Roberto Bruni, Ugo Montanari

Models of Computation

– Monograph –

May 20, 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.

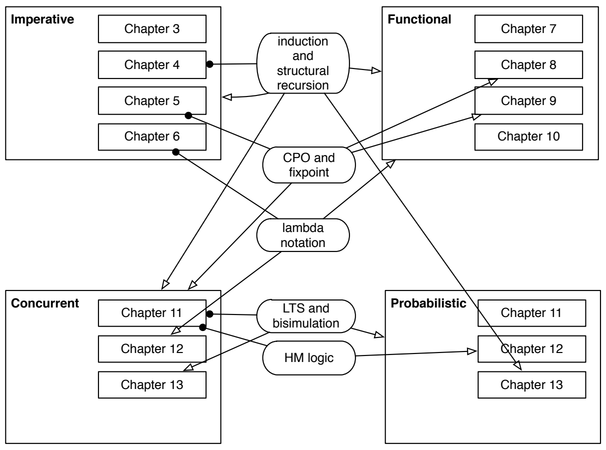
Each of the above models can be studied in separation from the others, but previous parts introduce a body of notions and techniques that are also applied and extended in later parts.

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,
February 2016

Roberto Bruni
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)
$\approxeq$	CCS weak observational congruence (see Section 11.8.2)
$\cong$	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 16
PEPA - Performance Evaluation Process Algebra

He who is slowest in making a promise is most faithful in its performance. (Jean Jacques Rousseau)

Abstract The probabilistic and stochastic models we have presented in previous chapters represent system behaviour but not its structure, i.e., they take a monolithic view and do not make explicit how the system is composed and what are the interacting components of which it is made. In this last chapter we introduce a language, called PEPA (Performance Evaluation Process Algebra), for composing stochastic processes and carry out their quantitative analysis. PEPA builds on CSP (Calculus for Sequential Processes), a process algebra similar to CCS but with slightly different primitives. In particular, it relies on multiway communication instead of binary (i/o) one. PEPA actions are labelled with rates and a CTMC can be derived from the LTS of a PEPA process without much efforts to evaluate quantitative properties of the modelled system. The advantage is that the PEPA description of the CTMC remains as a blueprint of the system and allows direct re-use of processes.

16.1 From Qualitative to Quantitative Analysis

To understand the differences between qualitative analysis and quantitative analysis, we remark that qualitative questions like:

- Will the system reach a particular state?
- Does the system implementation match its specification?
- Does a given property ϕ hold within the system?

are replaced by quantitative questions like:

- How long will the system take on average to reach a particular state?
- With what probability does the system implementation match its specification?
- Does a given property ϕ hold within the system within time t with probability p?

Jane Hillston defined the PEPA language in 1994. PEPA has been developed as a high-level language for the description of continuous time Markov chains. Over the years PEPA has been shown to provide an expressive formal language for modelling distributed systems. PEPA models are obtained as the structure assembly of components that perform individual activities at certain rates and can cooperate on shared actions. The most important features of PEPA w.r.t. other approaches to performance modelling are:

compositionality:	the ability to model a system as the interaction of subsystems, as opposed to poorly modular approaches;
formality:	a rigorous semantics giving a precise meaning to all terms in the language and solving any ambiguities;
abstraction:	the ability to build up complex models from components, disregarding the details when it is appropriate to do so;
separation of concerns:	the ability to model the components and the interaction separately;
structure:	the ability to impose a clear structure to models, which makes them more understandable and easier to maintain;
refinement:	the ability to construct models systematically by refining their specifications;
reusability:	the ability to maintain a library of model components.

For example, queueing networks offer compositionality but not formality; stochastic extensions of Petri nets offer formality but not compositionality; neither offer abstraction mechanisms.

PEPA was obtained by extending CSP (Calculus for Sequential Processes) with probabilities. We start with a brief introduction to CSP, then we will conclude with the presentation of the syntax and operational semantics of PEPA.

16.2 CSP

Communicating Sequential Processes (CSP) is a process algebra introduced by Tony Hoare in 1978 and is a very powerful tool for systems specification and verification. Contrary to CCS, CSP actions have no dual counterpart and the synchronisation between two or more processes is possible when they all perform the same action α (in which case the observable label of the synchronisation is still α). Since during communication the joint action remains visible to the environment, it can be used to interact with other (more than two) processes, realising multiway synchronisation.

