Roberto Bruni, Ugo Montanari

Models of Computation

– Monograph –

May 18, 2016

Springer

Mathematical reasoning may be regarded rather schematically as the exercise of a combination of two facilities, which we may call intuition and ingenuity.

Alan Turing[1]

[1] The purpose of ordinal logics (from Systems of Logic Based on Ordinals), Proceedings of the London Mathematical Society, series 2, vol. 45, 1939.

Preface

The origins of this book lie their roots on more than 15 years of teaching a course on formal semantics to graduate Computer Science to students in Pisa, originally called *Fondamenti dell'Informatica: Semantica* (*Foundations of Computer Science: Semantics*) and covering models for imperative, functional and concurrent programming. It later evolved to *Tecniche di Specifica e Dimostrazione* (*Techniques for Specifications and Proofs*) and finally to the currently running *Models of Computation*, where additional material on probabilistic models is included.

The objective of this book, as well as of the above courses, is to present different *models of computation* and their basic *programming paradigms*, together with their mathematical descriptions, both *concrete* and *abstract*. Each model is accompanied by some relevant formal techniques for reasoning on it and for proving some properties.

To this aim, we follow a rigorous approach to the definition of the *syntax*, the *typing* discipline and the *semantics* of the paradigms we present, i.e., the way in which well-formed programs are written, ill-typed programs are discarded and the way in which the meaning of well-typed programs is unambiguously defined, respectively. In doing so, we focus on basic proof techniques and do not address more advanced topics in detail, for which classical references to the literature are given instead.

After the introductory material (Part I), where we fix some notation and present some basic concepts such as term signatures, proof systems with axioms and inference rules, Horn clauses, unification and goal-driven derivations, the book is divided in four main parts (Parts II-V), according to the different styles of the models we consider:

IMP: imperative models, where we apply various incarnations of well-founded induction and introduce λ-notation and concepts like structural recursion, program equivalence, compositionality, completeness and correctness, and also complete partial orders, continuous functions, fixpoint theory;

HOFL: higher-order functional models, where we study the role of type systems, the main concepts from domain theory and the distinction between lazy and eager evaluation;

CCS, π: concurrent, non-deterministic and interactive models, where, starting from operational semantics based on labelled transition systems, we introduce the notions of bisimulation equivalences and observational congruences, and overview some approaches to name mobility, and temporal and modal logics system specifications;

PEPA: probabilistic/stochastic models, where we exploit the theory of Markov chains and of probabilistic reactive and generative systems to address quantitative analysis of, possibly concurrent, systems.

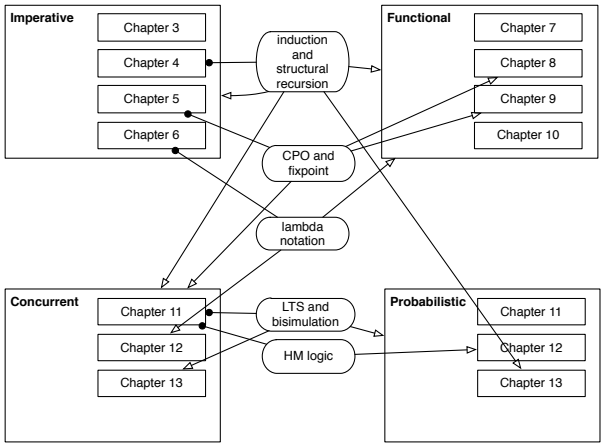
Each of the above models can be studied in separation from the others, but previous parts introduce a body of notions and techniques that are also applied and extended in later parts.

Parts I and II cover the essential, classic topics of a course on formal semantics.

Part III introduces some basic material on process algebraic models and temporal and modal logic for the specification and verification of concurrent and mobile systems. CCS is presented in good detail, while the theory of temporal and modal logic, as well as π-calculus, are just overviewed. The material in Part III can be used in conjunction with other textbooks, e.g., on model checking or π-calculus, in the context of a more advanced course on the formal modelling of distributed systems.

Part IV outlines the modelling of probabilistic and stochastic systems and their quantitative analysis with tools like PEPA. It poses the basis for a more advanced course on quantitative analysis of sequential and interleaving systems.

The diagram that highlights the main dependencies is represented below:



The diagram contains a squared box for each chapter / part and a rounded-corner box for each subject: a line with a filled-circle end joins a subject to the chapter where it is introduced, while a line with an arrow end links a subject to a chapter or part where it is used. In short:

Induction and recursion: various principles of induction and the concept of structural recursion are introduced in Chapter 4 and used extensively in all subsequent chapters.

