between $\mathcal{E}$ and $\mathcal{F}$ here is resolved at exactly the same time as the occurrence of an event from $\mathbf{in}(\mathcal{E} + \mathcal{F})$.

Definition 3.3.2. *The* sequential composition $\mathcal{E} \cdot \mathcal{F}$ *is defined by the BES*

$$(E \cup F, \#_{\mathcal{E}} \cup \#_{\mathcal{F}}, \mapsto_{\mathcal{E}} \cup \mapsto_{\mathcal{F}} \cup \{\mathbf{\Phi}_{\mathcal{E}} \mapsto e \mid e \in \mathbf{in}(\mathcal{F})\}, \lambda_{\mathcal{E}} \cup \lambda_{\mathcal{F}}, \mathbf{\Phi}_{\mathcal{F}}).$$

We define the asynchronous concurrency operation as follows.

Definition 3.3.3. *The* concurrent composition $\mathcal{E} \| \mathcal{F}$ *is defined by the BES*

$$(E \cup F \cup \{e, f\}, \#_{\mathcal{E}} \cup \#_{\mathcal{F}}, \mapsto_{\mathcal{E}\|\mathcal{F}}, \lambda_{\mathcal{E}} \cup \lambda_{\mathcal{F}}, \{f\})$$

where $e, f \notin E \cup F$ *and*

$$\mapsto_{\mathcal{E}\|\mathcal{F}} = \mapsto_{\mathcal{E}} \cup \mapsto_{\mathcal{F}} \cup \{\{e\} \mapsto e' \mid e' \in \mathbf{in}(\mathcal{E}) \cup \mathbf{in}(\mathcal{F})\} \cup \{\mathbf{\Phi}_{\mathcal{E}} \mapsto f, \mathbf{\Phi}_{\mathcal{F}} \mapsto f\}.$$

The fresh events e, f are unlabelled and are called *delimiters*. We assume that $(\|)$ has higher priority than $(+)$ but lower priority than $(\cdot)$ when parsing expressions involving multiple operations.

Example 3.3.4. The event structure of Example 3.2.5 is algebraically expressed as $w{:}0 + w{:}1 \cdot (r{:}1 \| r{:}1)$ up to a delimiting event (the initial event produced by $(\|)$). ■

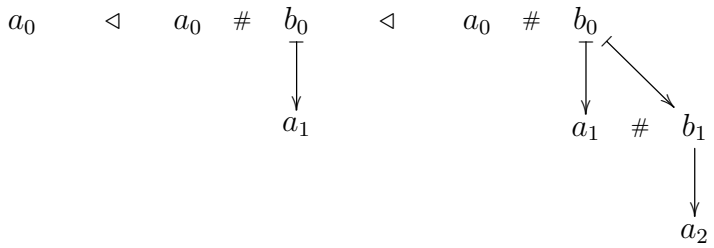
3.3.3 Substructure of a bundle event structure

To compute the Kleene star of a given bundle event structure, we define the sub-BES order $\mathcal{E} \lhd \mathcal{F}$ in Figure 3.1.

Proposition 3.3.5. $(\mathbf{BES}, \lhd)$ *is an* ω*-complete partially ordered set, that is, any countable ascending chain has a least upper bound in* **BES**.

$$E \subseteq F \tag{3.23}$$
$$\#_{\mathcal{E}} = \#_{\mathcal{F}} \cap (E \times E) \tag{3.24}$$
$$\mapsto_{\mathcal{E}} \subseteq \mapsto_{\mathcal{F}} \tag{3.25}$$
$$x \mapsto_{\mathcal{F}} e \wedge e \in E \Rightarrow x \subseteq E \wedge x \mapsto_{\mathcal{E}} e \tag{3.26}$$
$$\lambda_{\mathcal{E}} = \lambda_{\mathcal{F}}|_E \tag{3.27}$$
$$\mathbf{\Phi}_{\mathcal{E}} = \mathbf{\Phi}_{\mathcal{F}} \cap E \tag{3.28}$$

Figure 3.1: Definition of sub-BES relation $\mathcal{E} \lhd \mathcal{F}$.



An arrow $\mapsto$ denotes a bundle relation and # is the conflict relation. The events a_i are labelled by a while the b_is are labelled by b.

Figure 3.2: The first three terms in the construction of $b * a$.

Proof. The proof that $\lhd$ is a partial order amounts to checking reflexivity, antisymmetry and transitivity which is clear. As for ω-completeness, given a countable increasing sequence of BES $\mathcal{E}_0 \lhd \mathcal{E}_1 \lhd \mathcal{E}_2 \lhd \cdots$, we construct a BES $\mathcal{E} = (\cup_i E_i, \cup_i \#_i, \cup_i \mapsto_i, \cup_i \lambda_i, \cup \mathbf{\Phi}_i)$. It follows from standard set theory that $\mathcal{E}$ is indeed the least upper bound with respect to $\lhd$ of the countable sequence $(\mathcal{E}_i)_i$. ■

Definition 3.3.6. *Let $\mathcal{E}, \mathcal{F}$ be two BES. The Kleene product of $\mathcal{E}$ by $\mathcal{F}$, denoted by $\mathcal{E} * \mathcal{F}$, is the limit of the $\lhd$-increasing (countable) sequence of BES*

$$\mathcal{F} \lhd \mathcal{F} + \mathcal{E} \cdot \mathcal{F} \lhd \mathcal{F} + \mathcal{E} \cdot (\mathcal{F} + \mathcal{E} \cdot \mathcal{F}) \lhd \cdots$$

where adequate events renaming is needed to ensure that the bundle event structures in sequence are syntactically similar (see Figure 3.2 for a concrete example).

Equivalently, $\mathcal{E} * \mathcal{F}$ is the least fixed point of $\lambda X.\mathcal{F} + \mathcal{E} \cdot X$ in $(\mathbf{BES}, \lhd)$. The

Kleene star of $\mathcal{E}$ is then defined by $\mathcal{E}^* = \mathcal{E} * 1$. For convenience, we denote each component of the above sequence by $\mathcal{E} *_{\leq 0} \mathcal{F} = \mathcal{F}$, $\mathcal{E} *_{\leq 1} \mathcal{F} = \mathcal{F} + \mathcal{E} \cdot \mathcal{F}$, $\mathcal{E} *_{\leq 2} \mathcal{F} = \mathcal{F} + \mathcal{E} \cdot (\mathcal{F} + \mathcal{E} \cdot \mathcal{F})$,.... The following proposition ensures that these operations are well defined.

3.3.4 A soundness result

This section shows that the set of bundle event structure endowed with the previous operations is indeed a model of concurrent Kleene algebra. Firstly, we show that the operations $(+, \cdot, \|, *)$ are well defined. We only show that the set of final events $\mathbf{\Phi}$ contains mutually conflicting events as required.

Proposition 3.3.7. *Let $\mathcal{E}, \mathcal{F}$ be BES. Then for every $\circ \in \{+, \cdot, \|, *\}$ $\mathbf{\Phi}_{\mathcal{E}\circ\mathcal{F}} \# \mathbf{\Phi}_{\mathcal{E}\circ\mathcal{F}}$.*

Proof. We have $\mathbf{\Phi}_{\mathcal{E}+\mathcal{F}} = \mathbf{\Phi}_{\mathcal{E}} \cup \mathbf{\Phi}_{\mathcal{F}}$ and since $\mathbf{\Phi}_{\mathcal{E}} \times \mathbf{\Phi}_{\mathcal{F}} \subseteq \#_{\mathcal{E}+\mathcal{F}}$, it follows that $\mathbf{\Phi}_{\mathcal{E}+\mathcal{F}} \#_{\mathcal{E}+\mathcal{F}} \mathbf{\Phi}_{\mathcal{E}+\mathcal{F}}$. The result is clear for the case of $\mathcal{E} \cdot \mathcal{F}$ and $\mathcal{E} \| \mathcal{F}$ because $\mathbf{\Phi}_{\mathcal{E}.\mathcal{F}} = \mathbf{\Phi}_{\mathcal{F}}$ and $\mathbf{\Phi}_{\mathcal{E}\|\mathcal{F}} = \{f\}$ where f is the fresh final event in the construction of $\mathcal{E} \| \mathcal{F}$. For the Kleene star, we have $\mathbf{\Phi}_{\mathcal{E}*\mathcal{F}} = \cup_i \mathbf{\Phi}_{\mathcal{E}*_{\leq i}\mathcal{F}}$ (increasing union). Therefore, for every pair of events $(e, e') \in \mathbf{\Phi}_{\mathcal{E}*_{\leq i}\mathcal{F}} \times \mathbf{\Phi}_{\mathcal{E}*_{\leq j}\mathcal{F}}$, e and e' are in conflict with respect to the conflict relation of $\mathcal{E} *_{\leq \max(i,j)} \mathcal{F}$. ■

We end this section by observing that $(\mathbf{BES}, +, \cdot, \|, 0, 1)$ is a concurrent quantale where the operation $\circ \in \{\cdot, \|\}$ was defined so that $\mathcal{E} \circ 0 = 0 \circ \mathcal{E} = 0$. The following proposition essentially follows from Gischer's results [23]. In fact, Gischer proves that the axioms of concurrent Kleene algebra without the Kleene star completely axiomatise the pomset-language equivalence.

Proposition 3.3.8. *For each $\circ \in \{\cdot, \|\}$, the structure $(\mathbf{BES}, +, \circ, 0, 1)$ is a quantale under the pomset language equivalence.*

Proof. The Axioms (3.1-3.11) and (3.15-3.19) of Tables 3.1 and 3.2, outlining the semiring structures, were proven by Gischer [23] for the pomset language and his proofs can be translated to bundle event structures in a straightforward manner. Similarly for the interchange law (3.20).

Existence of arbitrary suprema (resp. infinima) is obtained by constructing a bundle event structure $\cup_i \mathcal{E}_i$ whose pomset language is the union (resp. intersection) of the languages of the family $\mathcal{E}_i$. A straightforward way is to create disjoint lposets for each distinct maximal pomsets. The bundle relation is then obtained from the order of the lposet and two events belonging to two different lposets are in conflict. To obtain a proper set of final events, we can append a fresh maximal unlabelled event for each of the lposets constructed. These fresh events are in conflict with each other.

The proof of the distribution of $(\cdot)$ and $(\|)$ through $(+)$ follows a standard construction from set theory. Let us show it for the case of $\mathcal{F} \cdot \cup_i \mathcal{E}_i$ and $\cup_i (\mathcal{F} \cdot \mathcal{E}_i)$; the other cases are proven in similar ways. It follows from the definition of sequential composition that if $u' \in \mathcal{P}(\mathcal{F} \cdot \cup_i \mathcal{E}_i)$ (the pomset language of $\mathcal{F} \cdot \cup_i \mathcal{E}_i$), then $u' = u \cdot v$ (that is, all events in u precede every event of v) for some finite maximal lposet $u \in \mathcal{P}(\mathcal{F})$ and $v \in \mathcal{P}(\cup_i \mathcal{E}_i)$. But $\mathcal{P}(\cup_i \mathcal{E}) = \cup_i (\mathcal{P}(\mathcal{E}_i))$, so there exists i such that $v \in \mathcal{P}(\mathcal{E}_i)$. Hence, $\mathcal{P}(\mathcal{F} \cdot \cup_i \mathcal{E}_i) \subseteq \cup_i \mathcal{P}(\mathcal{F} \cdot \mathcal{E}_i)$ and the other inclusion is shown similarly. ■

Corollary 3.3.9. *The structure* $(\mathbf{BES}, +, \cdot, \|, *, 0, 1)$ *modulo the pomset-language equivalence is a concurrent Kleene algebra, where* $\mathcal{E}^* = \mathcal{E} * 1$.

The interchange law (3.20) is ensured by the subsumption property in the definition of the pomset-language (Equation 3.22).

3.4 Schedulers and finishers on bundle event structures

In this section, we provide a novel technique to express the lposet semantics of a bundle event structure. We show how every finite (subsumed) lposet of a bundle event structure (i.e. every member of the set $\downarrow\mathcal{L}$ defined below) is computed by a particular *scheduler* and *finisher*.

Given a set of lposets $\mathcal{L}$, we define $\downarrow\mathcal{L} = \{u \mid \exists v \in \mathcal{L} : u \sqsubseteq_s v\}$ as the set of all lposets subsumed by some element of $\mathcal{L}$. Down-closure is a main property for obtaining the interchange law (3.20). As a consequence, the inclusion $\downarrow\mathcal{L}(\mathcal{E}) \subseteq \downarrow\mathcal{L}(\mathcal{F})$

means that $\mathcal{E}$ implements $\mathcal{F}$ with partially interleaved behaviours. Throughout this section, $\mathcal{E}$ is a fixed bundle event structure whose set of finite lposets (resp. event traces and configurations) is denoted by $\mathcal{L}$ (resp. $\mathcal{T}$ and $\mathcal{C}$).

3.4.1 Prefix of an lposet

Similar to the case of traces, we can define a prefixing on lposets which is analogous to the sub-bundle event structure relation of Figure 3.1.

Definition 3.4.1. *We say that* $(x, \leq, \lambda)$ *is a* prefix *of* $(y, \leq', \lambda')$*, written* $(x, \leq, \lambda) \trianglelefteq (y, \leq', \lambda')$*, if* $x \subseteq y$ *and* $\lambda = \lambda' \cap (x \times \Sigma)$ *and*

$$e \leq' e' \wedge e' \in x \Rightarrow e \in x \wedge e \leq e'. \tag{3.29}$$

The first two conditions in Definition 3.4.1 say that a prefix u of v is a restriction of v. The third property ensures that no new causal dependencies are introduced in u when it "evolves" into v (i.e. $u \trianglelefteq v$).

Proposition 3.4.4 stated below shows that prefixing entails configuration inclusion when restricted to configurations endowed with their respective canonical orders. Consequently, the lposets (with the canonical order) associated to two configurations x, y are comparable iff x and y are comparable with respect to set inclusion. Let us first prove two technical lemmas.

Given a trace $\alpha \in \mathcal{T}$, we write $\overline{\alpha}$ for the set of events occurring in α.

Lemma 3.4.2. *Let* $\alpha \in \mathcal{T}$ *and* $x \in \mathcal{C}$ *such that* $x \subseteq \bar{\alpha}$*, then the restriction* $\alpha|_x$ *of* α *to events in* x *is an event trace.*

Proof. Let $\alpha = e_1 e_2 \cdots e_n$, $x \in \mathcal{C}$ and write $\alpha|_x = e_{i_1} e_{i_2} \cdots e_{i_m}$. Let us show that $\alpha|_x$ is an event trace of $\mathcal{E}$. Let $e_{i_k} \in x$ and $z \mapsto e_{i_k}$ be a bundle of $\mathcal{E}$. Since α is an event trace, there exists an event e_j such that $e_j \in z$ and $j < i_k$. Since x is a configuration and $e_{i_k} \in x$, there exists $e_{i_l} \in z$ and $l < k$. By definition, the bundle set z contains mutually conflicting events only and since $\overline{\alpha}$ is conflict free, $e_{i_l} = e_j$. That is, $z \cap \{e_{i_1}, \ldots, e_{i_{k-1}}\} \neq \emptyset$ for every bundle $z \mapsto e_{i_k}$. Hence, $\alpha|_x$ is

an event trace because it is conflict free. ■

Lemma 3.4.3. *Let $x, y \in \mathcal{C}$ such that $x \subseteq y$. For every event trace α such that $\overline{\alpha} = x$, there exists an event trace α' satisfying $\overline{\alpha'} = y$ and $\alpha'|_x = \alpha$.*

Proof. Let α, β be any event traces such that $\overline{\alpha} = x$, $\overline{\beta} = y$ and $x \subseteq y$. Let β' be the concatenation of two sequences $\beta_1\beta_2$, where events in β_1 are exactly those of x ordered with $\leq_\beta$ and β_2 is composed of events from $y \setminus x$ ordered again with $\leq_\beta$. We now show that the concatenation $\alpha' = \alpha\beta_2$ is an event trace. That α' is conflict-free comes from the configuration y. To show the second property of an event trace, we need to show that every bundle pointing to an event e_2 in β_2 has to intersect $\overline{\alpha} \cup \overline{\beta}_2$ at an event occurring before e_2 with respect to the order $\leq_\beta$. That is clear because β is an event trace (notice that if $z \mapsto e_2$ holds, it is possible that the sole event in $z \cap \overline{\beta}$ belongs to $\overline{\alpha}$). Moreover, it is enough to show the property for events in β_2 only because α is already an event trace. Hence α' is an event trace and $\alpha'|_x = \alpha$. ■

With the aid of these two lemmas, we now prove the aimed characterisation of prefixing with configuration inclusion.

Proposition 3.4.4. *If $x, y \in \mathcal{C}$ and $x \subseteq y$, then $(x, \leq_x, \lambda_x) \trianglelefteq (y, \leq_y, \lambda_y)$ where $\leq_x$ and $\leq_y$ are the respective canonical orders of x and y.*

Proof. Let $x \subseteq y$. Let us first show that $\leq_x = \leq_y \cap (x \times x)$. Let $e, e' \in x$ such that $e \leq_x e'$. Lemma 3.4.2 implies that $e \leq_y e'$ because every event trace for y restricts to an event trace for x. For the converse inclusion, let $e, e' \in x$ such that $e \leq_y e'$. Lemma 3.4.3 implies that every event trace for x can be obtained as a restriction of some event trace for y. Hence, $e \leq_x e'$. Therefore $\leq_x = \leq_y \cap (x \times x)$.

It remains to show that Property (3.29) holds. Let $e, e' \in y$, $e \leq_y e'$ and $e' \in x$. It now suffices to show that $e \in x$ because once that is established, we use $\leq_x = \leq_y \cap (x \times x)$ to deduce that $e \leq_x e'$. For a contradiction, assume that $e \notin x$. Then there exists an event trace $\beta' = \beta_1\beta_2$ as specified in the proof of Lemma 3.4.3, that is, $\overline{\beta'} = y$, $\overline{\beta_1} = x$ and $e \in \overline{\beta_2}$. Therefore, $e \not\leq_{\beta'} e'$, which contradicts the fact that $e \leq_y e'$. ■

3.4.2 Scheduling and finishing events in BES

The notion of scheduler is a standard technique in theoretical studies as well as the practical implementation of software. It is particularly useful when generating averaged, worst-case, best-case or other specific behaviours.

In this thesis, a scheduler is used to generate a particular configuration from a given bundle event structure. In contrast to the interleaving concept, where generated behaviours are totally ordered execution traces, we assume that when an event is scheduled, then the action associated with it is ready to run (or has partially run) but has not necessarily terminated. Therefore, we introduce the dual notion of *finisher* to account for terminated events.

Definition 3.4.5. *A* scheduler *on the BES $\mathcal{E}$ is a map $\sigma : \downarrow\mathcal{L} \to \mathbb{P}(E)$ such that for every $u = (x, \leq, \lambda) \in \downarrow\mathcal{L}$, we have:*

- $\sigma(u) \cap x = \emptyset$, *and*

- $\sigma[u] = (x \cup \sigma(u), \leq \cup \leq_{x\cup\sigma(u)}, \lambda_{x\cup\sigma(u)})$ *is in* $\downarrow\mathcal{L}$

where $\leq_{x\cup\sigma(u)}$ is the canonical order associated to the configuration $x \cup \sigma(u) \in \mathcal{C}$ and $\lambda_{x\cup\sigma(u)}$ is the restriction of $\lambda_{\mathcal{E}}$ to $x \cup \sigma(u)$.

Iterating from the empty lposet, a scheduler σ generates a (countable) sequence of (finite) lposets $\emptyset \trianglelefteq \sigma[\emptyset] \trianglelefteq \sigma[\sigma[\emptyset]] \trianglelefteq \cdots$.

A finisher is a way to determine when events have terminated. A finisher can only say that an event has finished if it started sometime in the past.

Definition 3.4.6. *A finisher is a map $\varphi : \downarrow\mathcal{L} \to \downarrow\mathcal{L}$ such that for every $u, v \in \downarrow\mathcal{L}$, we have:*

- $\varphi(u) \trianglelefteq u$, *and*

- φ *is* $\trianglelefteq$*-monotonic.*

Intuitively, an lposet u can be thought of as a set of scheduled events ordered by causal dependencies. The lposet $\varphi(u)$ then captures the set of events in u that have terminated (when the last event in u was scheduled). Therefore, all

events scheduled after this "point of time" will causally depend on the events in $\varphi(u)$. Monotonicity ensures that, as we schedule new events, we cannot unfinish terminated events but can only finish the scheduled ones.

Observe that a special case of finisher is the identity function on $\downarrow\mathcal{L}$. Intuitively, this finisher ensures that every scheduled event will be finished "instantaneously". When the identity is used with a scheduler that schedules one event at a time, the resulting model reduces to an interleaving model of concurrency.

Example 3.4.7. Recall the bundle event structure

$$\mathcal{E} = (\{e_w, e'_w, e_r, e'_r, e\}, \#, \mapsto, \{(e'_w, w{:}0)(e_w, w{:}1), (e_r, r{:}1), (e'_r, r{:}1)\}, \{e, e'_w\})$$

outlined in the previous examples. We define a scheduler σ on $\mathcal{E}$ such that $\sigma(\emptyset) = \{e_w\}$, $\sigma(\{e_w\}) = \{e_r\}$, $\sigma(\{e_w, e_r\}) = \{e'_r\}$ and $\sigma(\{e_w, e_r, e'_r\}) = \emptyset$ where each set in the argument of σ should be read as the lposet composed of the configuration and its canonical order. This scheduler schedules e_r before e'_r but that does not mean that e_r will happen before e'_r because that order is not enforced by the bundle relation of $\mathcal{E}$. A scheduled event should be thought of as "ready to happen anytime from now".

An example of a finisher on $\mathcal{E}$ is given by the map φ such that $\varphi(\emptyset) = \emptyset$, $\varphi(\{e_w\}) = \{e_w\}$,$\varphi(\{e'_w\}) = \{e'_w\}$, $\varphi(\{e_w, e_r\}) = \{e_w\}$, $\varphi(\{e_w, e'_r\}) = \{e_w\}$, $\varphi(\{e_w, e_r, e'_r\}) = \{e_w, e_r, e'_r\}$ and $\varphi(\{e_w, e_r, e'_r, e\}) = \{e_w, e_r, e'_r, e\}$. Again, the sets in the argument of φ and on the right hand side of the equality should be read as the lposet composed of the configurations and their respective canonical orders. This finisher specifies that e_r and e'_r will only happen once both have been scheduled. ■

3.4.3 Generating lposets from schedulers and finishers

The dynamic of a bundle event structure is traced through the interaction between a fixed pair of a scheduler and a finisher. The state of the bundle event structure $\mathcal{E}$ is described by a tuple $(u, v) \in \downarrow\mathcal{L}^2$ such that $v \trianglelefteq u$ ($\downarrow\mathcal{L}^2$ stands

for $(\downarrow \mathcal{L}) \times (\downarrow \mathcal{L})$). Intuitively, u is the scheduled lposet while the carrier set of v describes all "finished" events (the order in which these events occurred is constrained by the order of v).

Example 3.4.8. By Proposition 3.4.4, the pair $(\{e_w, e'_r, e_r\}, \{e_w\})$ is a state of the bundle event structure $\mathcal{E}$ of Example 3.2.5 (both components of the pair are lposets with the corresponding canonical orders). Intuitively, it says that the events e_w, e_r and e'_r have been scheduled and e_w has happened. ■

In the remainder of this chapter, if u is a lposet, then we write $\mathbf{set}(u)$ for its carrier. Let σ be a scheduler on $\mathcal{E}$, we define $\overline{\sigma} : \downarrow \mathcal{L}^2 \rightarrow \downarrow \mathcal{L}^2$ such that $\overline{\sigma}(u, v) = (u', v)$ if

$$\begin{array}{rcl} \mathbf{set}(u') & = & \mathbf{set}(u) \cup \sigma(u) \\ \leq_{u'} & = & \leq_u \ \cup \leq_{\mathbf{set}(u')} \cup \ (\mathbf{set}(v) \times \sigma(u)) \\ \lambda_{u'} & = & \lambda_{\mathbf{set}(u')} \end{array}$$

where $u = (\mathbf{set}(u), \leq_u, \lambda_u), u' = (\mathbf{set}(u'), \leq_{u'}, \lambda_{u'})$, $\leq_{\mathbf{set}(u')}$ is the canonical order of the configuration $\mathbf{set}(u')$ and $\lambda_{\mathbf{set}(u')}$ is the restriction of the labelling function of $\mathcal{E}$ to $\mathbf{set}(u')$.

In other words, $e \leq_{u'} e'$ holds in the new lposet u' if:

- either e and e' have been scheduled (i.e. they belong to $\mathbf{set}(u)$) and $e \leq_u e'$,
- or e' is a newly scheduled event and the order $e \leq_{u'} e'$ is enforced by the transitive closure of the bundle relation of $\mathcal{E}$ (i.e. $e \leq_{\mathbf{set}(u')} e'$),
- or e has already happened (i.e. $e \in \mathbf{set}(v)$) and e' is newly scheduled after u.

In the sequel, we denote the lposet u' in this construction by $\sigma[u \leftarrow v]$. Notice that if $v \trianglelefteq u$ then $u \trianglelefteq \sigma[u \leftarrow v]$.

Similarly, every finisher φ generates a map $\overline{\varphi} : \downarrow \mathcal{L}^2 \rightarrow \downarrow \mathcal{L}^2$ such that $\overline{\varphi}(u, v) = (u, \varphi(u))$. It is clear from these definitions that if (u, v) is a state of $\mathcal{E}$, i.e. $v \trianglelefteq u$, then $\overline{\sigma}(u, v)$ and $\overline{\varphi}(u, v)$ are also states of $\mathcal{E}$. The following propositions and lemma provide a "sanity check" for the scheduler-finisher definition. The prefix relation $\trianglelefteq$ is applied to tuples in a component-wise manner.

Lemma 3.4.9. *For every $m, n \in \mathbb{N}$ and $(u, v) \in{\downarrow}\mathcal{L}^2$, if $n \geq 1$ then $\overline{\sigma}^m \circ \overline{\varphi}^n(u, v) = \overline{\sigma}^m \circ \overline{\varphi}(u, v)$*

Proof. $\overline{\varphi}$ is idempotent. ■

Hence, computing the repeated application of $\overline{\sigma}$ and $\overline{\varphi}$ in the expression $(u, v) \trianglelefteq (\overline{\sigma}^m \circ \overline{\varphi}^n)^k(u, v)$ is the same as computing $(u, v) \trianglelefteq (\overline{\sigma}^m \circ \overline{\varphi})^k(u, v)$ when $n \geq 1$.

Proposition 3.4.10. *Given σ and φ, if $v \trianglelefteq \varphi(u)$ then*

$$(u, v) \trianglelefteq (\overline{\sigma}^m \circ \overline{\varphi})^n(u, v)$$

for every $m, n \in \mathbb{N}$ and $\trianglelefteq$ is defined component-wise.

Proof. It suffices to show that $(u, v) \trianglelefteq \overline{\sigma}(u, v)$, $(u, v) \trianglelefteq \overline{\varphi}(u, v)$, $(u, v) \trianglelefteq (\overline{\sigma} \circ \overline{\varphi})(u, v)$ and $(u, v) \trianglelefteq (\overline{\varphi} \circ \overline{\sigma})(u, v)$. The result with arbitrary m and n then follows by a double induction and the transitivity of $\trianglelefteq$.

The first two cases are clear from the definition of $\overline{\sigma}$ and the hypothesis $v \trianglelefteq \varphi(u)$. For the third case, we have

$$(u, v) \trianglelefteq (\sigma[u{\leftarrow}\varphi(u)], \varphi(u)) = \overline{\sigma}(u, \varphi(u)) = \overline{\sigma} \circ \overline{\varphi}(u, v)$$

and similarly for the last case. ■

Proposition 3.4.11. *For every $(u, v) \in{\downarrow}\mathcal{L}^2$ such that $v \trianglelefteq u$, if $v \trianglelefteq \varphi(u)$ then $v^n_m \trianglelefteq \varphi(u^n_m)$ where $(u^n_m, v^n_m) = (\overline{\sigma}^m \circ \overline{\varphi})^n(u, v)$ and $m, n \in \mathbb{N}$.*

Proof. The case $m = n = 0$ is clear because $u^0_0 = u$, $v^0_0 = v$ and $v \trianglelefteq \varphi(u)$.

Now, assume that $v^l_k \trianglelefteq \varphi(u^l_k)$ holds for every $k \leq m$ and and $l \leq n$. We need to show that $v^{n+1}_m \trianglelefteq \varphi(u^{n+1}_m)$ and $v^n_{m+1} \trianglelefteq \varphi(u^n_{m+1})$. For the first case, we have

$$(\overline{\sigma}^m \circ \overline{\varphi})^{n+1}(u, v) = (\overline{\sigma} \circ \overline{\varphi})(u^n_m, v^n_m) = (\sigma[u^n_m{\leftarrow}v^n_m], \varphi(u^n_m)).$$

Since $u^n_m \trianglelefteq \sigma[u^n_m{\leftarrow}v^n_m]$ (because $v^n_m \trianglelefteq u^n_m$ follows from the induction hypothesis

$v_m^n \trianglelefteq \varphi(u_m^n))$ and φ is monotonic, we obtain

$$v_m^{n+1} = \varphi(u_m^n) \trianglelefteq \varphi(\sigma[u_m^n \leftarrow v_m^n]) = \varphi(u_m^{n+1}).$$

For the second case, we have

$$(\overline{\sigma}^{m+1} \circ \overline{\varphi})^n(u, v) = \sigma^{m+1} \circ \overline{\varphi}(u_{m+1}^{n-1}, v_{m+1}^{n-1}) = \overline{\sigma}^{m+1}(u_{m+1}^{n-1}, \varphi(u_{m+1}^{n-1})).$$

Notice that $\overline{\sigma}^{m+1}$ acts only on the first component u_{m+1}^{n-1} and since $\varphi(u_{m-1}^{n-1}) \trianglelefteq u_{m-1}^{n-1}$, it follows that

$$u_{m-1}^{n-1} \trianglelefteq \sigma[u_{m-1}^{n-1} \leftarrow \varphi(u_{m-1}^{n-1})] \trianglelefteq w$$

where $w = \sigma[\sigma[\ldots\sigma[u_{m-1}^{n-1} \leftarrow \varphi(u_{m-1}^{n-1})] \leftarrow \varphi(u_{m-1}^{n-1})] \leftarrow \varphi(u_{m-1}^{n-1})]$. The second inequality can be shown easily by induction. Hence, we deduce

$$v_{m+1}^n = \varphi(u_{m-1}^{n-1}) \trianglelefteq \varphi(w) = \varphi(u_{m+1}^n)$$

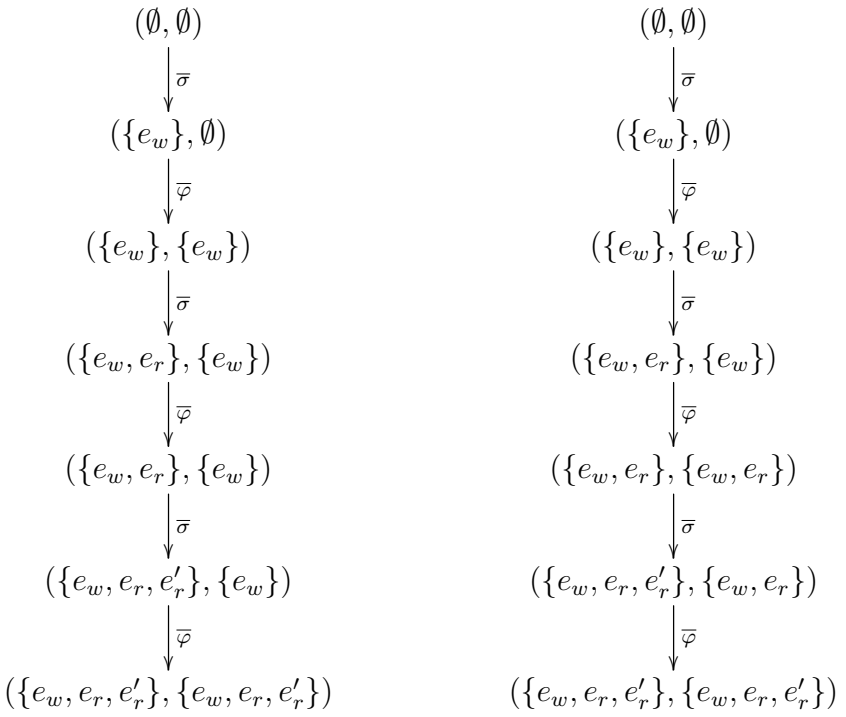
from the monotonicity of φ. ■

The above proposition says that the property $v \trianglelefteq \varphi(u)$ is an invariant for every state (u, v) generated from schedulers and finishers. In particular, if v was finished when u was scheduled then v remains finished after any subsequent scheduling and finishing applied to $\mathcal{E}$ from u.

We now define the resolution of $\mathcal{E}$ with respect to $\sigma \circ \varphi$.

Definition 3.4.12. *Given a finisher φ and scheduler σ, the resolution of $\mathcal{E}$ with $\sigma \circ \varphi$ is the directed graph whose nodes are states of $\mathcal{E}$ and whose edges are specified by the set $\{((u, v), \sigma \circ \varphi(u, v)) \mid (u, v)$ is a state of $\mathcal{E}\}$. The subgraph composed of states that are reachable with a finite path from $(\emptyset, \emptyset)$ is denoted $\sigma \circ \varphi(\mathcal{E})$.*

Example 3.4.13. Reconsider again the bundle event structure defined in Example 3.2.5. The scheduler σ and finisher φ of Example 3.4.7 generate the resolution illustrated in Figure 3.3 (a). The limit lposet will order the events as $e_w \leq e_r$ and



(a) (b)

For simplification, we only write the carrier set of the lposets involved in the states. The ordering between events can be recovered from the bundle relation of $\mathcal{E}$ and the interaction between the scheduler and the finisher.

Figure 3.3: Two examples of resolution $\sigma \circ \varphi(\mathcal{E})$ and $\sigma \circ \mathrm{id}(\mathcal{E})$.

$e_w \leq e'_r$, that is, the reading can happen concurrently. In contrast to that concurrent reading, Figure 3.3 (b) shows the resolution of $\mathcal{E}$ with the same scheduler and the identity function as a finisher. It is clear from the diagram that when e'_r is scheduled, the events e_w and e_r are already on the right hand side of the state. Hence, e'_r is forced to occur only after e_r has happened. The limit lposet will have the order $e_w \leq e_r \leq e'_r$. ■

A pair of scheduler/finisher $\sigma \circ \varphi$ generates an increasing sequence of states $(\emptyset, \emptyset) \trianglelefteq \sigma \circ \varphi(\emptyset, \emptyset) \trianglelefteq \cdots$. If we denote by $(u_0, v_0) \trianglelefteq (u_1, v_1) \trianglelefteq \cdots$ the states involved in that sequence, then we write $\lim(\sigma \circ \varphi(\mathcal{E})) = (\cup_i u_i, \cup_i v_i)$ where the

union of two lposets is obtained by taking the union of the carrier sets, the union of the orders relations and the union of the labelling functions. Each labelling function λ_i of u_i is usually the restriction of the labelling function of a given event structure. Hence the union of the λ_i will again be a function. Moreover, since the sequence is increasing, the union of the order relations will again be a partial order on the union of the carrier sets.