16.2.1 Syntax of CSP

We assume that a set Λ of actions α is given. The syntax of CSP processes is defined the below, where $L \subseteq \Lambda$ is any set of actions:

$$P,Q \quad ::= \quad \mathbf{nil} \quad | \quad \alpha.P \quad | \quad P+Q \quad | \quad P \underset{L}{\bowtie} Q \quad | \quad P/L \quad | \quad C$$

We briefly comment on each operator:

nil:	is the inactive process;
$\alpha.P$:	is a process which can perform an action α and then behaves like P;
$P+Q$:	is a process which can choose to behave like P or like Q;
P/L:	is the hiding operator; if P performs an action $\alpha \in L$ then P/L performs an unobservable silent action τ;
$P \underset{L}{\bowtie} Q$:	is a synchronisation operator, also called *cooperation combinator*. More precisely, it denotes an indexed family of operators, one for each possible set of actions L. The set L is called *cooperation set* and fixes the set of *shared actions* between P and Q. Processes P and Q can use the actions in L to synchronise each other. The actions not included in L are called *individual activities* and can be performed separately by P and Q. As a special case, if $L = \varnothing$ then all the actions of P and Q are just interleaved.
C:	is the name, called *constant*, of a recursively defined process that we assume given in a separate set $\Delta = \{C_i \stackrel{\text{def}}{=} P_i\}_{i \in I}$ of declarations.

16.2.2 Operational Semantics of CSP

Now we present the semantics of CSP. As we have done for CCS and π-calculus, we define the operational semantics of CSP as an LTS derived by a set of inference rules. As usual, theorems take the form $P \xrightarrow{\alpha} P'$, meaning that the process P in one step evolves to the process P' by executing the action α.

16.2.2.1 Inactive Process

There is no rule for the inactive process **nil**.

16.2.2.2 Action Prefix and Choice

The rules for action prefix and choice operators are the same as in CCS.

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$$

16.2.2.3 Hiding

The hiding operator should not be confused with the restriction operator of CCS: first, hiding takes a set L of labels as a parameter, while restriction takes a single action; second, when $P \xrightarrow{\alpha} P'$ with $\alpha \in L$ we have that $P/L \xrightarrow{\tau} P'/L$, while $P\backslash\alpha$ blocks the action. Instead, P/L and $P\backslash\alpha$ behaves similarly w.r.t. actions not included in $L \cup \{\alpha\}$.

$$\frac{P \xrightarrow{\alpha} P' \quad \alpha \notin L}{P/L \xrightarrow{\alpha} P'/L} \qquad \frac{P \xrightarrow{\alpha} P' \quad \alpha \in L}{P/L \xrightarrow{\tau} P'/L}$$

16.2.2.4 Cooperation Combinator

There are three rules for the cooperation combinator $\underset{L}{\bowtie}$: the first two rules allow the interleaving of actions not in L, while the third rule forces the synchronisation of the two processes when performing actions in L. Differently from CCS, when two processes synchronises on α the observed label is still α and not τ.

$$\frac{P \xrightarrow{\alpha} P' \quad \alpha \notin L}{P \underset{L}{\bowtie} Q \xrightarrow{\alpha} P' \underset{L}{\bowtie} Q} \qquad \frac{Q \xrightarrow{\alpha} Q' \quad \alpha \notin L}{P \underset{L}{\bowtie} Q \xrightarrow{\alpha} P \underset{L}{\bowtie} Q'} \qquad \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\alpha} Q' \quad \alpha \in L}{P \underset{L}{\bowtie} Q \xrightarrow{\alpha} P' \underset{L}{\bowtie} Q'}$$

Note that the cooperation combinator is not associative. For example

$$(\alpha.\beta.\mathbf{nil} \underset{\{\alpha\}}{\bowtie} \mathbf{nil}) \underset{\varnothing}{\bowtie} \alpha.\mathbf{nil} \neq (\alpha.\beta.\mathbf{nil}) \underset{\{\alpha\}}{\bowtie} (\mathbf{nil} \underset{\varnothing}{\bowtie} \alpha.\mathbf{nil}).$$