CPO and fixpoint: the notion of complete partial order and fixpoint computation are first presented in Chapter 5. They provide the basis for defining the denotational semantics of IMP and HOFL. In the case of HOFL, a general theory of product and functional domains is also introduced (Chapter 8). The notion of fixpoint is also used to define a particular form of equivalence for concurrent and probabilistic systems, called bisimilarity, and to define the semantics of modal logic formulas.

Lambda-notation: λ-notation is a useful syntax for managing anonymous functions. It is introduced in Chapter 6 and used extensively in Part III.

LTS and bisimulation: Labelled transition systems are introduced in Chapter 11 to define the operational semantics of CCS in terms of the interactions performed. They are then extended to deal with name mobility in Chapter 13 and with probabilities in Part V. A bisimulation is a relation over the states of an LTS that is closed under the execution of transitions. The before mentioned bisimilarity is the coarsest bisimulation relation. Various forms of bisimulation are studied in Part IV and V.

HM-logic: Hennessy-Milner logic is the logic counterpart of bisimilarity: two state are bisimilar if and only if they satisfy the same set of HM-logic formulas. In the context of probabilistic system, the approach is extended to Larsen-Skou logic in Chapter 15.

Each chapter of the book is concluded by a list of exercises that span over the main techniques introduced in that chapter. Solutions to selected exercises are collected at the end of the book.

Pisa, *Roberto Bruni*
February 2016 *Ugo Montanari*

Acknowledgements

We want to thank our friend and colleague Pierpaolo Degano for encouraging us to prepare this book and submit it to the EATCS monograph series. We thank Ronan Nugent and all the people at Springer for their editorial work. We acknowledge all the students of the course on *Models of Computation (MOD)* in Pisa for helping us to refine the presentation of the material in the book and to eliminate many typos and shortcomings from preliminary versions of this text. Last but not least, we thank Lorenzo Galeotti, Andrea Cimino, Lorenzo Muti, Gianmarco Saba, Marco Stronati, former students of the course on *Models of Computation*, who helped us with the LaTeX preparation of preliminary versions of this book, in the form of lecture notes.

Contents

Acronyms

$\sim$	operational equivalence in IMP (see Definition 3.3)
$\equiv_{den}$	denotational equivalence in HOFL (see Definition 10.4)
$\equiv_{op}$	operational equivalence in HOFL (see Definition 10.3)
$\simeq$	CCS strong bisimilarity (see Definition 11.5)
$\approx$	CCS weak bisimilarity (see Definition 11.16)
$\cong$	CCS weak observational congruence (see Section 11.8.2)
$\mathrel{\hat{=}}$	CCS dynamic bisimilarity (see Definition 11.18)
$\sim_E$	π-calculus strong early bisimilarity (see Definition 13.3)
$\sim_L$	π-calculus strong late bisimilarity (see Definition 13.4)
$\simeq_E$	π-calculus strong early full bisimilarity (see Section 13.5.3)
$\simeq_L$	π-calculus strong late full bisimilarity (see Section 13.5.3)
$\approx_E$	π-calculus weak early bisimilarity (see Section 13.5.4)
$\approx_L$	π-calculus weak late bisimilarity (see Section 13.5.4)
$\mathscr{A}$	interpretation function for the denotational semantics of IMP arithmetic expressions (see Section 6.2.1)
ack	Ackermann function (see Example 4.18)
Aexp	set of IMP arithmetic expressions (see Chapter 3)
$\mathscr{B}$	interpretation function for the denotational semantics of IMP boolean expressions (see Section 6.2.2)
Bexp	set of IMP boolean expressions (see Chapter 3)
$\mathbb{B}$	set of booleans
$\mathscr{C}$	interpretation function for the denotational semantics of IMP commands (see Section 6.2.3)
CCS	Calculus of Communicating Systems (see Chapter 11)
Com	set of IMP commands (see Chapter 3)
CPO	Complete Partial Order (see Definition 5.11)
$\text{CPO}_\perp$	Complete Partial Order with bottom (see Definition 5.12)
CSP	Communicating Sequential Processes (see Section 16.2)
CTL	Computation Tree Logic (see Section 12.2.2)
CTMC	Continuous Time Markov Chain (see Definition 14.15)
DTMC	Discrete Time Markov Chain (see Definition 14.14)