Every pair of lposets that occurs in a resolution has a particular property which is established by the following proposition:

Proposition 3.4.14. *Let σ be a scheduler, φ a finisher on $\mathcal{E}$ and (u, v) a node of $\sigma \circ \varphi(\mathcal{E})$. The implication*

$$\forall e, e' \in \mathbf{set}(u) : e \leq_u e' \Rightarrow e \leq_{\mathbf{set}(u)} e' \ \vee \ e \in \mathbf{set}(v) \tag{3.30}$$

holds where $\leq_{\mathbf{set}(u)}$ is the canonical order of the carrier set $\mathbf{set}(u)$ of u.

Proof. By induction on the reachability of (u, v). ■

Proposition 3.4.14 states that every state in $\sigma \circ \varphi(\mathcal{E})$ satisfies Property (3.30). In fact, that condition is necessary and sufficient for a pair (u, v) satisfying $v \trianglelefteq u$ to be a node in some resolution as the following theorem shows. In particular, for every $u \in \downarrow\mathcal{L}$, there exists a scheduler σ and a finisher φ such that $(u, u) = \lim \sigma \circ \varphi(\mathcal{E})$.

Theorem 3.4.15. *Let $(u, v) \downarrow\mathcal{L}^2$ such that $v \trianglelefteq u$. If u and v satisfy Property 3.30, then there exists a scheduler σ and a finisher φ such that $(u, v) = \lim(\sigma \circ \varphi(\mathcal{E}))$.*

Proof. We reason by induction on the size of $\mathbf{set}(u)$:

- the empty pair $(\emptyset, \emptyset)$ is obtained from the empty scheduler $u \mapsto \emptyset$ and the finisher $\mathrm{id}_{\downarrow\mathcal{L}}$.
- Let u be a finite lposet and $v \trianglelefteq u$. Let e be a maximal event in the lposet $u = (x, \leq, \lambda)$, therefore lposet u' obtained by removing e from $\mathbf{set}(u)$ belongs to $\downarrow\mathcal{L}$. We denote by $\downarrow e = (\{e' \mid e' < e\}, \leq_e, \lambda_e)$ where $\leq_e$ is the restriction

of $\leq$ on the downclosed set of events and similarly for λ_e. The lposet $v' = v \cap \downarrow e$ (component-wise intersection) is again a lposet in $\downarrow\mathcal{L}$ and $v' \trianglelefteq u'$. By induction hypothesis, there exists a scheduler σ' and a finisher φ' such that $(u', v') = \lim(\sigma' \circ \varphi'(\mathcal{E}))$ and $\sigma'(u') = \emptyset$. We construct a scheduler σ such that $\sigma(u') = \{e\}$, $\sigma(u) = \emptyset$ and it coincides with σ' otherwise. As for the finisher, we have $\varphi(u) = v$ and it coincides with φ' otherwise. Since the u' and v' are finite lposets, $\sigma' \circ \varphi'$ will generate a finite resolution that can be extended to cover (u, v). In fact, we have

- $\mathbf{set}(\sigma[u' \leftarrow v']) = \mathbf{set}(u') \cup \{e\} = \mathbf{set}(u)$,
- $\leq_{\sigma[u' \leftarrow v']} = \leq_{u'} \cup \leq_{\mathbf{set}(u)} \cup\ \mathbf{set}(v') \times \{e\}$ which coincides with the order of u because of prefixing and if $e' \leq_u e$ then $e' \leq_u e$ or $e' \in v'$ (Property (3.30)).
- $\lambda_{\sigma[u', v']} = \lambda_u = \lambda_{\mathbf{set}(u)}$.

Since $\varphi(u) = v$, we deduce that $(u, v) = \lim(\sigma \circ \varphi(\mathcal{E}))$. ■

3.4.4 Full resolution of a bundle event structure

In the previous subsection, the interaction between a scheduler and a finisher was sequential in the sense that scheduling always happens before finishing. In general, these two processes can happen in any order (or even concurrently) and the most important feature is that scheduled events causally depend on finished events.

The goal of this subsection is to prove that the sequential resolution can be used to generate all possible interaction between a scheduler and a finisher.

Definition 3.4.16. *Given a finisher φ and a scheduler σ, the* full resolution *of $\mathcal{E}$ with σ and φ is the directed graph*

$$(\downarrow\mathcal{L}^2, \{((u, v), \overline{\varphi}(u, v)), ((u, v), \overline{\sigma}(u, v)) \mid (u, v) \in \downarrow\mathcal{L}^2\}).$$

The subgraph composed of nodes that are reachable with a finite path from $(\emptyset, \emptyset)$ *is denoted by* $\sigma \| \varphi(\mathcal{E})$ *where self-loops (generated by* $\overline{\varphi}$*) are removed.*

We start by showing that $\sigma \| \varphi(\mathcal{E})$ is a directed acyclic graph.

Proposition 3.4.17. $\sigma \| \varphi(\mathcal{E})$ *is acyclic.*

Proof. Assume that $\sigma \| \varphi(\mathcal{E})$ has a cycle that is not a self-loop. Since $\overline{\varphi}$ is idempotent, that cycle needs to contain at least one application of $\overline{\sigma}$. Moreover, if there is such a cycle, then it contains a state (u, v) such that u is exactly the same as the first component of the state obtained after a finite application of σ and φ. But σ strictly increases the left lposet of an arbitrary pair, which makes it impossible to find such a state (u, v). ■

To prove the main result of this section, we need the following lemma.

Lemma 3.4.18. *Let* $\mathcal{E}$ *be a BES, if* $f : \downarrow\mathcal{L} \to \downarrow\mathcal{L}$ *is a partial function defined on a increasing sequence of lposet* $\emptyset = u_0 \trianglelefteq u_1 \trianglelefteq \cdots$ *and satisfies the properties of a finisher then there exists a finisher* φ *(i.e. totally defined) such that* $\varphi(u_i) = f(u_i)$ *for every* i*.*

Proof. Let $\mathcal{E}$ be a BES and f be a function satisfying the hypothesis of the lemma. We construct φ as follows

$$\varphi(u) = \begin{cases} f(u_i) & \text{if there is a maximal } i \text{ such that } u_i \trianglelefteq u \\ u & \text{if } u_i \trianglelefteq u \text{ for every } i \\ \emptyset & \text{otherwise} \end{cases}$$

Firstly, we show the prefixing property of finishers. Let $u \in \downarrow\mathcal{L}$:

- If there exists a maximal i such that $u_i \trianglelefteq u$ then $\varphi(u) = f(u_i) \trianglelefteq u_i \trianglelefteq u$.
- If $u_i \trianglelefteq u$ for all i, then $\varphi(u) = u \trianglelefteq u$.
- Otherwise, $\varphi(u) = \emptyset \trianglelefteq u$.

Secondly, we show that φ is monotonic. Let $u \trianglelefteq v$.

- If there exists a maximal i such that $u_i \trianglelefteq u$, then $\varphi(u) = f(u_i)$. There are three cases based on the value of $\varphi(v)$.
  - There exists a maximal j such that $u_j \trianglelefteq v$ and $\varphi(v) = f(u_j)$. Since $u \trianglelefteq v$, maximality of j implies that $u_i \trianglelefteq u_j$ and hence $\varphi(u) = f(u_i) \trianglelefteq f(u_j) = \varphi(v)$ by monotonicity of f.
  - For all j, $u_j \trianglelefteq v$ and therefore $\varphi(u) = f(u_i) \trianglelefteq u_i \trianglelefteq v = \varphi(v)$.
  - The empty case is impossible because $u_i \trianglelefteq v$.
- If $u_i \trianglelefteq u$ for all i, then $u_i \trianglelefteq v$ for all i because $u \trianglelefteq v$. Hence $\varphi(u) = u \trianglelefteq v = \varphi(v)$.
- Otherwise, $\varphi(u) = \emptyset \trianglelefteq \varphi(v)$, whatever $\varphi(v)$ is. ■

Theorem 3.4.19. *For every scheduler σ and finisher φ, there exists a (countable) family of finishers $\varphi_0, \varphi_1, \ldots$ such that the full resolution $\sigma \| \varphi(\mathcal{E})$ is the union of the family of resolutions $\sigma \circ \varphi_0(\mathcal{E}), \sigma \circ \varphi_1(\mathcal{E}), \ldots$.*

Proof. Let σ and φ be some scheduler and finisher on a BES $\mathcal{E}$. The full resolution of $\sigma \| \varphi(\mathcal{E})$ is depicted in Figure 3.4.

Given a path π in the dag of Figure 3.4, we generate a partial function f_π such that $f(u) = v$ iff $(u, v) \in \pi$. Therefore, f_π satisfies the first property of a finisher because each node of the tree is a state of $\mathcal{E}$ and it is monotonic because if $(u_i, v_i), (u_j, v_j) \in \pi$ such that $u_i \trianglelefteq u_j$, then there exist two indices k_i, k_j such that $f(u_i) = \varphi(u_{k_i})$ and $f(u_j) = \varphi(u_{k_j})$ and $u_{k_i} \trianglelefteq u_{k_j}$. Hence $f(u_i) \trianglelefteq f(u_j)$ and it extends to a finisher φ_π by the previous lemma. Since the dag can be recovered from the union of all paths, we deduce that

$$\sigma \| \varphi(\mathcal{E}) = \cup_\pi \sigma \circ \varphi_\pi(\mathcal{E})$$

where π ranges over all paths in $\sigma \| \varphi(\mathcal{E})$ (which is of course countable). ■

Example 3.4.20. The full resolution of our running example using the scheduler and finisher of Example 3.4.7 is depicted in Figure 3.5. ■

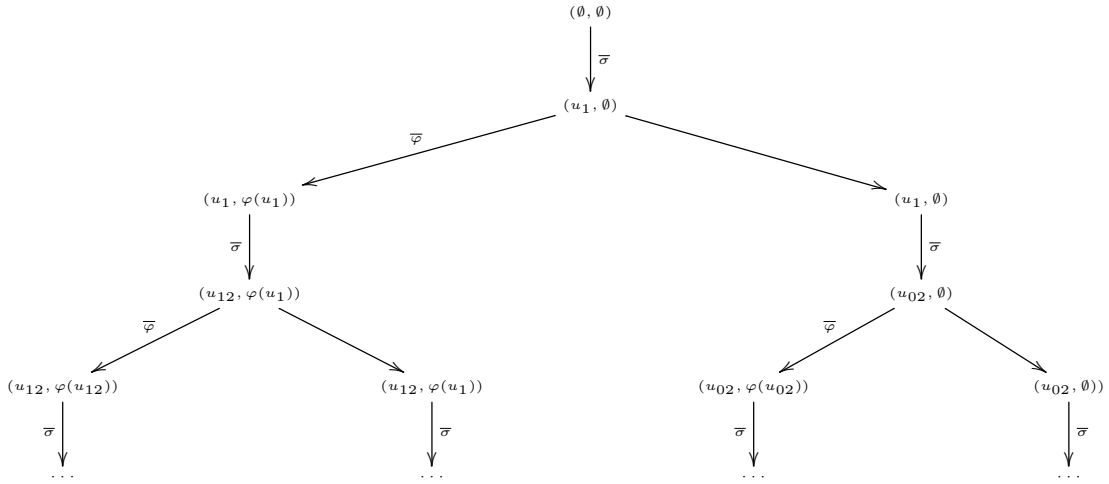


Figure 3.4: Full resolution where unlabelled arrows are added for unchanging states.

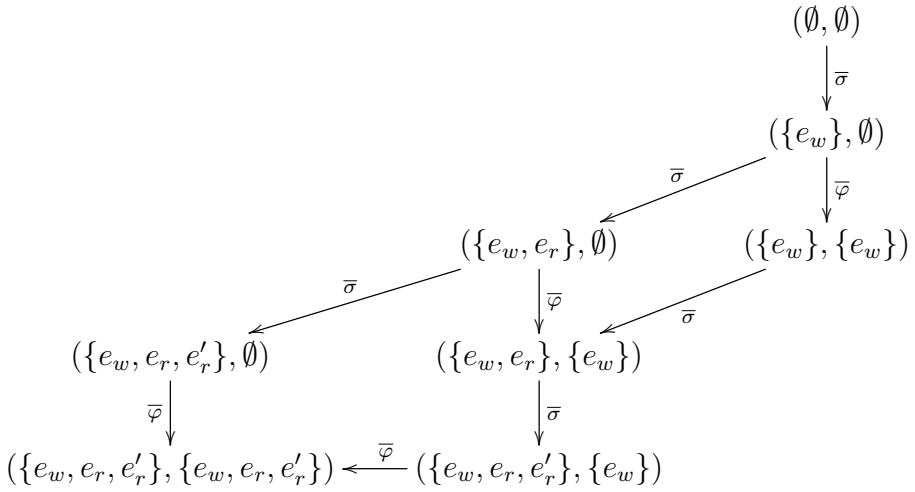


Figure 3.5: The full resolution of bundle event structure.

3.5 Discussion

The main contribution of this chapter is twofold. Firstly, it provides an alternative concrete model for Hoare et al.'s axiomatisation of concurrent Kleene algebra. It should be noted that only a soundness result was given and the completeness of such an axiomatisation is still an open question. The most comprehensive work on this problem dates back to Gischer in his paper on the axiomatic characterisation of the subsumption theory of pomsets [23]. Note however that his result only holds for the bi-semiring fragment of concurrent Kleene algebra i.e., without the Kleene star. Indeed, continuity (in the style of Chapter 2) may be of tremendous use for the general case but a complete axiomatisation in the style of Kozen, as in standard Kleene algebra, is much more desirable.

Secondly, the notion of finisher developed in Section 3.4 is a novel idea in its own right. Together with our definition of scheduler, it provides a complete characterisation of the lposet semantics for bundle event structures. That is, it provides an alternative view of the same computational model. Moreover, we have shown in Theorem 3.4.19 that a limited form of interaction between schedulers and finishers is enough. That is, all states involved in a full resolution $\sigma \| \varphi(\mathcal{E})$ can be locally studied within the resolutions $\sigma \circ \varphi_\pi(\mathcal{E}))$ where π ranges on the set of paths containing the observed state. Indeed, the limited interaction $\sigma \circ \varphi(\mathcal{E})$ was used to establish that every (subsumed) lposet of $\downarrow\mathcal{L}$ is effectively computed from a scheduler and a finisher.

The notion of finisher can be used to define properties such as "acceptable" lposets. For instance, in our read/write examples, we can define a lposet $u = (\{e_w, e_r\}, \{(e_w, e_r)\}, \{(e_w, w{:}1), (e_r, r{:}1)\})$ to be acceptable because the reading of 1 indeed occurs after the writing of 1. The lposets $v = (\{e_w, e_r\}, \emptyset, \{(e_w, w{:}1), (e_r, r{:}1)\})$ and $w = (\{e_w, e_r\}, \{(e_r, e_w)\}, \{(e_w, w{:}1), (e_r, r{:}1)\})$ are not acceptable. In the first case, the reading is not guaranteed to happen after the writing. In the second case, the writing and reading are achieved in the wrong order. Then a finisher can be defined such that $\varphi(u) = u$, $\varphi(v) = \{e_w\}$ (endowed with the canonical order)

and $\varphi(w) = \emptyset$. Hence, a restricted class of schedulers and finishers can be used to study specific properties of systems.

Finally, further and deeper explorations are needed to relate the concept of scheduler/finisher to other mature techniques, such as interval logics. This will however go beyond the original scope of the present thesis and we leave it for future works.

Chapter 4

Probabilistic Concurrent Kleene Algebra

In this chapter, we set out the algebraic foundation of *probabilistic concurrent Kleene algebra*, which combines probabilistic Kleene algebra and concurrent Kleene algebra from Chapters 2 and 3. We start by outlining the axiomatisation of the "combined algebra" in Section 4.1, whose soundness is proved with respect to the set of probabilistic automata modulo forward simulation equivalence. Most of the axioms are derived respectively from probabilistic and concurrent Kleene algebra, where the properties captured by each individual axiom were motivated and discussed in the previous respective chapters. The new piece that we add is the probabilistic choice, which is shown to satisfy the usual properties such as idempotence and quasi-associativity [15, 73, 77]. An important property of probabilistic choice operation $\oplus_p$ is given by the Inequality 4.21 which intuitively says that an early resolution of a probabilistic choice provides an over-specification to a late resolution. This usually allows us to write a specification in the form of $good \oplus_p bad$ or $good \oplus_p (good + bad)$, which provides a probabilistic bound on the occurrence of a bad behaviour. Small examples will be provided to illustrate that point as well as the general use of the algebra for the verification of action-based

$$X + X = X \tag{4.1}$$
$$X + Y = Y + X \tag{4.2}$$
$$X + (Y + Z) = (X + Y) + Z \tag{4.3}$$
$$X + 0 = X \tag{4.4}$$
$$X \cdot 1 = X \tag{4.5}$$
$$1 \cdot X = X \tag{4.6}$$
$$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z \tag{4.7}$$
$$0 \cdot X = 0 \tag{4.8}$$
$$(X + Y) \cdot Z = X \cdot Z + Y \cdot Z \tag{4.9}$$
$$X \cdot Y + X \cdot Z \leq X \cdot (Y + Z) \tag{4.10}$$
$$X^* = 1 + X \cdot X^* \tag{4.11}$$
$$X \cdot Y \leq Y \Rightarrow X^* \cdot Y \leq Y \tag{4.12}$$
$$X \| Y = Y \| X \tag{4.13}$$
$$X \| (Y \| Z) = (X \| Y) \| Z \tag{4.14}$$
$$(X \| Y) \cdot (X' \| Y') \leq (X \cdot X') \| (Y \cdot Y') \tag{4.15}$$
$$X \| Y + X \| Z \leq X \| (Y + Z) \tag{4.16}$$

Figure 4.1: Axioms of weak concurrent probabilistic Kleene algebra.

probabilistic systems.

4.1 Axiomatisation of probabilistic concurrent Kleene algebra

A concurrent Kleene algebra has four algebraic operations, namely, $(+, \cdot, \|)$ and $(*)$. These operations were given a bundle event structure semantics in Section 3.3 but the axiomatisation with respect to pomset equivalence does not allow a successful manipulation of probability. In contrast, a probabilistic Kleene algebra has the usual operations of Kleene algebra, namely, $(+, \cdot, *)$ (c.f. Section 2.1) where probabilities are treated implicitly. We provide a new probabilistic concurrent Kleene algebra that extends both structures. Without explicit probabilistic choice, we refer to such a structure as a *weak concurrent Kleene Algebra*.

Definition 4.1.1. *A* weak concurrent Kleene algebra *is an algebraic structure with signature* $(K, +, \cdot, \|, *, 0, 1)$ *where* K *is a set closed under the operations and satisfies the axioms for Figure 4.1.*

The equation $1\|X = X$ is not usually satisfied if X is an automaton which contains actions that are synchronised by the CSP parallel composition. Similarly, the right annihilator axiom $X \cdot 0 = 0$ is left out because, in the automata model of weak concurrent Kleene algebra, the entity 0 will capture *deadlocks* instead of the *abort* of relational and logical interpretation. Finally, the distribution of $\|$ through $+$ also fails in the automata model(c.f. [15] Axiom **L6**).

To gain complete control over the quantitative information, we append explicit probabilistic choices to weak concurrent Kleene algebras. That is, the choice operation $\oplus_p$ satisfies the axioms shown in Figure 4.2.

$$
\begin{aligned}
X &= X \oplus_p X && (4.17)\\
X \oplus_p Y &= Y \oplus_{1-p} X && (4.18)\\
X \oplus_p (Y \oplus_q Z) &= (X \oplus_{p'} Y) \oplus_{q'} Z && (4.19)\\
(X \oplus_p Y) \cdot Z &= (X \cdot Z) \oplus_p (Y \cdot Z) && (4.20)\\
X \cdot (Y \oplus_p Z) &\leq (X \cdot Y) \oplus_p (X \cdot Z) && (4.21)\\
X \| (Y \oplus_p Z) &\leq (X \| Y) \oplus_p (X \| Z) && (4.22)
\end{aligned}
$$

where $p, q, p', q' \in [0, 1]$ such that $q' = pq$ and $(1 - q')p' = (1 - q)p$. We assume the following precedence between the operations. The Kleene star $*$ binds more tightly than $\cdot$ which binds more tightly than $\|$. The operation $\|$ binds more tightly than $+$ and $\oplus_p$ and we use parenthesis to parse expressions having $+$ and $\oplus_p$ at the same level.

Figure 4.2: Axioms for the probabilistic choice $\oplus_p$.

Definition 4.1.2. *A probabilistic concurrent Kleene algebra is a weak concurrent Kleene algebra with a collection of probabilistic choices* $\oplus_p$, $p \in [0, 1]$, *satisfying the axioms of Figure 4.2.*

4.2 Operations on probabilistic automata

The standard constructions from automata theory have been generalised to capture probabilistic behaviour. We summarise these constructions briefly in this subsection.

4.2.1 Basic definitions

Probabilistic information is encoded as distributions over the set of states. A transition in a (nondeterministic) probabilistic automaton starts from a source state, executes an action from a given alphabet Σ and ends in a target distribution [73]. Such a distribution is then resolved into a probabilistic choice which specifies the new state of the automaton. Given a countable set P, $\mathbb{D}P$ denotes the set of probability distributions over P.

Definition 4.2.1. *A probabilistic automaton is a tuple* $(P, \rightarrow, \mu_0, F_P)$ *where*

- P *is a set of states,*

- $\rightarrow$: $P \times \Sigma \times \mathbb{D}P$ *is a set of probabilistic transitions where,*

- $\mu_0 \in \mathbb{D}P$ *is the start or initial distribution of* P*,*

- *and* $F_P \subseteq P$ *is a set of final states.*

The set of labels or actions Σ is the same for all probabilistic automata we consider. As in Chapter 2, we identify an automaton with its set of states and explicit distinction will be made only when confusion could arise.

Definition 4.2.1 provides a specialised version of probabilistic automata. Generally, a transition is composed of a state and a distribution over $\Sigma \times P$ but we restrict ourselves to automata with simple transitions (a subset of $P \times \Sigma \times \mathbb{D}P$) so that the parallel composition of two automata is easily defined. Moreover, simple probabilistic automata are expressive enough to model most practical applications. We denote by **PAut** the set of simple probabilistic automata.

The set of actions Σ is divided into two parts, namely, *internal* and *external* actions. Internal actions are either local or invisible and are usually intrinsic to the automaton where they are defined. They are not shared with other automata in the sense that they can be executed independently from the environment. A special case is the silent action τ which does not belong to the set of internal actions I and we write $I_\tau = I \cup \{\tau\}$. The silent action τ is treated differently from the ε label of Definition 2.2.1. The main difference is that τ usually carries internal computation while ε is just a device to simplify the definition of the regular operations on non-probabilistic automata. Moreover, ε could be removed while preserving standard simulation equivalence but τ cannot usually be removed without violating our definition of probabilistic automata.[1]

External actions are visible to the environment and may be synchronised. We denote the set of external actions by E, and define $\Sigma = I \cup E$ and $\Sigma_\tau = I_\tau \cup E$. The set Σ is assumed implicitly and is fixed for every automaton.

Example 4.2.2. Figure 4.3 depicts two probabilistic automata. The automaton V on the left represents a faulty vending machine with a button called `tea` which gets stuck with probability 0.2. The automaton U on the right represents the behaviours of a user interacting with the vending machine by kicking it if he fails to get his tea. Two kicks means the machine is broken.[2]

The states of the two automata are labelled by x_i, y_i respectively and distributions are not labelled unless they are initial and their components correspond to dotted arrows labelled with the probability. The set of actions is

$$\Sigma = \{\texttt{coin}, \texttt{tea}, \texttt{kick}, \texttt{fail}, \texttt{stuck}\}$$

where `stuck` is the only internal action. Notice both automata have no final state and a failure is tracked with occurrences of the action `fail`. ■

The linear run of a probabilistic automaton yields a *path*, as in the standard

[1] For instance, the τs introduced by $+$ in the expression $(a \oplus_p b) + c$ cannot be removed unless a probabilistic automata is defined to have more than a single initial distribution.

[2] This example was suggested by Steve Schneider [71].

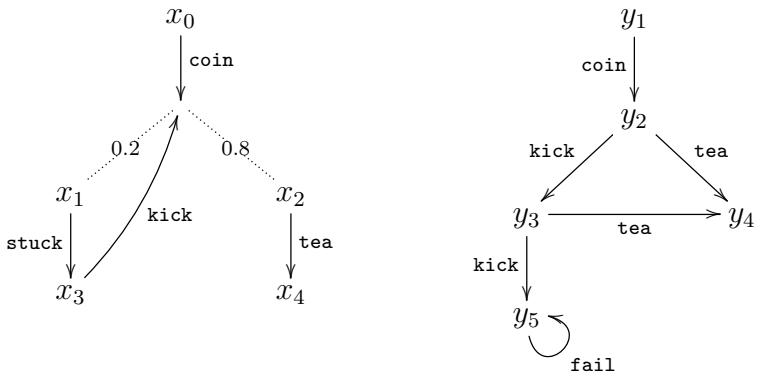


Figure 4.3: A probabilistic vending machine $V = \texttt{coin}\cdot M$ and a user $U = \texttt{coin}\cdot U'$ who is kicking the machine if it gets stuck and expecting his tea after the first kick.

---

case, which is quantified with respect to a family of probability measures. Formally,

Definition 4.2.3. *A path is a sequence $x_0 a_1 x_1 a_2 x_2 \cdots$ of alternating states and actions such that there is a sequence of transitions $x_i \xrightarrow{a_{i+1}} \mu_{i+1}$, $i \geq 0$, where $x_i \in \mathrm{supp}(\mu_i)$ (the support of μ_i) for every $i \geq 0$.*

A path α always starts with a state and, if it is finite, ends with another state denoted by $\mathrm{last}(\alpha)$. Usually, we want a path to start from a state in the support of the initial distribution. We denote $\mathrm{Path}(P)$ the set of all finite paths of the automaton P.

4.2.2 Algebraic operations on probabilistic automata

This section revises and extends the definition of algebraic operations over probabilistic automata. The standard operations of Section 2.2 are still valid up to replacing each initial state x with the point distribution δ_x in the automata $0, 1, a, P + Q$ and P^* and using τ instead of ε. The reason we use τ instead of ε is that, in this transition model, we assume that τ is actually doing some internal computation, unlike the syntactic construct ε. Secondly, we cannot usually remove the τs in our definition of probabilistic automata. In the case of sequential

composition, the initial distribution of P becomes the initial distribution of $P \cdot Q$. Therefore, we only give explicit definitions for the probabilistic choice $\oplus_p$ and the framed parallel composition $\|_A$.

4.2.2.1 Probabilistic choice

The implementation of a probabilistic choice $P \oplus_p Q$ between two automata is obtained from the convex combination $p\mu_0 + (1-p)\nu_0$ of the initial distributions μ_0 of P and ν_0 of Q.

Definition 4.2.4. *Given two probabilistic automata P and Q, we define*

$$P \oplus_p Q = (P \cup Q, \rightarrow_P \cup \rightarrow_Q, p\mu_0 + (1-p)\nu_0, F_P \cup F_Q).$$

4.2.2.2 Parallel composition

Let P and Q be two probabilistic automata. The parallel composition is defined using a probabilistic version of CSP parallel composition operation that synchronises the actions in some fixed $A \subseteq E$ [14, 15, 76]. Given $\mu \in \mathbb{D}P$ and $\nu \in \mathbb{D}Q$, the product $\mu \times \nu$ is a distribution over $P \times Q$ such that $(\mu \times \nu)(x, y) = \mu(x)\nu(y)$.

Definition 4.2.5. *We define the parallel composition of P and Q as*

$$P \|_A Q = (P \times Q, \rightarrow_{P\|_A Q}, \mu_0 \times \nu_0, F_P \times F_Q)$$

where, for each $a \in \Sigma_\tau$, a transition $(x, y) \xrightarrow{a} \mu \times \nu$ belongs to $\rightarrow_{P\|Q}$ if one of the following conditions holds:

- *$a \in A$ and $x \xrightarrow{a} \mu$ and $y \xrightarrow{a} \nu$,*
- *$a \notin A$ and $x \xrightarrow{a} \mu$ and $\nu = \delta_y$,*
- *$a \notin A$ and $y \xrightarrow{a} \nu$ and $\mu = \delta_x$.*

where δ_x is the point mass distribution centred at x i.e. $\delta_x(y) = 1$ if $x = y$ and $\delta_x(y) = 0$ otherwise.

Intuitively, transitions labelled with the same action from the frame set A are synchronised while transitions labelled with actions from $\Sigma_\tau \setminus A$ are interleaved.

Example 4.2.6. We can express the automata from Figure 4.3 using the algebraic language provided. The right hand side automaton of Figure 4.3 corresponds to the algebraic expression

$$\texttt{coin} \cdot (\texttt{kick} \cdot (\texttt{kick} \cdot \texttt{fail}^* + \texttt{tea}) + \texttt{tea})$$

where we have abused notation by denoting the automaton that does a single action, say `coin`, and then terminates successfully with the same notation `coin`.

The left hand side is obtained as a sequential composition $\texttt{coin} \cdot M$ where M corresponds to the least fixed point of

$$f(X) = \texttt{stuck} \cdot \texttt{kick} \cdot X \cdot 0 \oplus_{0.8} \texttt{tea} \cdot 0.$$

We will compute the least fixed point of f algebraically in Section 4.4. ■

4.3 Probabilistic forward simulation

In this section, we define an inequality $\leq$ on the set **PAut** as per the constructions of [14, 15, 45, 74]. The equivalence relation associated with $\leq$ is based on weak forward simulation. We are mainly interested in the soundness of probabilistic concurrent Kleene algebras with respect to this model.

We give two equivalent definitions of simulation. The first definition is based on the probabilistic simulation of Deng et al. [15], while the second is Segala's probabilistic weak forward simulation [74]. The equivalence ensures that the results can be translated from one to the other. Both definitions of simulation rely on the lifting of relations from states to distributions.

4.3.1 Lifting from $X \times \mathbb{D}Y$ to $\mathbb{D}X \times \mathbb{D}Y$

Let X, Y be two (countable) sets.

Definition 4.3.1 ([15]). *Given a relation $S \subseteq X \times \mathbb{D}Y$, the lifting of S is a relation $\overline{S} \subseteq \mathbb{D}X \times \mathbb{D}Y$ such that $(\mu, \nu) \in \overline{S}$ iff there exists a (countable) family of real number $\{p_i \mid i \in I\} \subseteq [0,1]$ such that $\sum_i p_i = 1$ and*

1. *$\mu = \sum_i p_i \delta_{x_i}$, for some family of $x_i \in X$ and*
2. *for each $i \in I$, there exists $\nu_i \in \mathbb{D}Y$ such that $(x_i, \nu_i) \in S$, and*
3. *$\nu = \sum_i p_i \nu_i$.*

These sums are over the set I, which we leave implicit to simplify the notations.

Lifting is a probability preserving function that associates to each probabilistic relation S a relation $\overline{S}$ over the set of distributions. It is important to notice that the decomposition of μ is not necessarily canonical, that is, there may be some repetition in the x_i.

The most important properties of the lifting transformation is summarised by the following proposition (the proof is given in [15]) where the sums run over a fixed finite set of indices.

Proposition 4.3.2 ([15]). *Let $S \subseteq X \times \mathbb{D}Y$ be a relation and $\sum_i p_i = 1$. We have*

- *if the tuple (μ_i, ν_i) is in $\overline{S}$ for all i, then $(\sum_i p_i \mu_i, \sum_i p_i \nu_i) \in \overline{S}$,*
- *if $(\sum_i p_i \mu_i, \nu) \in \overline{S}$ then there exists a finite collection of distributions ν_i such that $(\mu_i, \nu_i) \in \overline{S}$ and $\nu = \sum_i p_i \nu_i$.*

Lifting also applies to labelled transitions because $\cdot \xrightarrow{a} \cdot \subseteq P \times \mathbb{D}P$ for any probabilistic automaton P and any action $a \in \Sigma_\tau$. Hence, we denote $\xrightarrow{\overline{a}}$ the lifting of this transition relation, which corresponds to the notion of *combined transition* [45, 74]. That is, a family of transitions $x_i \xrightarrow{a} \mu_i$ induces a lifted transition $\sum_i p_i \delta_{x_i} \xrightarrow{\overline{a}} \sum_i p_i \mu_i$.

We extend internal transitions with stuttering, that is, we write $x \xrightarrow{\tau} \mu$ if such a transition exists in the automaton or $\mu = \delta_x$. Stuttering implies that a simulation allows a sequence of τs to be executed in one automaton while staying in the same state in the other. The lifted version is again denoted $\xrightarrow{\overline{\tau}} \subseteq \mathbb{D}P \times \mathbb{D}P$. Finally, weak transitions are obtained from the reflexive transitive closure of $\xrightarrow{\overline{\tau}}$, denoted $\Longrightarrow$, and we write $\mu \overset{a}{\Longrightarrow} \mu'$ if there exist μ_1, μ_2 such that $\mu \Longrightarrow \mu_1 \xrightarrow{\overline{a}} \mu_2 \Longrightarrow \mu'$. We now give the formal definition of simulation by straightforwardly generalising [15] to automata with final states.