In fact the leftmost process can perform only an action α

$$(\alpha.\beta.\mathbf{nil} \underset{\{\alpha\}}{\bowtie} \mathbf{nil}) \underset{\varnothing}{\bowtie} \alpha.\mathbf{nil} \xrightarrow{\alpha} (\alpha.\beta.\mathbf{nil} \underset{\{\alpha\}}{\bowtie} \mathbf{nil}) \underset{\varnothing}{\bowtie} \mathbf{nil}$$

after which it is deadlock, whereas the rightmost process can perform a synchronisation on α and then it can perform another action β

$$(\alpha.\beta.\mathbf{nil}) \underset{\{\alpha\}}{\bowtie} (\mathbf{nil} \underset{\varnothing}{\bowtie} \alpha.\mathbf{nil}) \xrightarrow{\alpha} (\beta.\mathbf{nil}) \underset{\{\alpha\}}{\bowtie} (\mathbf{nil} \underset{\varnothing}{\bowtie} \mathbf{nil}) \xrightarrow{\beta} \mathbf{nil} \underset{\{\alpha\}}{\bowtie} (\mathbf{nil} \underset{\varnothing}{\bowtie} \mathbf{nil}).$$

16.2.2.5 Constants

Finally, the rule for constants unfolds the recursive definition $C \overset{\text{def}}{=} P$, so that C has all transitions that P has.

$$\frac{(C \overset{\text{def}}{=} P) \in \Delta \quad P \xrightarrow{\alpha} P'}{C \xrightarrow{\alpha} P'}$$

16.3 PEPA

As we said, PEPA is obtained by adding probabilities to the execution of actions. As we will see, PEPA processes are stochastic: there are not explicit probabilistic operators in PEPA, but the probabilistic behaviour is obtained by associating an exponentially distributed continuous random variable to each action prefix; this random variable represents the time needed to execute the action. These random variables lead to a clear relationship between the process algebra model and a CTMC. Via this underlying Markov process, performance measures can then be extracted from the model.

16.3.1 Syntax of PEPA

In PEPA an action is a pair (α, r), where α is the action type and r is the rate of the continuous random variable associated with the action. The rate r can be any positive real number. The grammar for PEPA process is given below:

$$P,Q \quad ::= \quad \mathbf{nil} \quad | \quad (\alpha,r).P \quad | \quad P+Q \quad | \quad P \underset{L}{\bowtie} Q \quad | \quad P/L \quad | \quad C$$

$(\alpha,r).P$: is a process which can perform an action α and then behaves like P. In this case the rate r is used to define the exponential variable which describes the duration of the action. A component may have a purely sequential behaviour, repeatedly undertaking one activity after another and possibly returning to its initial state. As a simple example, consider a web server in a distributed system that can serve one request at a time:

$$WS \stackrel{\text{def}}{=} (request, \top).(serve, \mu).(respond, \top).WS$$

In some cases, as here, the rate of an action falls out of the control of this component: such actions are carried out jointly with another component, with the current component playing some sort of passive role. For example, the web server is passive with respect to the request and respond actions, as it cannot influence the rate at which applications execute these actions. This is recorded by using the distinguished rate $\top$ which we can assume to represent an extremely high value that cannot influence the rate of interacting components.

$P+Q$: has the same meaning of the CSP operator for choice. For example, we can consider an application in a distributed system that can either access a locally available method (with probability p_1) or access to a remote web service (with probability $p_2 = 1 - p_1$). The decision is taken by performing a think action which is parametric to the rate λ:

$$\begin{aligned} Appl &\stackrel{\text{def}}{=} (think, p_1 \cdot \lambda).(local, m).Appl \\ &\quad + (think, p_2 \cdot \lambda).(request, rq).(respond, rp).Appl \end{aligned}$$

$P \bowtie_L Q$: has the same meaning of the CSP operator. In the web service example, we can assume that the application and the web server interact over the set of shared actions $L = \{request, respond\}$:

$$Sys \stackrel{\text{def}}{=} (Appl \bowtie_{\varnothing} Appl) \bowtie_L WS$$

During the interaction, the resulting action will have the same type of the shared action and a rate reflecting the rate of the slowest action.

P/L: is the same as the CSP hiding operator: the duration of the action is unaffected, but its type becomes τ. In our running example, we may want to hide the local computation of *Appl* to the environment:

$$Appl' \stackrel{\text{def}}{=} Appl/\{local\}$$

C: is the name of a recursively defined process such as $C \stackrel{\text{def}}{=} P$ that we assume given in a separate set Δ of declarations. Using recursive definitions as the ones given above for *Appl* and WS, we are able to describe components with infinite behaviour without introducing an explicit recursion or replication operator.