Env	set of HOFL environments (see Chapter 9)
fix	(least) fixpoint (see Definition 5.2.2)
FIX	(greatest) fixpoint
gcd	greatest common divisor
HML	Hennessy-Milner modal Logic (see Section 11.6)
HM-Logic	Hennessy-Milner modal Logic (see Section 11.6)
HOFL	A Higher-Order Functional Language (see Chapter 7)
IMP	A simple IMPerative language (see Chapter 3)
int	integer type in HOFL (see Definition 7.2)
Loc	set of locations (see Chapter 3)
LTL	Linear Temporal Logic (see Section 12.2.1)
LTS	Labelled Transition System (see Definition 11.2)
lub	least upper bound (see Definition 5.7)
$\mathbb{N}$	set of natural numbers
$\mathscr{P}$	set of closed CCS processes (see Definition 11.1)
PEPA	Performance Evaluation Process Algebra (see Chapter 16)
Pf	set of partial functions on natural numbers (see Example 5.13)
PI	set of partial injective functions on natural numbers (see Problem 5.12)
PO	Partial Order (see Definition 5.1)
PTS	Probabilistic Transition System (see Section 14.4.2)
$\mathbb{R}$	set of real numbers
$\mathscr{T}$	set of HOFL types (see Definition 7.2)
Tf	set of total functions from $\mathbb{N}$ to $\mathbb{N}_\perp$ (see Example 5.14)
Var	set of HOFL variables (see Chapter 7)
$\mathbb{Z}$	set of integers

Part V
Probabilistic Systems

This part focuses on models and logics for probabilistic and stochastic systems. Chapter 14 presents the theory of random processes and Markov chains. Chapter 15 studies (reactive and generative) probabilistic models of computation with observable actions and sources of non-determinism together with a specification logic. Chapter 16 defines the syntax, operational and abstract semantics of PEPA, a well-known high-level language for the specification and analysis of stochastic, interactive systems.

Chapter 15
Discrete Time Markov Chains with Actions and Non-determinism

A reasonable probability is the only certainty. (E.W. Howe)

Abstract In this chapter we introduce some advanced probabilistic models that can be defined by enriching the transition functions of PTSs. As we have seen for Markov chains, the transition system representation is very useful since it comes with a notion of bisimilarity. In fact, using the advanced, categorical notion of *coalgebra*, which however we will not develop further, there is a standard method to define bisimilarity just according to the type of the transition function. Also a corresponding notion of Hennessy-Milner logic can be defined accordingly. First we will see two different ways to add observable actions to our probabilistic models, then we will present extensions which combine non-determinism, actions and probabilities.

15.1 Discrete Markov Chains With Actions

In this section we show how it is possible to change the transition function of PTSs in order to extend Markov chains with labels that represent actions performed by the system. There are two main cases to consider, called *reactive models* and *generative models*, respectively:

Reactive: In the first case we add actions that are used by the controller to stimulate the system. When we want the system to change its state we give an input action to it which could affect its future state (its reaction). This is the reason why this type of models is called "reactive". Formally, we have that a *reactive probabilistic transition system* (also called *Markov decision process*) is determined by a transition function of the form:[1]

$$\alpha_{\mathsf{r}} : S \to L \to (D(S) + 1)$$

[1] The subscript r stands for "reactive".

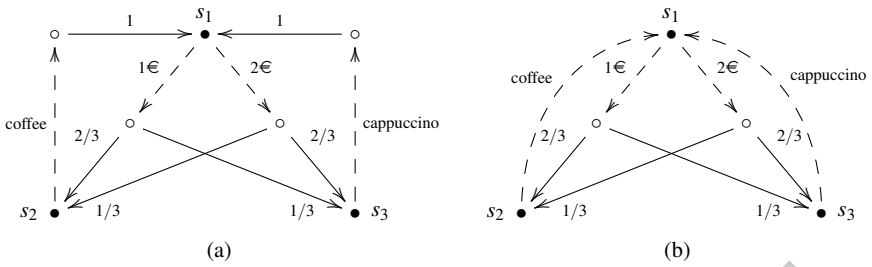


Fig. 15.1: A reactive PTS which represents a coffee maker

where we recall that S is the set of states, $1 = \{*\}$ is a singleton used to represent the deadlock states, and that $D(S)$ is the set of discrete probability distributions over S.