Definition 4.3.3 ([15]). *A probabilistic simulation S from P to Q is a relation $S \subseteq P \times \mathbb{D}Q$ satisfying the following properties:*

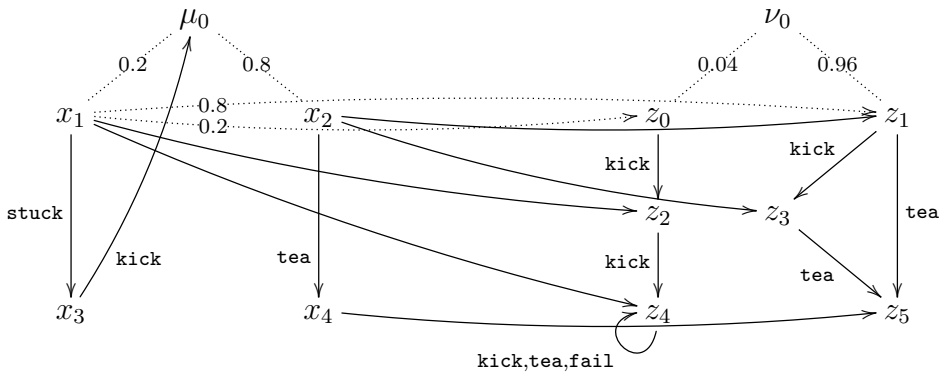
1. *there exists ν_0' such that $(\mu_0, \nu_0') \in \overline{S}$ and $\nu_0 \Longrightarrow \nu_0'$,*
2. *if $x \xrightarrow{a} \mu'$ is a valid transition of P and $(x, \nu) \in S$, there exists $\nu' \in \mathbb{D}Q$ such that $\nu \overset{a}{\Longrightarrow} \nu'$ and $(\mu', \nu') \in \overline{S}$,*
3. *if $x \in F_P$ and $(x, \nu) \in S$ then there exists $\nu' \in \mathbb{D}F_Q$ such that $\nu \Longrightarrow \nu'$.*

Property (1) ensures that preceding τ actions do not interfere with probabilistic choices (i.e. $P \oplus_p Q$ and $\tau \cdot (P \oplus_p Q)$ are simulation equivalent). Property (2) is the usual co-inductive definition of simulation and property (3) ensures that if a state $x \in P$ is simulated by a distribution $\nu \in \mathbb{D}Q$ and P can terminate successfully at x, then Q can also terminate successfully from ν after a *finite number* of internal transitions.

A simulation is always total on reachable states, that is, if $S \subseteq P \times \mathbb{D}Q$ is a simulation and $x \in P$ such that $x_0 a_1 x_1 \cdots x$ is a path that occurs with positive maximal probability, then there exists $\nu \in \mathbb{D}Q$ such that $(x, \nu) \in S$.

Example 4.3.4. Figure 4.4 depicts two automata related by a simulation relation i.e. $M \leq H$ where M (resp. H) is the left (resp. right) automaton. The simulation is obtained from the relation $S = S' \cup \{(x_3, \mu) \mid (x_1, \mu) \in S'\}$ where

$$S' = \{(x_1, 0.2\delta_{z_0} + 0.8\delta_{z_1}), (x_1, \delta_{z_2}), (x_1, \delta_{z_4}), (x_2, \delta_{z_1}), (x_2, \delta_{z_3}), (x_4, \delta_{z_5})\}.$$



(M) (H)

The action `stuck` is an internal action so we have removed the arrows from x_3 because they are exactly the same as for x_1. The dotted arrow represents non-trivial distribution again.

Figure 4.4: Two automata related by a simulation.

---

where we recall that δ_x is the point distribution concentrated at the state x. To see that S is indeed a simulation, let us write $\nu_0 = 0.2(0.2\delta_{z_0} + 0.8\delta_{z_1}) + 0.8\delta_{z_1}$ where $(x_1, (0.2\delta_{z_0} + 0.8\delta_{z_1})) \in S$ and $(x_2, \delta_{z_1}) \in S$. Hence, $(\mu_0, \nu_0) \in \overline{S}$. Since `stuck` is an internal action, it follows that $(x_3, \mu) \in S$ for every distribution μ such that $(x_1, \mu) \in S$. Next, we have $(x_3, (0.2\delta_{z_0} + 0.8\delta_{z_1})) \in S$ and $x_3 \xrightarrow{\texttt{kick}} \mu_0$. Since $\mu_0 = 0.2\delta_{x_1} + 0.8\delta_{x_2}$ and $(x_1, \delta_{z_2}) \in S$ and $(x_2, \delta_{z_3}) \in S$, it follows that $(\mu_0, (0.2\delta_{z_2} + 0.8\delta_{z_3})) \in \overline{S}$ and $(0.2\delta_{z_0} + 0.8\delta_{z_1}) \stackrel{\overline{\texttt{kick}}}{\Longrightarrow} (0.2\delta_{z_2} + 0.8\delta_{z_3})$. The other inductive cases are proved in similar fashion. Moreover, an algebraic proof is given in the next section. ■

We write $P \leq Q$ if there is a simulation from P to Q and $P \equiv Q$ iff $P \leq Q$ and $Q \leq P$ and it is indeed a preorder [15].

In this chapter, any probabilistic relation satisfying Definition 4.3.3 will be referred to as a simulation.

4.3.1.1 Segala's simulation

Another formulation is given by Segala's probabilistic forward simulation which corresponds to the coarsest precongruence included in the trace distribution equiv-

alence [74].

This notion of simulation relies on *double-liftings*. Given two countable sets X, Y and a relation $S \subseteq X \times Y$, the double-lifting of S, denoted $\widehat{S}$, is a subset of $\mathbb{D}X \times \mathbb{D}Y$ where $(\mu, \nu) \in \widehat{S}$ iff there exists a function $w : X \times Y \to [0, 1]$ such that

1. if $w(x, y) > 0$ then $(x, y) \in S$,
2. for every $x \in X$, $\sum_{y \in Y} w(x, y) = \mu(x)$,
3. for every $y \in Y$, $\sum_{x \in X} w(x, y) = \nu(y)$.

The function w is a probability preserving function that provides corresponding decompositions for μ and ν. Since a probabilistic weak forward simulation is again defined as a subset of $P \times \mathbb{D}Q$, double-lifting generates an element of $\mathbb{D}P \times \mathbb{D}\mathbb{D}Q$ which complicates the lifting of transitions. To obtain a standard relation over the set of distributions, Segala [45, 74] provided a flat version of a distribution in $\mathbb{D}\mathbb{D}Q$ through the use of a projection $\pi : \mathbb{D}\mathbb{D}Q \to \mathbb{D}Q$ such that

$$\pi(\phi) = \sum_{\mu \in \mathrm{supp}(\phi)} \phi(\mu)\mu.$$

where $\phi \in \mathbb{D}\mathbb{D}Q$. We now give the modified version of Segala's probabilistic weak forward simulation.

Definition 4.3.5. *A relation $S \subseteq P \times \mathbb{D}Q$ is a probabilistic weak forward simulation if*

1. *there exists $\psi_0 \in \mathbb{D}\mathbb{D}Q$, such that $(\mu_0, \psi_0) \in \widehat{S}$ and $\nu_0 \Longrightarrow \pi(\psi_0)$,*
2. *if $x \xrightarrow{a} \mu'$ is a valid transition of P and $(x, \nu) \in S$, there exists $\psi \in \mathbb{D}\mathbb{D}Q$ such that $\nu \overset{a}{\Longrightarrow} \pi(\psi)$ and $(\mu', \psi) \in \widehat{S}$.*
3. *if $x \in F_P$ and $(x, \nu) \in S$, then there exists $\psi \in \mathbb{D}\mathbb{D}F_Q$ such that $\nu \Longrightarrow \pi(\psi)$.*

We now show that Definitions 4.3.5 and 4.3.3 are equivalent.

Proposition 4.3.6. *Let X, Y be two sets, $S \subseteq X \times \mathbb{D}Y$, $\mu \in \mathbb{D}X$ and $\psi \in \mathbb{D}\mathbb{D}Y$. If $(\mu, \psi) \in \widehat{S}$ then $(\mu, \pi(\psi)) \in \overline{S}$.*

Proof. If $(\mu, \psi) \in \widehat{S}$, then there exists $w : X \times \mathbb{D}Y \to [0,1]$ satisfying the condition above. Then by considering $I = \mathrm{supp}(w)$, it directly follows that $\mu = \sum_{i \in I} w(i)\delta_{x_i}$, each x_i is related to some ν_i by S and $\pi(\psi) = \sum_{i \in I} w(i)\nu_i$. ■

Corollary 4.3.7. *A relation is a probabilistic simulation iff it is a probabilistic weak forward simulation on* **PAut**.

Proof. That Definition 4.3.5 implies Definition 4.3.3 follows directly from the previous proposition.

Conversely, assume that $S \subseteq P \times \mathbb{D}Q$ satisfies Definition 4.3.3. If $(\mu, \nu) \in \overline{S}$, then there exits a decomposition $\mu = \sum_{i \in I} p_i \delta_{x_i}$ such that for each i, there exists $\nu_i \in \mathbb{D}Q$ such that $(x_i, \nu_i) \in \overline{S}$ for each i, and $\nu = \sum_{i \in I} p_i \nu_i$. Hence $(\mu, \sum_{i \in I} p_i \delta_{\nu_i}) \in \widehat{S}$. We just apply this simple construction to each of the three cases of Definition 4.3.5. ■

We conclude this section with a remark about the two definitions of probabilistic simulation and their relationship to the theory of testing [63]. Corollary 4.3.7 shows that the corresponding definitions of [73] and [15] coincide (notice that we can replace final states with some special external action and obtain a formulation closer to those given in [15, 74]).

On one hand, Segala has shown that the largest precongruence included in the trace distribution equivalence coincides with "vector may testing" [74]. On the other hand, Deng et al. have shown that vector and scalar testings coincide on the recursion-free fragment of probabilistic automata and that with the same restriction, Definition 4.3.3 is complete for testing equivalence [14]. Using Proposition 4.3.7 and Segala's result, we conclude that Deng's completeness for may testing extends to automata with countable state spaces (this is particularly important when unfolding the automata P^*). However, it is still unknown whether the equivalence between scalar and vector testing in the infinite case is valid.

These equivalences are the main motivation for our use of simulation in order to create an interleaving model for our algebra. It should be noted that probabilistic simulation equivalence is decidable for finite automata but it is unknown whether

an efficient decision procedure exists. This is in contrast to other related results in the literature showing that strong simulation is decidable in polynomial time [35].

4.4 Interleaving interpretation of pCKA

In this section, we show that the set of **PAut** of simple probabilistic automata yields a denotational model for probabilistic concurrent Kleene algebra. For simplification, we assume synchronisation over all external actions and denote that operation simply with $\|$ without any frame set.

Equations (4.1-4.4) of Figure 4.1 amd (4.17-4.19) of Figure 4.2 are standard and the proofs are omitted (they can be found in [15]). Moreover, the equivalence $P \leq Q$ iff $P + Q \equiv Q$ follows from these equations, that is, simulation coincides with the natural order of the algebra. Recall that in our interpretation Q has more behaviours than P. A complete characterisation of the consequences of Equation (4.17-4.19) defining the probabilistic choice $\oplus_p$ with respect to probabilistic bisimulation can be found in the work of Stark and Smolka on the axiomatisation of finite state probabilistic processes [77].

We now verify the axioms of probabilistic concurrent Kleene algebra against the structure $(\mathbf{PAut}, +, \cdot, \oplus_p, \|, *, 0, 1)$.

Proposition 4.4.1. *The structure* $(\mathbf{PAut}, +, \cdot, \oplus_p, \leq)$ *satisfies equations (4.5-4.10) of Figure 4.1 and (4.20-4.21) of Figure 4.2.*

Proof. In this proof, we mostly show the construction of the simulations as the proofs of their properties are achieved by straightforward case analysis on the definition of simulation.

Equation (4.5) and (4.6) are clear and (4.8) follows form the fact that $P \equiv Q$ iff their reachable parts are simulation equivalent.

Associativity (4.7) is evident because the left and right hand side automata are exactly the same.

For distributivity (4.9), let us write the left hand side term as $P \cdot R + Q \cdot R_c$ where R_c is a copy of R whose states are renamed to x_c for every state x of R. We

construct a relation $S \subseteq (P \cup Q \cup \{z\} \cup R \cup R_c) \times \mathcal{D}(P \cup Q \cup \{z\} \cup R)$ such that $S = \{(x, \delta_x), (x_c, \delta_x) \mid x \in R \wedge x_c \text{ is the copy of } x\} \cup id_{P+Q}$. It is straightforward to show that S is a simulation and so is its inverse.

For subdistributivity (4.10), we consider the relation

$$S = \{(x, \delta_x), (x_c, \delta_x) \mid x \in P \wedge x_c \text{ is the copy of } x\} \cup \{(z, \mu_0)\} \cup id_Q \cup id_R$$

where z is the initial state of $P \cdot Q + P \cdot R$ and μ_0 is the initial distribution of P. It is again straightforward to prove that S is indeed a simulation.

We refer to Stark and Smolka [77] or the more recent work of Deng et al. [15] for the proof of equations (4.17-4.19).

Equation (4.20) is proved using the exact same simulation constructed in the case of Equation (4.9).

For the last equation 4.21, let

$$S = \{(x, \delta_x \oplus_p \delta_{x_c}) \mid x \in P \wedge x_c \text{ is the copy of } x\} \cup id_Q \cup id_R$$

This simulation essentially says that we carry down the probabilistic choice $\oplus_p$ on the left hand side until it needs to be resolved.

- By construction of the simulation, we have $(\mu_0, (\mu_0 \oplus_p \mu_{0c})) \in \overline{S}$ where μ_0 and μ_{0c} are the respective initial distributions of P and P_c.
- Let $x \xrightarrow{a}_{P \cdot (Q \oplus_p R)} \mu$ and $(x, \nu) \in S$, there are three cases:
  - $x \xrightarrow{a}_P \mu$, therefore $\nu = \delta_x \oplus_p \delta_{x_c}$ and $\nu \xrightarrow{\overline{a}}_{P \cdot Q \oplus_p P \cdot R} \mu \oplus_p \mu_c$ where μ_c is the copy of μ.
  - $x \xrightarrow{a}_Q \mu$ or $x \xrightarrow{a}_R \mu$, then we obtain the desired result because $id_Q \cup id_R \subseteq S$.
  - $x \xrightarrow{\tau}_{P \cdot (Q \oplus_p R)} \mu_{0Q} \oplus_p \mu_{0R}$ and $x \in F_P$, then $x \xrightarrow{\tau}_{P \cdot Q} \mu_{0Q}$ and $x \xrightarrow{\tau}_{P_c \cdot R} \mu_{0R}$ are valid transitions of $P \cdot Q$ and $P_c \cdot R$. But $\nu = \delta_x \oplus_p \delta_{x_c}$ because $x \in P$, therefore $\nu \xrightarrow{\overline{\tau}}_{P \cdot Q \oplus_p P \cdot R} \mu_{0Q} \oplus_p \mu_{0R}$.

- Let $xS\nu$ and x is a final state. By definition of $\oplus_p$, $x \in F_Q \cup F_R$ and hence $\nu = \delta_x \in \mathbb{D}F_Q \cup \mathbb{D}F_R$. ■

Proposition 4.4.2. *The structure* $(\mathbf{PAut}, +, \cdot, *, \leq)$ *satisfies the unfolding axiom (4.11) and induction law 4.12.*

Proof. Let x_0 be the initial state of $1 + P \cdot P^*$ and y_0 be the initial state of P^*. Since we add only one state and some transition in the construction of P^*, we denote $x_* \in P^*$ the state corresponding to x, for each $x \in P$. To prove Equation (4.11), we consider the relation

$$S = \{(x_*, \delta_{x_*}), (x_*, \delta_x) \mid x \in P\} \cup \{(y_0, \delta_{y_0}), (y_0, \delta_{x_0})\}$$

from P^* to $1 + P \cdot P^*$ (Notice that y_0 is a state of $1 + P \cdot P^*$). We now prove that S is a simulation.

- For the initial distribution, we have $(y_0, \delta_{x_0}) \in S$.
- Let $y \xrightarrow{a} \mu$ be a valid transition in P^* and $(y, \nu) \in S$. There are two cases:
  - $y = y_0$ and the transition is $y_0 \xrightarrow{\tau} \mu_0$ where μ_0 is the initial distribution of P. If $\nu = \delta_{y_0}$ then we are done because $\{(y_0, \delta_{y_0})\} \cup \{(x_*, \delta_{x_*}) \mid x \in P\} = id_{P^*}$. Else, $\nu = \delta_{x_0}$ and $x_0 \xrightarrow{\tau} \mu_0$ is a valid transition in $1 + P \cdot P^*$.
  - $y = x_*$ for some $x \in P$ and:
    - $x_* \xrightarrow{a} \mu_*$ is the copy of a transition of P. Therefore, if $\nu = \delta_{x_*}$ then the same transition belongs to $P \cdot P^*$. If $\nu = \delta_x$ then $x \xrightarrow{a} \mu$ is a transition of P and $\mu_* \overline{S} \mu$.
    - or, $x_* \xrightarrow{\tau} \delta_{y_0}$ and in this case, if $\nu = \delta_{x^*}$ then that transition belongs to $P \cdot P^*$ again, else $\nu = \delta_x$ and $x \in F_P$. Therefore, $\delta_x \xrightarrow{\overline{\tau}} \delta_{y_0}$ is a lifted transition in $P \cdot P^*$.
- The preservation of final state is obvious because $F_{P^*} = \{y_0\}$ and $x_0 \xrightarrow{\tau} \delta_z$ where z is the final state of 1 in $1 + P \cdot P^*$ (which justify the case $(y_0, \delta_{x_0}) \in S$).

With the similar reasoning, it holds that the inverse relation S^{-1} of S is a simulation from $1 + P \cdot P^*$ to P^*.

We now prove the induction law (4.12). Firstly, let us introduce the notion of unfolding which will simplify the proof considerably. [3]

We denote $\mathbf{unfold}(P)$ the unfold of any automaton P [45], that is, the automaton

$$(\mathrm{Path}(P), \to, \mu_0, F)$$

where a transition $\alpha \xrightarrow{a} \mu$ holds for $\alpha \in \mathrm{Path}(P)$ and $\mu \in \mathbb{D}\mathrm{Path}(P)$ iff there exists $\mu' \in \mathbb{D}P$ such that $\mathrm{last}(\alpha) \xrightarrow{a}_P \mu'$ and $\mu(\alpha a x) = \mu'(x)\}$. The set of final states is

$$F = \{\alpha \in \mathrm{Path}(P) \mid \mathrm{last}(\alpha) \in F_P\}.$$

and μ_0 is the initial distribution of P. This construction provides us with an automaton whose states are finite paths in P and there is a transition between two paths α, α' iff $\alpha' = \alpha a x$ where $a \in \Sigma_\tau$ and $x \in P$. Such a transition is labelled by a. It is now easy to show that the relation $\{(\alpha, \delta_{\mathrm{last}(\alpha)}) \mid \alpha \in \mathrm{Path}(P)\}$ is a simulation from $\mathbf{unfold}(P)$ to P and the inverse is also a simulation from P to $\mathbf{unfold}(P)$ [45].

Now, we can assume that P is loop-free by unfolding it and therefore $1 + P \cdot \mathbf{unfold}(P^*)$ is again loop-free and simulation equivalent to P^*. Let $f(X) = 1 + P \cdot X$. Since $P \cdot 0 \leq P$, we show easily by induction that $\mathbf{unfold}(f^n(0)) \subset \mathbf{unfold}(f^{n+1}(0))$ where $\subset$ is the inclusion of automata. We define $X \subset Y$ if the state space of X is a subset of the state space of Y, transitions of X are transitions of Y and $F_X \subseteq F_Y$. We can then construct a limit automaton $\sup_n f^n(0) = f^*(0)$ obtained as the countable union of component by component (the set of states is the union of the sets of states, the set of transitions is the union of sets of transitions,...). Since P has no cycle, it follows that $f^*(0) = \mathbf{unfold}(P^*)$.

Now assume that $P \cdot Q \leq Q$, then $(1 + P \cdot 0) \cdot Q \leq (1 + P) \cdot Q \leq Q$ and by induction, $f^n(0) \cdot Q \leq Q$ for every $n \in \mathbb{N}$. Moreover, since $\mathbf{unfold}(f^n(0)) \subset$

[3] It is essentially a cleaner version of our construction in [49]

$\mathbf{unfold}(f^{n+1}(0))$, we have $\mathbf{unfold}(f^n(0)) \cdot Q \subset \mathbf{unfold}(f^{n+1}(0)) \cdot Q$ and since $F_{\mathbf{unfold}(f^n(0))} \subseteq F_{\mathbf{unfold}(f^{n+1}(0))}$ (inclusion of final states), $\sup_n(\mathbf{unfold}(f^n(0)) \cdot Q) = f^*(0) \cdot Q$ (the two automaton are equal by construction). Hence $f^*(0) \cdot Q \leq Q$. ■

The proof that the simulation order provided by Definition 4.3.3 is indeed a pre-congruence respecting $(+, \oplus_p)$ and $(\|)$ occurs abundantly in the literature [14, 15, 73, 75].

Proposition 4.4.3. *Simulation is a precongruence i.e. if $P \leq Q$ then $P + R \leq Q + R$, $P \cdot R \leq Q \cdot R$, $P^* \leq Q^*$, $P \oplus_p R \leq Q \oplus_p R$, $P \| R \leq Q \| R$ and the same holds for binary operations when the order of the arguments is reversed.*

Proof. We only provide algebraic proofs of congruence for the sequential composition and Kleene star, which depends on the explicit construction of simulations in the proof of Proposition 4.4.1 and 4.4.2.

Congruence with respect to the sequential composition $(\cdot)$ is a standard consequences of Equation 4.9 and 4.10.

For Kleene star, Let $P \leq Q$. By Equation 4.11, we have $Q \cdot Q^* \leq Q^*$. Therefore, the congruence of $\leq$ with respect to $(\cdot)$ (or monotonicity of $(\cdot)$) implies that $P \cdot Q^* \leq Q \cdot Q^* \leq Q^*$. By the induction law 4.12, we obtain $P^* \cdot Q^* \leq Q^*$ and since $1 \leq Q^*$, we deduce $P^* \leq P^* \cdot Q^* \leq Q^*$ using Equation 4.5. ■

Proposition 4.4.4. *The structure $(\mathbf{PAut}, +, \cdot, \|, \oplus_p)$ satisfies equations (4.13-4.16) of Figure 4.1 and $\|$ distributes through $\oplus_p$*

$$P \| Q \oplus_p P \| R \quad \equiv \quad P \| (Q \oplus_p R) \tag{4.23}$$

Proof. Proof of Equation (4.13),(4.16) and (4.23) follows from a simple adaptation of the proofs in [15] (where final states need to be taken care of).

For the associativity (4.14), recall that when the frame is fixed, then there is a standard simulation between $P \| (Q \| R)$ and $(P \| Q) \| R$ by associating each tuple $(x, (y, z))$ to $((x, y), z)$. That simulation is lifted to $(P \times (Q \times R)) \times \mathbb{D}((P \times Q) \times R)$ using point distributions. The converse simulation is obtained by symmetry.

As for the interchange law (4.15), we consider the injection

$$S = \{((x, y), \delta_{(x,y)}) \mid (x, y) \in (P \times Q) \cup (P' \times Q')\}$$

from $U = (P \| Q) \cdot (P' \| Q')$ to $V = P \cdot P' \| Q \cdot Q'$.

- Using the definition of $\|$ and $\cdot$, we deduce that the initial distributions of U and V are the same.
- Let $((x, y), \delta_{(x,y)}) \in S$ and $(x, y) \xrightarrow{a}_U \mu$. There are three cases:
  - $(x, y) \in P \times Q$ and $\mu = \mu_P \times \mu_Q \in \mathcal{D}(P \times Q)$. In all three cases in the definition of $\|$, we have $(x, y) \xrightarrow{a}_V \mu_P \times \mu_Q$.
  - $(x, y) \in P' \times Q'$ and $\mu = \mu_{P'} \times \mu_{Q'} \in \mathcal{D}(P' \times Q')$. This is the same as the previous case.
  - $(x, y) \in F_P \times F_Q$ and the transition is $(x, y) \xrightarrow{\tau}_U \mu_{0P'} \times \mu_{0Q'}$ where $\mu_{0P'}, \mu_{0Q'}$ are the respective initial distributions of P', Q'. Since $x \in F_P$, $x \xrightarrow{\tau}_{P \cdot P'} \mu_{0P'}$ and similarly for $y \in F_Q$. Therefore,

    $$(x, y) \xrightarrow{\tau}_V \mu_{0P'} \times \delta_y \xrightarrow{\overline{\tau}}_V \mu_{0P'} \times \mu_{0Q'}$$

    i.e. $(x, y) \Longrightarrow_V \mu_{0P'} \times \mu_{0Q'}$ is a weak lifted transition in V.
- Finally, $F_U = F_V$, so the preservation of final states is clear. ■

Notice that we cannot have equality for the interchange law (4.15) even with a fully synchronised $\|$. For example, if we let $a \in \Sigma$ be an external synchronised action, we have $(a \cdot 1) \| (1 \cdot a) > (a \| 1) \cdot (1 \| a)$. On the left hand side of the inequality, the sequential composition $1 \cdot a$ will generate a single τ action which is not synchronised. Hence, the action a will be synchronised. On the right hand side, the action a appears only in one side of the $\|$ in the expression $a \| 1$. Hence, it will be "blocked", resulting in a *deadlock*. Indeed, this happens because of our CSP style parallel composition.

The previous three propositions are summarised in the following theorem:

Theorem 4.4.5. *The structure* $(\mathbf{PAut}, +, \cdot, \|, *, 0, 1)$ *forms a probabilistic concurrent Kleene algebra.*

Example 4.4.6. We end this section by providing an algebraic proof for the existence of a simulation between the automata in Figure 4.4. First, we compute the least fixed point of the function f of Example 4.2.6 algebraically. We have

$$f(X) = (\mathtt{stuck} \cdot \mathtt{kick} \oplus_{0.8} \mathtt{tea} \cdot \mathbf{0}) \cdot X \cdot 0$$

using equations (4.8) and (4.20). Next, we show that the least fixed point of $f(X) = P \cdot X \cdot 0$ is $P^* \cdot 0$, where $P = \mathtt{stuck} \cdot \mathtt{kick} \oplus_{0.8} \mathtt{tea} \cdot 0$. In fact $f(P^* \cdot 0) = P \cdot P^* \cdot 0 = (1 + P \cdot P^*) \cdot 0 = P^* \cdot 0$ because of equations (4.6), (4.9) and (4.11).

Now let Q be a suffix-point of f, i.e., $P \cdot Q \cdot 0 \leq Q$, then monotonicity and Equation (4.8) imply $P \cdot Q \cdot 0 \leq Q \cdot 0$. Therefore, $P^* \cdot Q \cdot 0 \leq Q \cdot 0 \leq Q$ because of the induction law (4.12) and $0 \leq 1$. Hence $P^* \cdot 0 \leq Q$ follows from Equation (4.8) and monotonicity of $(\cdot)$.

Therefore, the left hand side automaton is simulation equivalent to

$$M = (\mathtt{stuck} \cdot \mathtt{kick} \oplus_{0.8} \mathtt{tea} \cdot 0)^* \cdot 0$$

The algebraic proof in Figure 4.5 shows that $M \leq H$. ■

4.5 Completing a proof of correctness

Our ultimate goal for the system illustrated by Example 4.2.2 is to compute the probability that the client will successfully obtain his tea. Indeed, it is possible to compute that probability by a direct computation of the parallel composition between the specification of the machine V and the user U. However, we will establish the postcondition Q described in Figure 4.6 using the algebraically established simulation $M \leq H$ of Figure 4.5. In plain English, the postcondition Q says that a user will `fail` to get his `tea` with (maximal) probability 0.04.

$$
\begin{aligned}
& M \\
=\quad & && \text{One unfold of } M \\
& (\mathtt{stuck} \cdot \mathtt{kick} \oplus_{0.8} \mathtt{tea} \cdot 0) \cdot M \\
\equiv\quad & && \text{Equations (4.20) and (4.8)} \\
& \mathtt{stuck} \cdot \mathtt{kick} \cdot M \oplus_{0.8} \mathtt{tea} \cdot 0 \\
\equiv\quad & && \text{Unfolding of } (*) \text{ and definition of } M \\
& \mathtt{stuck} \cdot \mathtt{kick} \cdot (\mathtt{stuck} \cdot \mathtt{kick} \oplus_{0.8} \mathtt{tea} \cdot 0) \cdot M \oplus_{0.8} \mathtt{tea} \cdot 0 \\
\leq\quad & && \text{Equation (4.21)} \\
& (\mathtt{stuck} \cdot \mathtt{kick} \cdot \mathtt{stuck} \cdot \mathtt{kick} \cdot M \oplus_{0.8} \mathtt{stuck} \cdot \mathtt{kick} \cdot \mathtt{tea} \cdot 0) \oplus_{0.8} \mathtt{tea} \cdot 0 \\
\equiv\quad & && \text{Equation (4.19)} \\
& \mathtt{stuck} \cdot \mathtt{kick} \cdot \mathtt{stuck} \cdot \mathtt{kick} \cdot M \oplus_{0.96} (\mathtt{stuck} \cdot \mathtt{kick} \cdot \mathtt{tea} \cdot 0 \oplus_{0.8/0.96} \mathtt{tea} \cdot 0) \\
\leq\quad & && P \oplus_p Q \leq P + Q \\
& \mathtt{stuck} \cdot \mathtt{kick} \cdot \mathtt{stuck} \cdot \mathtt{kick} \cdot M \oplus_{0.96} (\mathtt{stuck} \cdot \mathtt{kick} \cdot \mathtt{tea} \cdot 0 + \mathtt{tea} \cdot 0) \\
\leq\quad & && M \leq \mathbf{Run}(\{\mathtt{kick}, \mathtt{tea}, \mathtt{fail}\}) \\
& \mathtt{kick} \cdot \mathtt{kick} \cdot \mathbf{Run}(\{\mathtt{kick}, \mathtt{tea}, \mathtt{fail}\}) \oplus_{0.96} (\mathtt{kick} \cdot \mathtt{tea} \cdot 0 + \mathtt{tea} \cdot 0) \\
=\quad & && \text{Definition of } H \\
& H
\end{aligned}
$$

where $\mathbf{Run}(A) = (+_{a \in A} a)^*$ where $A \subseteq \Sigma$.

Figure 4.5: Example of algebraic proof in pCKA.

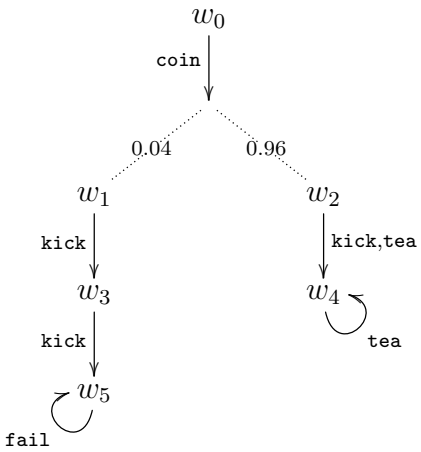


Figure 4.6: The postcondition for the system in the form $Q = \texttt{coin}\cdot(bad \oplus_{0.96} good)$.

By computing the straightforward parallel composition $H\|U'$ (where we recall that U' is the subsequent behaviour of a user in its specification $U = \texttt{coin} \cdot U'$), we readily obtain the simulation

$$\texttt{coin} \cdot (H\|U') \leq Q.$$

Therefore, by the congruence properties of $\leq$ (Proposition 4.4.3), we have

$$V\|U = \texttt{coin} \cdot (M\|U') \leq \texttt{coin} \cdot (H\|U') \leq Q.$$

The reasoning used here to establish $V\|U \leq Q$ is a particular case of a more general framework, namely, the rely/guarantee calculus or, more precisely, assume/guarantee technique in the case of action-based systems [35, 43]. In other words, H can be seen as a rely condition for the user and a guarantee condition for the machine (`coin` and Q are the usual pre/postconditions). The full development of such a tool in the probabilistic case is delayed until Chapter 7 where all algebraic proof is done within the framework of probabilistic concurrent Kleene algebra, hence establishing the various rely/guarantee rules in all models of that algebra. In particular, these rules can be used in the context of probabilistic

automata modulo probabilistic simulation.

4.6 Discussion

This chapter established our foundation of probabilistic concurrent Kleene algebra and illustrated how it can be used in verification tasks. The axiom system was proven sound with respect to the set of probabilistic automata under simulation equivalence and no completeness result is inferred whatsoever. Segala and Parma have given a complete axiomatisation of probabilistic simulation where they have used general recursion instead of the tail iteration resulting from the use of Kleene star [64]. It should be noted that the restriction to the regular operations $(+, \cdot, *)$ (in addition to $(\|)$) usually complicates the algebraic characterisation. That issue dates back to Milner [56]. Moreover, the exchange law does not appear in Segala and Parma's axiomatisation but it is an essential ingredient for an algebraic view of concurrency as shown by Gischer's completeness result [23]. In the next chapter, we introduce a true concurrent model of probabilistic concurrent Kleene algebra.

Chapter 5

True Concurrency in Probabilistic Concurrent Kleene Algebra

This chapter gives an adaptation and extension of Katoen's [33] and Varacca's [85] approaches to probabilistic event structures. The aim is to obtain a truly concurrent semantics for the axioms of probabilistic Kleene algebra.

The main idea of the probabilistic bundle event structure model hinges tightly on the notion of clusters. Clusters are sets of events that are in conflict with each other. Therefore, only a single event from a cluster can occur in a consistent behaviour. Hence, clusters were first introduced by Katoen to provide supports for probabilistic distributions. Notice that clusters are also used to obtain a successful algebraic interpretation of nondeterminism on event structures, as discussed in Definition 3.3.1.

This chapter is based on Katoen's probabilistic bundle event structure. However, Katoen did not provide an adequate equality or inequality for comparing these structures and Varacca [85] has given a semantics based on valuation on configurations which is the counterpart of Segala's trace distribution for prime

event structures. In this thesis, we adapt the notion of probabilistic simulation for true concurrency and devise a slightly different definition of a probabilistic bundle event structure.

5.1 Probabilistic bundle event structure

The key idea in adding probabilistic information to event structures is to use probability as a mechanism to resolve a conflict between events. However, not all conflicts can be resolved probabilistically [33]. The cases when this occurs are referred to as *confusions*.

Example 5.1.1. A typical example of confusion is given by three events e_1, e_2 and e_3 where $e_1 \# e_2$, $e_2 \# e_3$ and $\neg e_1 \# e_3$ hold, allowing e_1 and e_3 to occur simultaneously in a single run. If the conflict $e_1 \# e_2$ is resolved with a coin flip and if it yields e_2, then $e_2 \# e_3$ cannot be resolved probabilistically as it may produce e_3. In contrast, if e_1 happens then $e_2 \# e_3$ has to be resolved into e_3. ■

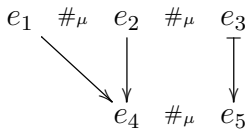
The goal of this section is to eliminate these confusions by introducing the notion of confusion-free bundle event structures and obtain the desired Definition 5.1.13 of probabilistic bundle event structures.

5.1.1 Immediate conflict

Following Varacca [85], we start by characterising conflicts that may be resolved probabilistically.

Definition 5.1.2. *Given a BES $\mathcal{E}$, two events $e, e' \in E$ are in immediate conflict if $e \# e'$ and there exists a configuration x such that $x \cup \{e\}$ and $x \cup \{e'\}$ are again configurations. We write $e \#_\mu e'$ when e and e' are in immediate conflict.*

Immediate conflict represents a conflict that has not been resolved by the execution history. When there is no confusion, these conflicts are obtained from nondeterministic constructs and ultimately from probabilistic choices.