Usually we are interested only in those agents which have an ergodic underlying Markov process, since we want to apply the steady state analysis. It has been shown that it is possible to ensure ergodicity by using syntactic restrictions on the agents. In particular, the class of PEPA terms which satisfy these syntactic conditions are called *cyclic components* and they can be described by the following grammar:

$$\begin{array}{lcl} P, Q & ::= & S \quad | \quad P \bowtie_L Q \quad | \quad P/L \\ S, T & ::= & (\alpha, r).S \quad | \quad S + T \quad | \quad C \end{array}$$

where *sequential processes* S, T can be distinguished from general processes P, Q and it is required that each recursive process C is sequential, i.e., it must be $(C \stackrel{\text{def}}{=} S) \in \Delta$ for some sequential process S.

16.3.2 Operational Semantics of PEPA

PEPA operational semantics is defined by a rule system similar to the one for CSP. In the case of PEPA, well formed formulas have the form $P \xrightarrow{(\alpha,r)} Q$ for suitable PEPA processes P and Q, activity α and rate r. We assume a set Δ of declarations is available.

16.3.2.1 Inactive Process

As usual, there is no rule for the inactive process **nil**.

16.3.2.2 Action Prefix and Choice

The rules for action prefix and choice are essentially the same as the ones for CSP: the only difference is that the rate r is recorded in the label of transitions.

$$\frac{}{(\alpha,r).P \xrightarrow{(\alpha,r)} P} \qquad \frac{P \xrightarrow{(\alpha,r)} P'}{P+Q \xrightarrow{(\alpha,r)} P'} \qquad \frac{Q \xrightarrow{(\alpha,r)} Q'}{P+Q \xrightarrow{(\alpha,r)} Q'}$$

16.3.2.3 Constants

The rule for constants is the same as that of CSP, except for the fact transition labels carry also the rate.

$$\frac{(C \overset{\text{def}}{=} P) \in \Delta \quad P \xrightarrow{(\alpha,r)} P'}{C \xrightarrow{(\alpha,r)} P'}$$

16.3.2.4 Hiding

Also the rules for hiding resemble the ones for CSP. Note that when $P \xrightarrow{(\alpha,r)} P'$ with $\alpha \in L$, the rate r is associated with τ in $P/L \xrightarrow{(\tau,r)} P'/L$.

$$\frac{P \xrightarrow{(\alpha,r)} P' \quad \alpha \notin L}{P/L \xrightarrow{(\alpha,r)} P'/L} \qquad \frac{P \xrightarrow{(\alpha,r)} P' \quad \alpha \in L}{P/L \xrightarrow{(\tau,r)} P'/L}$$

16.3.2.5 Cooperation Combinator

As for CSP, we have three rules for the cooperation combinator. The first two rules are for action interleaving and deserve no further comment.

$$\frac{P \xrightarrow{(\alpha,r)} P' \quad \alpha \notin L}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P' \underset{L}{\bowtie} Q} \qquad \frac{Q \xrightarrow{(\alpha,r)} Q' \quad \alpha \notin L}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P \underset{L}{\bowtie} Q'}$$

The third rule, called *cooperation* rule (see below), is the most interesting one, because it deals with synchronisation and with the need to combine rates. The cooperation rule exploits the so-called *apparent rate* of action α in P, written $r_\alpha(P)$, which is defined by structural recursion as follows:

$$\begin{aligned}
r_\alpha(\mathbf{nil}) &\stackrel{\text{def}}{=} 0 \\
r_\alpha((\beta,r).P) &\stackrel{\text{def}}{=} \begin{cases} r & \text{if } \alpha = \beta \\ 0 & \text{if } \alpha \neq \beta \end{cases} \\
r_\alpha(P+Q) &\stackrel{\text{def}}{=} r_\alpha(P) + r_\alpha(Q) \\
r_\alpha(P/L) &\stackrel{\text{def}}{=} \begin{cases} r_\alpha(P) & \text{if } \alpha \notin L \\ 0 & \text{if } \alpha \in L \end{cases} \\
r_\alpha(P \underset{L}{\bowtie} Q) &\stackrel{\text{def}}{=} \begin{cases} min(r_\alpha(P), r_\alpha(Q)) & \text{if } \alpha \in L \\ r_\alpha(P) + r_\alpha(Q) & \text{if } \alpha \notin L \end{cases} \\
r_\alpha(C) &\stackrel{\text{def}}{=} r_\alpha(P) \quad \text{if } (C \stackrel{\text{def}}{=} P) \in \Delta
\end{aligned}$$