Generative: In the second case the actions represent the outcomes of the system, this means that whenever the system changes its state it shows an action, whence the terminology "generative". Formally we have that a *generative probabilistic transition system* is determined by a transition function of the form:[2]

$$\alpha_{\mathsf{g}} : S \to (D(L \times S) + 1)$$

Remark 15.1. We have that in a reactive system, for any $s \in S$ and for any $\ell \in L$:

$$\sum_{s' \in S} \alpha_{\mathsf{r}}\, s\, \ell\, s' = 1$$

Instead, in a generative system, for any $s \in S$:

$$\sum_{(\ell,s') \in L \times S} \alpha_{\mathsf{g}}\, s\, (\ell, s') = 1$$

This means that in reactive systems, fixed the source state and the action, the next state probabilities must sum to 1, while in a generative system, fixed the source state, the distribution of all transitions must sum to 1 (i.e., given an action ℓ the sum of probabilities to reach any state is less or equal to 1).

15.1.1 Reactive DTMC

Let us illustrate how a reactive probabilistic system works by using a simple example.

[2] The subscript g stands for "generative".

Example 15.1 (Random coffee maker). Let us consider a system which we call *random coffee maker*, in which the user can insert a coin (1 or 2 euros) then, the coffee maker, based on the value of the input, makes a coffee or a cappuccino with larger or smaller probabilities. The system is represented in Figure 15.1(a). Note that, since we want to allow the system to take input from the environment we have chosen a reactive system to represent the coffee maker. The set of labels is $L = \{1€, 2€, \text{coffee}, \text{cappuccino}\}$ and the corresponding transitions are represented as dashed arrows. There are three states s_1, s_2 and s_3, represented with black-filled circles. If the input $1€$ is received in state s_1, then we can reach state s_2 with probability $2/3$ or s_3 with probability $1/3$, as illustrated by the solid arrows departing from the white-filled circle associated with the distribution. Vice versa, if the input $2€$ is received in state s_1, then we can reach state s_2 with probability $1/3$ or s_3 with probability $2/3$. From state s_2 there is only one transition available, with label coffee, that leads to s_1 with probability 1. Figure 15.1(b) shows a more compact representation of the random coffee maker where the white-filled circle is omitted because the probability distribution is trivial. Similarly, from state s_3 there is only one transition available, with label cappucino, that leads to s_1 with probability 1.

As we have shown in the previous chapter, using LTSs we have a standard method to define bisimilarity between probabilistic systems. In the following, we shall abuse the notation by writing $\alpha_r\ s\ \ell\ I$ in place of $\sum_{s' \in I} \alpha_r\ s\ \ell\ s'$, i.e., by extending α_r to deal with equivalence classes I of states.

Definition 15.1 (Bisimulation in reactive systems). Let $\alpha_r : S \to L \to (D(S)+1)$ be a reactive probabilistic system. A relation $R \subseteq S \times S$ is a bisimulation if for all $s_1, s_2 \in S$ we have:

$$s_1\ R\ s_2 \quad \Rightarrow \quad \forall \ell \in L, I \in S_{/\equiv_R}.\ \alpha_r\ s_1\ \ell\ I = \alpha_r\ s_2\ \ell\ I$$

where I ranges over the equivalence classes induced by R. Two states are said to be bisimilar if there exists a bisimulation in which they are related.

Note that any two bisimilar states s_1 and s_2 must have, for each action, the same probability to reach the states in any other equivalence class.

15.1.1.1 Larsen-Skou Logic

Now we will present a probabilistic version of Hennessy-Milner logic. This logic has been introduced by Larsen and Skou, and provides a new version of the modal operator. As usual we start from the syntax of Larsen-Skou logic formulas

Definition 15.2 (Larsen-Skou logic). The formulas of *Larsen-Skou logic* are generated by the following grammar:

$$\varphi \quad ::= \quad \mathit{true} \quad | \quad \varphi_1 \wedge \varphi_2 \quad | \quad \neg\varphi \quad | \quad \langle \ell \rangle_q\, \varphi$$

We let $\mathscr{S}$ denote the set of Larsen-Skou logic formulas. The novelty resides in the new modal operator $\langle \ell \rangle_q \varphi$ that takes three parameters: a formula φ, an action ℓ and a real number $q \leq 1$. It corresponds to a refined variant of the usual HM-logic diamond operator $\Diamond_\ell$. Informally, the formula $\langle \ell \rangle_q \varphi$ requires the ability to reach a state satisfying the formula φ by performing the action ℓ with probability at least q.