In this BES, the bundles are $\{e_1, e_2\} \mapsto e_4$ and $\{e_3\} \mapsto e_5$. The conflict relation is $e_1 \# e_2$ and $e_2 \# e_3$. Therefore, e_1 and e_3 are concurrent. An arrow $\rightarrow$ represents some part of a bundle (i.e. $\{e_1, e_2\} \mapsto e_4$ is the completed bundle) whilst $\mapsto$ represents a bundle.

Figure 5.1: Immediate conflict in a BES.

Example 5.1.3. In Figure 5.1, e_4 and e_5 are in immediate conflict because $\{e_1, e_3, e_4\}$ and $\{e_1, e_3, e_5\}$ are configurations. In fact, every conflict in that BES is immediate. Notice that the conflict $e_4 \#_\mu e_5$ is resolved when e_2 occurs. ■

5.1.2 Clusters

Events can be grouped into clusters of events that are pairwise in immediate conflict. Moreover, if an event in a cluster can occur (i.e. all preceding event have happened) then every other event in that cluster may occur. More precisely, we define a cluster as follow.

Definition 5.1.4. *A partial cluster is a set of events $K \subseteq E$ satisfying*

$$\begin{aligned} \forall e, e' \in K : e \neq e' \;\; &\Rightarrow \;\; e \#_\mu e' \qquad \textit{and} \\ \forall e, e' \in K, x \subseteq E : x \mapsto e \;\; &\Rightarrow \;\; x \mapsto e' \end{aligned}$$

A cluster is a maximal partial cluster (with respect to set inclusion).

Given an event $e \in E$, the singleton $\{e\}$ is a partial cluster. Therefore, there is always at least one cluster (i.e. maximal) containing e and we write $\langle e \rangle$ the intersection of all clusters containing e.

Example 5.1.5. In Figure 5.1, $\{e_1, e_2\}$ and $\{e_2, e_3\}$ are clusters and $\langle e_2 \rangle = \{e_2\}$. ■

The following proposition characterises clusters:

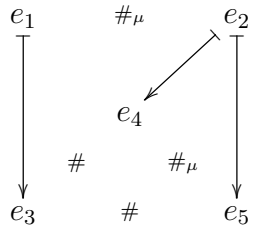


Figure 5.2: A BES where $\{e_1, e_2\}$, $\{e_3\}$ and $\{e_4, e_5\}$ are clusters.

Proposition 5.1.6. *A partial cluster K is maximal (i.e. a cluster) iff*

$$\forall e \in E : (\forall e' \in K : e \#_\mu e' \ \wedge \ \forall x \subseteq E : x \mapsto e \Leftrightarrow x \mapsto e') \Rightarrow e \in K$$

Proof. The forward implication follows from Definition 5.1.4 and maximality of K. Conversely, let us assume that K is a partial cluster satisfying the above property. Let H be a partial cluster such that $K \subseteq H$ and $e \in H$. Then, for all $e' \in K$, $e \#_\mu e'$ and

$$\forall x \subseteq E : x \mapsto e \Leftrightarrow x \mapsto e'$$

because H is a partial cluster. By the hypothesis, $e \in K$ and hence $H = K$. ■

Similar to Katoen's and Varacca's approaches, clusters are used to carry probability and they can be intuitively seen as providing a choice between events where the chosen event happens instantaneously. Our notion of cluster is weaker than Katoen's original definition [33]: the BES in Fig. 5.2 contains three clusters $\{e_1, e_2\}$, $\{e_3\}$ and $\{e_4, e_5\}$ and only $\{e_1, e_2\}$ satisfies Katoen's definition. This modification ensures that we can use clusters to partition the set of events in all *confusion-free bundle event structures* defined in the next section (Proposition 5.1.8).

5.1.3 Confusion-free bundle event structure

We are now ready to define confusion freeness on bundle event structures.

Definition 5.1.7. *A BES $\mathcal{E}$ is confusion free if for all events $e, e' \in E$,*

- *if* $e\#_\mu e'$ *then* $e \in \langle e' \rangle$, *and*
- *if* $\langle e \rangle \cap x = \emptyset$ *and* $x \cup \{e\} \in \mathcal{C}(\mathcal{E})$ *for some configuration* $x \in \mathcal{C}(\mathcal{E})$, *then* $x \cup \{e''\} \in \mathcal{C}(\mathcal{E})$ *for all events* $e'' \in \langle e \rangle$.

The first property implies that $\langle e \rangle$ contains all events in immediate conflict with e and hence the confusion introduced by e_1, e_2 and e_3 in Fig. 5.1 is avoided. The second property says that all events in $\langle e \rangle$ are enabled at the same time. Hence, confusion freeness ensures that all conflicts in $\langle e \rangle$ can be resolved probabilistically regardless of the execution history.

The proof of the following proposition is the same as for prime event structures [85].

Proposition 5.1.8. *For a confusion free BES* $\mathcal{E}$, *the set* $\{\langle e \rangle \mid e \in E\}$ *defines a partition of* E. *That is, the reflexive closure of* $\#_\mu$ *is an equivalence relation and the equivalence classes are of the form* $\langle e \rangle$.

The second property of Definition 5.1.7 is usually hard to check. We give a static and simpler sufficient condition for confusion freeness.

Proposition 5.1.9. *If a BES* $\mathcal{E}$ *satisfies*

$$\forall e, e' \in E : (e\#_\mu e' \Rightarrow e \in \langle e' \rangle) \wedge (\langle e \rangle \cap \underline{\mathbf{cfl}}(e') \neq \emptyset \Rightarrow \langle e \rangle \subseteq \underline{\mathbf{cfl}}(e')),$$

then it is confusion free.

Proof. Let $e \in E$ and $x \in \mathcal{C}(\mathcal{E})$ such that $\langle e \rangle \cap x = \emptyset$ and $x \cup \{e\} \in \mathcal{C}(\mathcal{E})$. We need to show that $x \cup \{e'\} \in \mathcal{C}(\mathcal{E})$ for every $e' \in \langle e \rangle$.

Let $e' \in \langle e \rangle$ and $z \mapsto e'$ be a bundle of $\mathcal{E}$. By Definition 5.1.4, $z \mapsto e$ is also a bundle and since x and $x \cup \{e\}$ are configurations, $e_1 \cdots e_n e$ is again a linearisation of $x \cup \{e\}$ for every linearisation $e_1 \cdots e_n$ of x. Therefore, $z \cap \{e_1, \ldots, e_n\} \neq \emptyset$. If $e' \in \underline{\mathbf{cfl}}(e_i)$ for some i, then $\langle e \rangle \subseteq \underline{\mathbf{cfl}}(e_i)$ by the hypothesis and hence $e \in \underline{\mathbf{cfl}}(e_i)$, which is impossible because $e_1 \cdots e_n e$ is an event trace. Hence $e_1 \cdots e_n e'$ is also an event trace, that is, $x \cup \{e'\} \in \mathcal{C}(\mathcal{E})$. ■

The second argument of the conjunction says that if some event in $\langle e \rangle$ is in conflict with an event e', then all events in $\langle e \rangle$ are in conflict with e'.

Example 5.1.10. Figure 5.2 depicts a confusion free BES that satisfies Proposition 5.1.9. The partition generated by $\#_\mu$ contains $\langle e_1 \rangle = \langle e_2 \rangle$, $\langle e_4 \rangle = \langle e_5 \rangle$ and $\langle e_3 \rangle$. We can see from Figure 5.2 that $\langle e_4 \rangle \cap \underline{\mathbf{cfl}}(e_3) = \{e_3\}$ and $\langle e_4 \rangle = \{e_4, e_5\} \subseteq \underline{\mathbf{cfl}}(e_3)$. ■

A more interesting application of Proposition 5.1.9 is to prove that any *regular* bundle event structure is confusion free. A bundle event structure $\mathcal{E}$ is *regular* if it is inductively constructed from the basic constructs 0, 1 and the event structure associated to each $a \in \Sigma$, with the operations of Section 3.3.

Lemma 5.1.11. *If $\mathcal{E}$ is a regular bundle event structure with set of events E then the properties*

$$\forall e \in E : e \in \mathbf{in}(\mathcal{E}) \Rightarrow \langle e \rangle = \mathbf{in}(\mathcal{E}) \tag{5.1}$$

and

$$\forall e \in E : \underline{\mathbf{cfl}}(e) \cap \mathbf{in}(\mathcal{E}) \neq \emptyset \Rightarrow e \in \mathbf{in}(\mathcal{E}) \tag{5.2}$$

hold. The same result holds if we replace $\mathbf{in}(\mathcal{E})$ with $\mathbf{\Phi}_{\mathcal{E}}$.

Proof. These two properties are clear by induction on the structure of $\mathcal{E}$ and the definition of $\mathbf{in}(\mathcal{E})$. ■

Proposition 5.1.12. *A regular bundle event structure is confusion free.*

Proof. The result follows by induction on the structure of $\mathcal{E}$. We show that the property

$$\forall e, e' \in E : \langle e \rangle \cap \underline{\mathbf{cfl}}(e') \neq \emptyset \Rightarrow \langle e \rangle \subseteq \underline{\mathbf{cfl}}(e') \tag{5.3}$$

is preserved by the operations $(+, \cdot, \|)$ and $(*)$. The base cases are clear. Let $\mathcal{E}, \mathcal{F}$ be two regular bundle event structures satisfying the Property 5.3. Let e, e' be two events from $E \cup F$ (the set of events of $\mathcal{E} + \mathcal{F}$) such that $\langle e \rangle \cap \underline{\mathbf{cfl}}(e') \neq \emptyset$ holds in $\mathcal{E} + \mathcal{F}$. Let us assume that $e \in E$, then there are three cases:

- If $e \in \mathbf{in}(\mathcal{E})$, then $\langle e \rangle = \mathbf{in}(\mathcal{E} + \mathcal{F})$ by Property 5.1 because $\mathcal{E} + \mathcal{F}$ is regular. By Property 5.2, we have $\underline{\mathbf{cfl}}(e') \cap \mathbf{in}(\mathcal{E} + \mathcal{F}) \neq \emptyset$. Hence, $e' \in \mathbf{in}(\mathcal{E} + \mathcal{F})$ and we are done because $\langle e \rangle = \underline{\mathbf{cfl}}(e')$.

- Similarly for $e \in \mathbf{\Phi}_{\mathcal{E}}$.

- If $e \in E \setminus (\mathbf{in}(\mathcal{E}) \setminus \mathbf{\Phi}(\mathcal{E})$, then $\langle e \rangle \subseteq E$ (by definition of the conflict relation $\#_{\mathcal{E}+\mathcal{F}}$). Thus, $e' \in E$ and the result follows directly from the induction hypothesis on $\mathcal{E}$.

The cases of the operations $(\cdot, \|, *)$ are proven in the similar ways. ■

5.1.4 Probabilistic bundle event structure

With confusion freeness, we are now able to define probability distributions supported by clusters. Given an event structure $\mathcal{E} = (E, \#, \mapsto, \lambda, \Phi)$, we say that $\mu \in \mathbb{D}E$ is a *probability distribution on* $\mathcal{E}$ if $\mathrm{supp}(\mu) \subseteq \langle e \rangle$ for some event $e \in E$. That is, if μ is supported in $\langle e \rangle$, then it is used to resolve probabilistically the immediate conflict between events in $\langle e \rangle$.

Definition 5.1.13. *A probabilistic BES is a tuple $(\mathcal{E}, \pi)$ where $\mathcal{E}$ is a confusion free BES and π is a set of probability distributions on $\mathcal{E}$ such that for every $e \in E$, there exists $\mu \in \pi$ such that $e \in \mathrm{supp}(\mu)$.*

We write **pBES** for the collection of all probabilistic bundle event structures that are constructed from a given countable set of actions Σ (this set of actions is usually left implicit).

The intuition behind this definition is simple: if there is no $\mu \in \pi$ such that $e \in \mathrm{supp}(\mu)$, then e is an impossible event and it can be removed (this may affect any event e' such that $e \preceq_x e'$ for some $x \in \mathcal{C}(\mathcal{E})$). The set of configurations of $(\mathcal{E}, \pi)$ is defined to be $\mathcal{C}(\mathcal{E})$.

Our approach differs from both Varacca's [85] and Katoen's [33] in that nondeterminism is modelled concretely as a set of probabilistic choices. This is a

usual concept in sequential probabilistic programs where nondeterminism is introduced from the fact that we have partial information about the distribution (rather than the state) [48]. This approach will mainly contribute to the definition of the probabilistic choice operation $\oplus_p$, $p \in [0,1]$. For instance, while the expression $a + (b \oplus_p c)$ does not have any meaning in Katoen's probabilistic bundle event structure, we will see that it has a precise semantics in our case.

5.2 Probabilistic simulation on pBES

The weakest interpretation of a refinement order on pBES is Varacca's configuration distributions inclusion [85]. However, as in the interleaving case, that order is not a congruence with respect to the concurrency operation $\|$ [73], that is, it is possible to construct three event structures $\mathcal{E}, \mathcal{F}$ and $\mathcal{G}$ such that $\mathcal{E}$ is a refinement of $\mathcal{F}$ but $\mathcal{E}\|\mathcal{G}$ is not a refinement of $\mathcal{F}\|\mathcal{G}$. We then use probabilistic simulations which are based on the notion of lifting given in Definition 4.3.1.

We start by defining the analogue of transition applied to pBES.

5.2.1 Prefixing on Distributions over Configurations

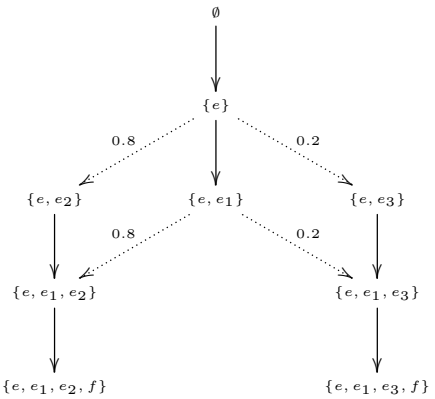
Since the notion of configuration for a pBES $(\mathcal{E}, \pi)$ is independent of π, we keep the notation $\mathcal{C}(\mathcal{E})$ for the set of all finite configurations. An example of relation on $\mathcal{C}(\mathcal{E}) \times \mathbb{DC}(\mathcal{E})$ is given by the probabilistic prefixing which is an extension of Chapter 3 Definition 3.4.1.

Definition 5.2.1. *We say that $x \in \mathcal{C}(\mathcal{E})$ is a prefix of $\Delta \in \mathbb{DC}(\mathcal{E})$, denoted (again) by $x \trianglelefteq \Delta$, if there exists $\mu \in \pi$ such that* $\mathrm{supp}(\mu) \cap x = \emptyset$ *and* $\Delta = \sum_{e \in \mathrm{supp}(\mu)} (\mu(e)) \delta_{x \cup \{e\}}$.

In particular, if $\langle e \rangle = \{e\}$, $e \notin x$ and $x \cup \{e\} \in \mathcal{C}(\mathcal{E})$ then $x \trianglelefteq \delta_{x \cup \{e\}}$.

The relation $\trianglelefteq$ is lifted to $\overline{\trianglelefteq} \subseteq \mathbb{DC}(\mathcal{E}) \times \mathbb{DC}(\mathcal{E})$ and the reflexive transitive closure of the lifted relation is denoted by $\overline{\trianglelefteq}^*$. Probabilistic prefixing allows us to

construct a *configuration-tree* for every pBES. An example is depicted in Fig. 5.3.



The dotted arrows with common source are parts of a probabilistic prefix relation (e.g. $\{e\} \trianglelefteq 0.8\delta_{\{e,e_2\}} + 0.2\delta_{\{e,e_3\}}$). The events e, f are the delimiters introduced by $\|$.

Figure 5.3: The configurations-tree of the pBES $e_1 \| (e_2 \oplus_{0.2} e_3)$ ($\oplus_{0.2}$ is defined later).

5.2.2 Probabilistic Simulation on pBES

To simplify the presentation, we restrict ourselves to bundle event structures satisfying

$$\forall x \subseteq E : (\exists e \in E : x \mapsto e) \Rightarrow \mathbf{\Phi} \cap x = \emptyset \tag{5.4}$$

In other words, no event is enabled by a final event. This allows a simpler specification of the preservation of final events by a simulation. Notice that all regular bundle event structures satisfy the property 5.4.

Recall from Chapter 3 Section 3.2 that configurations (endowed with the canonical order) are compared using the subsumption order $\sqsubseteq_s$.

Definition 5.2.2. *A (probabilistic) simulation from* $(\mathcal{E}, \pi)$ *to* $(\mathcal{F}, \rho)$ *is a relation* $S \subseteq \mathcal{C}(\mathcal{E}) \times \mathbb{D}\mathcal{C}(\mathcal{F})$ *such that:*

- $(\emptyset, \delta_\emptyset) \in S$,

- *if* $(x, \Theta) \in S$ *then for every* $y \in \mathrm{supp}(\Theta)$, $x \sqsubseteq_s y$,
- *if* $(x, \Theta) \in S$ *and* $x \trianglelefteq \Delta'$ *then there exists* $\Theta' \in \mathbb{D}\mathcal{C}(\mathcal{F})$ *such that* $\Theta \overline{\trianglelefteq}^* \Theta'$ *and* $(\Delta', \Theta') \in \overline{S}$.
- *if* $(x, \Theta) \in S$ *and* $x \cap \mathbf{\Phi}_{\mathcal{E}} \neq \emptyset$ *then for every* $y \in \mathrm{supp}(\Theta)$ *we have* $y \cap \mathbf{\Phi}_{\mathcal{F}} \neq \emptyset$.

We write $(\mathcal{E}, \pi) \sqsubseteq_{p\mathrm{sim}} (\mathcal{F}, \rho)$ *if there is a simulation from* $(\mathcal{E}, \pi)$ *to* $(\mathcal{F}, \rho)$.

Indeed, Definition 5.2.2 is akin to probabilistic forward simulation on automata (Definition 4.3.3). The main difference is the use of the order $x \sqsubseteq_s y$ which holds iff there exists a label preserving monotonic bijection from $(\hat{y}, \preceq_y, \lambda_y)$ to $(\hat{x}, \preceq_x, \lambda_x)$.

Another consequence of this definition is that concurrent events can be linearised while preserving simulation.

Proposition 5.2.3. $\sqsubseteq_{p\mathrm{sim}}$ *is a preorder.*

The proof is the same as in [15], hence, we provide only a sketch.

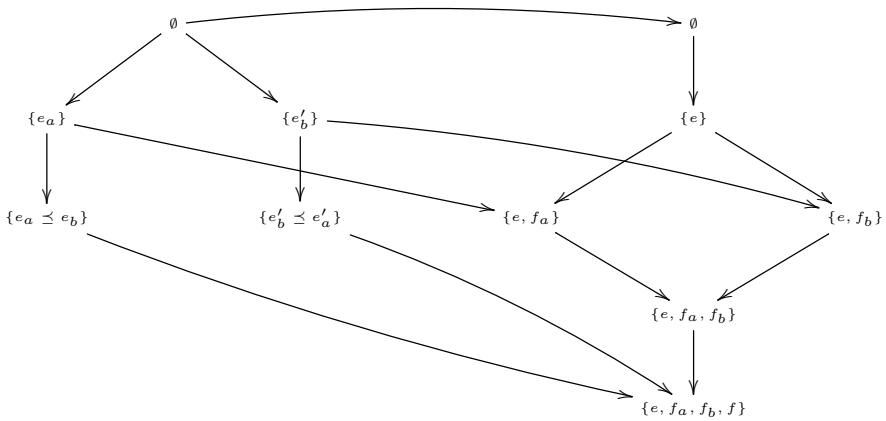
Proof. Reflexivity is clear by considering the relation $\{(x, \delta_x) \mid x \in \mathcal{C}(\mathcal{E})\}$, which is indeed a simulation. If R, S are probabilistic simulations from $(\mathcal{E}, \pi)$ to $(\mathcal{F}, \rho)$ and $(\mathcal{F}, \rho)$ to $(\mathcal{G}, r)$ respectively, then we can show, using Proposition 4.3.2 and a similar proof as in the interleaving case, that $R \circ \overline{S}$ is a probabilistic simulation from $(\mathcal{E}, \pi)$ to $(\mathcal{G}, r)$. ■

The following example shows that our notion of simulation differentiates between interleaving and true concurrency.

Example 5.2.4. In Figure 5.4, it is shown that $a \cdot b + b \cdot a \sqsubseteq_{p\mathrm{sim}} a \| b$, but the converse does not hold. ■

5.3 True concurrent interpretation of pCKA

In this section, we show that the set **pBES** endowed with $(+, \cdot, \|, *, 0, 1)$ satisfies the axioms of a probabilistic concurrent Kleene algebra.



Since $\{e, f_a, f_b, f\} \not\sqsubseteq_s \{e_a \preceq e_b\}$ nor $\{e, f_a, f_b, f\} \not\sqsubseteq_s \{e'_b \preceq e'_a\}$, it is impossible to find a simulation from $a\|b$ to $a \cdot b + b \cdot a$. In the configuration tree on the left, the order $\preceq$ is made explicit and primes are introduced for disjointness.

Figure 5.4: A simulation from $a \cdot b + b \cdot a$ to $a\|b$.

5.3.1 Operations on probabilistic bundle event structures

We start by defining and extending the constants and operations defined on bundle event structures.

We generate the probabilistic bundle event structures $(0, \emptyset)$, $(1, \{\delta_e\})$ and $(a, \{\delta_{e_a}\})$ from the basic bundle event structures. To simplify the notations, these basic pBES are again denoted by $0, 1$ and a. The other standard operations are extended as follows:

$$\begin{aligned}(\mathcal{E}, \pi) + (\mathcal{F}, \rho) &= (\mathcal{E} + \mathcal{F}, \pi \cup \rho)\\ (\mathcal{E}, \pi) \cdot (\mathcal{F}, \rho) &= (\mathcal{E} \cdot \mathcal{F}, \pi \cup \rho)\\ (\mathcal{E}, \pi) \| (\mathcal{F}, \rho) &= (\mathcal{E} \| \mathcal{F}, \pi \cup \rho \cup \{\delta_e, \delta_f\})\end{aligned}$$

where e and f are the fresh events delimiting $\mathcal{E}\|\mathcal{F}$. Recall that $\mathcal{E}$ and $\mathcal{F}$ are assumed to be disjoint in these definitions. The probabilistic choice that chooses $\mathcal{E}$ with probability $1 - p$ and $\mathcal{F}$ with probability p is

$$(\mathcal{E}, \pi) \oplus_p (\mathcal{F}, \rho) = (\mathcal{E} + \mathcal{F}, \pi \oplus_p \rho)$$

where $\mu \in \pi \oplus_p \rho$ iff:

- if $\mathrm{supp}(\mu) \subseteq \mathbf{in}(\mathcal{E}) \cup \mathbf{in}(\mathcal{F})$ then $\mu = (1-p)\mu_0 + p\nu_0$ for some $\mu_0 \in \pi$ and $\nu_0 \in \rho$,
- else $\mu \in \pi \cup \rho$.

Intuitively, nondeterminism is resolved first by choosing a probability distribution, then a probabilistic choice is resolved based on that distribution. Indeed, the nondeterministic and probabilisic choices introduce clusters.

Example 5.3.1. The bundle event structure $a \| (b \oplus_{0.2} c)$ contains four clusters $\langle e \rangle, \langle e_b, e_c \rangle, \langle e_a \rangle$ and $\langle f \rangle$ where e, f are the delimiter events. It has a set of probability distributions $\{0.8\delta_{e_b} + 0.2\delta_{e_c}, \delta_{e_a}, \delta_e, \delta_f\}$. In contrast, the event structure $a + (b \oplus_{0.2} c)$ has a single cluster $\langle e_a, e_b, e_c \rangle$ with a set of probability distributions $\{0.8\delta_{e_b} + 0.2\delta_{e_c}, \delta_{e_a}\}$. ■

To construct the binary Kleene star, we need the following partial order:

$$(\mathcal{E}, \pi) \lhd (\mathcal{F}, \rho) \qquad \text{iff} \qquad \mathcal{E} \lhd \mathcal{F} \wedge \pi = \{p \in \rho \mid \mathrm{supp}(p) \subseteq E\}.$$

The proof that $\lhd$ is indeed ω-complete is essentially the same as in the standard case (Chapter 3 Section 3.3). Hence the Kleene product $(\mathcal{E}, \pi) * (\mathcal{F}, \rho)$ is again the limit of the increasing sequence of probabilistic bundle event structures:

$$(\mathcal{F}, \rho) \lhd (\mathcal{F}, \rho) + (\mathcal{E}, \pi) \cdot (\mathcal{F}, \rho) \lhd (\mathcal{F}, \rho) + (\mathcal{E}, \pi) \cdot ((\mathcal{F}, \rho) + (\mathcal{E}, \pi)) \lhd \cdots .$$

More precisely, $(\mathcal{E}, \pi) * (\mathcal{F}, \rho) = (\mathcal{E} * \mathcal{F}, \pi * \rho)$ where $\pi * \rho = \cup_i \pi *_{\leq i} \rho$ and each set $\pi *_{\leq i} \rho$ is obtained from the construction of $\mathcal{E} *_{\leq i} \mathcal{F}$.

5.3.2 Proof of soundness for $(\mathbf{pBES}, \sqsubseteq_{p\mathrm{sim}})$

We start by ensuring that $\sqsubseteq_{p\mathrm{sim}}$ is indeed a precongruence with respect to all the algebraic operations on **pBES**. We write $\equiv_{p\mathrm{sim}}$ the equivalence relation associated to $\sqsubseteq_{p\mathrm{sim}}$.

Proposition 5.3.2. *The order $\sqsubseteq_{p\text{sim}}$ is a precongruence, i.e., for every pBES $(\mathcal{E},\pi),(\mathcal{F},\rho)$ and $(\mathcal{G},\eta)$, if $(\mathcal{E},\pi) \sqsubseteq_{p\text{sim}} (\mathcal{F},\rho)$ then $(\mathcal{E},\pi)\circ(\mathcal{G},\eta) \sqsubseteq_{p\text{sim}} (\mathcal{F},\rho)\circ(\mathcal{G},\eta)$ (and symmetrically) for every $\circ \in \{+,\cdot,\|,*\}$.*

Proof. Let $(\mathcal{E},\pi) \sqsubseteq_{p\text{sim}} (\mathcal{F},\rho)$ be witnessed by a simulation $S \subseteq \mathcal{C}(\mathcal{E}) \times \mathbb{DC}(\mathcal{F})$ and $(\mathcal{G},\eta)$ be any pBES. The congruence properties are proven by extending the simulation S to the events of $\mathcal{G}$. For instance, that $(\mathcal{E},\pi)+(\mathcal{G},\eta) \sqsubseteq_{p\text{sim}} (\mathcal{F},\rho)+(\mathcal{G},\eta)$ is deduced by showing that $S \cup \{(x,\delta_x) \mid x \in \mathcal{C}(\mathcal{G})\}$ is indeed a simulation and similarly for the other operations. ■

The axioms (4.1-4.10) referring to the left semiring property and the standard properties of $(\|)$ such as the interchange law (4.15), the commutativity (4.13), the associativity (4.14) and the subdistributivity (4.16) are proven using simulations akin to the interleaving case. The same is achieved for the properties of $\oplus_p$ (axioms 4.17-4.22) [51]. The proof similarities are due to the fact that the extra change introduced in Definition 5.2.2 is the property $x \sqsubseteq_s y$ for every $(x,\Theta) \in S$ and $y \in \Theta$, where S is a simulation. Hence, it suffices to check only that property for each of the cases, and this usually is a direct consequence of the construction. Moreover, the definition of the operations $(+,\cdot,\oplus_p)$ and $(*)$ are conceptually the same.

The existence of simulations that establishes that 1 is a unit for $\|$ (Axiom 3.16) is clear from the definitions of $\|$ and 1. It follows from the axioms of $+$ and Proposition 5.3.2 that $(\mathcal{E},\pi) \sqsubseteq_{p\text{sim}} (\mathcal{F},\rho)$ iff $(\mathcal{E},\pi)+(\mathcal{F},\rho) \equiv_{p\text{sim}} (\mathcal{F},\rho)$.

Proposition 5.3.3. *The binary Kleene star satisfies the fixed point equations:*

$$\mathcal{F} + \mathcal{E}\cdot(\mathcal{E}*\mathcal{F}) \quad \equiv_{p\text{sim}} \quad (\mathcal{E}*\mathcal{F}), \tag{5.5}$$

$$\mathcal{G} + \mathcal{E}\cdot\mathcal{F} \sqsubseteq_{p\text{sim}} \mathcal{F} \quad \Rightarrow \quad \mathcal{E}*\mathcal{G} \sqsubseteq_{p\text{sim}} \mathcal{F} \tag{5.6}$$

Proof. The first equation is proven using the simulation construction of Chapter 4 Theorem 4.4.2. For the second implication, let $S \subseteq \mathcal{C}(\mathcal{E}\cdot\mathcal{F}) \times \mathbb{DC}(\mathcal{F})$ be a probabilistic simulation from $\mathcal{G}+(\mathcal{E},\pi)\cdot(\mathcal{F},\rho)$ to $(\mathcal{F},\pi)$. By monotonicity of $\cdot$

and $+$, there exists a simulation $S^{(i)} \subseteq \mathcal{C}(\mathcal{E} *_{\leq i} \mathcal{G}) \times \mathbb{DC}(\mathcal{F})$ from $(\mathcal{E}, \pi) *_{\leq i} (\mathcal{G}, \rho)$ to $(\mathcal{F}, \rho)$, for every $i \in \mathbb{N}$. Moreover, we can find a family of simulations such that $S^{(i-1)}$ is the restriction of $S^{(i)}$ to $(\mathcal{E}, \pi) *_{\leq i-1} (\mathcal{G}, \rho)$. Thus, we can consider the union $S = \cup_i S^{(i)}$ and show that it is indeed a simulation from $(\mathcal{E}, \pi) * (\mathcal{G}, \rho)$ to $(\mathcal{F}, \rho)$. Hence, Equation (5.6) holds. ■

These results are then summarised in the following soundness theorem.

Theorem 5.3.4. *The structure* $(\mathbf{pBES}, +, \cdot, *, \|, 0, 1)$ *modulo probabilistic simulation forms a probabilistic concurrent Kleene algebra with a binary Kleene star.*

Notice that this theorem includes the properties of the binary Kleene star rather than the unary Kleene star for probabilistic concurrent Kleene Algebra. The unfold Axiom 4.11 follows immediately from the definition $\mathcal{E}^* = \mathcal{E} * 1$ and substituting $\mathcal{F}$ with 1 in 5.5. As for the induction Axiom 4.12, if $\mathcal{E} \cdot \mathcal{F} \sqsubseteq_{p\text{sim}} \mathcal{F}$, then $\mathcal{F} + \mathcal{E} \cdot \mathcal{F} \sqsubseteq_{p\text{sim}} \mathcal{F}$. Therefore, $\mathcal{E} * \mathcal{F} \sqsubseteq_{p\text{sim}} \mathcal{F}$. A simulation from $(\mathcal{E} * 1) \cdot \mathcal{F}$ to $\mathcal{E} * \mathcal{F}$ can be constructed because the expression 1 will introduce unlabelled events, which are removed when comparing configurations with $\sqsubseteq_s$. A configuration x of $(\mathcal{E} * 1) \cdot \mathcal{F}$ is then implemented by the corresponding configuration of $\mathcal{F}$ (i.e. removing the event introduced by 1 from x). Moreover, the probabilistic prefixing is preserved. Hence, we have $(\mathcal{E} * 1) \cdot \mathcal{F} \sqsubseteq_{p\text{sim}} (\mathcal{E} * \mathcal{F}) \sqsubseteq_{p\text{sim}} \mathcal{F}$. [1]

5.4 Discussion

This chapter shows that our proposed axiomatisation of probabilistic concurrent Kleene algebra is sound with respect to a true concurrent interpretation. That is, the axiom systems presented in Chapter 4 Figure 4.1 and Figure 4.2 can be used to verify probabilistic systems having truly concurrent behaviours. To obtain a congruence, we have redefined the notion of probabilistic simulation which can be seen as a probabilistic extension of Ferroudja's construction [7] on a variation of Katoen's probabilistic bundle event structure. Simulation is used

[1] A better proof is obtained by applying the fixed point fusion theorem [1]. However, we do not need to construct the Galois connection needed for that theorem to show this particular case.

because the configuration distribution equivalence is not a congruence as in the case of trace distribution [73]. However, simulation is usually a very strong form of refinement order and usually unsuitable for systems modelled primarily using relations or predicates transformers. Therefore, we introduce the notion of event structure with implicit probability where both sequential and simulation orders are defined. Such a model will be used to provide a suitable denotational semantics for probabilistic rely/guarantee calculus.

Chapter 6

Bundle Event Structure with Implicit Probability

In this section, we provide an alternative definition of probabilistic bundle event structures by assuming implicit probabilities (ipBES). That is, on a higher level, the event structure is a standard bundle event structure where each event is labelled with a probabilistic program on a finite state space.

The set of bundle event structures with implicit probabilities is essential to the development of probabilistic rely/guarantee calculus in the style of [31]. These structures are studied modulo two forms of equivalence: a simulation (qualitative) order which is a particular case of Definition 5.2.2, and a sequential (quantitative) order based on inclusion of the set of distributions.

6.1 Sequential probabilistic programs

In this section, we give a brief summary of the denotation of sequential probabilistic programs using the powerdomain construction of McIver and Morgan [48]. All probabilistic programs will be considered to have a finite state space denoted by Ω. A distribution over the set Ω is a function $\mu : \Omega \to [0, 1]$ such that

$\sum_{s\in\Omega}\mu(s) = 1$. The set of distributions over Ω is denoted by $\mathbb{D}\Omega$. Since Ω is a finite set, we will identify a distribution with the associated measure. For every $\mu \in \mathbb{D}\Omega$ and $O \subseteq \Omega$, we write $\mu(O) = \sum_{s\in O}\mu(s)$. An important example of distribution is the point mass distribution δ_s, centred at the state $s \in \Omega$, such that

$$\delta_s(s') = \begin{cases} 1 & \text{if } s = s', \\ 0 & \text{otherwise.} \end{cases}$$

A (nondeterministic) probabilistic program r is denoted by a map of type $\Omega \to \mathbb{P}\mathbb{D}\Omega$ such that $r(s)$ is a non-empty, topologically closed and convex subset of $\mathbb{D}\Omega$ for every $s \in \Omega$. The set $\mathbb{D}\Omega$ is a topological sub-space of the finite product $\mathbb{R}^\Omega$ (endowed with the usual product topology), and the topological closure is considered with respect to the induced topology on Ω. [1] We denote by $\mathbb{H}_1\Omega$ the set of probabilistic programs. Notice that the set $\mathbb{D}\Omega$ contains only distributions instead of the subdistributions considered by McIver and Morgan [48]. Therefore, we can only model nondeterministic programs that are terminating with probability 1.