Roughly, the apparent rate $r_\alpha(S)$ is the sum of the rates of all distinct actions α that can be performed by S, thus $r_\alpha(S)$ expresses the overall rate of α in S (because of the property of rates of exponentially distributed variables in Theorem 14.2). Notably, in the case of shared actions $P \underset{L}{\bowtie} Q$ the apparent rate is the slowest of the apparent rates of P and Q. The cooperation rule is:

$$\frac{P \xrightarrow{(\alpha,r_1)} P' \quad Q \xrightarrow{(\alpha,r_2)} Q' \quad \alpha \in L}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P' \underset{L}{\bowtie} Q'} \quad \text{where } r = r_\alpha(P \underset{L}{\bowtie} Q) \times \frac{r_1}{r_\alpha(P)} \times \frac{r_2}{r_\alpha(Q)}$$

Let us now explain the calculation

$$r = r_\alpha(P \underset{L}{\bowtie} Q) \times \frac{r_1}{r_\alpha(P)} \times \frac{r_2}{r_\alpha(Q)}$$

that appears in the cooperation rule. The best way to resolve what should be the rate of the shared action has been a topic of some debate. The definition of cooperation in PEPA is based on the assumption that a component cannot be made to exceed its bounded capacity for carrying out the shared actions, where the bounded capacity consists of the apparent rate of the action. The underlying assumption is that the choice of a specific action (with rate r_i) to carry on the shared activity occurs independently in the two cooperating components P and Q. Now, the probability that a specific action (α, r_i) is chosen by P is (see Theorem 14.3)

$$\frac{r_i}{r_\alpha(P)}.$$

Then, from the choice independence we obtain the combined probability

$$\frac{r_1}{r_\alpha(P)} \times \frac{r_2}{r_\alpha(Q)}.$$

Finally, the resulting rate is the product of the apparent rate

$$r_\alpha(P \underset{L}{\bowtie} Q) = \min(r_\alpha(P), r_\alpha(Q))$$

and the above probability. Notice that if we sum up the rates of all possible synchronisations on α of $P \underset{L}{\bowtie} Q$ we just get $min(r_\alpha(P), r_\alpha(Q))$ (see the example below).

Example 16.1. Let us define two PEPA agents as follows:

$$P \overset{\text{def}}{=} (\alpha, r).P_1 + \ldots + (\alpha, r).P_n \qquad\qquad Q \overset{\text{def}}{=} (\alpha, r).Q_1 + \ldots + (\alpha, r).Q_m$$

for some $n \leq m$. So we have the the following apparent rates:

$$\begin{aligned} r_\alpha(P) &\overset{\text{def}}{=} n \times r \\ r_\alpha(Q) &\overset{\text{def}}{=} m \times r \\ r_\alpha(P \underset{\{\alpha\}}{\bowtie} Q) &\overset{\text{def}}{=} min(r_\alpha(P), r_\alpha(Q)) = n \times r \end{aligned}$$

By the rules for action prefix and choice, we have transitions:

$$P \xrightarrow{(\alpha,r)} P_i \quad \text{for } i \in [1,n] \qquad Q \xrightarrow{(\alpha,r)} Q_j \quad \text{for } j \in [1,m]$$

Then we have $m \times n$ possible ways of synchronising P and Q, :

$$P \underset{\{\alpha\}}{\bowtie} Q \xrightarrow{(\alpha,r')} P_i \underset{\{\alpha\}}{\bowtie} Q_j \quad \text{for } i \in [1,n] \text{ and } j \in [1,m]$$

where

$$r' = (n \times r) \times \frac{r}{n \times r} \times \frac{r}{m \times r} = \frac{r}{m}.$$

So we have $m \times n$ transitions with rate r/m and, in fact, the apparent rate of the synchronisation is:

$$m \times n \times \frac{r}{m} = n \times r = r_\alpha(P \underset{\{\alpha\}}{\bowtie} Q).$$

Remark 16.1. The selection of the exponential distribution as the governing distribution for action durations in PEPA has profound consequences. In terms of the underlying stochastic process, it is the only choice which gives rise to a Markov process. This is due to the memoryless properties of the exponential distribution: the time until the next event is independent of the time since the last event, because the exponential distribution forgets how long it has already waited. For instance, if we consider the process $(\alpha, r).\,\mathbf{nil} \underset{\varnothing}{\bowtie} (\beta, s).\,\mathbf{nil}$ and the system performs the action α, the time needed to complete β from $\mathbf{nil} \underset{\varnothing}{\bowtie} (\beta, s).\,\mathbf{nil}$ does not need to consider the time already taken to carry out the action α.