As we have done for Hennessy-Milner logic we present the Larsen-Skou logic by defining a satisfaction relation $\models \subseteq S \times \mathscr{S}$.

Definition 15.3 (Satisfaction relation). Let $\alpha_r : S \to L \to (D(S)+1)$ be a reactive probabilistic system. We say that the state $s \in S$ satisfies the Larsen-Skou formula φ and write $s \models \varphi$, if satisfaction can be proved using the (inductively defined) rules:

$$
\begin{array}{ll}
s \models \mathit{true} & \\
s \models \varphi_1 \wedge \varphi_2 & \text{if } s \models \varphi_1 \text{ and } s \models \varphi_2 \\
s \models \neg\varphi & \text{if } \neg s \models \varphi \\
s \models \langle \ell \rangle_q \varphi & \text{if } \alpha\, s\, \ell\, [\![\varphi]\!] \geq q \text{ where } [\![\varphi]\!] = \{s' \in S \mid s' \models \varphi\}
\end{array}
$$

A state s satisfies the formula $\langle \ell \rangle_q \varphi$ if the (sum of the) probability to pass in any state s' that satisfies φ from s with an action labelled ℓ is greater than or equal to q. Note that the corresponding modal operator of the HM-logic can be obtained by setting $q = 1$, i.e., $\langle \ell \rangle_1 \varphi$ means $\Diamond_\ell \varphi$ and we write just $\langle \ell \rangle \varphi$ when this is the case.

Likewise HM-logic, the equivalence induced by Larsen-Skou logic formulas coincide with bisimilarity. Moreover, we have an additional, stronger result: It can be shown that it is enough to consider only the version of the logic without negation.

Theorem 15.1 (Larsen-Skou bisimilarity characterization). *Two states of a reactive probabilistic system are bisimilar if and only if they satisfy the same formulas of Larsen-Skou logic without negation.*

Example 15.2 (Larsen-Skou logic). Let us consider the reactive system in Figure 15.1. We would like to prove that:

$$s_1 \models \langle 1€ \rangle_{1/2} \langle \text{coffee} \rangle \mathit{true}$$

By definition of the satisfaction relation, we must check that:

$$\alpha_r\ s_1\ 1€\ I_1 \geq 1/2 \qquad \text{where} \quad I_1 \stackrel{\text{def}}{=} \{s \in S \mid s \models \langle \text{coffee} \rangle \mathit{true}\}$$

Now we have that $s \models \langle \text{coffee} \rangle \mathit{true}$ if:

$$\alpha_r\ s\ \text{coffee}\ I_2 \geq 1 \qquad \text{where} \quad I_2 \stackrel{\text{def}}{=} \{s \in S \mid s \models \mathit{true}\} = \{s_1, s_2, s_3\}$$

Therefore:

$$I_1 = \{s \in S \mid \alpha_r\ s\ \text{coffee}\ I_2 \geq 1\} = \{s \in S \mid \alpha_r\ s\ \text{coffee}\ \{s_1, s_2, s_3\} \geq 1\} = \{s_2\}$$

Finally:

$$\alpha_r\ s_1\ 1€\ \{s_2\} = 2/3 \geq 1/2.$$

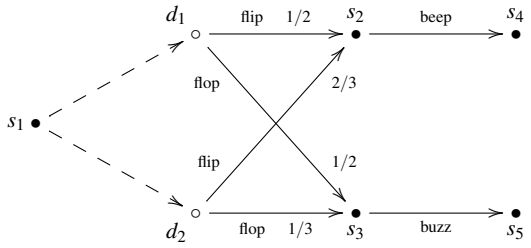


Fig. 15.2: A Segala automata

15.1.2 DTMC With Non-determinism

In this section we add non-determinism to generative and reactive systems. Correspondingly, we will introduce two classes of models called *Segala automata* and *simple Segala automata*, after the name of Roberto Segala who studied them. In both cases we use non-determinism to allow the system to choose between different probability distributions.

15.1.2.1 Segala Automata

Segala automata have been developed by Roberto Segala in 1995. They are generative systems that combine probability and non-determinism. When the system has to move from a state to another, first of all it has to choose non-deterministically a probability distribution, then it uses this information to perform the transition. Formally the transition function of a *Segala automaton* is defined as follows:

$$\alpha_s : S \to \mathscr{P}(D(L \times S))$$

As we can see, to each state it corresponds a set of probability distributions $D(L \times S)$ that are defined on pairs of labels and states. Note that in this case it is not necessary to have the singleton 1 to model explicitly deadlock states, because we can use the empty set to the purpose.