Programs in $\mathbb{H}_1\Omega$ are ordered by pointwise inclusion, i.e., $r \sqsubseteq_{\mathbb{H}} r'$ if for every $s \in \Omega$, $r(s) \subseteq r'(s)$. A program r is deterministic if, for every s, $r(s) = \{\mu_s\}$ (i.e. a singleton) for some distribution $\mu_s \in \mathbb{D}\Omega$. The set of deterministic programs is denoted by $\mathbb{J}_1\Omega$ (as in Jones' spaces [30]). If $f \in \mathbb{J}_1\Omega$ is a deterministic program such that $f(s) = \{\mu_s\}$, then we usually just write $f(s) = \mu_s$.

Example 6.1.1. Let us consider the assignment $x := h \oplus_{0.5} t$ which assigns the value h to x with probability 0.5. The value t is assigned to x with the same probability. The state space is defined by the set $\Omega = \{h, t\}$ containing all possible values for x. This assignment is denoted by a deterministic program r such that $r(h) = 0.5\delta_h + 0.5\delta_t$. ■

The probabilistic combination of two probabilistic programs r and r' is defined

[1] These healthiness conditions are set out and fully explained in the work of McIver and Morgan [48]. In Example 6.1.3, we show another motivation behind the topological closure.

as ([48] Definition 5.4.5)

$$(r \oplus_p r')(s) = \{\mu \oplus_p \mu' \mid \mu \in r(s) \wedge \mu' \in r'(s)\}, \tag{6.1}$$

where $(\mu \oplus_p \mu')(s) = p\mu(s) + (1-p)\mu'(s)$ for every state $s \in \Omega$.

Nondeterminism is obtained as the set of all probabilistic choices ([48] Definition 5.4.6). That is,

$$(r + r')(s) = \cup_{p\in[0,1]}(r \oplus_p r')(s). \tag{6.2}$$

The sequential composition of r by r' is defined as ([48] Definition 5.4.7):

$$r \cdot r'(s) = \{\, f \star \mu |\, f \sqsubseteq_{\mathbb{H}} r' \wedge f \in \mathbb{J}_1\Omega \wedge \mu \in r(s)\} \tag{6.3}$$

where

$$(f \star \mu)(s') = \sum_{s''\in\Omega} f(s'')(s')\mu(s'')$$

for every state $s' \in \Omega$.

A particularly important example of probabilistic program is the ineffectual program `skip`, which we denote by δ. That is, $\delta(s) = \{\delta_s\}$.

Example 6.1.2. We reconsider the program r associated to the probabilistic assignment $x := h \oplus_{0.5} t$ of Example 6.1.1. A deterministic refinement f of $\delta + r$ is characterised by choosing a weight function $w : \{h, t\} \to [0, 1]$ such that $f(s) = w(s)\delta_h + (1 - w(s))(0.5\delta_h + 0.5\delta_t)$ for every state $s \in \Omega$.

A distribution $\nu \in [r \cdot (\delta + r)](h)$ satisfies

$$\nu(s) = (f \star [0.5\delta_h + 0.5\delta_t])(h)(s) = 0.5f(h)(s) + 0.5f(t)(s)$$

where $f \sqsubseteq_{\mathbb{H}} 1 + r$. We have $f(h)(h) = w(h) + 0.5(1 - w(h)) = 0.5(1 + w(h))$ and $f(t)(h) = 0.5(1 - w(t))$. Therefore, the quantity $\nu(h)$ attains its maximal value with the weight function $w(h) = 1$ and $w(t) = 0$. Intuitively, if the output of the first run of r is t, then we execute the second r. This gives a maximal probability of $\nu(h) = 0.5 + 0.5^2$ as expected.

Dually, that weight function will minimise the probability of getting t to the lowest bound $1 - 0.5 - 0.5^2 = 0.5^2$. ■

The algebraic constant 1 is defined to be δ. We define the constant $\bot$ such that $r \cdot \bot = \bot \cdot r = \bot$, $\bot + r = r$ and $\bot \sqsubseteq_{\mathbb{H}} r$ for every $r \in \mathbb{H}_1\Omega \cup \{\bot\}$.

Let $r \in \mathbb{H}_1\Omega$, the binary Kleene star $r * r'$ is again the least fixed point of the function $f_{r,r'}(X) = r' + r \cdot X$ in $\mathbb{H}_1\Omega$. The function $r' \mapsto r \cdot r'$ is continuous —it preserves directed suprema (c.f. Section 2.2.4)— because of the bounded continuity of the associated expectation transformer ([48] Lemma 5.6.6). Notice that a topological closure is sometimes needed to ensure that we obtain an element of $\mathbb{H}_1\Omega$ (c.f. Example 6.1.3). Hence, the Kleene star $r * r'$ is the program such that

$$r * r'(s) = \overline{\cup_n f^n_{r,r'}(\bot)(s)}$$

where $\overline{A}$ is the topological closure of the set $A \subseteq \mathbb{D}\Omega$. The unary Kleene star is defined as $r^* = r * 1$. Notice that $\bot * \delta = \delta$ follows from $\bot \cdot \bot * \delta = \bot$ and $\delta + \bot = \delta$. When endowed with these operations and constants, the set $\mathbb{H}_1(\Omega)$ forms a model of probabilistic Kleene algebra [53].

Example 6.1.3. We use again the program r denoting the probabilistic assignment $x := h \oplus_{0.5} t$. From the reasoning in Example 6.1.2, the set $[r \cdot (\delta + r)](t)$ is formed of probability distribution μ such that $0.5 \geq \mu(t) \geq 0.5^2$. Therefore, $[\delta + r \cdot (\delta + r)](t) = \{p\delta_t + (1-p)\mu_2 \mid p \in [0,1]\}$ where $\mu_2 = (0.5 + 0.5^2)\delta_h + 0.5^2\delta_t$. By reapplying the reasoning in Example 6.1.2 inductively, we can show that

$$[\underbrace{\delta + r \cdot (\delta + r \cdot \cdots \cdot (\delta + r))}_{r \text{ occurs } n \text{ times}}](s) = \{p\delta_t + (1-p)\mu_n \mid p \in [0,1]\}$$

where $\mu_n = (1 - 0.5^n)\delta_h + 0.5^n\delta_t$. Notice that the union $\cup_n\{p\delta_t + (1-p)\mu_n \mid p \in [0,1]\}$ does not contain the distribution δ_h. Therefore, a topological closure is necessary to obtain the expected equality $r^* = \{p\delta_t + (1-p)\delta_h \mid p \in [0,1]\}$, which states that r^* indeed behaves in the same way as the nondeterministic choice between t and h. In other words, the topological closure ensures that we

do not differentiate between a program that terminates with probability 1 (such as the assignment $x := h$) and a program that terminates with a probability that "converges" to 1 (such as the program $x := t; \mathbf{while}(x \neq h)\{x := h \oplus_{0.5} t\})$. ■

We introduce tests, that are used for conditional constructs, following the idea adopted in algebras. We define a test to be a map $b : \Omega \to \mathbb{PD}\Omega$ such that $b(s) \subseteq \{\delta_s\}$. Indeed, an "if statement" is modelled algebraically as $b \cdot r + (\neg b) \cdot r'$. The sub-expression $b \cdot r(s)$ still evaluates to $\emptyset$ if $b(s)$ is empty, but care should be taken to avoid expressions such as $r \cdot b$ (if f is a deterministic refinement of b, then $f(s'')(s')$ may have no meaning if $b(s'') = \emptyset$).

We denote by $\overline{\mathbb{H}}_1\Omega$ the set of tests together with the set of probabilistic programs. The refinement order $\sqsubseteq_{\mathbb{H}}$ is extended to $\overline{\mathbb{H}}_1\Omega$ in a straightforward manner. For every test b, we have $b \sqsubseteq_{\mathbb{H}} \delta$, hence, we refer to tests as *subidentities*. We refer to the elements of $\overline{\mathbb{H}}_1\Omega$ as programs, unless otherwise specified.

Definition 6.1.4. *A structure $\mathcal{E} = (E, \mapsto, \#, \lambda, \mathbf{\Phi})$ is a bundle event structure with implicit probability (i.e. an ipBES) if*

- *$\lambda : E \to \overline{\mathbb{H}}_1\Omega$, i.e. λ labels event with (atomic) probabilistic programs.*

- *$\mathbf{\Phi} \subseteq \mathbb{P}\Omega$ such that, for every $x \in \mathbf{\Phi}$, we have $x \# x$.*

The finite state space Ω of the programs used as labels is fixed.

The intuition behind this definition is that, if a program fragment is considered atomic (i.e. it will happen without interferences from an environment), then it is the label of an event. Hence, we need to distinguish all atomic program fragments when translating a program into a bundle event structure. Atomic programs can be achieved by creating a construct that forces atomicity. Examples of such a technique include the "atomic bracket" used by Jones and Hayes [24]. In this chapter and the next, we will state which actions are atomic rather than using such a device.

In Chapter 5, the set $\mathbf{\Phi}_{\mathcal{E}}$ contained events that are pairwise in conflict. In Definition 6.1.4, $\mathbf{\Phi}_{\mathcal{E}}$ contains sets of events, and each $x \in \mathbf{\Phi}_{\mathcal{E}}$ is seen as a (local)

set of final events. The intuition is that, in the sequential composition $\mathcal{E} \cdot \mathcal{F}$, each of these sets $x \in \mathbf{\Phi}_{\mathcal{E}}$ will become a bundle set pointing to all initial events of $\mathcal{F}$. This new definition is mainly used to simplify the construction of the operation $(\|)$, which is necessary to obtain the associativity with respect to the t-simulation of Section 6.5.

6.2 Probabilistic scheduler on ipBES

To obtain a sequential equivalence on bundle event structures with implicit probability, we define the notion of scheduler that will provide a distribution on states, from maximal execution traces.

Firstly, we need the notion of subdistributions [48]. A subdistribution is a map $\mu : \Omega \to [0,1]$ such that $\sum_{s\in\Omega} \mu(s) \leq 1$. The set of subdistributions over Ω is denoted by $\mathbb{D}_{\leq 1}\Omega$.

Definition 6.2.1. *A scheduler σ on an ipBES $\mathcal{E}$ is a map*

$$\sigma : \mathcal{T}(\mathcal{E}) \to [(E \times \Omega) \rightharpoonup \mathbb{D}_{\leq 1}\Omega]$$

such that for all $\alpha \in \mathcal{T}(\mathcal{E})$:

1. $\mathrm{dom}(\sigma(\alpha)) = \{(e,s) \mid \alpha e \in \mathcal{T}(\mathcal{E}) \wedge s \in \Omega\}$,
2. *there exists a function $w : E \times \Omega \to [0,1]$ such that, for every $(e,s) \in \mathrm{dom}(\sigma(\alpha))$, $\sigma(\alpha)(e,s) = w(e,s)\mu$ for some $\mu \in \lambda(e)(s)$.*
3. *for every $s \in \Omega$, we have $\sum_{(e,s)\in\mathrm{dom}(\sigma(\alpha))} w(e,s) = 1$,*
4. *for every $(e,s) \in \mathrm{dom}(\sigma(\alpha))$, if $\lambda(e) = \emptyset$, then $w(e,s) = 0$ and $\sigma(\alpha)(e,s) = 0$ (the subdistribution that evaluates to 0 everywhere).*

The set of all schedulers on $\mathcal{E}$ is denoted by $\mathbf{Sched}(\mathcal{E})$.

Property 1 says that we may schedule an event if it does not depend on unscheduled events.

Property 2 states that, given a trace α, the scheduler will resolve the nondeterminism between events enabled after α, using the weight function w. This may include immediate conflicts (Chapter 5 Definition 5.1.2) or interleaving of concurrent events. Moreover, the scheduler has access to the current program state when resolving that nondeterminism. That is, $w(e, s)$ is the probability that the event e is scheduled, knowing that the program state is s. If the event e is successfully scheduled, then the scheduler performs a last choice of distribution, say μ from $\lambda(e)$, to generate the next state of the program.

Property 3 ensures that when the state s is known, then the choice between the events, enabled after the trace α, is indeed probabilistic.

Property 4 says that a scheduler is forced to choose events whose label does not evaluate to the empty set, at the current state of the program. This is particularly important when the program contains conditionals and the label of an event is a test. A scheduler is forced to choose the branch whose test holds. If two tests hold at state s, then a branch is chosen probabilistically.

The motivation behind Property 4 is to ensure that, for every trace α such that $\mathrm{dom}(\alpha) \neq \emptyset$, and every state $s \in \Omega$, we have

$$\sum_{(e,s)\in \mathrm{dom}(\sigma(\alpha))} \sigma(\alpha)(e, s) \in \mathbb{D}\Omega$$

i.e. that sum is indeed a distribution. To ensure that a scheduler satisfying that condition can be constructed, we restrict ourselves to *feasible* event structures. Given an element $r \in \overline{\mathbb{H}}_1\Omega$, we write $\mathrm{dom}(r) = \{s \mid r(s) \neq \emptyset\}$.

Definition 6.2.2. *A BES $\mathcal{E}$ is* feasible *if for every $\alpha \in \mathcal{T}(\mathcal{E})$ such that* $\mathrm{dom}(\alpha) \neq \emptyset$*, we have* $\cup_{\alpha e \in \mathcal{T}(\mathcal{E})} \mathrm{dom}(\lambda(e)) = \Omega$.

A consequence of this assumption is that an "if clause" always needs to have a corresponding "else clause".

Example 6.2.3. The bundle event structure with implicit probability associated to the program `if` $x = h$ `then` $x = h \oplus_{0.5} t$ `else skip fi` is directly translated as

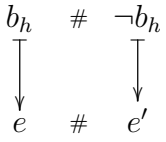


Figure 6.1: A simple example of bundle event structure with implicit probability.

the bundle event structure of Figure 6.1. The construct `skip` is the ineffectual program interpreted as δ. The events b_h and $\neg b_h$ are respectively labelled by the subidentities associated to the tests $x = h$ and $x \neq h$. The event e is labelled by r and e' is labelled by δ. In this example, the checking of the test $x = h$ and the execution of the probabilistic assignment r are both atomic. The only scheduler of this event structure satisfies $\sigma(\emptyset)(b_h, h) = \delta_h$, $\sigma(\emptyset)(b_h, t) = 0$ and $\sigma(b_h)(e, s) = 0.5\delta_h + 0.5\delta_t$. ■

6.3 Computation function on ipBES

We define the runs of a bundle event structure $\mathcal{E}$ against a given scheduler σ as follows. Let $\mathcal{T}_n(\mathcal{E})$ be the set of traces of length $n \in \mathbb{N}$. Let $\sigma \in \mathbf{Sched}(\mathcal{E})$, the *computation sequence* of $\mathcal{E}$ with respect to σ is a sequence of partial functions $\varphi_n : \mathcal{T}(\mathcal{E}) \rightharpoonup \mathbb{D}_{\leq 1}\Omega$ such that $\mathrm{dom}(\varphi_n) = \cup_{k \leq n} \mathcal{T}_n(\mathcal{E})$

1. $\varphi_0(\emptyset) = \delta_s$ where s is the initial state,
2. if $\alpha e \in \mathcal{T}_{n+1}(\mathcal{E})$ then

$$\varphi_{n+1}(\alpha e)(s) = \sum_{t \in \Omega} \sigma(\alpha)(e, t)(s)\varphi_n(\alpha)(t)$$

and $\varphi_{n+1}(\alpha e) = \varphi_n(\alpha e)$ otherwise.

In this inductive construction, the initial term φ_0 is defined using a given initial state s. Therefore, all subsequent terms also depend on s. When emphasis about s is needed, we write $\varphi_{n,s}$ instead of φ_n. This notation is mainly used when considering sequential compositions.

The *complete run* of $\mathcal{E}$ with respect to σ is the limit φ of that sequence i.e. $\varphi = \cup_n \varphi_n$ which exists because φ_n defines a sequence of partial functions such that φ_n is the restriction of φ_{n+1} on $\mathrm{dom}(\varphi_n)$. Since we consider finite traces only, we have $\mathrm{dom}(\varphi) = \mathcal{T}(\mathcal{E})$. The *behaviour* of $\mathcal{E}$ with respect to σ from the initial state s is defined by the sum

$$\sigma_s(\mathcal{E}) = \sum_{\alpha \in \mathcal{T}_{\max}(\mathcal{E})} \varphi(\alpha),$$

where $\mathcal{T}_{\max}(\mathcal{E})$ is the set of finite, maximal (with respect to the prefix ordering $\trianglelefteq$) traces of $\mathcal{E}$.

Proposition 6.3.1. *For every bundle event structure $\mathcal{E}$, scheduler $\sigma \in \mathbf{Sched}(\mathcal{E})$ and initial state s, $\sigma_s(\mathcal{E})$ is a subdistribution.*

Proof. Let φ be the complete run of $\mathcal{E}$ with respect to a given scheduler σ. We show by induction on n that

$$\mu_n(\Omega) = \sum_{\alpha \in \mathcal{T}_n \cup (\mathcal{T}_{\max} \cap \mathrm{dom}(\varphi_n))} \varphi(\alpha)(\Omega) = \sum_{t \in \Omega} \sum_{\alpha \in \mathcal{T}_n \cup (\mathcal{T}_{\max} \cap \mathrm{dom}(\varphi_n))} \varphi(\alpha)(t) = 1$$

The set $\mathcal{T}_n$ contains all traces of length n and the set $\mathcal{T}_{\max} \cap \mathrm{dom}(\varphi_n)$ contains maximal traces that may be of size less than n.

For the base case $n = 0$, we have $\mu_0(\Omega) = \varphi(\emptyset)(\Omega) = \delta_s(\Omega) = 1$, where s is the initial state. Assume the induction hypothesis $\mu_n(\Omega) = 1$. We have

$$\begin{aligned}
\mu_{n+1}(\Omega) &= \sum_{\alpha \in \mathcal{T}_{n+1} \cup (\mathcal{T}_{\max} \cap \mathrm{dom}(\varphi_{n+1})} \varphi(\alpha)(\Omega) \\
&= \sum_{\alpha \in \mathcal{T}_{n+1}} \varphi(\alpha)(\Omega) + \sum_{\alpha \in \mathcal{T}_{\max} \cap \mathrm{dom}(\varphi_n)} \varphi(\alpha)(\Omega) \\
&= \sum_{\alpha e \in \mathcal{T}_{n+1}} \sum_{t \in \Omega} \sigma(\alpha)(e,t)(\Omega)\varphi(\alpha)(t) + \sum_{\alpha \in \mathcal{T}_{\max} \cap \mathrm{dom}(\varphi_n)} \varphi(\alpha)(\Omega) \\
&= \sum_{\alpha \in \mathcal{T}_n \setminus \mathcal{T}_{\max}} \sum_{\alpha e \in \mathcal{T}} \sum_{t \in \Omega} \sigma(\alpha)(e,t)(\Omega)\varphi(\alpha)(t) + \sum_{\alpha \in \mathcal{T}_{\max} \cap \mathrm{dom}(\varphi_n)} \varphi(\alpha)(\Omega)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{\alpha \in \mathcal{T}_n \setminus \mathcal{T}_{\max}} \left[\sum_{(e,t) \in \mathrm{dom}(\sigma(\alpha))} \sigma(\alpha)(e,t)(\Omega) \right] \varphi(\alpha)(t) + \sum_{\alpha \in \mathcal{T}_{\max} \cap \mathrm{dom}(\varphi_n)} \varphi(\alpha)(\Omega) \\
&= \sum_{\alpha \in \mathcal{T}_n \setminus \mathcal{T}_{\max}} \varphi(\alpha)(\Omega) + \sum_{\mathcal{T}_{\max} \cap \mathrm{dom}(\varphi_n)} \varphi(\alpha)(\Omega) \\
&= \mu_n(\Omega) = 1.
\end{aligned}
$$

The square-bracketed term equals 1 because of Properties 2 and 3 of the scheduler σ. Therefore, each partial computation φ_n is a probability distribution when restricted on $\mathcal{T}_n \cup (\mathcal{T}_{\max} \cap \mathrm{dom}(\varphi_n))$, and hence the limit φ is a subdistribution on $\mathcal{T}_{\max}$. It does not necessarily add up to 1 because elements of $\mathcal{T}_{\max}$ are finite maximal traces only and non-termination will decrease that quantity. [2] ■

As a consequence of this proposition, we will denote by $\mathbf{Sched}_1(\mathcal{E})$ the set of schedulers of $\mathcal{E}$ such that, for every initial state s, $\sigma_s(\mathcal{E})$ is a distribution. A scheduler from $\mathbf{Sched}_1(\mathcal{E})$ will generate a sequential behaviour that terminates with probability 1.

Example 6.3.2. Let us reconsider the program $r \cdot (\delta + r)$. In this program, r is an atomic action, so the associated event structure is again constructed in a straightforward manner. Let us denote by r_i the event associated to the i^{th} occurrence of r ($i \in \{1, 2\}$). A scheduler σ on the event structure associated to $r \cdot (\delta + r)$ is characterised by the weight function w of Example 6.1.3. In fact we can write $\sigma(r_1)(r_1, s) = (1 - w(s))[0.5\delta_h + 0.5\delta_t]$. ■

6.4 Sequential semantics from ipBES

In this section, the sequential behaviour of a bundle event structure with implicit probability is computed as a sequential probabilistic program from $\mathbb{H}_1\Omega$.

[2] We assume that the empty sum is 0. This occurs when there are no maximal traces.

6.4.1 Functional Interpretation of ipBES

We construct a semantics map $[\![\ ]\!]$ that transforms each event structure to an element of $\mathbb{H}_1\Omega$. Recall that $\mathbb{H}_1\Omega$ can express probabilistic programs that terminate with probability 1. Hence, this thesis is restricted to partial correctness.

Given a feasible event structure $\mathcal{E}$, we define

$$[\![\mathcal{E}]\!](s) = \overline{\mathrm{conv}\{\sigma_s(\mathcal{E}) \mid \sigma \in \mathbf{Sched}_1(\mathcal{E})\}}$$

where $\mathrm{conv}(A)$ (resp. $\overline{A}$) is the convex (resp. topological) closure of the set of distributions A. We restrict ourselves to feasible and *terminating* event structures, i.e., such that $\mathbf{Sched}_1(\mathcal{E})$ is non-empty (except for the special element 0 defined below, which will not be interpreted sequentially).

Definition 6.4.1. *Let $\mathcal{E}, \mathcal{F}$ be two feasible event structures, we say that $\mathcal{E}$ (sequentially) refines $\mathcal{F}$, denoted by $\mathcal{E} \sqsubseteq \mathcal{F}$, if $[\![\mathcal{E}]\!] \sqsubseteq_{\mathbb{H}} [\![\mathcal{F}]\!]$ holds in $\mathbb{H}_1\Omega$.*

The relation $\sqsubseteq$ is indeed a preorder on bundle event structure with implicit probabilities. Whilst this order is not a congruence, it is used to specify the desired sequential correctness using event structures.

Recall that the operations $(+, \cdot, *)$ and constants $0, 1$ have been defined in Chapter 3 Section 3.3. We provide a simplified version as follows.

- The algebraic constant 1 is interpreted as $(e, \emptyset, \emptyset, \{(e, 1)\}, \{e\})$ where the 1 in the expression $(e, 1)$ is the identity of $\mathbb{H}_1\Omega$.
- The algebraic constant 0 is interpreted as $(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$.
- Each atomic action $r \in \overline{\mathbb{H}}_1\Omega$ is associated to $(\{e\}, \emptyset, \emptyset, \{(e, r)\}, \{e\})$. This event structure is again denoted by r.
- The nondeterministic choice between $\mathcal{E}$ and $\mathcal{F}$ is constructed in a similar way as Definition 3.3.1, that is,

$$\mathcal{E} + \mathcal{F} = (E \cup F, \#_{\mathcal{E}+\mathcal{F}}, \mapsto_{\mathcal{E}} \cup \mapsto_{\mathcal{F}}, \lambda_{\mathcal{E}} \cup \lambda_{\mathcal{F}}, \{x \cup y \mid x \in \mathbf{\Phi}_{\mathcal{E}} \wedge y \in \mathbf{\Phi}_{\mathcal{F}}\})$$

where $\#_{\mathcal{E}+\mathcal{F}} = [\cup_{x\in\mathbf{\Phi}_\mathcal{E}\wedge y\mathbf{\Phi}_\mathcal{F}}\mathrm{sym}(x\times y)]\cup\#_\mathcal{E}\cup\#_\mathcal{F}\cup\mathrm{sym}(\mathbf{in}(\mathcal{E})\times\mathbf{in}(\mathcal{F}))$ and sym is again the symmetric closure of a relation on $E\cup F$. The square-bracketed set ensures that every final event in $\mathcal{E}$ is in conflict with every final event in $\mathcal{F}$. This ensures that, if $z\in\mathbf{\Phi}_{\mathcal{E}+\mathcal{F}}$, then $z\#z$.

- The sequential composition is

$$\mathcal{E}\cdot\mathcal{F} = (E\cup F, \#_\mathcal{E}\cup\#_\mathcal{F}, \mapsto_\mathcal{E}\cup\mapsto_\mathcal{F}\cup\{x\mapsto e \mid e\in\mathbf{in}(\mathcal{F})\wedge x\in\mathbf{\Phi}_\mathcal{E}\}, \lambda_\mathcal{E}\cup\lambda_\mathcal{F}, \mathbf{\Phi}_\mathcal{F}).$$

- The concurrent composition of $\mathcal{E}$ and $\mathcal{F}$ is

$$\mathcal{E}\|\mathcal{F} = (E\cup F, \#_\mathcal{E}\cup\#_\mathcal{F}, \mapsto_\mathcal{E}\cup\mapsto_\mathcal{F}, \lambda_\mathcal{E}\cup\lambda_\mathcal{F}, \mathbf{\Phi}_\mathcal{E}\cup\mathbf{\Phi}_\mathcal{F})$$

In the application of each of these binary operations, we assume that the set of events of the operands are again disjoint.

Example 6.4.2. Given a program $r_1, r_2, r_3 \in \overline{\mathbb{H}}_1\Omega$, the event structure associate to $(r_1\|r_2)+r_3$ is

$$(\{e_1,e_2,e_3\}, \mathrm{sym}(\{(e_i,e_3)\mid i\in\{1,2\}\}), \emptyset, \lambda, \{\{e_i,e_3\}\mid i\in\{1,2\}\})$$

where $\lambda(e_i)=r_i$. Notice that $e_1\#e_3$ and $e_2\#e_3$, but e_1 and e_2 are concurrent. This generates a "confusion" from the point of view of the previous chapter but, that is not a problem in this variant of bundle event structures because probabilities are associated to labels instead of clusters. ■

These changes ensure that all events of a bundle event structure with implicit probability are labelled with an element of $\overline{\mathbb{H}}_1\Omega$. This is necessary when constructing schedulers. Notice that, for every bundle event structure $\mathcal{E}$, $0+\mathcal{E}=\mathcal{E}$, $0\cdot\mathcal{E}=\mathcal{E}\cdot 0=\mathcal{E}$, and in particular, $0\cdot 1=1$. The constant 0 was only introduced to have a bottom element on the set of bundle event structures with implicit probabilities, rather than to obtain all equations valid in probabilistic concurrent Kleene algebra. It will again ensures that we can compute the Kleene star inductively

from the least element. Moreover, 0 will disappear in mixed expressions because of these properties. The important algebraic properties that are necessary for the rely/guarantee setting of the next section are proven in Proposition 6.5.4.

We now show that the operations $(+, \cdot)$ are preserved by the map $[\![\ ]\!]$. The case of the binary Kleene star $(*)$ is proven in Proposition 6.5.7.

Proposition 6.4.3. *Let $\mathcal{E}, \mathcal{F}$ be non-zero, feasible and terminating event structures, for every operation $\circ \in \{+, \cdot\}$, we have $[\![\mathcal{E} \circ \mathcal{F}]\!] = [\![\mathcal{E}]\!] \circ [\![\mathcal{F}]\!]$.*

Proof. For the case of nondeterminism $(+)$, let $s \in \Omega$ be the initial state and $\mu \in [\![\mathcal{E}+\mathcal{F}]\!](s)$. Let us firstly assume that $\mu = \sigma_s(\mathcal{E})$ for some $\sigma \in \mathbf{Sched}_1(\mathcal{E}+\mathcal{F})$. By definition of the sum $\mathcal{E}+\mathcal{F}$, the set of events E and F are disjoint, so we can define two schedulers $\sigma^{\mathcal{E}} \in \mathbf{Sched}_1(\mathcal{E})$ and $\sigma^{\mathcal{F}} \in \mathbf{Sched}_1(\mathcal{F})$ as follows. Let $\alpha \in \mathcal{T}(\mathcal{E}+\mathcal{F})$ and $(e,t) \in \mathrm{dom}(\sigma(\alpha))$, we define

$$\sigma^{\mathcal{E}}(\alpha)(e,t) = \begin{cases} \sigma(\alpha)(e,t) & \text{if } \alpha \in \mathcal{T}(\mathcal{E}) \setminus \{\emptyset\}, \\ \frac{\sigma(\emptyset)(e,t)}{p_t^{\mathcal{E}}} & \text{if } \alpha = \emptyset \end{cases}$$

where $p_t^{\mathcal{E}} = \sum_{e' \in \mathbf{in}(\mathcal{E})} w(e,t)$, w is the weight function associated to σ at the trace $\emptyset$ and s is the initial state. The real number $p_t^{\mathcal{E}}$ is just a normalisation constant required by Property 3 in the definition of schedulers. [3] The scheduler $\sigma^{\mathcal{F}}$ is similarly defined. It follows directly from these definition of $\sigma^{\mathcal{E}}$ and $\sigma^{\mathcal{F}}$ that $\sigma(\emptyset)(e,t) = p_t^{\mathcal{E}}\sigma(\emptyset)(e,t) + p_t^{\mathcal{F}}\sigma(\emptyset)(e,t)$ where $p_t^{\mathcal{E}} + p_t^{\mathcal{F}} = 1$ because of Property 3. Hence, $\sigma_s(\mathcal{E}) = p_s^{\mathcal{E}}\sigma_s^{\mathcal{E}}(\mathcal{E}) + p_s^{\mathcal{F}}\sigma_s^{\mathcal{F}}(\mathcal{F})$ i.e. $\sigma_s(\mathcal{E}) \in [\![\mathcal{E}]\!] + [\![\mathcal{F}]\!]$. Since $[\![\mathcal{E}]\!] + [\![\mathcal{F}]\!]$ is convex and topologically closed, we deduce that $[\![\mathcal{E}+\mathcal{F}]\!](s) \subseteq ([\![\mathcal{E}]\!] + [\![\mathcal{F}]\!])(s)$.

For the converse inclusion $([\![\mathcal{E}]\!] + [\![\mathcal{F}]\!])(s) \subseteq [\![\mathcal{E}+\mathcal{F}]\!](s)$, notice that $\overline{\mathrm{conv}(A)} = \mathrm{conv}(\overline{A})$ holds for every subset $A \subseteq \mathbb{R}^{\Omega}$. If we write $A = \{\sigma_s(\mathcal{E}) \mid \sigma \in \mathbf{Sched}_1(\mathcal{E})\}$ and $B = \{\sigma_s(\mathcal{F}) \mid \sigma \in \mathbf{Sched}_1(\mathcal{F})\}$, then

$$([\![\mathcal{E}]\!] + [\![\mathcal{F}]\!])(s) = \overline{\mathrm{conv}(\overline{\mathrm{conv}(A)} \cup \overline{\mathrm{conv}(B)})} = \overline{\mathrm{conv}(A \cup B)}$$

But it is clear that $A \subseteq [\![\mathcal{E}+\mathcal{F}]\!](s)$ (a scheduler that does not choose $\mathcal{F}$ is

[3] If $p_t^{\mathcal{E}} = 0$, then $\sigma \in \mathbf{Sched}_1(\mathcal{F})$.

possible because $\mathcal{E}$ is feasible) and $B \subseteq [\![\mathcal{E}+\mathcal{F}]\!](s)$. Therefore, $([\![\mathcal{E}]\!]+[\![\mathcal{F}]\!])(s) = \overline{\mathrm{conv}(A \cup B)} \subseteq [\![\mathcal{E}+\mathcal{F}]\!](s)$ because the last set is convex and topologically closed.

The sequential composition is proven using a similar reasoning. Let $\mathcal{E}, \mathcal{F}$ be two bundle event structures satisfying the hypothesis, and $\mu \in [\![\mathcal{E}\cdot\mathcal{F}]\!](s)$ for some initial state $s \in \Omega$. Firstly, let us assume that there is a scheduler σ on $\mathcal{E}\cdot\mathcal{F}$ such that $\mu = \sigma_s(\mathcal{E}\cdot\mathcal{F})$. Since schedulers are inductively constructed, there exists $\sigma^{\mathcal{E}} \in \mathbf{Sched}(\mathcal{E})$ and $\sigma^{\mathcal{F}} \in \mathbf{Sched}(\mathcal{F})$ such that

$$\sigma(\alpha)(e,t) = \begin{cases} \sigma^{\mathcal{E}}(\alpha)(e,t) & \text{if } \alpha e \in \mathcal{T}(\mathcal{E}), \\ \sigma^{\mathcal{F}}(\alpha'')(e,t) & \text{if } \alpha = \alpha'\alpha'' \text{ and } (\alpha',\alpha'') \in \mathcal{T}_{\max}(\mathcal{E}) \times \mathcal{T}(\mathcal{F}). \end{cases}$$

Let us denote by φ_n and $\varphi^{\mathcal{E}}_n$ (resp. $\varphi^{\mathcal{F}}_{n,t}$) the computation sequences associated to the respective schedulers σ and $\sigma^{\mathcal{E}}$ (resp. $\sigma^{\mathcal{F}}$) from the initial state s (resp. t). It follows directly that $\varphi_n(\alpha) = \varphi^{\mathcal{E}}_n(\alpha)$ for every $\alpha \in \mathcal{T}_n(\mathcal{E})$. If $\alpha' \in \mathcal{T}_{\max}(\mathcal{E}) \cap \mathcal{T}_n(\mathcal{E})$ and $e \in \mathbf{in}(\mathcal{F})$ then, for every state $u \in \Omega$,

$$\varphi_{n+1}(\alpha' e)(u) = \sum_{t\in\Omega} \sigma^{\mathcal{F}}(\emptyset)(e,t)(u)\varphi^{\mathcal{E}}(\alpha')(t).$$

Similarly, we have

$$\begin{aligned} \varphi_{n+1}(\alpha' e e')(u) &= \sum_{t'\in\Omega} \sigma^{\mathcal{F}}(e)(e,t')(u) \left[\sum_{t\in\Omega} \sigma^{\mathcal{F}}(\emptyset)(e,t)(t')\varphi^{\mathcal{E}}(\alpha')(t)\right] \\ &= \sum_{t\in\Omega} \left[\sum_{t'\in\Omega} \sigma^{\mathcal{F}}(e)(e,t')(u)\sigma^{\mathcal{F}}(\emptyset)(e,t)(t')\right] \varphi^{\mathcal{E}}(\alpha')(t) \\ &= \sum_{t\in\Omega} \varphi^{\mathcal{F}}_{2,t}(u)\varphi^{\mathcal{E}}(\alpha')(t). \end{aligned}$$

By simple induction on the length of α'', we deduce that

$$\varphi(\alpha'\alpha'')(u) = \sum_{t\in\Omega} \varphi^{\mathcal{F}}_t(\alpha'')(u)\varphi^{\mathcal{E}}(\alpha')(t),$$

where $\varphi_t^{\mathcal{F}}$ is the complete run obtained from the sequence $\varphi_{n,t}^{\mathcal{F}}$[4]. It follows by definition of the sequential composition on $\mathbb{H}_1\Omega$ (Equation 6.3), that

$$\sigma_s(\mathcal{E})(u) = \sum_{t\in\Omega} \sigma_t^{\mathcal{F}}(\mathcal{F})(u)\sigma_s^{\mathcal{E}}(\mathcal{E})(t) \in [\![\mathcal{E}]\!] \cdot [\![\mathcal{F}]\!](s),$$

for every state $u \in \Omega$. Secondly, since $[\![\mathcal{E}]\!] \cdot [\![\mathcal{F}]\!](s)$ is upclosed and topologically closed, we deduce that $[\![\mathcal{E} \cdot \mathcal{F}]\!](s) \subseteq [\![\mathcal{E}]\!] \cdot [\![\mathcal{F}]\!](s)$. Conversely, if $\mu \in [\![\mathcal{E}]\!] \cdot [\![\mathcal{F}]\!](s)$, then either $\mu(u) = \sum_{t\in\Omega} \sigma_t^{\mathcal{F}}(\mathcal{F})(u)\sigma_s^{\mathcal{E}}(\mathcal{E})(t)$ or μ is in the closure of the set of these distributions. Either way, the closure properties of $[\![\mathcal{E} \cdot \mathcal{F}]\!](s)$ imply that $[\![\mathcal{E}]\!] \cdot [\![\mathcal{F}]\!](s) \subseteq [\![\mathcal{E} \cdot \mathcal{F}]\!](s)$. ■

6.5 Simulation for ipBES with tests

The simulation order constructed in this subsection remedies to the congruence problem of the sequential refinement given in Definition 6.4.1.