The underlying CTMC is obtained from the LTS by associating a (global) state with each process, and the transitions between states are derived from the transitions of the LTS. Since all activity durations are exponentially distributed, the total transition rate between two states will be the sum of the activity rates labelling arcs connecting the corresponding nodes in the LTS.

The PEPA language is supported by a range of tools and by a wide community of users. PEPA application areas span the subject areas of informatics and engineering including, e.g., cellular telephone networks, database systems, diagnostic expert systems, multiprocessor access-contention protocols, protocols for fault-tolerant systems, software architectures. Additional information and a PEPA Eclipse Plug-in are freely available at `http://www.dcs.ed.ac.uk/pepa/`.

We conclude this section by showing an example of modelling with PEPA.

Example 16.2 (Roland the gunman). We want to model a Far West duel. We have two main characters: Roland the gunman and his enemies. Upon its travels Roland will encounter some enemies with whom he will have no choice but to fight back. For simplicity we assume that Roland has two guns with one bullet in each and that each hit is fatal. We also assume that a sense of honour prevents an enemy from attacking Roland if he is already involved in a gun fight. We model the behaviour of Roland as follows. Normally, Roland is in an idle state $Roland_{\text{idle}}$, but when he is attacked (attacks) he moves to state $Roland_2$, where he has two bullets available in his gun:

$$Roland_{\text{idle}} \stackrel{\text{def}}{=} (\text{attack}, \top).Roland_2$$

In front of his enemies, Roland can act in three ways: if he hits them then he reloads his gun and returns idle; if he misses the enemies he tries a second attack (see $Roland_1$); finally if an enemy hits him, Roland dies.

$$\begin{aligned} Roland_2 \stackrel{\text{def}}{=}\ & (\text{hit}, r_{\text{hit}}).(\text{reload}, r_{\text{reload}}).Roland_{\text{idle}} \\ & +(\text{miss}, r_{\text{miss}}).Roland_1 \\ & +(\text{e-hit}, \top).Roland_{\text{dead}} \end{aligned}$$

The second attempt to shoot by Roland is analogous to the first one, but this time it is the last bullet in Rolands gun and if the enemy is missed no further shot is possible in $Roland_{\text{empty}}$ until the gun is reloaded.

$$\begin{aligned} Roland_1 \quad & \stackrel{\text{def}}{=} (\text{hit}, r_{\text{hit}}).(\text{reload}, r_{\text{reload}}).Roland_{\text{idle}} \\ & \quad +(\text{miss}, r_{\text{miss}}).Roland_{\text{empty}} \\ & \quad +(\text{e-hit}, \top).Roland_{\text{dead}} \\ Roland_{\text{empty}} & \stackrel{\text{def}}{=} (\text{reload}, r_{\text{reload}}).Roland_2 \\ & \quad +(\text{e-hit}, \top).Roland_{\text{dead}} \end{aligned}$$

Finally if Roland is dead he cannot perform any action.

$$Roland_{\text{dead}} \overset{\text{def}}{=} \mathbf{nil}$$

We describe enemies behaviour as follows. If the enemies are idle they can try to attack Roland:

$$Enemies_{\text{idle}} \overset{\text{def}}{=} (\text{attack}, r_{\text{attack}}).Enemies_{\text{attack}}$$

Enemies shoot once and either get hit or they hit Roland.

$$\begin{aligned} Enemies_{\text{attack}} \overset{\text{def}}{=} & (\text{e-hit}, r_{\text{e-hit}}).Enemies_{\text{idle}} \\ & +(hit, \top).Enemies_{\text{idle}} \end{aligned}$$

The rates involved in the model are measured in seconds, so a rate of 1.0 would indicate that the action is expected to occur once every second. We define the following rates:

$\top =$	about ∞	
$r_{\text{fire}} =$	1	one shot per second
$r_{\text{hit-success}} =$	0.8	80% of success
$r_{\text{hit}} =$	0.8	$r_{\text{fire}} \times r_{\text{hit-success}}$
$r_{\text{miss}} =$	0.2	$r_{\text{fire}} \times (1 - r_{\text{hit-success}})$
$r_{\text{reload}} =$	0.3	3 seconds to reload
$r_{\text{attack}} =$	0.01	Roland is attacked once every 100 seconds
$r_{\text{e-hit}} =$	0.02	Enemies can hit once every 50 seconds