Example 15.3 (Segala automata). Let us consider the system in Figure 15.2. We have an automata with five states, named s_1 to s_5, represented as usual by black-filled circles. When in the state s_1, the system can choose non-deterministically (dashed arrows) between two different distributions d_1 and d_2:

$$\alpha_{\mathsf{s}}\, s_1 = \{d_1, d_2\} \quad \text{where} \quad \begin{array}{ll} d_1(\text{flip}, s_2) = 1/2 & d_1(\text{flop}, s_3) = 1/2 \\ d_2(\text{flip}, s_2) = 2/3 & d_2(\text{flop}, s_3) = 1/3 \end{array}$$

(we leave implicit that $d_1(l,s) = 0$ and $d_2(l,s) = 0$ for all other label-state pairs).

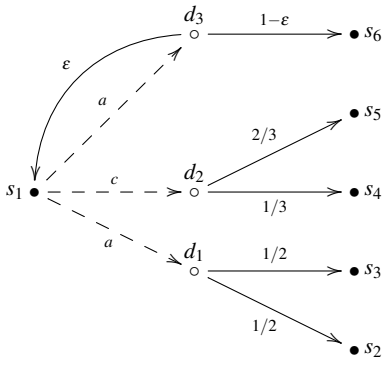


Fig. 15.3: A Simple Segala automaton

From states s_2 and s_3 there is just one choice available, respectively the trivial distributions d_3 and d_4 that are omitted from the picture for conciseness of the representation:

$$\alpha_{\mathsf{s}}\ s_2 = \{d_3\} \quad \text{where } d_3(\text{beep}, s_4) = 1$$
$$\alpha_{\mathsf{s}}\ s_3 = \{d_4\} \quad \text{where } d_4(\text{buzz}, s_5) = 1$$

Finally, from states s_4 and s_5 there are simply no choices available, i.e., they are deadlock states:

$$\alpha_{\mathsf{s}}\ s_4 = \alpha_{\mathsf{s}}\ s_5 = \varnothing$$

15.1.2.2 Simple Segala Automata

Now we present the reactive version of Segala automata. In this case we have that the system can react to an external stimulation by using a probability distribution. Since we can have more than one distribution for each label, the system uses non-determinism to choose between different distributions for the same label. Formally the transition function of a *simple Segala automaton* is defined as follows:

$$\alpha_{\mathsf{simS}} : S \to \mathscr{P}(L \times D(S))$$

Example 15.4 (A Simple Segala Automata). Let us consider the system in Figure 15.3, where we assume some suitable probability value ε has been given. We have six states (represented by black-filled circles, as usual): the state s_1 has two possible inputs, a and c, moreover the label a has associated two different distributions d_1 and d_3, while c has associated a unique distribution d_2. All the other states are deadlock. Formally the system is defined by letting:

$$\begin{aligned}\alpha_{\mathsf{simS}}\ s_1 = \{&(a,d_1),(c,d_2),(a,d_3)\} \text{ where}\\ &d_1(s_2) = d_1(s_3) = 1/2\\ &d_2(s_4) = 1/3 \quad d_2(s_5) = 2/3\\ &d_3(s_1) = \varepsilon \quad d_3(s_6) = 1-\varepsilon\\ \alpha_{\mathsf{simS}}\ s_2 = \alpha_{\mathsf{simS}}\ s_3 = \alpha_{\mathsf{simS}}\ s_4 = \alpha_{\mathsf{simS}}\ s_5 = \alpha_{\mathsf{simS}}\ s_6 = \varnothing\end{aligned}$$

15.1.2.3 Non-determinism, Probability and Actions

As we saw, there are many ways to combine probability, non-determinism and actions. We conclude this chapter by mentioning two other interesting models which can be obtained by redefining the transition function of a PTS.

The first class of systems is that of *alternating transition systems*. In this case we allow the system to perform two types of transition: one using probability distributions and one using non-determinism. An alternating system can be defined formally by a transition function of the form:

$$\alpha_{\mathsf{a}} : S \to (D(S) + \mathscr{P}(L \times S))$$

So in this kind of systems we can alternate probabilistic and non-deterministic choices and can partition the states accordingly. (Again, a state s is deadlock when $\alpha_{\mathsf{a}}\ s = \varnothing$).