We say that a trace α is *weakly maximal* if it is maximal or there exist some events $e_1, \ldots, e_n$ such that $\alpha e_1 \cdots e_n \in \mathcal{T}_{\max}(\mathcal{E})$ and $\delta \sqsubseteq_{\mathbb{H}} \lambda(e_i)$ for every $1 \leq i \leq n$.

Definition 6.5.1. *A function $f : \mathcal{T}(\mathcal{E}) \to \mathcal{T}(\mathcal{F})$ is called a t-simulation:*

- *if $f(\emptyset) = \emptyset$ and $f^{-1}(\beta)$ is a finite set for every $\beta \in \mathcal{T}(\mathcal{F})$,*
- *if $\alpha e \in \mathcal{T}(\mathcal{E})$ then either:*
  - *$f(\alpha e) = f(\alpha)$ and $\lambda(e) \sqsubseteq_{\mathbb{H}} \delta$ holds in $\mathbb{H}_1\Omega$,*
  - *or there exists $e' \in F$ such that $\lambda(e) \sqsubseteq_{\mathbb{H}} \lambda(e')$ and $f(\alpha e) = f(\alpha)e'$.*
- *if αe is maximal in $\mathcal{T}(\mathcal{E})$ then $f(\alpha e) = f(\alpha)e'$, for some e' (with $\lambda(e) \sqsubseteq_{\mathbb{H}} \lambda(e')$), and $f(\alpha e)$ is weakly maximal in $\mathcal{T}(\mathcal{F})$.*

We say that $\mathcal{E}$ is simulated by $\mathcal{F}$, written $\mathcal{E} \sqsubseteq_{\text{sim}} \mathcal{F}$, if there exists a simulation from $\mathcal{E}$ to $\mathcal{F}$. The equivalence generated by this preorder is denoted $\equiv_{sim}$.

[4]Remind that $\varphi_{n,t}$ is the computation sequence that starts with $\varphi_{0,t}(\emptyset) = \delta_t$.

Notice that if $f(\alpha)$ is maximal then α is necessarily maximal.

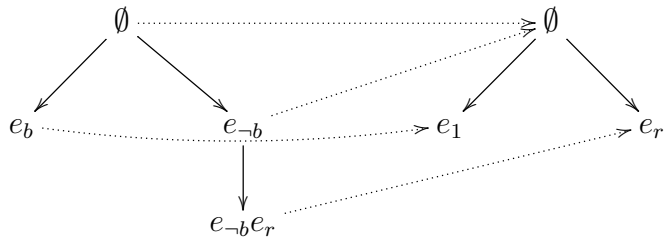
The notion of t-simulation has been designed to correctly simulate event structures in presence of tests. For instance, given a test b, the simulation $\delta \sqsubseteq_{\text{sim}} (b+\neg b)$ fails. A t-simulation is a total function and it does not allow the removal of "internal" events labelled with sub-identities during a refinement step . The finiteness condition on $f^{-1}(\beta)$ ensures that we do not refine a terminating specification with a diverging implementation. For instance, without that constraint, we may write the refinement

$$\texttt{if } (0 == 1) \texttt{ then } s := 0 \texttt{ else}[\texttt{if } (0 == 1) \texttt{ then } s := 0 \texttt{ else}[\dots]] \sqsubseteq_{\text{sim}} s := 0.$$

This should not hold because the left hand side is a non-terminating program and cannot refine the terminating assignment $s := 0$.

A t-simulation is used to compare bundle event structures without looking in details at the labels of events. It can be seen as a refinement order on the higher level structure of a concurrent program. Once a sequential behaviour has to be checked, we use the previously defined functional equivalence (Definition 6.4.1) on event structures with implicit probabilities.

Example 6.5.2. A t-simulation from $b + \neg b \cdot r$ to $1 + r$ is given by the dotted arrow in the following diagram:



This shows the refinement of a nondeterministic choice with a conditional. Notice that t-simulations allow the introduction of subidentities. ■

Lemma 6.5.3. *The t-simulation relation $\sqsubseteq_{\text{sim}}$ is a preorder.*

Proof. Reflexivity follows from the identity function and transitivity is obtained

by composing t-simulation which will generate a new t-simulation. Notice that care should be taken with respect to the third property of a t-simulation. If $f : \mathcal{T}(\mathcal{E}) \to \mathcal{T}(\mathcal{F})$, $g : \mathcal{T}(\mathcal{F}) \to \mathcal{T}(\mathcal{G})$ are t-simulations, $\alpha e \in \mathcal{T}_{\max}(\mathcal{E})$ and $\lambda(e) \sqsubseteq_{\mathbb{H}} \delta$, then $f(\alpha e) = f(\alpha)e'$ for some $e' \in F$ such that $\lambda(e) \sqsubseteq_{\mathbb{H}} \lambda(e')$. If $\lambda(e') \sqsubseteq_{\mathbb{H}} \delta$, then it is possible that $g(f(\alpha)e') = g(f(\alpha))$. However, since $f(\alpha)e'$ is weakly maximal, $g(f(\alpha)e')$ is also weakly maximal and we can find an event $e'' \in G$ such that $f(\alpha)e''$ is weakly maximal and $\lambda(e') \sqsubseteq_{\mathbb{H}} \lambda(e'')$. We then map αe to $g(f(\alpha))e''$ in the generated t-simulation. ■

Proposition 6.5.4. *If $\mathcal{E}, \mathcal{F}, \mathcal{G}$ are bundle event structures with implicit probability, then*

$$\mathcal{E}\|\mathcal{F} \equiv_{\text{sim}} \mathcal{F}\|\mathcal{E} \tag{6.4}$$

$$\mathcal{E}\|(\mathcal{F}\|\mathcal{G}) \equiv_{\text{sim}} (\mathcal{E}\|\mathcal{F})\|\mathcal{G} \tag{6.5}$$

$$\mathcal{E} \sqsubseteq_{\text{sim}} \mathcal{F} \Rightarrow \mathcal{E}\|\mathcal{G} \sqsubseteq_{\text{sim}} \mathcal{F}\|\mathcal{G} \tag{6.6}$$

$$\mathcal{E} \sqsubseteq_{\text{sim}} \mathcal{F} \Rightarrow \mathcal{G} \cdot \mathcal{E} \sqsubseteq_{\text{sim}} \mathcal{G} \cdot \mathcal{F} \tag{6.7}$$

Proof. The constructions $\mathcal{E}\|\mathcal{F}$ and $\mathcal{F}\|\mathcal{E}$ result in the same event structure and similarly for the associativity.

For the Implication 6.6, let $f : \mathcal{T}(\mathcal{E}) \to \mathcal{T}(\mathcal{F})$ be a t-simulation. Let us construct a t-simulation $g : \mathcal{T}(\mathcal{E}\|\mathcal{G}) \to \mathcal{T}(\mathcal{F}\|\mathcal{G})$ inductively. We set $g(\emptyset) = \emptyset$. Let $\alpha \in \mathcal{T}(\mathcal{E}\|\mathcal{G})$ and $e \in E \cup G$ such that αe is a trace of $\mathcal{E}\|\mathcal{G}$. We write $\alpha|_E$ the restriction of α to the events occurring in $\mathcal{E}$. The inductive definition of g is:

$$g(\alpha e) = \begin{cases} g(\alpha)e & \text{if } e \in G, \\ g(\alpha) & \text{if } e \in E \text{ and } f(\alpha|_E e) = f(\alpha|_E), \\ g(\alpha)e' & \text{if } e \in E \text{ and } f(\alpha|_E e) = f(\alpha|_E)e' \end{cases}$$

Since the set of events of $\mathcal{E}$ and $\mathcal{G}$ are disjoint, the cases in the above definition of g are disjoint. That is, g is indeed a function and it satisfies the second property of a *t*-simulation. The last property is clear because if αe is maximal in $\mathcal{T}(\mathcal{E}\|\mathcal{G})$,

then either $\alpha|_E$ is maximal in $\mathcal{E}$ and $\alpha|_G e$ is maximal in $\mathcal{T}(\mathcal{G})$, or $\alpha|_E e$ is maximal in $\mathcal{T}(\mathcal{E})$ and $\alpha|_G$ is maximal in $\mathcal{T}(\mathcal{G})$. In both cases, $g(\alpha e) = g(\alpha)e'$ for some $e' \in E \cup G$ and $g(\alpha e)$ is weakly maximal in $\mathcal{T}(\mathcal{F}\|\mathcal{G})$.

For the last case, let f be a t-simulation from $\mathcal{E}$ to $\mathcal{F}$. It is clear that the function $g : \mathcal{T}(\mathcal{G} \cdot \mathcal{E}) \to \mathcal{T}(\mathcal{G} \cdot \mathcal{F})$, such that $g(\alpha) = \alpha|_G f(\alpha|_E)$ is a t-simulation. ■

We now prove the main result of this chapter, which is the backbone of our probabilistic rely/guarantee calculus.

Theorem 6.5.5. *Let $\mathcal{E}$ and $\mathcal{F}$ be feasible and terminating, if $\mathcal{E} \sqsubseteq_{\text{sim}} \mathcal{F}$ then $\mathcal{E} \sqsubseteq \mathcal{F}$.*

Proof. Let f be a t-simulation from $\mathcal{E}$ to $\mathcal{F}$, $s \in \Omega$ be the initial state, $\sigma \in \mathbf{Sched}_1(\mathcal{E})$ and φ is the complete run of σ on $\mathcal{E}$ from s. We have to generate a scheduler $\tau \in \mathbf{Sched}_1(\mathcal{F})$ such that the measures $\sigma_s(\mathcal{E})$ and $\tau_s(\mathcal{F})$ are equal i.e. they produce the same value for every state $u \in \Omega$.

For every $\beta \in \mathcal{T}(\mathcal{F})$, we define $f^{-1}_{\mathbf{min}}(\beta)$ to be the set of minimal traces in $f^{-1}(\beta)$, that is,

$$f^{-1}_{\mathbf{min}}(\beta) = \{\alpha \mid \forall e \in E : \alpha = \alpha' e \in f^{-1}(\beta) \Rightarrow \alpha' \notin f^{-1}(\beta)\}.$$

We now construct the scheduler τ. Let $\beta \in \mathcal{T}(\mathcal{F})$. We consider two cases:

- If $f^{-1}(\beta) = \emptyset$ then we set $\tau(\beta)(e, t) = 0 \in \mathbb{D}_{\leq 1}\Omega$ (the subdistribution that produces 0 on every state), except for some particular maximal traces that are handled in (†) below.

- Otherwise, given a state $t \in \Omega$, we define a normalisation factor

$$C_{\beta,t} = \sum_{\alpha \in f^{-1}_{\mathbf{min}}(\beta)} \varphi(\alpha)(t),$$

and we set [5]

$$\tau(\beta)(e,t) = \frac{1}{C_{\beta,t}} \left(\sum_{\alpha \in f_{\mathbf{min}}^{-1}(\beta)} \varphi(\alpha)(t) \sum_{\alpha e_1 \cdots e_k \in f_{\mathbf{min}}^{-1}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,t)\mu_k \right)$$

where $w_{i-1}(e_i,t)$ is the quantity such that $\sigma(\alpha e_1 \cdots e_{i-1})(e_i,t) = w_{i-1}(e_i)\mu$, and $\mu \in \lambda(e_i)$ (if $\lambda(e_i)(t)$ is empty then $w_{i-1}(e_i,t) = 0$). The distribution μ_k is chosen by σ from $\lambda(e_k)(t)$, when scheduling e_k.

Firstly, we show that τ is indeed a scheduler on $\mathcal{F}$. Definition 6.2.1 Property 1 is clear. Let us show the other properties. Let $\beta e \in \mathcal{T}(\mathcal{E})$ and let $W : E \times \Omega \to \mathbb{R}$ be the weight function such that

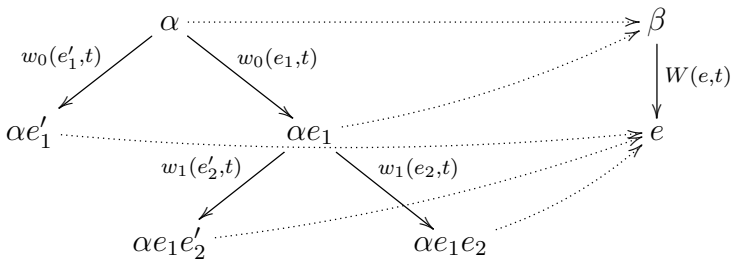
$$W(e,t) = \frac{1}{C_{\beta,t}} \sum_{\alpha \in f_{\mathbf{min}}^{-1}(\beta)} \varphi(\alpha)(t) \sum_{\alpha e_1 \cdots e_k \in f_{\mathbf{min}}^{-1}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,t)$$

Indeed, $\mu = \frac{\tau(\beta)(e,t)}{W(e,t)} \in \lambda(e)(t)$ [6] because $\lambda(e)(t)$ is convex and for each $\alpha e_1 \cdots e_k \in f_{\mathbf{min}}^{-1}(\beta e)$, $\mu_k \in \lambda(e_k)(t) \subseteq \lambda(e)(t)$. Hence $\tau(\beta)(e) = W_e f_e$ and τ satisfies the Definition 6.2.1 Property 2. As for Definition 6.2.1 Property 3, let $s \in \Omega$ and let us compute the quantity

$$V(t) = \sum_{(e,t) \in \mathrm{dom}(\tau(\beta))} W(e,t),$$

[5] Notice if $C_{\beta,t} = 0$ for some $t \in \Omega$ then $\varphi(\alpha)(t) = 0$ for every $\alpha \in f_{\mathbf{min}}^{-1}(\beta)$. In other words, none of these α will be scheduled at all. Hence, β need not be scheduled either.

[6] The case $W(e,t) = 0$ can be adapted easily because the numerator in the definition of $\tau(\beta)(e)$ is also 0. For instance, we can assume that $\frac{0}{0} = 1$.



we have $V(t) = w_0(e'_1,t) + w_0(e_1,t)w_1(e'_2,t)w_0(e_1,t)w_1(e_2,t) = 1$ because $w_1(e'_2,t)+w_1(e_2,t) = 1$ and $w_0(e'_1,t)+w_0(e_1,t) = 1$ (Definition 6.2.1 Property 3).

Figure 6.2: An example showing that $V(t) = 1$

---

for a fixed $t \in \Omega$. Let us write $\mathrm{dom}(\beta) = \{e \mid \beta e \in \mathcal{T}(\mathcal{F})\}$.

$$
\begin{aligned}
V(t) &= \sum_{(e,t)\in\mathrm{dom}(\tau(\beta))} \frac{1}{C_{\beta,t}} \sum_{\alpha\in f^{-1}_{\mathbf{min}}(\beta)} \varphi(\alpha)(t) \sum_{\alpha e_1\cdots e_k \in f^{-1}_{\mathbf{min}}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,t) \\
&= \frac{1}{C_{\beta,t}} \sum_{\alpha\in f^{-1}_{\mathbf{min}}(\beta)} \varphi(\alpha)(t) \sum_{(e,t)\in\mathrm{dom}(\tau(\beta))} \sum_{\alpha e_1\cdots e_k \in f^{-1}_{\mathbf{min}}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,t) \\
&= \frac{1}{C_{\beta,t}} \sum_{\alpha\in f^{-1}_{\mathbf{min}}(\beta)} \varphi(\alpha)(t) \sum_{\alpha e_1\cdots e_k \in \bigcup_{e\in\mathrm{dom}(\beta)} f^{-1}_{\mathbf{min}}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,t)
\end{aligned}
$$

From the second to the third expression, the two rightmost sums were merged into a single one because $f^{-1}_{\mathbf{min}}(\beta e) \cap f^{-1}(\beta e') = \emptyset$ (f is a function). It follows from Definition 6.2.1 Property 3, applied on the weight $w_{i-1}(e_i,t)$ of σ, that

$$
\sum_{\alpha e_1\cdots e_k \in \bigcup_{e\in\mathrm{dom}(\beta)} f^{-1}_{\mathbf{min}}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,s) = 1
$$

and hence $V = 1$ (c.f. Figure 6.2 for a concrete example). Definition 6.2.1 Property 4 is clear because if $\lambda(e)(t) = \emptyset$, then the coefficient of $\sigma(\alpha e_1 \cdots e_{k-1})(e_k,t)$ is 0 because $\lambda(e_k)(t) = \emptyset$. Hence, the product is also 0.

Secondly, let ψ be the complete run of $\mathcal{F}$ with respect to τ. We now show by

induction on β that

$$\psi(\beta) = \sum_{\alpha \in f_{\mathbf{min}}^{-1}(\beta)} \varphi(\alpha) = C_{\beta,t}, \tag{6.8}$$

where the empty sum evaluates to the identically zero distribution. The base case is clear because $\psi(\emptyset) = \delta_s = \phi(\emptyset)$ where s is the initial state. Let us assume the above identity for $\beta \in \mathcal{T}(\mathcal{F})$ and let $e \in F$ such that $\beta e = \mathcal{T}(\mathcal{E})$ and $f_{\mathbf{min}}^{-1}(\beta e) \neq \emptyset$. By definition of ψ, if $u \in \Omega$, we have:

$$\begin{aligned}
\psi(\beta e)(u) &= \sum_{t\in\Omega} \frac{1}{C_{\beta,t}} \sum_{\alpha \in f_{\mathbf{min}}^{-1}(\beta)} \varphi(\alpha)(t) \sum_{\alpha e_1 \cdots e_k \in f_{\mathbf{min}}^{-1}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,t)\mu_k(u)\psi(\beta)(t) \\
&= \sum_{t\in\Omega} \sum_{\alpha \in f_{\mathbf{min}}^{-1}(\beta)} \sum_{\alpha e_1 \cdots e_k \in f_{\mathbf{min}}^{-1}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,t)\mu_k(u)\varphi(\alpha)(t) \\
&= \sum_{\alpha \in f_{\mathbf{min}}^{-1}(\beta)} \sum_{\alpha e_1 \cdots e_k \in f_{\mathbf{min}}^{-1}(\beta e)} \sum_{t\in\Omega} \prod_{i=1}^{k} w_{i-1}(e_i,t)\mu_k(u)\varphi(\alpha)(t) \\
&= \sum_{\alpha \in f_{\mathbf{min}}^{-1}(\beta)} \sum_{\alpha e_1 \cdots e_k \in f_{\mathbf{min}}^{-1}(\beta e)} \sum_{t\in\Omega} \sum_{t'\in\Omega} w_0(e_1,t')\delta_{t'}(t) \left[\prod_{i=2}^{k} w_{i-1}(e_i,t)\mu_k(u)\right] \varphi(\alpha)(t') \\
&= \sum_{\alpha \in f_{\mathbf{min}}^{-1}(\beta)} \sum_{\alpha e_1 \cdots e_k \in f_{\mathbf{min}}^{-1}(\beta e)} \sum_{t\in\Omega} \prod_{i=2}^{k} w_{i-1}(e_i,t)\mu_k(u)\varphi(\alpha e_1)(t) \\
&= \sum_{\alpha \in f_{\mathbf{min}}^{-1}(\beta)} \sum_{\alpha e_1 \cdots e_k \in f_{\mathbf{min}}^{-1}(\beta e)} \sum_{t\in\Omega} \sum_{t'\in\Omega} w_1(e_2,t')\delta_{t'}(t) \left[\prod_{i=3}^{k} w_{i-1}(e_i,t)\mu_k(u)\right] \varphi(\alpha e_1)(t') \\
&= \cdots .
\end{aligned}$$

By continuing the above reasoning for all e_i (induction), $i \leq k-1$, we obtain

$$\begin{aligned}
\psi(\beta e)(u) &= \sum_{\alpha \in f_{\mathbf{min}}^{-1}(\beta)} \sum_{\alpha e_1 \cdots e_k \in f_{\mathbf{min}}^{-1}(\beta e)} \sum_{t\in\Omega} w_{k-1}(e_k,t)\mu_k(u)\varphi(\alpha e_1 \cdots e_{k-1})(t) \\
&= \sum_{\alpha \in f_{\mathbf{min}}^{-1}(\beta)} \sum_{\alpha e_1 \cdots e_k \in f_{\mathbf{min}}^{-1}(\beta e)} \varphi(\alpha e_1 \cdots e_k)(u)
\end{aligned}$$

Hence,

$$\psi(\beta e)(u) \;=\; \sum_{\alpha' \in f^{-1}_{\mathbf{min}}(\beta e)} \varphi(\alpha')(u).$$

(†) We finally compute the sum $\tau_s(\mathcal{F}) = \sum_{\beta \in \mathcal{T}_{\max}(\mathcal{F})} \psi(\beta)$. Notice firstly that τ may not schedule some traces of $\mathcal{F}$. In particular, the third property in the definition of simulation implies that a maximal element of $\mathcal{T}(\mathcal{E})$ may be mapped to a weakly maximal element of $\mathcal{T}(\mathcal{F})$. Hence, we need to extend the scheduler τ so that it is non-zero for exactly one maximal element from that weakly maximal trace. More precisely, if $\beta' = f(\alpha)$ is weakly maximal for some maximal trace $\alpha \in \mathcal{T}_{\max}(\mathcal{E})$, then there exists a sequence $e_1, \ldots, e_n$ such that $\beta = \beta' e_1 \cdots e_n \in \mathcal{T}_{\max}(\mathcal{F})$ and $\delta \sqsubseteq_{\mathbb{H}} \lambda(e_i)$. We extend τ such that $\tau(\beta' e_1 \cdots e_i)(e_{i+1}, t) = \delta_t$. This implies that $\psi(\beta)(t) = \psi(\beta')(t)$. The other case is that β is maximal and belongs to the image of f. In both cases, we have

$$\psi(\beta)(t) = \sum_{\alpha \in A_\beta} \varphi(\alpha)(t),$$

where $A_\beta = f^{-1}_{\mathbf{min}}(\beta)$ if β is in the image of f, or $A_\beta = f^{-1}_{\mathbf{min}}(\beta')$ if there is such a β' as above, otherwise, $A_\beta = \emptyset$. Thus, A_β contains maximal traces only (if it is not empty). Since, f is a total function, the set $\{A_\beta \mid \beta \in \mathcal{T}_{\max}(\mathcal{F})\}$ is a partition of $\mathcal{T}_{\max}(\mathcal{E})$ and we have

$$\sum_{\beta \in \mathcal{T}_{\max}(\mathcal{E})} \psi(\beta)(t) = \sum_{\beta \in \mathcal{T}_{\max}(\mathcal{E})} \sum_{\alpha \in A_\beta} \varphi(\alpha)(t) = \sum_{\alpha \in \mathcal{T}_{\max}(\mathcal{E})} \varphi(\alpha)(t),$$

i.e., we obtain $\tau_s(\mathcal{F}) = \sigma_s(\mathcal{E})$. ■

Example 6.5.6. Let us reconsider the simulation of Example 6.5.2. By definition, the unique scheduler σ on $b + \neg b \cdot r$ is characterised by the weight function w such that, $w(\emptyset)(e_b, t) = 1$ if $b(t)$ holds and 0 otherwise. From the illustrated simulation f, we have $f^{-1}_{\mathbf{min}}(e_r) = \{e_{\neg b} e_r\}$ and hence, $\tau(\emptyset)(e_r, t) = w(e_{\neg b}, t)\mu$ where $\mu \in r(t)$

is the distribution chosen by σ. Similarly, for $\tau(\emptyset)(e_b, t)$. ■

We now show that the binary Kleene star is preserved by the semantics map.

Proposition 6.5.7. *For every non-zero, feasible and terminating event structure $\mathcal{E}$ and $\mathcal{F}$, we have $[\![\mathcal{E} * \mathcal{F}]\!] = [\![\mathcal{E}]\!] * [\![\mathcal{F}]\!]$.*

Proof. For the binary Kleene product, since $[\![\mathcal{E}]\!] * [\![\mathcal{F}]\!]$ is the least fixed point of $f(X) = [\![\mathcal{F}]\!] + [\![\mathcal{E}]\!] \cdot X$ in $\mathbb{H}_1\Omega$, [7] and $\mathcal{E} * \mathcal{F}$ satisfies

$$\mathcal{F} + \mathcal{E} \cdot (\mathcal{E} * \mathcal{F}) \equiv_{\text{sim}} \mathcal{E} * \mathcal{F}$$

by construction of the sequences of bundle event structures defining $\mathcal{E} * \mathcal{F}$. Therefore, Theorem 6.5.5 and Proposition 6.4.3 imply that $[\![\mathcal{E}]\!] * [\![\mathcal{F}]\!] \sqsubseteq_{\mathbb{H}} [\![\mathcal{E} * \mathcal{F}]\!]$.

Conversely, let $\mu \in [\![\mathcal{E} * \mathcal{F}]\!](s)$ for some initial state $s \in \Omega$. As in the case of Proposition 6.4.3, we assume that μ is computed from a scheduler σ on $\mathcal{E} * \mathcal{F}$. We construct a sequence of schedulers σ_n that "converges" to σ as follows. We set σ_0 to be any element of $\mathbf{Sched}_1(\mathcal{F})$, $\sigma_1(\alpha) = \sigma(\alpha)$ if α is a trace of $\mathcal{F}$ or $\mathcal{E}$, otherwise, we set $\sigma_1(\alpha'\alpha'') = \sigma_0(\alpha'')$ where $\alpha' \in \mathcal{T}_{\max}(\mathcal{E})$ (notice that σ_0 is applied to a different copy of $\mathcal{F}$ but this is not important as event names can be abstracted.). Inductively, we define

$$\sigma_n(\alpha) = \begin{cases} \sigma(\alpha) & \text{if } \alpha \in \mathcal{T}(\underbrace{\mathcal{F} + \mathcal{E} \cdot (\ldots \mathcal{E} \cdot (\mathcal{F} + \mathcal{E}))}_{n \text{ occurrences of } \mathcal{E}}), \\ \sigma_0(\alpha|_F) & \text{otherwise} \end{cases}$$

Again, σ_0 is applied to the $n + 1^{\text{th}}$ copy of $\mathcal{F}$. Indeed, we have

$$\sigma_n \in \mathbf{Sched}_1(\underbrace{\mathcal{F} + \mathcal{E} \cdot (\cdots \mathcal{E} \cdot (\mathcal{F} + \mathcal{E} \cdot \mathcal{F}))}_{n \text{ occurrences of } \mathcal{E}})$$

by construction. On the one hand, the sequence of distributions $\sigma_{n,s}(\mathcal{E})$ forms a

[7] Notice that the least fixed point is in $\mathbb{H}_1\Omega$ but not $\overline{\mathbb{H}}_1\Omega$. The reason is that $[\![\mathcal{E}]\!]$ and $[\![\mathcal{F}]\!]$ are elements of $\mathbb{H}_1\Omega$ because of feasibility and termination.

subset of $[\![\mathcal{E}]\!] * [\![\mathcal{F}]\!](s)$. On the other hand, let $u \in \Omega$ and let us denote

$$\mathcal{T}_{\leq n} = \mathcal{T}(\underbrace{\mathcal{F} + \mathcal{E} \cdot (\ldots \mathcal{E} \cdot (\mathcal{F} + \mathcal{E} \cdot \mathcal{F}))}_{n \text{ occurrences of } \mathcal{E}}).$$

If we denote by φ_n the complete run of σ_n on $\mathcal{E} * \mathcal{F}$, then we have

$$\begin{aligned} |\sigma_s(\mathcal{E})(u) - \sigma_{n,s}(\alpha)(u)| &= \left| \sum_{\alpha \in \mathcal{T}_{\max}(\mathcal{E}*\mathcal{F})} \varphi(\alpha)(u) - \sum_{\alpha \in \mathcal{T}_n \cap \mathcal{T}_{\max}(\mathcal{E}*\mathcal{F})} \varphi_n(\alpha)(u) \right| \\ &= \left| \sum_{\alpha \in \mathcal{T}_{\max}(\mathcal{E}*\mathcal{F}) \setminus \mathcal{T}_{\leq n-1}} (\varphi(\alpha)(u) - \varphi_n(\alpha)(u)) \right| \\ &\leq \sum_{\alpha \in \mathcal{T}_{\max}(\mathcal{E}*\mathcal{F}) \setminus \mathcal{T}_{\leq n-1}} |\varphi(\alpha)(u) - \varphi_n(\alpha)(u)| \end{aligned}$$

The set $\mathcal{T}_{\max}(\mathcal{E} * \mathcal{F}) \setminus \mathcal{T}_{\leq n-1}$ is strictly decreasing, when n increases, because every finite trace of $\mathcal{E} * \mathcal{F}$ belongs to some set $\mathcal{T}_{\leq k}$. Therefore, the last sum above is decreasing to 0. Hence, since Ω is a finite set, the sequence $\sigma_{n,s}(\mathcal{E} * \mathcal{F})$ converges to $\sigma_s(\mathcal{E} * \mathcal{F})$ in $\mathbb{D}\Omega$. Since $[\![\mathcal{E}]\!] * [\![\mathcal{F}]\!](s)$ is topologically closed, we deduce that $\sigma_s(\mathcal{E}) \in [\![\mathcal{E}]\!] * [\![\mathcal{F}]\!](s)$. Therefore, $[\![\mathcal{E} * \mathcal{F}]\!] \sqsubseteq_{\mathbb{H}} [\![\mathcal{E}]\!] * [\![\mathcal{F}]\!]$. ■

The main properties of t-simulation are summarised in the following proposition.

Proposition 6.5.8. *Let $r, r' \in \mathbb{H}_1\Omega$ be two atomic programs and let $\mathcal{E}, \mathcal{F}$ be two bundle event structures with implicit probability, then*

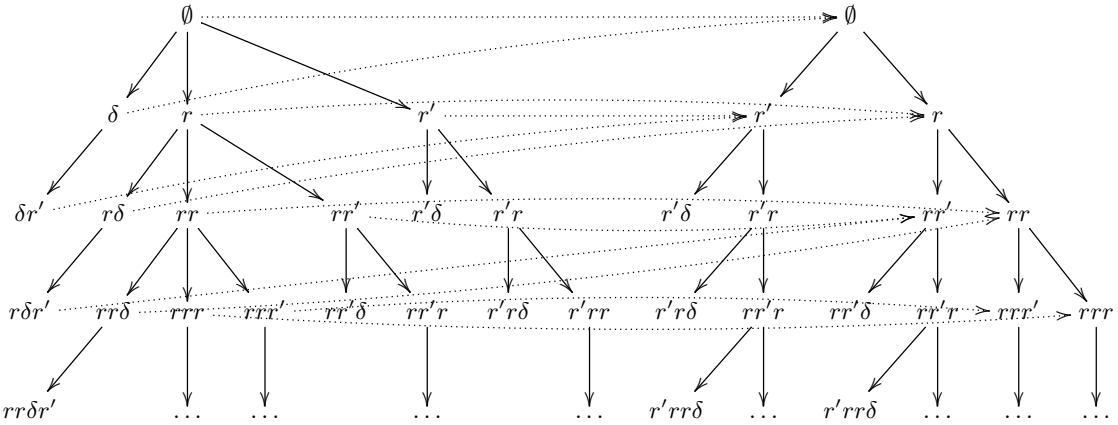
$$r^* \| r^* \sqsubseteq_{\text{sim}} r^* \tag{6.9}$$

$$r^* \| r' \sqsubseteq_{\text{sim}} r * (r' \cdot r^*) \tag{6.10}$$

$$r^* \| (b \cdot \mathcal{E} + c \cdot \mathcal{F}) \sqsubseteq_{\text{sim}} r * (b \cdot (r^* \| \mathcal{E}) \tag{6.11}$$

$$r^* \| (r' \cdot \mathcal{E}) \sqsubseteq_{\text{sim}} r * (r' \cdot (r^* \| \mathcal{E})) \tag{6.12}$$

where $r^* = r * 1$.



The "obvious" arrows, such as an arrow from $r'\delta$ to $r'\delta$, have been left out to keep the picture clear.

Figure 6.3: The t-simulation from $r^*\|r'$ to $r*(r'\cdot r^*)$.

Proof. Let us denote by e_1 and e_2 (resp. e) the events that are labelled by δ in the event structure associated to $r^*\|r^*$ (resp. r^*). Given a trace α of $r^*\|r^*$ that does not contain any of the e_is, we denote by α' unique trace corresponding to α in r^* (i.e. with the same number of events labelled by r).