So we model the duel as follows:

$$\text{Duel} \overset{\text{def}}{=} Roland_{idle} \underset{\{hit,attack,e-hit\}}{\bowtie} Enemies_{\text{idle}}$$

We can perform various types of analysis of the system by using standard methods. Using the steady state analysis, that we have seen in the previous chapters, we can prove that Roland will always die and the system will deadlock, because there is an infinite supply of enemies (so the system is not ergodic). Moreover we can answer many other questions by using the following techniques:

- Transient analysis: we can ask for the probability that Roland is dead after 1 hour, or the probability that Roland will have killed some enemy within 30 minutes.
- Passage time analysis: we can ask for the probability of passing at least 10 seconds from the first attack to Roland to the time it has hit 3 enemies, or the probability that 1 minute after he is attacked Roland has killed his attacker (i.e., the probability that the model performs a *hit* action within 1 minute after having performed an *attack* action).

Problems

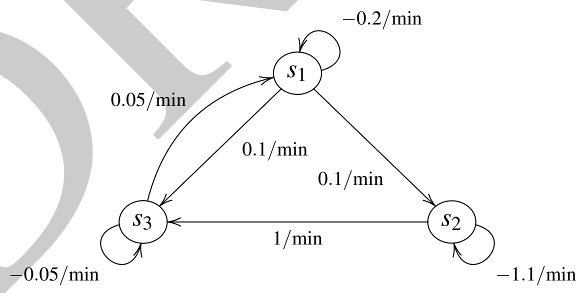
16.1. We have defined CTMC bisimilarity in the case of *unlabeled* transition systems, while PEPA transition system is labeled. Extend the definition of bisimilarity to the labeled version.

16.2. Consider a simple system in which a process repeatedly carries out some task. In order to complete its task the process needs access to a resource for part, but not all, of the time. We want to model the process and the resource as two separate PEPA agents: *Process* and *Resource*, respectively. The *Process* will undertake two activities consecutively: *get* with some rate *rg*, in cooperation with the *Resource*, and *task* at rate *rt*, representing the remainder of its processing task. Similarly the *Resource* will engage in two activities consecutively: *get*, at a rate $rg' > 2rg$ and *update*, at rate *ru*.

1. Give the PEPA specification of a system composed with two *Process*es that compete for one shared *Resource*.
2. What is the apparent rate of action *get* in the initial state of the system?
3. Draw the complete LTS (eight states) of the system and list all its transitions.

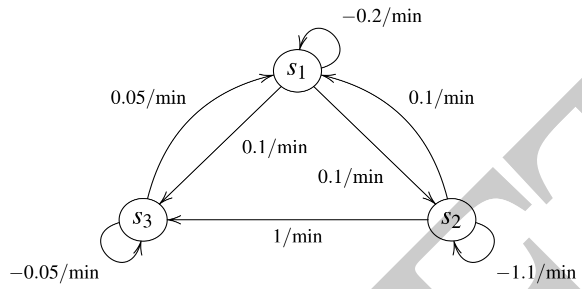
16.3. In a multiprocessor system with shared memory, processes must compete to use the memory bus. Consider the case of two identical processes. Each process has cyclic behaviour: it performs some local activity (local action *think*), accesses the bus (synchronization action *get*), operates on the memory (local action *use*) and then releases the bus (synchronization action *rel*). The bus has cyclic behavior with actions *get* and *rel*. Define a PEPA program representing the system and derive the corresponding CTMC (with actions). Find the bisimilar states according to the notion of bisimilarity in Problem 16.1 and draw the minimal CTMC.

16.4. Consider the taxi driver scenario from Problem 14.3, but this time represented as the CTMC in the Figure below, where rates are defined in 1/minutes, e.g., costumers show up every 10 minutes (rate 0.1/min) and rides last 20 minutes (rate 0.05/min) .



Assuming a unique label l for all the transitions, and disregarding self loops, define a PEPA agent for the system, and show that all states are different in terms of bisimilarity. Finally, to study the steady state behaviour of the system, introduce the self loops,[1] and write and solve a system of linear equations similar to the one seen for DTMC: $\pi\, Q = 0$ and $\sum_i \pi_i = 1$. The equations express the fact that, for every state s_i, the probability flow from the other states to state s_i is the same as the probability flow from state s_i to the other states.