The second type of systems that we present is that of *bundle transition systems*. In this case the system associates a distribution to subsets of non-deterministic choices. Formally, the transition function has the form:

$$\alpha_{\mathsf{b}} : S \to D(\mathscr{P}(L \times S))$$

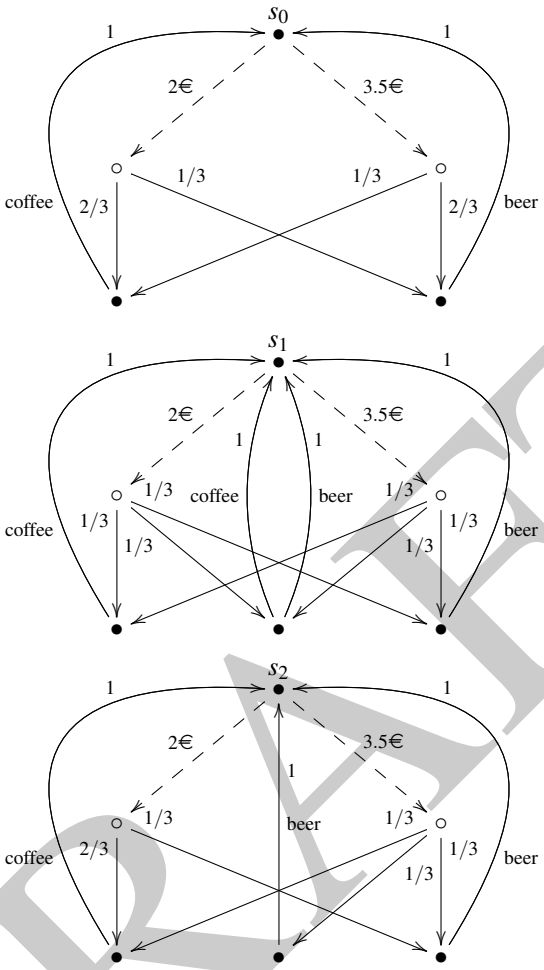
So when a bundle transition system has to perform a transition, first of all it chooses by using a probability distribution a set of possible choices, then non-deterministically it picks one of these.

Problems

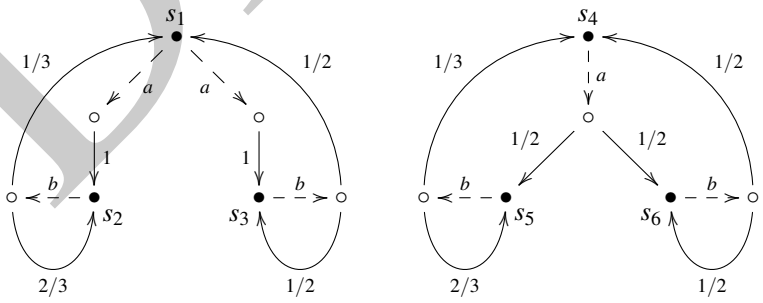
15.1. In which sense is a Segala automaton the most general model?

1. Show in which way a generative LTS, a reactive LTS and a simple Segala automaton can be interpreted as (generative) Segala automata.
2. Explain the difficulties in representing a generative LTS as a simple Segala automaton.

15.2. Consider the following three reactive LTSs. For every pair of systems, check whether their initial states are bisimilar. If they are, describe the bisimulation, if they are not, find a formula of the Larsen-Skou logic that distinguishes them.



15.3. Define formally the notion of bisimulation/bisimilarity for simple Segala automata. Then apply the partition refinement algorithm to the automata below to check which are the bisimilar states.



15.4. Let a *non-stopping*, reactive, probabilistic labelled transition systems (PLTS) be the reactive PLTS with $\alpha : S \to L \to D(S)$ (rather than $\alpha : S \to L \to (D(S)+1)$).

1. Prove that all the states of a non-stopping, reactive PLTS are bisimilar.
2. Then give the definition of bisimilarity also for *generative* PLTS.
3. Furthermore, consider the non-stopping subclass of generative PLTS and show an example where some states are not bisimilar.
4. Moreover, give the definition of bisimilarity also for Segala PLTS, and show that Segala bisimilarity reduces to generative PLTS bisimilarity in the deterministic case (namely when, for every state s, $\alpha(s)$ is a singleton).