A t-simulation from $r^*\|r^*$ to r^* is obtained by considering a function f such that

$$f(\alpha) = \begin{cases} (\alpha \setminus \{e_1, e_2\})' & \text{if } e_1 \notin \alpha or e_2 \notin \alpha \\ (\alpha \setminus \{e_1, e_2\})'e & \text{if } e_1, e_2 \in \alpha \end{cases}$$

The Simulation (6.10) is constructed as follows. Let us abstract the event names, i.e., r^k would be a trace where each r is the label of a unique event. Every trace of $r^*\|r'$ is a prefix of $r^m r' r^n \delta$ or $r^m \delta r'$, for some $m, n \geq 0$. Every prefix of either trace corresponds to a unique trace of $r*(r'\cdot r^*)$. For instance, the maximal trace $r^m\delta r'$ is associated to the weakly maximal trace $r^m r'$ of $r*(r'\cdot r^*)$. Figure 6.3 shows an explicit construction of the t-simulation.

The Simulation (6.11)is similar. Every trace of $r^*\|(b\cdot\mathcal{E} + c\cdot\mathcal{F})$ is a prefix of $r^m b\alpha$ or $r^m c\beta$ or $r^m\delta b\gamma$ or $r^m\delta c\zeta$, where $\alpha \in \mathcal{T}(r^*\|\mathcal{E})$, $\beta \in \mathcal{T}(r^*\|\mathcal{F})$, $\gamma \in \mathcal{T}(\mathcal{E})$, $\zeta \in \mathcal{F}$ and $n \geq 0$. Again, prefixes of the first two traces correspond to a unique

trace of $r * (b \cdot (r^* \| \mathcal{E}) + c \cdot (r^* \| \mathcal{F}))$. The maximal trace $r^m \delta b \gamma$ is again mapped to the weakly maximal trace $r^m b \gamma$. Similarly for the fourth case. This indeed results in a t-simulation.

The Simulation (6.12) is constructed as follows. Every trace of $r^* \| (r' \cdot \mathcal{E})$ is a prefix of $r^m r' \alpha$ or $r^m \delta r' \beta$ for some trace $\alpha \in \mathcal{T}(r^* \| \mathcal{E})$ and $\beta \in \mathcal{T}(\mathcal{E})$. We continue as in the previous case. ■

Notice that the binary Kleene star is used on the right hand side of Equation (6.10-6.12). The reason is that the expression $r^* \cdot r' \cdot r^*$ will introduce two events labelled by δ while $r^* \| r'$ will contain only one. The binary Kleene star enables the construction of the t-simulation in the previous proposition.

The Proposition 6.5.8 is used mainly to interleave the right operand r^* systematically through the internal structure of $\mathcal{E}$, while preserving the simulation order. More precisely, these equations are applied to generate algebraic proofs for the reduction of one expression into another where the occurrence of $\|$ is pushed deeper into the sub-expressions (and possibly removed).

6.6 Discussion

This chapter presented a new denotational model for probabilistic concurrent systems. It uses a simulation to refine or rewrite the specification of a concurrent program, mainly using the congruence properties of Proposition 6.5.4 and the inequalities of Proposition 6.5.8. The latter proposition ensures that all interleaving behaviours, in the left hand side of $\sqsubseteq_{\text{sim}}$, are present in the right hand side expression. Equivalently, it can be used to remove every occurrence of the operation ($\|$) in the expression $r^* \| \mathcal{E}$, when $\mathcal{E}$ denotes a probabilistic concurrent program. Multiple applications of Proposition 6.5.8 on that expression will result in a probabilistic sequential program. Hence, all the beautiful techniques developed in [48] are applicable to the $\mathbb{H}_1\Omega$-denotation of that expression.

The constructed semantics space has two main restrictions. (a) The state space Ω is assumed to be finite. In the infinite (countable) case, McIver and Morgan

have presented an extension of the space $\mathbb{H}_1\Omega$ by assuming the compactness of the convex set of distributions $r(s)$ [48]. This allows the modelling of programs having variables ranging over the natural numbers, without any upper bound restriction. With the compactness condition, it should be possible to work with concurrent probabilistic programs whose state space is a countable set. (b) The semantic map $[\![\ ]\!]$ does not model non-terminating behaviours. The sequential behaviours of a program are expressed by schedulers that are generating probability distributions. Total correctness is achieved by considering all possible schedulers, which are generating subdistributions. However, care should be taken in the interpretation of Proposition 6.5.8, where the set $[\![r * 1]\!](s)$ will contain the identically zero subdistribution. In other words, some "fairness condition" should be assumed about the parallel composition $r^*\|\mathcal{E}$, so that r^* indeed behaves sequentially as the Kleene star. These two problems will be part of our future works.

The main motivation of this chapter is to provide a denotational semantics for the extension of rely/guarantee calculus in the next chapter. The two orders $\sqsubseteq$ (Definition 6.4.1) and $\sqsubseteq_{\text{sim}}$ (Definition 6.5.1) are of primordial importance in the definition of a rely/guarantee specification. These two orders are linked by Theorem 6.5.5, which ensures a transition from a concurrent specification to the quantification of all sequential behaviours.

Chapter 7

Probabilistic Rely/guarantee Calculus

7.1 Standard rely/guarantee technique

The algebraic formulation of rely/guarantee was first achieved by Hoare et al. and is based on an algebraic axiomatisation of Hoare triples as follows. A system $\mathcal{E}$ satisfies a postcondition Q when run from precondition P iff the triple $\{P\}\mathcal{E}\{Q\}$ holds with respect to a given semantics. A rely/guarantee specification, denoted by $\{P\ R\}\mathcal{E}\{G\ Q\}$, holds iff the triple $\{P\}R\|\mathcal{E}\{Q\}$ holds and $\mathcal{E}$ *guarantees* G, for a given rely condition R and a guarantee condition G. The precise semantics of this notation will be given in Section 7.3 and it should be noted that, in the non-probabilistic case, R, G (resp. Q) are interpreted as binary relations between current (resp. initial) and next (resp. final) states, while P is a unary predicate.

The semantics of these notations depend on the interpretation of the concurrency operation $\|$ and the meaning given to "$\mathcal{E}$ *guarantees* G". With Jones's original definition, $\|$ interleaves the rely condition R through the internal structure of $\mathcal{E}$ and $\mathcal{E}$ guarantees G iff every (atomic) action in $\mathcal{E}$ satisfies the property G.

In the standard case, Jones [24, 31], as well as others [17, 28], have proven the following composition rule:

$$\frac{\{P\ R\}\mathcal{E}\{G\ Q\} \quad \{P\ R'\}\mathcal{E}'\{G'\ Q'\} \quad G \text{ guarantees } R' \quad G' \text{ guarantees } R}{\{P\ R \cap R'\}\mathcal{E}\|\mathcal{E}'\{G \cup G'\ Q \cap Q', \}}. \quad (7.1)$$

This rule implies if $\mathcal{E}$ and $\mathcal{E}'$ satisfy the premises then $\mathcal{E}\|\mathcal{E}'$ will satisfy both Q and Q' when run in an environment satisfying $R \cap R'$. Moreover, $\mathcal{E}\|\mathcal{E}'$ will guarantee the condition $G \cup G'$.

Example 7.1.1. Consider the program $x := x + 2$ that increments a variable x by 2. If the program is executed in a single step, then it satisfies the guarantee conditions "x never decreases" and "x is always even" (assuming x initially holds an even value). If this program is implemented with the sequence $x := x + 1; x := x + 1$, then the first condition is still preserved while the second condition is not. Consequently, both programs will behave in the same way when run concurrently with another program r that relies on the property "x never decreases". In contrast, if r needs "x is always even" to perform its tasks correctly, then $r\|(x := x + 1; x := x + 1)$ may result in unexpected behaviours because x can hold an odd value in-between the two increments. ■

While the approaches taken in [17, 24, 28, 31] are all related to non-probabilistic programs, the main difference can be found in the respective proofs of the rely/guarantee rules. Jones and Dingel's proofs are directly based on the semantics of concurrent programs, namely, sets of execution traces. In contrast, Hoare et al. provided an algebraic proof which is more elegant and is valid for all models satisfying the basic algebraic axioms. The goal of this chapter is then to follow Hoare et al.'s development but for probabilistic programs.

7.2 Probabilistic rely and guarantee conditions

7.2.1 Using standard conditions

Our first task towards the extension of rely/guarantee to probabilistic systems is to provide a suitable definition for a rely condition that contains sufficient quantitative information about the environment and the components of a system. Such an extension should reduce to the standard properties of rely and guarantee when probabilities are not present.

From a relational point of view, as in Jones's thesis, a guarantee condition expresses a constraint between a state and its successor by running the relation as a nondeterministic program. Therefore, it is important to know when some action is executed atomically or is split into smaller components. For instance, when run in the same environment, a probabilistic choice between $x := x + 1$ and $x := x - 1$ produced from an `if...then...else` clause may behave differently from an atomic probabilistic assignment that assigns $x + 1$ and $x - 1$ to x with the exact same probability.

In this chapter, the implementation of a probabilistic concurrent program is directly denoted by the bundle event structure with implicit probability corresponding to it. In a given implementation, all atomic events are always explicitly stated and, unless otherwise specified, all assignments are considered atomic.[1] They will be used as the labels of the event and every other program construct is translated to the corresponding operation in $(+, \cdot, \|, *)$.

Example 7.2.1. Let us consider the programs

$$\mathcal{E}_1 \stackrel{\text{def}}{=} \texttt{if}(0.7 < \texttt{rand}())\ \texttt{then}\ x := x + 1\ \texttt{else}\ x := x - 1,$$

where `rand`() is a procedure that returns a uniformly distributed random number between 0 and 1, and

$$\mathcal{E}_2 \stackrel{\text{def}}{=} x := x + 1 \oplus_{0.7} x - 1,$$

[1] This provides a higher level look at concurrent programs rather than the low level examples of Chapter 3.

which resolves the probabilistic choice and assigns the value $x+1$ or $x-1$ to x atomically. Thus $\mathcal{E}_2$ will have a single event and $\mathcal{E}_1$ will have four events, corresponding to $0.7 < \texttt{rand}()$, $0.7 \not< \texttt{rand}()$, $x := x+1$ and $x := x-1$. Assume that both programs are run concurrently with $x := 2{*}x$ from an initial state $x = 0$. Assume furthermore that each assignment is executed without any interference between the reading and writing to the variable x. On the one hand, $\mathcal{E}_1 \| (x := 2{*}x)$ may terminate with a value of x in $\{1, -2\}$ with maximal probability 1 because it is bounded from below by the interleaving behaviour

$$\texttt{if}(0.7 < \texttt{rand}())\ \texttt{then}\ x := 2 * x; x := x + 1\ \texttt{else}\ x := x - 1; x := 2 * x.$$

On the other hand, $\mathcal{E}_2 \| (x := 2 * x)$ will yield a state in $\{1, -2\}$ with maximal probability 0.7, because the only possible sequential behaviours are $x := 2 * x; \mathcal{E}_2$ (which yields a state in $\{1, -2\}$ with probability 0.3) and $\mathcal{E}_2; x := 2{*}x$ (that yields a state in $\{1, -2\}$ with probability 0.7). ■

In the standard case, a common example of a guarantee condition for a given program is the reflexive transitive closure with respect to $(\|)$ of the union of all atomic actions in that program. That closure property plays a crucial role in the algebraic proof of Rule 7.1 and is achieved through Proposition 6.5.8.

The transitive closure with respect to $(\cdot)$ is another desirable property. To obtain a probabilistic guarantee condition from a total relation $\rho \subseteq \Omega \times \Omega$, we construct a probabilistic program $r \in \mathbb{H}_1\Omega$ such that

$$r(s) = \{\mu \in \mathbb{D}\Omega \mid \mu(\{s' \mid (s, s') \notin \rho\}) = 0\},$$

where $\mathbb{D}\Omega$ is the set of probability distribution over Ω. Equivalently, r is the convex closure of ρ. The following proposition then follows easily from that construction.

Proposition 7.2.2. *If a relation $\rho \subseteq \Omega \times \Omega$ is transitive then $r \cdot (r + \delta) \sqsubseteq_{\mathbb{H}} r$.*

Proof. Let ρ be a transitive relation, r its associated probabilistic program, $s \in \Omega$ a state and $\mu \in [r \cdot (r + \delta)](s)$. We need to show that $\mu \in r(s)$. By

definition of the sequential composition $(\cdot)$ (Equation 6.3), there exists $\nu \in r(s)$ and a deterministic program $f \sqsubseteq_{\mathbb{H}} (1 + r)$ such that $\mu = f \star \nu$. Let $u \in \Omega$ such that $(s, u) \notin \rho$, we are going to show that $\mu(u) = 0$. We have:

$$\mu(u) = \sum_{t \in \Omega} f(t)(s)\nu(t) = \sum_{t \in \Omega \wedge (s,t) \in \rho} f(t)(u)\nu(t) = \sum_{t \in \Omega \wedge (s,t) \in \rho \wedge (t,u) \in \rho} f(t)(u)\nu(t).$$

The second equality follows from $\nu(t) = 0$ for every $(s, t) \notin \rho$. Similarly, the last equality follows from $f(t)(u) = 0$ for $(t, u) \notin \rho$. The last expression reduces to $\sum_{t \in \Omega \wedge (s,u) \in \rho} f(t)(u)\nu(t)$, by transitivity of ρ, which is an empty sum because $(s, u) \notin \rho$. Therefore, $\mu(u) = 0$ for every $(u, s) \notin \rho$, which is equivalent to $\mu \in r(s)$. ■

7.2.2 Limitations of standard rely/guarantee conditions

The convex closure of a relation provided by the previous subsection sometimes provides a very general rely condition that is not very useful in the probabilistic case.

In practice, a probabilistic assignment is considered atomic and the correctness of many protocols is based on that crucial assumption. That is, the random choice and the writing of the chosen value into x is assumed to happen instantaneously and no other program can modify x during and in-between these two operations. Thus, probabilistic rely and guarantee conditions need to capture the probabilistic information in such an assignment.

Example 7.2.3. Let us write $x := \texttt{uniform}(0, x)$ the program that assigns a random integer between two integers 0 and x to the variable x. A probabilistic guarantee condition for that assignment is obtained from the probabilistic program r such that

$$r(x) = \left\{ \mu \mid \mu(\{0, n\}) \geq \frac{1}{n+1} \right\}. \tag{7.2}$$

The condition r specifies the convex set of all probabilistic deterministic programs

whose atomic actions establish a state in $\{0, n\}$ with probability at least $\frac{1}{n+1}$. Notice that the exact probability for the assignment to yield a state in $\{0, n\}$ is $\frac{1}{x+1}$. The condition r is transitive and "completely" probabilistic. ■

In practice, constructing a useful transitive probabilistic rely/guarantee condition is difficult but the standard technique is still valid. That is, the strongest guarantee condition of a given program is the nondeterministic choice of all one-step actions found in that program. This construction was introduced by Jones [31] and later refined by others [17, 24, 27] and in this thesis, is extended to the probabilistic case. In general, we define:

Definition 7.2.4. *A probabilistic rely or guarantee condition R is a bundle event structure with implicit probability such that $R \| R \sqsubseteq_{\text{sim}} R$.*

In particular, the bundle event structure $r^* = r * 1$, where r is an atomic program, is a rely condition. This illustrates the idea that a rely condition specifies an environment that can stutter or execute a sequence of actions that are bounded by r. The targeted properties of rely and guarantee conditions are then ensured by Proposition 6.5.8.

7.3 Probabilistic rely/guarantee rules

In this section, we develop the rely guarantee rules governing programs involving probability and concurrency. An example is given by Rule 7.1, which allows us to compositionally check the safety properties of the subsystems and infer the correctness of the whole system. Hence, we will provide a probabilistic counterpart for that rule.

In the previous chapter, we have developed the mathematical foundations that are needed for our interpretation of Hoare triple and *guarantee* relation, namely, the sequential refinement in Definition 6.4.1 and simulation relation of Definition 6.5.1. Following [27], a rely/guarantee quintuple $\{P\ R\}\mathcal{E}\{G\ Q\}$ is

valid iff

$$P \cdot (R \| \mathcal{E}) \sqsubseteq Q \qquad \text{and} \qquad \mathcal{E} \sqsubseteq_{\text{sim}} G,$$

where $P, R, \mathcal{E}, G$ and Q are programs denoted by bundle event structures with implicit probability.

The terms R and G specify how the component $\mathcal{E}$ interacts with its environment. As we have discussed in the previous section, rely and guarantee conditions are obtained by taking r^* for some atomic probabilistic program r. Therefore, $\mathcal{E} \sqsubseteq_{\text{sim}} r^*$ implies that all actions carried by events in $\mathcal{E}$ are either stuttering or satisfying the specification r. This corresponds to the standard approach of Jones [24, 31].

The following rules are probabilistic extensions of the related rely/guarantee rules developed in [28].

7.3.1 Atomic action

The rely/guarantee rule for atomic statements is provided by the equation

$$r^* \| r' \sqsubseteq_{\text{sim}} r * (r' \cdot r^*)$$

of Proposition 6.5.8. This equation shows that a (background) program satisfying the rely condition r will not interfere with the low level operations involved in the atomic execution of r'. The programs will be interleaved.

7.3.2 Conditional statement

The rely/guarantee rule for conditional statement is provided by the equation

$$r^* \| (b \cdot \mathcal{E} + c \cdot \mathcal{F}) \sqsubseteq_{\text{sim}} r * (b \cdot (r^* \| \mathcal{E}) + c \cdot (r^* \| \mathcal{F}))$$

of Proposition 6.5.8. This equation shows how a rely condition r^* distributes through branching structures. In this rule, the tests b and c are assumed to be

atomic and their disjunction is be always *true* (this is necessary for feasibility). This assumption may be too strong in general because b may involve the reading of some large data that is too expensive to be performed atomically. However, we may assume that such a reading is done before the guard b is checked and the non-atomic evaluation of the variables involved in b may be assigned to some auxiliary variable that is then checked atomically by b.

7.3.3 Prefixing

The sequential rely/guarantee rule for a probabilistic program expressed using prefixing. That is,

$$r^* \| (r' \cdot \mathcal{E}) \sqsubseteq_{\text{sim}} r * (r' \cdot (r^* \| \mathcal{E}).$$

It tells us that a rely condition r^* can be distributed through a prefixing operation. In other words, the program r' and $\mathcal{E}$ should tolerate the same rely condition in order to prove any meaningful property of $r \cdot \mathcal{E}$. This is mainly a consequence of our interpretation of $\|$ where no synchronisation is assumed.

7.3.4 Concurrent execution

In Rule 7.1, the concurrent composition $\mathcal{E} \| \mathcal{E}'$ requires an environment that satisfies $R \cap R'$ to establish the postcondition $Q \cap Q'$. However, such an intersection is not readily accessible at the structural level of event structures. Therefore, the most general probabilistic extension of Rule 7.1 which applies in the setting of bundle event structures is:

$$\frac{\{P\ R\}\mathcal{E}\{G\ Q\} \qquad \{P\ R'\}\mathcal{E}'\{G'\ Q'\} \qquad G \sqsubseteq_{\text{sim}} R' \qquad G' \sqsubseteq_{\text{sim}} R}{\{P\ R''\}\mathcal{E}\|\mathcal{E}'\{G\|G'\ Q\}}, \tag{7.3}$$

where R'' is a rely condition such that $R'' \sqsubseteq_{\text{sim}} R$ and $R'' \sqsubseteq_{\text{sim}} R'$. The proof of this rule is exactly the same as in [27, 52].

We have $R'' \sqsubseteq_{\text{sim}} R$, $\mathcal{E}' \sqsubseteq_{\text{sim}} R$, $R \| R \sqsubseteq_{\text{sim}} R$, therefore Equations 6.5 and 6.6

imply

$$R''\|(\mathcal{E}'\|\mathcal{E}) \sqsubseteq_{\text{sim}} R\|(R\|\mathcal{E}) \sqsubseteq_{\text{sim}} R\|\mathcal{E},$$

and we obtain $P \cdot R''\|(\mathcal{E}'\|\mathcal{E}) \sqsubseteq_{\text{sim}} P \cdot (R\|\mathcal{E})$ by Equation (6.7). It follows from Theorem 6.5.5 that $P \cdot R''\|(\mathcal{E}'\|\mathcal{E}) \sqsubseteq Q$.

The conclusion does not contain any occurrence of Q', but by symmetry, it is also valid if Q is substituted with Q'. The combined rely condition R'' is constructed such that it is below R and R'. Indeed, if R, R' have a greatest lower bound with respect to $\sqsubseteq_{\text{sim}}$, then R'' is that lower bound so that the strengthening of the rely is not too strong.

The above rule can be specialised by considering rely/guarantee conditions of the form r^*, where r is an atomic probabilistic program.

The following rule is expressed in exactly the same way as the standard case [28]. This is due to the fact that probabilities are implicit.

Proposition 7.3.1. *The following rule is valid in BES*

$$\frac{\{P\ r_1^*\}\mathcal{E}_1\{g_1^*\ Q_1\} \qquad \{P\ r_2^*\}\mathcal{E}_2\{g_2^*\ Q_2\} \qquad g_1 \sqsubseteq_{\mathbb{H}} r_2 \qquad g_2 \sqsubseteq_{\mathbb{H}} r_1}{\{P\ (r_1 \cap r_2)^*\}\mathcal{E}_1\|\mathcal{E}_2\{(g_1 + g_2)^*\ Q_1\}} \tag{7.4}$$

where $r, r', g, g' \in \mathbb{H}_1\Omega$, $g + g'$ is the nondeterministic choice on $\mathbb{H}_1\Omega$ and $\cap$ is the pointwise intersection of two probabilistic denotations of programs.

Proof. This follows from substituting R and G by respectively r^* and g^* in Rule 7.3. Moreover $g^*\|g'^* \sqsubseteq_{\text{sim}} (g + g')^*$ holds because $(g + g')^*\|(g + g')^* \sqsubseteq_{\text{sim}} (g + g')^*$ (Proposition 6.5.8 Equation 6.9). ■

Recall that the nondeterministic choice of $\mathbb{H}_1\Omega$ is obtained by the pointwise union followed by the necessary closure properties for the elements of $\mathbb{H}_1\Omega$. The intersection $r \cap r'$ is obtained by pointwise intersection which also preserves the same closure properties.

Rule 7.4 can further be specialised to give us some statistical information about the system $\mathcal{E}\|\mathcal{E}'$. Our second formulation shows how the probabilities of correctness are combined.

Given a BES $\mathcal{E}$, an initial state s satisfying the precondition, a subset $O \subseteq \Omega$ and $p \in [0,1]$, we write $[\![\mathcal{E}]\!](s)(O) \geq p$, if for every $\mu \in [\![\mathcal{E}]\!](s)$, $\mu(O) \geq p$ i.e. the correctness property O is established by $\mathcal{E}$ with probability *at least* p.

We assume that the precondition P is standard, i.e., a set of initial states. The corresponding probabilistic program maps $s \notin P$, to $\emptyset \subseteq \mathcal{D}\Omega$ and $s \in P$ to $\{\delta_s\}$.

Proposition 7.3.2. *For every $s \in P$ and subsets $O_1, O_2 \subseteq \Omega$, we have*

$$\frac{[\![r_1^* \| \mathcal{E}_1]\!](s)(O_1) \geq p_1 \quad [\![r_2^* \| \mathcal{E}_2]\!](s)(O_2) \geq p_2 \quad \mathcal{E}_1 \sqsubseteq_{\text{sim}} g^* \sqsubseteq_{\text{sim}} r_2^* \quad \mathcal{E}_2 \sqsubseteq_{\text{sim}} g'^* \sqsubseteq_{\text{sim}} r_1^*}{[\![(r_1 \cap r_2)^* \| \mathcal{E}_1 \| \mathcal{E}_2]\!](s)(O_1 \cap O_2) \geq p_1 + p_2 - q \qquad \mathcal{E}_1 \| \mathcal{E}_2 \sqsubseteq_{\text{sim}} (g + g')^*}$$

where $q \geq \sup_{\mu \in [\![(r_1 \cap r_2)^* \| \mathcal{E}_1 \| \mathcal{E}_2]\!](s)} \mu(O_1 \cup O_2)$.

Proof. Let $\mu \in [\![(r_1 \cap r_2)^* \| \mathcal{E} \| \mathcal{E}_2]\!](s)$, we need to show that $\mu(O_1 \cap O_2) \geq p_1 + p_2 - q$ with the above definition of p_1, p_2 and q.

Let us define Q_1 to be the (single event) BES associated to the probabilistic program u_1 such that if $s \in P$, then $u_1(s) = \{\mu \mid \mu(O) \geq p\}$ else $u_1(s) = \emptyset$. Similarly, we define Q_2. Then the premises imply $P \cdot (r_1^* \| \mathcal{E}_1) \sqsubseteq Q_1$ and $P \cdot (r_2^* \| \mathcal{E}_2) \sqsubseteq Q_2$. By Proposition 7.3.1, we have

$$[\![P \cdot ((r_1 \cap r_2)^* \| \mathcal{E}_1 \| \mathcal{E}_2)]\!] \sqsubseteq_{\mathbb{H}} [\![Q_1]\!] \qquad \text{and} \qquad [\![P \cdot ((r_1 \cap r_2)^* \| \mathcal{E}_1 \| \mathcal{E}_2)]\!] \sqsubseteq_{\mathbb{H}} [\![Q_2]\!].$$

Therefore $\mu(O_1) \geq p_1$ and $\mu(O_2) \geq p_2$. Modularity of finite measures implies that $\mu(O_1 \cap O_2) + \mu(O_1 \cup O_2) = \mu(O_1) + \mu(O_2) \geq p_1 + p_2$. Hence, $\mu(O_1 \cap O_2) \geq p_1 + p_2 - \mu(O_1 \cup O_2) \geq p_1 + p_2 - \sup_{\mu \in [\![(r_1 \cap r_2)^* \| \mathcal{E}_1 \| \mathcal{E}_2]\!](s)} \mu(O_1 \cup O_2)$. ■

The lower bound given in this proposition is the best we can achieve in the most general case. Consequently, any improvement in that bound has to depend on particular properties of the system $\mathcal{E}_1 \| \mathcal{E}_2$ (e.g. commutativity of atomic actions). Proposition 7.3.4 provides an example of a such case.

Example 7.3.3. Let $\mathcal{E}_1$ (resp. $\mathcal{E}_2$) be the event structure associated to the atomic assignment $x := x + 1 \oplus_p \texttt{skip}$ (resp. $x := 1$) on the state space $\mathbb{Z}_2 = \{0, 1\}$ which represents the parity of integers, and where $p \leq 1/2$. Let the respective rely

conditions be obtained from r_1 and r_2 such that $r_1(1) = \{\delta_1\}, r_1(0) = \overline{\{\delta_1, \delta_0\}}$ and $r_2(x) = \mathbb{D}_1\mathbb{Z}_2$ for every state $x \in \mathbb{Z}_2$. Let us assume that we run the program $\mathcal{E}_1 \| \mathcal{E}_2$ from the initial state 0. We have $[\![r_1^* || \mathcal{E}_1]\!](0)(\{1\}) \geq \inf(p, 1-p) = p$ and $[\![r_2^* || \mathcal{E}_2]\!](0)(\mathbb{Z}_2) \geq 1$. We can verify easily that $\mathcal{E}_2 \sqsubseteq_{\text{sim}} r_2^*$ and $\mathcal{E}_1 \sqsubseteq_{\text{sim}} r_1^*$ and by applying Proposition 7.3.2, we have $[\![r_1 \| \mathcal{E}_1 \| \mathcal{E}_2]\!](x)(\{1\}) \geq p$. However, it is clear that $[\![\mathcal{E}_1 \| \mathcal{E}_2]\!](x)(\{1\}) = p$, i.e., the lower bound given by Proposition 7.3.2 cannot be improved in this example. ■

The last rule we establish for the case of concurrency explores some special properties of the studied system. We assumed that both components $\mathcal{E}_1$ and $\mathcal{E}_2$ have the same rely condition, that all sequential behaviours of $\mathcal{E}_1 \| \mathcal{E}_2$ are subsumed in $\mathcal{E}_1 \cdot \mathcal{E}_2$ and that O_1 is an *invariant* of $\mathcal{E}_2$.

Proposition 7.3.4. *For every $s \in P$ and subsets $O_1, O_2 \subseteq \Omega$, if the sequential refinement $r^* \| \mathcal{E}_1 \| \mathcal{E}_2 \sqsubseteq (r^* \| \mathcal{E}_1) \cdot (r^* \| \mathcal{E}_2)$ and, for every $t \in O_1$, $[\![r^* \| \mathcal{E}_2]\!](t)(O_1) = 1$ hold, then the rule*

$$\frac{\begin{array}{cc} [\![r^* \| \mathcal{E}_1]\!](s)(O_1) \geq p_1 & \inf_{t \in O_1} [\![r^* \| \mathcal{E}_2]\!](t)(O_2) \geq p_2 \\ \mathcal{E}_1 \sqsubseteq_{\text{sim}} g^* \sqsubseteq_{\text{sim}} r^* & \mathcal{E}_2 \sqsubseteq_{\text{sim}} g'^* \sqsubseteq_{\text{sim}} r^* \end{array}}{\begin{array}{cc} [\![r^* \| \mathcal{E}_1 \| \mathcal{E}_2]\!](s)(O_1 \cap O_2) \geq p_1 p_2 & \mathcal{E}_1 \| \mathcal{E}_2 \sqsubseteq_{\text{sim}} (g + g')^* \end{array}}$$

is valid.

Proof. We reason as follows:

$$
\begin{array}{ll}
& [\![r^* \| \mathcal{E}_1 \| \mathcal{E}_2]\!](s)(O_1 \cap O_2) \\
\geq & r^* \| \mathcal{E}_1 \| \mathcal{E}_2 \sqsubseteq (r^* \| \mathcal{E}_1) \cdot (r^* \| \mathcal{E}_2). \\
& \inf_{\mu \in [\![r^* \| \mathcal{E}_1]\!](s)} \inf_{f \sqsubseteq_{\mathbb{H}} [\![r^* \| \mathcal{E}_2]\!]} \sum_{t \in \Omega} f(t)(O_1 \cap O_2)\mu(t) \\
\geq & O_1 \subseteq \Omega,\ [\![r^* \| \mathcal{E}_2]\!](t)(O_1) = 1 \text{ for every } t \in O_1 \text{ and modularity of measure.} \\
& \inf_{\mu \in [\![r^* \| \mathcal{E}_1]\!](s)} \inf_{f \sqsubseteq_{\mathbb{H}} [\![r^* \| \mathcal{E}_2]\!]} \sum_{t \in O_1} f(t)(O_2)\mu(t) \\
\geq & \text{Expectation is monotonic.} \\
& \inf_{\mu \in [\![r^* \| \mathcal{E}_1]\!](s)} \inf_{f \sqsubseteq_{\mathbb{H}} [\![r^* \| \mathcal{E}_2]\!]} \sum_{t \in O_1} \inf_{t' \in O_1} f(t')(O_2)\mu(t) \\
= & \text{The factor } \inf_{t' \in O_1} f(t')(O_2) \text{ is independent of } t. \\
& \inf_{\mu \in [\![r^* \| \mathcal{E}_1]\!](s)} \inf_{f \sqsubseteq_{\mathbb{H}} [\![r^* \| \mathcal{E}_2]\!]} \inf_{t' \in O_1} f(t')(O_2)\mu(O_1) \\
= & \text{Swapping the second and last inf (†).} \\
& \inf_{\mu \in [\![r^* \| \mathcal{E}_1]\!](s)} \inf_{t' \in O_1} [\![r^* \| \mathcal{E}_2]\!](t')(O_2)\mu(O_1) \\
= & \text{Both infs are attained for some value of } t' \text{ and } \mu\text{, we can separate them.} \\
& [\![r^* \| \mathcal{E}_1]\!](s)(O_1) \inf_{t' \in O_1} [\![r^* \| \mathcal{E}_2]\!](t')(O_2) \\
\geq & \text{Definitions of } p_1 \text{ and } p_2. \\
& p_1 p_2
\end{array}
$$

(†) To ensure the replacement of the internal inf with $[\![r^* \| \mathcal{E}_2]\!](t')(O_2)$ in the presence of the multiplied constant $\mu(O_1)$, we can use the fact that the infinimum is attained for some deterministic refinement $f \sqsubseteq_{\mathbb{H}} [\![r^* \| \mathcal{E}]\!]$ because Ω if a finite set and $[\![r^* \| \mathcal{E}]\!](s)$ is a compact subset of $\mathbb{R}^{\Omega}$.

The guarantee part has been established in Proposition 7.3.1. ■

7.3.5 While loop

A while program is modelled using the binary Kleene star. The idea is to unfold the whole structure of the loop to obtain an event structure with infinitely many events. Proposition 6.5.8 can be applied on the unfolded structure to distribute the rely condition. That is, we write

$$\begin{aligned} r^* \| ((b \cdot \mathcal{E}) * c) &= r^* \| (c + b \cdot \mathcal{E} \cdot (b \cdot \mathcal{E} * c)) \\ &\sqsubseteq_{\text{sim}} r * (c \cdot r^* + b \cdot (r^* \| [\mathcal{E} \cdot (b \cdot \mathcal{E} * c)]))^2 \\ &\sqsubseteq_{\text{sim}} \cdots . \end{aligned}$$

The sequential correctness is achieved by the usual generation of probability distributions using terminating schedulers, or using the semantic map $[\![\ ]\!]$ and then apply other algebraic techniques. A bounded loop, such as a for loop, should be modelled using a sequence of sequential compositions or prefixing.

7.4 R,G-Preorder and extension to action refinement

The rely/guarantee rules that were developed in the previous section are most helpful when the implementations of components of the system are given in full. However, they do not tell us when we are allowed to refine parts of the given implementation into sequential components.

Example 7.1.1 provides a good illustration of this problem. It is obvious that $x := x + 2$ and $x := x + 1; x := x + 1$ have the exact same sequential behaviour. However, the condition "x is always even" is not satisfied by the second program. Therefore, it is unsafe to replace the implementation $x := x+2$ with $x := x+1; x := x + 1$ when it is part of an environment that is bound by the rely condition "x is always even". Hence, the guarantee condition should be preserved by the new implementation.

Definition 7.4.1. *Let r, g be two probabilistic programs, we say that $\mathcal{F}$ is a (r, g)-refinement of $\mathcal{E}$, written $\mathcal{F} \sqsubseteq_{r,g} \mathcal{E}$, if for every event structures P and Q, we*

have

$$\{P\ r^*\}\mathcal{E}\{g^*\ Q\} \Rightarrow \{P\ r^*\}\mathcal{F}\{g^*\ Q\}.$$

This is indeed a straightforward generalisation of the preorder obtained from the standard Hoare triple which reduces to the sequential refinement order $\sqsubseteq$.

When the rely/guarantee parameters are allowed to be arbitrary event structures then $(\forall R, G : \mathcal{E} \sqsubseteq_{R,G} \mathcal{E}') \Leftrightarrow \mathcal{E} \sqsubseteq_{\text{sim}} \mathcal{E}'$, which is also our interpretation of the guarantee satisfaction relation. However, Definition 7.4.1 provides a weaker comparison that allows us to establish a relationship between programs up to some assumption about the environment.