16.5. Consider the taxi driver scenario from Problem 16.4, but this time with the option of going back to state s_1 (parking) from state s_2 (moving slowly looking for costumers) as in the figure below.



Assuming a unique label l for all the transitions, and disregarding self loops, define a PEPA agent for the system, and show that all states are different in terms of bisimilarity. Finally, to study the steady state behaviour of the system, introduce the self loops, decorated with suitable negative rates, and write and solve a system of linear equations similar to the one seen for DTMC: $\pi\, Q = 0$ and $\sum_i \pi_i = 1$. The equations express the fact that, for every state s_i, the probability flow from the other states to state s_i is the same as the probability flow from state s_i to the other states (see Section 14.4.5).

16.6. Let the (infinitely many) PEPA processes $\{A_\alpha, B_\beta\}$, indexed by strings $\alpha, \beta \in \{0,1\}^*$ be defined as:

$$A_\alpha \stackrel{\text{def}}{=} (a,\lambda).B_{\alpha 0} + (a,\lambda).B_{\alpha 1} \qquad B_\beta \stackrel{\text{def}}{=} (b,\lambda).A_{\beta 0} + (b,\lambda).A_{\beta 1}.$$

Consider the (sequential) PEPA program $P \stackrel{\text{def}}{=} A_\varepsilon$, for ε the empty string:

1. draw (at least in part) the transition system of P;
2. find the states reachable from P;
3. determine the bisimilar states;
4. finally, find the smallest PEPA program bisimilar to P.

16.7. Let the (infinitely many) PEPA processes A_α, indexed by strings $\alpha \in \{0,1\}^*$ be defined as:

$$A_\alpha \stackrel{\text{def}}{=} (a,\lambda).A_{\alpha 0} + (a,\lambda).A_{\alpha 1}$$

Consider the (sequential) PEPA program $P \stackrel{\text{def}}{=} A_\varepsilon$, for ε the empty string:

[1] Remind that, in the infinitesimal generator matrix of a CTMC, self loops are decorated with negative rates which are negated apparent rates, namely the negated sums of all the outgoing rates.

1. draw (at least in part) the transition system of P;
2. find the states reachable from *P*;
3. determine the bisimilar states;
4. finally, find the smallest PEPA program bisimilar to *P*.

16.8. Consider the PEPA process *A* with

$$A \overset{\text{def}}{=} (\alpha,\lambda).B + (\alpha,\lambda).C \qquad B \overset{\text{def}}{=} (\alpha,\lambda).A + (\alpha,\lambda).C \qquad C \overset{\text{def}}{=} (\alpha,\lambda).A.$$

and derive the corresponding finite state CTMC.

1. What is the probability distribution of staying in *B*?
2. If $\lambda = 0.1\ sec^{-1}$, what it the probability that the system be still in *B* after 10 seconds?
3. Are there bisimilar states?
4. Finally, to study the steady state behaviour of the system, introduce the self loops, decorated with suitable negative rates, show that the system is ergodic and write and solve a system of linear equations similar to the one seen for DTMC.

16.9. Consider *n transmitters* $T_0, T_1, \ldots, T_{n-1}$ connected by a token ring. At any moment, a transmitter *i* can be *ready* to transmit or *not ready*. It becomes ready with a private action *arrive* and a rate λ. Once ready, it stays ready until it transmits, and then it becomes not ready with an action $serve_i$ and rate μ. To resolve conflicts, only the transmitter with the *token* can operate. There is only one token *K*, which at any moment is located at some transmitter T_i. If transmitter T_i is not ready, the token synchronises with it with an action $walkon_i$ and rate ω moving from transmitter T_i to transmitter $T_{i+1\ (mod\ n)}$. If transmitter T_i is ready, the token synchronises with it with action $serve_i$ and rate μ and stays at transmitter T_i.

Write a PEPA process modelling the above system as follows:

1. define recursively all the states of T_i, for $i \in [0, n-1]$ and of K;
2. define the whole system by choosing the initial state where all transmitters are not ready and the token in at T_0 and composing in parallel all of them with $\underset{L}{\bowtie}$, with *L* being the set of synchronised actions.
3. Then draw the transition system corresponding to $n = 2$, and compute the bisimilarity relation.
4. Finally define a function *f* such that $f(n)$ is the number of (reachable) states for the system with *n* transmitters.