It follows directly from Definition 7.4.1 that $\sqsubseteq_{R,G}$ is a preorder and if $\mathcal{F} \sqsubseteq_{r,g} \mathcal{E}$ then $\mathcal{E}$ can be replaced by $\mathcal{F}$ in all the conclusions of the rely/guarantee rules of the previous sections. More importantly, since the concurrency operation provided by the event structure framework supports refinement of an action with another bundle event structure, it is then possible to replace any atomic action with a composition of actions that preserve the guarantee condition.

Example 7.4.2. Let us show that the assignment $x := \texttt{uniform}(0, x)$ can be refined to $y := x; x := \texttt{uniform}(0, y)$ while preserving the overall correctness of the original assignment. To ensure that both programs have the same underlying state space, we assume that y is present in the first assignment but is unused. It is then clear that

$$y := x; x := \texttt{uniform}(0, y) \sqsubseteq_{r^*,r^*} x := \texttt{uniform}(0, x),$$

where r is the probabilistic program such that

$$r((x, y)) = \left\{\mu \in \mathbb{D}(\{0, n\}^2) \mid \mu(\{(0, y), (n, y) \mid y \in \{0, n\}\}) \geq \frac{1}{n+1}\right\}$$

■

We conclude this chapter with an example that uses the developed rely/guarantee techniques of this chapter. We study the probabilistic correctness of an adaptation

of the Eratosthenes sieve, where a measurable fault is introduced when removing a multiple of a given integer. The goal is to obtain a sensible bound for the probability of removing all composite numbers.

7.5 Concurrent Eratosthenes sieve

Let $n \geq 2$ be a natural number and $s_0 = \{2, 3, \ldots, n\}$. We present a variant of the concurrent Eratosthenes sieve that is originally due to Jones [31]. To obtain a quantitative system, we modify Jones's example by introducing a probabilistic failure. That is, the atomic action that removes a particular composite number may fail with a given probability. Indeed, the rely and guarantee conditions we generate are closely related to Jones's originals which have been modified to account for quantitative properties.

For each integer $i \in [2, \sqrt{n}]$, we consider a program $\texttt{thd}_i$ that sequentially removes all (strict) multiples of i from the shared set variable s with a fixed probability p. More precisely, each thread $\texttt{thd}_i$ is be implemented as the following program:

```
for(j = 2 to n/i)
    u_{i,j} : skip ⊕_p (remove i*j from s);
```

where n/i is the integer division of n by i. Each $u_{i,j}$ can be seen as a faulty action that removes the product ij from the current value of s with probability p. The state space of each atomic deterministic program $u_{i,j}$ is $\Omega = \{s \mid s \subseteq s_0\}$. In $\mathbb{H}_1\Omega$, $u_{i,j}$ is defined by $u_{i,j}(s) = (1-p)\delta_s + p\delta_{s\setminus\{ij\}}$. The whole system is specified by the concurrent execution

$$\texttt{thd}_2 \| \ldots \| \texttt{thd}_{\sqrt{n}} = \|_{i=2}^{\sqrt{n}} (u_{i,2} \cdots u_{i,n/i}).$$

where, in the sequel, $\sqrt{n}$ is computed without decimal.

Let us denote by $\pi = \{2, 3, 5 \ldots, p_n\}$ the set of prime numbers less than or equal to n (the largest being p_n). Our goal is to compute the minimal probability

that the final state is π, after executing the threads $\texttt{thd}_i$ concurrently, from the initial state s_0.

Let us denote by $O_{i,j} = \{s \mid ij \notin s\}$, $O_i = \cap_{j=2}^{n/i} O_{i,j}$ and r be the probabilistic program such that $r(s)$ is the convex closure of $\{\delta_{s'} \mid s' \subseteq s\}$. It is clear that $u_{i,j} \sqsubseteq_{\mathbb{H}} r$, for every i, j, and thus $\texttt{thd}_i \sqsubseteq_{\text{sim}} r^*$. Multiple applications of the prefix-case of Proposition 6.5.8 give us

$$r^* \| \texttt{thd}_i \sqsubseteq_{\text{sim}} r * (u_{i,2} \cdot (r * (u_{i,3} \cdot (\ldots r * (u_{i,n/i} \cdot r^*)))).$$

Since the right multiplication $X \mapsto X \cdot r$, by any program $r' \in \mathbb{H}_1\Omega$, is the lower adjoint in a Galois connection [48], the fixed point fusion theorem [1] implies that

$$r * (u_{i,2} \cdot (r * (u_{i,3} \cdot (\ldots r * (u_{i,n/i} \cdot r^*))))) = r^* \cdot u_{i,2} \cdot r^* \cdot u_{i,3} \cdot \ldots r^* \cdot u_{i,n/i} \cdot r^*$$

(the equivalence is in $\mathbb{H}_1\Omega$). Thus,

$$r^* \| \texttt{thd}_i \sqsubseteq r^* \cdot u_{i,2} \cdot r^* \cdot u_{i,3} \cdot \ldots r^* \cdot u_{i,n/i} \cdot r^*$$

follows from Theorem 6.5.5. The right hand side explicitly states the interleaving of the rely condition r^* in-between the atomic executions in $\texttt{thd}_i$ as in [24].

Moreover, since r is the probabilistic version of a transitive standard relation, Proposition 7.2.2 implies that $r \cdot (r + \delta) \sqsubseteq_{\mathbb{H}} r$. But $\mathbb{H}_1\Omega$ is a probabilistic Kleene algebra [53], therefore the right induction law (2.14) (Chapter 2 Figure 2.1) implies that $r \cdot r^* \sqsubseteq_{\mathbb{H}} \delta + r$ and therefore $r^* = \delta + r \cdot r^* \sqsubseteq_{\mathbb{H}} \delta + r$. The converse refinement also holds because $r \sqsubseteq_{\mathbb{H}} r^*$, $\delta \sqsubseteq_{\mathbb{H}} r^*$ and $(+)$ is idempotent. That reduction of r^* to $\delta + r$ shows the main use of a transitive rely condition. Therefore,

$$r^* \| \texttt{thd}_i \sqsubseteq (\delta + r) \cdot u_{i,2} \cdot (\delta + r) \cdot u_{i,3} \cdot \ldots (\delta + r) \cdot u_{i,n/i} \cdot (\delta + r).$$

Let us first focus on computing $[\![u_{i,j} \cdot (1 + r)]\!](s)(O_{i,j})$ and $[\![(\delta + r) \cdot u_{i,j}]\!](s)(O_{i,j})$.

By Proposition 6.4.3, we have $[\![u_{i,j} \cdot (\delta + r)]\!] = [\![u_{i,j}]\!] \cdot [\![(\delta + r)]\!] = u_{i,j} \cdot (\delta + r)$, and

$$[u_{i,j} \cdot (\delta + r)](s)(O_{i,j}) = \inf_{f \sqsubseteq_{\mathbb{H}} \delta + r} \sum_{t \in \Omega} f(t)(O_{i,j}) u_{i,j}(s)(t)$$

The definition $u_{i,j} = (1-p)\delta_s + p\delta_{s \setminus \{ij\}}$ implies that the sum is restricted to the set $\{t \mid t = s \vee t = s \setminus \{ij\}\}$. Therefore, we have

$$[u_{i,j} \cdot (\delta + r)](s)(O_{i,j}) = \begin{cases} \inf_{f \sqsubseteq_{\mathbb{H}} \delta + r} f(s)(O_{i,j}) & \text{if } ij \notin s \\ \inf_{f \sqsubseteq_{\mathbb{H}} \delta + r} (1-p) f(s)(O_{i,j}) + p f(s \setminus \{ij\})(O_{i,j}) & \text{if } ij \in s \end{cases}$$

The first case evaluates to 1, because the output state obtained by executing r from s will not contain ij. Therefore, we obtain lower bound $[u_{i,j} \cdot (\delta + r)](s)(O_{i,j}) \geq p$. A similar reasoning can be applied to prove that $[(\delta + r) \cdot u_{i,j}](s)(O_{i,j}) \geq p$.

More generally, for every $j \neq j'$, $O_{i,j}$ is an invariant for $u_{i,j'}$, i.e., $u_{i,j'}(s)(O_{i,j}) = 1$ for every $s \in O_{i,j}$. Therefore, the reasoning in the proof of Proposition 7.3.4 can be used to show that

$$[u_{i,j} \cdot (\delta + r) \cdot u_{i,j+1}](s)(O_{i,j} \cap O_{i,j+1}) \geq p^2$$

By induction, we obtain the expected lower bound

$$[\![r^* \| \mathtt{thd}_i]\!](s_0)(O_i) \geq p^{n/i-1}$$

Applying Proposition 7.3.2 (with $q = 1$ because r can be used to establish any particular state in $O_2 \cup O_3$) on $\mathtt{thd}_2 \| \mathtt{thd}_3$ gives

$$[\![r^* \| \mathtt{thd}_2 \| \mathtt{thd}_3]\!](s_0)(O_2 \cap O_3) \geq p^{n/2-1} + p^{n/3-1} - 1.$$

In fact, by repeating the process $\sqrt{n} - 1$ times (Corollary 7.3), we deduce that

$$[\![r^* \|_{i=2}^{\sqrt{n}} \mathtt{thd}_i]\!](s_0)(\cap_{i=2}^{\sqrt{n}} O_i) \geq \sum_{i=2}^{\sqrt{n}} p^{n/i-1} - (\sqrt{n} - 2) = f(p, n).$$

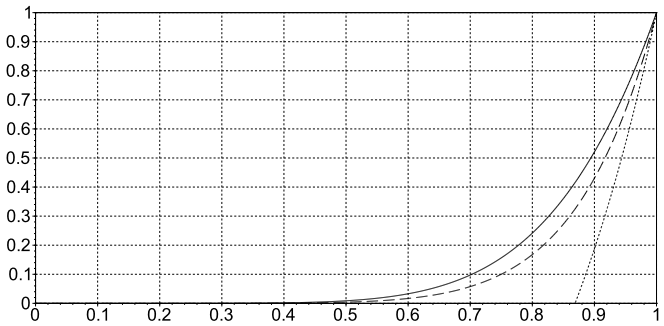


Figure 7.1: Comparison of the quantity $f(p, 15)$ (dotted), $g(p, 15)$ (dashed) and the actual probability $p^{10} + 4p^9(1-p) + 4p^8(1-p)^2$ (solid).

Firstly, the lower bound $f(p, n)$ is not usually the best approximation for the probability that the system establishes $\cap_{i=2}^{\sqrt{n}} O_i$, mainly because of the presence of r^*. However, it is clear that $\lim_{p\to 1} f(p, n) = f(1, n) = 1$.

In the particular case of $n = 15$, $f(p, 15) = p^6 + p^4 - 1$. A simple numerical calculation shows that $f(p, 15)$ gives a positive lower bound when $p \geq 0.868$, the exact probability being $p^{10} + 4p^9(1-p) + 4p^8(1-p)^2$.

We can use Proposition 7.3.4 to obtain a better lower bound. It is clear that O_i is an invariant for every $\texttt{thd}_j$ (for $j \neq i$) and that all actions $u_{i,j}$ (sequentially) commute with each other. Therefore, the assumption of Proposition 7.3.4 can be established easily. Inductively, we have

$$[\![r^* \| _{i=2}^{\sqrt{n}} \texttt{thd}_i]\!](s_0)(\cap_{i=2}^{\sqrt{n}} O_i) \geq p^{n/2-1} \left(\frac{p^{n/3-1}}{p^{n/6}}\right) \left(\frac{p^{n/4-1}}{p^{n/4-1}}\right) \left(\frac{p^{n/5-1}}{p^{n/10} + p^{n/15}}\right) \cdots = g(p, n)$$

In the particular case of $n = 15$, $g(p, 15) = p^8$. A graphic comparison of f, g and the actual probability is displayed in Figure 7.1 for this particular value of n.

Secondly, $\emptyset \in \cap_{i=2}^{\sqrt{n}} O_i$ which means that $r^* \| _{i=2}^{\sqrt{n}} \texttt{thd}_i$ can establish $s = \emptyset$ with positive probability. This issue is resolved by using proper guarantee relation such as $u_{i,j}$ never removes i and refining r^* to 1 in the concurrent execution. Therefore, $\|_{i=2}^{m} \texttt{thd}_i$ never removes any prime numbers i.e. any element of $\cap_{i=2}^{\sqrt{n}} O_i$ that does not contain all the prime numbers less than or equal to n are impossible states.

Chapter 8

Conclusion

The present thesis aimed to develop an algebraic framework for probabilistic concurrent systems. The goal was to provide an algebraic framework for a uniform treatment of concurrent programs that exhibit probabilistic behaviours. The proposed solution was divided into two parts: (a) a suitable "merging" of concurrent and probabilistic Kleene algebra that unifies these two computational behaviours into a single, algebraic setting, and (b) an extension of rely/guarantee rules for the verification of probabilistic concurrent programs. Most of the rules are again expressed and proven using algebraic properties. Both solutions depended heavily on the use of algebraic techniques. Therefore, this thesis does provide an answer to the question asked in the introduction:

> *Can we verify algebraically and compositionally probabilistic programs in the presence of interference?*

Results

Firstly, the development of probabilistic concurrent Kleene algebra required a deep understanding of both the probabilistic and the concurrent components. A completeness result, about the equational theory of probabilistic Kleene algebra,

was conjecture by Takai and Furusawa in [79]. Our first result was to prove that that conjecture holds for continuous probabilistic Kleene algebra and the set of automata modulo simulation equivalence. This then entailed a decision procedure, refined using a minimisation technique that preserves simulation, for the equality of two terms in a continuous probabilistic Kleene algebra.

Secondly, the other component, concurrent Kleene algebra, was also shown to be sound with respect to a bundle event structure model. An important result of that work is the characterisation of lposet semantics, of a bundle event structure, with schedulers and finishers. This characterisation was achieved by the definition of the resolution of a bundle event structure with respect to a pair of scheduler and finisher. In particular, we have shown that a special kind of interaction between schedulers and finishers is enough to generate "full interaction". Moreover, it was shown how a finishers can be used to define "acceptable behaviours", in the sense that unacceptable behaviour will never be finished.

Thirdly, the probabilistic concurrent Kleene algebra of Chapter 4 was devised to provide a unified framework for probabilistic concurrent systems. The new algebra underlines the interaction between nondeterminism, probability and concurrency and is sound with respect to an interleaving as well as a true concurrent model of computation. In contrast to the previous results on probabilistic Kleene algebra, no completeness property is conjectured or claimed but an answer to such a question would increase greatly the understanding of the axiomatisation of probabilistic concurrent Kleene algebra.

Lastly, we illustrated the utility of the developed algebraic frameworks by the derivation of a rely/guarantee technique for probabilistic programs with interferences. This forms a new extension of Jones rely guarantee calculus that allows the verification of probabilistic properties. The mathematical foundations of the extended calculus were laid out in Chapter 6, where a variant of bundle event structures labelled with atomic, probabilistic, sequential programs was developed. The main results include: the definition of the order $\sqsubseteq$ and $\sqsubseteq_{\text{sim}}$, which are the fundamental semantics used for the definition of a rely/guarantee specification;

a theorem (Theorem 6.5.5) linking these two orders that is, in particular, a necessary ingredient in the proof of the concurrency rules; and the rely/guarantee rules themselves. In the example, these rules were shown to be powerful enough to provide a sensible bound for the probability of correctness of a faulty simple system.

Limitations and future work

These contributions are by no mean a panacea for the formal study and verification of probabilistic and concurrent systems. The solutions have limitations as well as room for improvement.

In the constructions of bundle event structures with implicit probability, the state space of programs was assumed to be a finite set. Even if computer memories are finite, it is sometimes desirable to have a model that supports an infinite state space so as to ensure a more general correctness property. Indeed, sequential probabilistic programs on countable state spaces are already accounted for, using the denotational model of McIver and Morgan [48]. Therefore, an extension of the results on bundle event structures with implicit probability is part of our future works. Moreover, a further extension to probabilistic programs running on continuous state spaces (such ar $\mathbb{R}$) will be investigated.

The semantic map $[\![\ ]\!]$ that interprets concurrent probabilistic programs into a sequential one does not account for termination. This is due to the fact that $[\![\ ]\!]$ only allows schedulers that are terminating with probability 1 (i.e. a scheduler that generates a distribution over the state space). Non-termination can be achieved by considering all possible schedulers, which may generate subdistributions. This however requires a constraint on the definition of the concurrent composition so as to obtain a meaningful extension of the presented probabilistic rely/guarantee calculus. More precisely, the notion of "fair scheduler" should be formally defined to ensure that in the concurrent composition $r^* \| \mathcal{E}$, the rely condition r^* is indeed treated as the Kleene star (i.e. terminating behaviours only).

Another important use of rely/guarantee calculus is to provide a refinement

calculus for the stepwise development of concurrent programs [61]. That is, a program should be correct by construction rather than by verification. Such a framework has been investigated for standard concurrent programs [17, 24] and a probabilistic version is of comparable interest.

Finally, automation should be part of any serious theory of development and verification of systems, because a formal proof of correctness usually consists of a formidable number of easily mechanised steps. No automation has been attempted in this thesis as such a task would constitute an additional large amount of work on top of the presented theoretical development. However, reasoning within probabilistic Kleene algebra has been automated within Isabelle/HOL, which is a part of the proof archive on Kleene abgelras by Struth and Weber [78]. Additionally, a different approach which is directly based on the implementation of expectation transformer in Isabelle/HOL can also be achieved by following the works of Hurd [29] or the more recent implementation by Cock [8].

Notations

$\mathbb{R}$	Set of real numbers
$\mathbb{Z}$	Set of positive and negative integers
$\mathbb{N}$	Set of natural numbers
ω	The first infinite ordinal number
Ω	State space
$\mathbb{H}_1\Omega$	Set of sequential probabilistic programs with state space Ω (He spaces)
$\mathbb{J}_1\Omega$	Set of sequential probabilistic deterministic programs with state space Ω
$\overline{\mathbb{H}}_1\Omega$	Set of sequential probabilistic programs and tests
$\mathbb{D}X$	Set of probability distributions over X
$\mathbb{D}_{\leq 1}X$	Set of subdistributions over X
Σ	Set of labels or actions
PAut	Set of simple probabilistic automata
BES	Set of bundle event structures
pBES	Set of probabilistic bundle event structures
ipBES	Set of bundle event structures with implicit probability
POMSet	Set of pomsets
$\mathcal{T}(\mathcal{E})$	Set of traces of $\mathcal{E}$
$\mathcal{T}_n(\mathcal{E})$	Set of traces of $\mathcal{E}$ that are of length n
$\mathcal{T}_{\max}(\mathcal{E})$	Set of maximal traces of $\mathcal{E}$
$\mathcal{C}(\mathcal{E})$	Set of configurations of $\mathcal{E}$
$\mathcal{L}(\mathcal{E})$	Set of lposets of $\mathcal{E}$
$\mathcal{P}(\mathcal{E})$	Set of pomsets of $\mathcal{E}$
$\mathbf{Sched}_1(\mathcal{E})$	Set of schedulers of $\mathcal{E}$ that are generating distributions
$\downarrow X$	Down-closure of X with respect to a given order
$\uparrow X$	Up-closure of X with respect to a given order
$\mathrm{conv}(X)$	Convex closure of X
$\overline{X}$	Topological closure of X
$\mathrm{sym}(R)$	Symmetric closure of the homogeneous relation R
$\mathrm{supp}(f)$	Support of the real valued function f
$\mathrm{dom}(f)$	Domain of the function or relation f

$\tau(P)$	Set of trees (or dags) simulated by the automaton P
$\mathbf{unfold}(P)$	Unfolding of an automaton P
$\mathrm{Path}(P)$	Set of paths of an automaton P
$\mathcal{S}_P$	The maximal simulation of a standard automaton P
$\mathcal{B}$	The intersection of a maximal simulation and its inverse (standard case)
$\longrightarrow_P$	Set of transitions of an automaton P
$\Longrightarrow_P$	Set of extended (or weak) transitions of an automaton P
$\mathbf{set}(u)$	The carrier set x of a partial order structure $u = (x, \leq)$
$\cong$	Standard simulation equivalence
$\preceq$	Standard simulation order
$\leq$	A preorder or a partial order
$\trianglelefteq$	Prefix order on traces, lposets and configurations
$\triangleleft$	Sub-bundle event structure order
$\#$	Conflict relation
$\#_\mu$	Immediate conflict relation
$\mathbf{cfl}(x)$	The set of events that are in conflict with some event of x
$\underline{\mathbf{cfl}}(x)$	The set $\mathbf{cfl}(x) \cup \{x\}$
$\langle e \rangle$	Intersection of clusters containing the event e
$\sqsubseteq_s$	Subsumption (or implementation) order between lposets
$\sqsubseteq_{\mathbb{H}}$	Refinement order in $\mathbb{H}_1\Omega$
$\sqsubseteq$	Sequential refinment in ipBES
$\sqsubseteq_{\mathrm{sim}}$	(Funcional) Simulation order on ipBES
$\sqsubseteq_{p\mathrm{sim}}$	Probabilistic simulation order on pBES
$X \to Y$	Function from X to Y
$X \rightharpoonup Y$	Partial function from X to Y

Bibliography

[1] R. Backhouse. Galois connections and fixed point calculus. In R. Backhouse, R. Crole, and J. Gibbons, editors, *ACM/MPC*, volume 2297 of *LNCS*, pages 89–150. Springer Berlin Heidelberg, 2002.

[2] C. Baier and J. P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[3] J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

[4] M. Boffa. Une remarque sur les systmes complets d'identits rationnelles. *ITA*, 24:419–428, 1990.

[5] J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

[6] D. Bustan and O. Grumberg. Simulation-based minimization. *ACM Trans. Comput. Logic*, 4:181–206, April 2003.

[7] F. Cherief. Back and forth bisimulations on prime event structures. In D. Etiemble and J. C. Syre, editors, *PARLE*, volume 605 of *LNCS*, pages 843–858. Springer, 1992.

[8] D. Cock. Verifying probabilistic correctness in isabelle with pgcl. In F. Cassez, R. Huuck, G. Klein, and B. Schlich, editors, *SSV*, EPTCS, pages 167–178, 2012.

[9] E. Cohen. Separation and reduction. In R. C. Backhouse and J. N. Oliveira, editors, *MPC*, volume 1837 of *LNCS*, pages 45–59. Springer, 2000.

[10] E. Cohen. Weak Kleene algebra is sound and (possibly) complete for simulation. *CoRR*, abs/0910.1028, 2009.

[11] J. W. Coleman and C. B. Jones. A structural proof of the soundness of rely/guarantee rules. *J. Log. Comput.*, 17(4):807–841, 2007.

[12] J. H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, Mathematics series, 1971.

[13] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In S. Graf and M. Schwartzbach, editors, *TACAS'00*, volume 1785 of *LNCS*, pages 395–410. Springer, 2000.

[14] Y. Deng, R. J. van Glabbeek, M. Hennessy, C. Morgan, and C. Zhang. Characterising testing preorders for finite probabilistic processes. In *LICS*, pages 313–325. IEEE Computer Society, 2007.

[15] Y. Deng, R. J. van Glabbeek, M. Hennessy, C. Morgan, and C. Zhang. Remarks on testing probabilistic processes. *Electron. Notes Theor. Comput. Sci.*, 172:359–397, 2007.

[16] J. Desharnais and G. Struth. Internal axioms for domain semirings. *Sci. Comput. Program.*, 76(3):181–203, 2011.

[17] J. Dingel. A refinement calculus for shared-variable parallel and distributed programming. *Formal Asp. Comput.*, 14(2):123–197, 2002.

[18] W. Fokkink and H. Zantema. Basic process algebra with iteration: Completeness of its equational axioms. *Comput. J.*, 37(4):259–268, 1994.

[19] U. Frendrup and J. N. Jensen. A complete axiomatization of simulation for regular CCS expressions. Technical report, University of Aarhus, 2001.

[20] H. Furusawa and K. Nishizawa. Relational and multirelational representation theorems for complete idempotent left semirings. In H. C. M. de Swart, editor, *RAMICS 2011*, LNCS. Springer, 2011.

[21] H. Furusawa, N. Tsumagari, and K. Nishizawa. A non-probabilistic relational model of probabilistic Kleene algebras. In *RelMiCS'08/AKA'08*, RelMiCS'08/AKA'08, pages 110–122. Springer-Verlag, 2008.

[22] J. L. Gischer. *Partial Orders and the Axiomatic Theory of Shuffle (Pomsets).* PhD thesis, Stanford University, Stanford, CA, USA, 1985.

[23] J. L. Gischer. The equational theory of pomsets. *Theor. Comput. Sci.*, 61(23):199 – 224, 1988.

[24] I. J. Hayes, C. B. Jones, and Colvin R. J. Refining rely-guarantee thinking. Technical Report CS-TR-1334, Newcastle University, United Kingdom, 2012.

[25] M. R. Henzinger, T. A. Henzinger, and P. W Kopke. Computing simulations on finite and infinite graphs. pages 453–462. IEEE Computer Society Press, 1996.

[26] T. Herman. Probabilistic self-stabilization. *Inf. Process. Lett.*, 35(2):63–67, June 1990.

[27] C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra. In M. Bravetti and G. Zavattaro, editors, *CONCUR*, volume 5710 of *LNCS*, pages 399–414. Springer, 2009.

[28] C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra and its foundations. *J. Log. Algebr. Program.*, 80(6):266–296, 2011.

[29] J. Hurd, A. K. McIver, and C. Morgan. Probabilistic guarded commands mechanized in *hol. Electr. Notes Theor. Comput. Sci.*, 112:95–111, 2005.

[30] C. Jones. *Probabilistic non-determinism.* PhD thesis, Edinburgh, Scotland, UK, 1989. UMI Order No. GAXDX-94930.

[31] C. B. Jones. *Development Methods for Computer Programs including a Notion of Interference*. PhD thesis, Oxford University, June 1981.

[32] B. Jonsson, K. G. Larsen, and W. Yi. *Probabilistic Extensions of Process Algebras*, chapter 11. Elsevier, 2001.

[33] J. P. Katoen. *Quantitative and qualitative extensions of event structures*. PhD thesis, University of Twente, 1996.

[34] S. Kiefer, A. S. Murawski, J. Ouaknine, J.and Worrell, and L. Zhang. On stabilization in herman's algorithm. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP*, volume 6756 of *LNCS*, pages 466–477. Springer, 2011.

[35] A. Komuravelli, C. S. Pasareanu, and E. M. Clarke. Assume-guarantee abstraction refinement for probabilistic systems. In P. Madhusudan and S. A. Seshia, editors, *CAV*, volume 7358 of *LNCS*, pages 310–326. Springer, 2012.

[36] D. Kozen. On Kleene algebras and closed semirings. In Rovan, editor, *Proc. Math. Found. Comput. Sci.*, volume 452 of *LNCS*, pages 26–47. Springer-Verlag, 1990.

[37] D. Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, New York, 1991.

[38] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Infor. and Comput.*, 110(2):366–390, May 1994.

[39] D. Kozen. Kleene algebras with tests and the static analysis of programs. Technical Report TR2003-1915, Computer Science Department, Cornell University, November 2003.

[40] D. Kozen and M. C. Patron. Certification of compiler optimizations using kleene algebra with tests. In J. W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L. M. Pereira, and P. J. Sagiv, Y.and Stuckey, editors, *Computational Logic*, volume 1861 of *LNCS*, pages 568–582. Springer, 2000.

[41] D. Krob. Complete systems of b-rational identities. *Theor. Comput. Sci.*, 89(2):207–343, 1991.

[42] M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker. In T. Field, P.G. Harrison, J. T. Bradley, and U. Harder, editors, *CPE/TOOLS*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.

[43] M. Z. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In J. Esparza and R. Majumdar, editors, *TACAS*, volume 6015 of *LNCS*, pages 23–37. Springer, 2010.

[44] R. Langerak. *Bundle event structures: a non-interleaving semantics for LOTOS*. Memoranda informatica. University of Twente, 1992.

[45] N. A. Lynch, R. Segala, and F. W. Vaandrager. Compositionality for probabilistic automata. In *CONCUR*, pages 204–222, 2003.

[46] A. K. McIver, E. Cohen, and C. Morgan. Using probabilistic kleene algebra for protocol verification. In R. A. Schmidt, editor, *RelMiCS*, volume 4136 of *LNCS*, pages 296–310. Springer, 2006.

[47] A. K. McIver and C. Morgan. An elementary proof that Herman's ring is Theta (N2). *Inf. Process. Lett.*, 94(2):79–84, 2005.

[48] A. K. McIver and C. C. Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science)*. SpringerVerlag, 2004.

[49] A. K. McIver, T. M. Rabehaja, and G. Struth. On probabilistic Kleene algebras, automata and simulations. In H. C. M. de Swart, editor, *RAMICS*, volume 6663 of *LNCS*, pages 264–279. Springer, 2011.

[50] A. K. McIver, T. M. Rabehaja, and G. Struth. An event structure model for probabilistic concurrent Kleene algebra. In K. L. McMillan, A. Middeldorp, and A. Voronkov, editors, *LPAR*, volume 8312 of *LNCS*, pages 653–667. Springer, 2013.

[51] A. K. McIver, T. M. Rabehaja, and G. Struth. An event structure model for probabilistic concurrent Kleene algebra (with appendix). *CoRR*, abs/1310.2320, 2013.

[52] A. K. McIver, T. M. Rabehaja, and G. Struth. Probabilistic concurrent Kleene algebra. In L. Bortolussi and H. Wiklicky, editors, *QAPL*, volume 117 of *EPTCS*, pages 97–115, 2013.

[53] A. K. McIver and T. Weber. Towards automated proof support for probabilistic distributed systems. In G. Sutcliffe and A. Voronkov, editors, *LPAR*, volume 3835 of *LNAI*, pages 534–548. Springer, 2005.

[54] L. Meinicke and K. Solin. Probabilistic demonic refinement algebra. Technical Report SSE-2006-04, University of Queensland, Australia, 2006.

[55] L. Meinicke and K. Solin. Refinement algebra for probabilistic programs. *Electron. Notes Theor. Comput. Sci.*, 201:177–195, March 2008.

[56] R. Milner. An algebraic definition of simulation between programs. Technical report, Stanford, CA, USA, 1971.

[57] R. Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984.

[58] R. Milner. A complete axiomatisation for observational congruence of finite-state behaviors. *Inf. Comput.*, 81(2):227–247, May 1989.

[59] R. Mittermayr, J. Blieberger, and A. Schöbel. Kronecker Algebra based Deadlock Analysis for Railway Systems. *PROMET - Traffic & Transportation*, 24(5):359–369, 2012.

[60] B. Möller. Kleene getting lazy. *Sci. Comput. Program.*, 65:195–214, March 2007.

[61] C. C. Morgan. *On the Refinement Calculus*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.

[62] A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):pp. 541–544, 1958.

[63] R. De Nicola and M. Hennessy. Testing equivalence for processes. In J. Daz, editor, *ICALP*, volume 154 of *LNCS*, pages 548–560. Springer, 1983.

[64] A. Parma and R. Segala. Axiomatization of trace semantics for stochastic nondeterministic processes. In B. R. Franceschinis, G.and Haverkort, J. P. Katoen, and M. Woodside, editors, *QEST*, pages 294–303. IEEE Computer Society, 2004.

[65] V. Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[66] R. Qiao, Y. Wang, G. Gao, and J. Wu. Operational semantics of probabilistic Kleene algebra with tests. In *ISCC*, pages 706–713. IEEE, 2008.

[67] T. M. Rabehaja and Jeff W. Sanders. Refinement algebra with explicit probabilism. In W. N. Chin and S. Qin, editors, *TASE*, pages 63–70. IEEE, 2009.

[68] M. O. Rabin. The choice coordination problem. *Acta Inf.*, 17:121–134, 1982.

[69] J. J. M. M. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorgi and R. de Simone, editors, *CONCUR'98*, volume 1466 of *LNCS*, pages 194–218. Springer, 1998.

[70] A. Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13:158–169, January 1966.

[71] S. Schneider. Incorporating time to an integrated formal method, 2012. Presentation at the NII Shonan Meeting on Quantitative Methods in Security and Safety Critical Applications, Japan.

[72] M. P. Schtzenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190 – 194, 1965.

[73] R. Segala. A compositional trace-based semantics for probabilistic automata. In I. Lee and S. A. Smolka, editors, *CONCUR*, volume 962 of *LNCS*, pages 234–248. Springer, 1995.

[74] R. Segala. Testing probabilistic automata. In U. Montanari and V. Sassone, editors, *CONCUR*, volume 1119 of *LNCS*, pages 299–314. Springer, 1996.

[75] R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. *Nord. J. Comput.*, 2(2):250–273, 1995.

[76] K. Seidel and C. C. Morgan. Hierarchical reasoning in probabilistic CSP. *Programmirovanie (Russian Journal of Programming and Computer Software)*, 23, 1996.

[77] E. W. Stark and S. A. Smolka. A complete axiom system for finite-state probabilistic processes. In G. D. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction*, pages 571–596. The MIT Press, 2000.

[78] G. Struth. Isabelle Repository for Kleene algebras. http://staffwww.dcs.shef.ac.uk/people/G.Struth/, 2010.

[79] T. Takai and H. Furusawa. Monodic tree Kleene algebra. In R. A. Schmidt, editor, *Relations and Kleene Algebra in Computer Science*, volume 4136 of *LNCS*, pages 402–416. Springer Berlin Heidelberg, 2006.

[80] R. Tix, K. Keimel, and G. Plotkin. Semantic domains for combining probability and non-determinism. *Electron. Notes Theor. Comput. Sci.*, 222:3–99, February 2009.

[81] R. J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In J. C. M. Baeten and J. W. Klop, editors, *CONCUR 1990*, volume 458 of *LNCS*, pages 278–297. Springer, 1990.

[82] R. J. van Glabbeek. A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In A. M. Borzyszkowski and

S. Sokoowski, editors, *MFCS*, volume 711 of *LNCS*, pages 473–484. Springer Berlin Heidelberg, 1993.

[83] R. J. van Glabbeek and G. D. Plotkin. Configuration structures, event structures and Petri nets. *Theor. Comput. Sci.*, 410(41):4111–4159, 2009.

[84] R. J. van Glabbeek and F. W. Vaandrager. Bundle event structures and CCSP. In R. M. Amadio and D. Lugiez, editors, *CONCUR*, volume 2761 of *LNCS*, pages 57–71. Springer, 2003.

[85] D. Varacca. *Probability, nondeterminism and concurrency: two denotational models for probabilistic computation.* BRICS Dissertation Series. 2003.

[86] J. von Wright. Towards a refinement algebra. *Sci. Comput. Program.*, 51:23–45, May 2004.

[87] G. Winskel. *Events in Computation.* PhD thesis, Edinburgh, Scotland, UK, 1980.

[88] G. Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets*, pages 325–392, 1986.