Lecture Notes in Computer Science 5569

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Marco Bernardo Luca Padovani
Gianluigi Zavattaro (Eds.)

Formal Methods for Web Services

9th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2009
Bertinoro, Italy, June 1-6, 2009
Advanced Lectures

Springer

Volume Editors

Marco Bernardo
Università di Urbino "Carlo Bo"
Istituto di Scienze e Tecnologie dell'Informazione
Piazza della Repubblica 13, 61029 Urbino, Italy
E-mail: bernardo@sti.uniurb.it

Luca Padovani
Università di Urbino "Carlo Bo"
Istituto di Scienze e Tecnologie dell'Informazione
Piazza della Repubblica 13, 61029 Urbino, Italy
E-mail: padovani@sti.uniurb.it

Gianluigi Zavattaro
Università di Bologna
Dipartimento di Scienze dell'Informazione
Mura Anteo Zamboni 7, 40127 Bologna, Italy
E-mail: zavattar@cs.unibo.it

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D.2.4, H.3.5, D.3.1, F.4, H.3.5

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-642-01917-X Springer Berlin Heidelberg New York
ISBN-13 978-3-642-01917-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12676064 06/3180 5 4 3 2 1 0

Preface

This volume presents the set of papers accompanying the lectures of the 9th International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM).

This series of schools addresses the use of formal methods in computer science as a prominent approach to the rigorous design of computer, communication, and software systems. The main aim of the SFM series is to offer a good spectrum of current research in foundations as well as applications of formal methods, which can be of help for graduate students and young researchers who intend to approach the field.

SFM 2009 was devoted to formal methods for Web services and covered several aspects including choreography, orchestration, description techniques, interaction, synthesis, composition, session types, contracts, verification, security, and performance.

This volume comprises eight articles. Bruni's paper overviews some of the most recently proposed abstractions in the setting of process calculi tailored to the well-disciplined handling of issues such as long-running interactions, orchestration, and unexpected events. Van der Aalst, Mooij, Stahl, and Wolf provide some foundational notions related to service interaction and address in a Petri net setting challenges like how to expose a service, how to replace and refine services, and how to generate service adapters. The paper by Marconi and Pistore presents a survey of existing approaches to the synthesis of Web service compositions, a difficult and error-prone task that requires automated solutions. Vasconcelos's paper illustrates a reconstruction of session types in a linear π-calculus where types are qualified as linear or unrestricted, together with an algorithmic type-checking system. Carbone, Yoshida, and Honda explore two extensions of session types to interactional exceptions and multiparty sessions in the presence of asynchronous communications. Padovani's paper discusses a set-theoretic semantics of contracts, which is employed for defining a family of equivalence relations that can be effectively used for discovering and adapting Web services implementing specific contracts. The paper by Bravetti and Zavattaro also focusses on contracts by following the idea of designing a service system through the description of the behavior of each of its participants and then instantiating such participants by retrieving services exposing contracts that conform to the given behaviors. Clark, Gilmore, and Tribastone introduce quantitative methods for analyzing Web services with the goal of understanding how they will perform under increased demand or when asked to serve a larger pool of service subscribers.

We believe that this book offers a comprehensive view of what has been done and what is going on worldwide in the field of formal methods for Web services. We wish to thank all the speakers and all the participants for a lively and fruitful

school. We also wish to thank the entire staff of the University Residential Center of Bertinoro for the organizational and administrative support. Finally, we are very grateful to the University of Bologna, which kindly provided sponsorship for this event under the International Summer School Program.

June 2009

Marco Bernardo
Luca Padovani
Gianluigi Zavattaro

Table of Contents

Calculi for Service-Oriented Computing*

Roberto Bruni

Dipartimento di Informatica, Università di Pisa
bruni@di.unipi.it

Abstract. It is widely recognised that process calculi stay to concurrent computing as lambda-calculus stays to sequential computing; in fact, they lay abstract, rigorous foundations for the analysis of interactive, communicating systems. Nowadays, the increasing popularity of Service-Oriented Computing (SOC) challenges the quest for novel abstractions tailored to the well-disciplined handling of specific issues, like long running interactions, orchestration, and unexpected events. In fact, these features emerge neatly in most SOC applications and need to be studied as first-class aspects, whereas they would be obfuscated if dealt with by sophisticated encoding in traditional process calculi. This paper overviews some of the most recent proposals emerged in the literature, pointing out their main characteristics and presents in more detail one such proposal, called CaSPiS, by providing several examples to give evidence of its flexibility. No prior acquaintance with process calculi is assumed, indeed a gentle introduction to their basics is provided before the more advanced material be presented.

1 Introduction

Service-oriented computing has been one of the latest trend in the IT community, finding in Web Services (WS) technology its major realisation. Services are autonomous computational entities, that are developed separately, loosely coupled, globally available over a widely distributed network in a platform-independent way, and not fully reliable. Service computing consists of assembling services in well-engineered ways to form complex open-ended applications, and this must be done in a highly dynamic way, possibly on demand. To this aim, it is essential to find suitable abstractions to describe services, the so-called *service descriptors*, to be published in public registries. Such registries can be queried by other services and applications to locate those services that best match certain requirements, yielding a brokering architecture. When satisfactory matches are found, then the located services can be dynamically linked and invoked. Therefore, service engineering has to do with the development of methodologies, techniques, formal methods and tools able to guarantee a safe service composition, in the sense of being able to provide some strong guarantees on such dynamic, open-ended applications by applying some static or semi-static analysis.

WS technology has established *de facto* standards for naming schemes and service access (URI, URL), service descriptors (WSDL and BPEL in UDDI registries), communication protocols (SOAP over HTTP, TCP/IP and SMTP) and message format (XML)

* Research supported by the Project FET-GC II IST-2005-16004 Sensoria and by the Italian FIRB Project Tocai.it.

M. Bernardo, L. Padovani, and G. Zavattaro (Eds.): SFM 2009, LNCS 5569, pp. 1–41, 2009.

over the web. Existing infrastructures already enable providers to describe web services in terms of their interfaces, access policies and behaviour, and to combine simpler services into more structured and complex ones. However, some research and solid foundations are still needed to move WS technology from skilled handcrafting to an engineered practice, a step where formal methods must play a fundamental role. For example, it has been shown that the lack of unambiguous semantics of BPEL has led different BPEL engines to exhibit different behaviours under the same circumstances [43].

Research on formal methods for SOC can be roughly separated in two main strands, both equally worth the effort: one dedicated to establishing the missing theoretical foundations of state-of-the-art technologies, so to fix rigorous semantic and logic frameworks for the analysis and verification of SOC and WS systems; another one aimed to rethink the design and development of next generation technology, by understanding the key distinguishing features of SOC, assessing the necessary bits of theory for them in technology agnostic terms, and paving the way to their well-disciplined engineering. In both cases, the mathematical models and tools from the literature that seem to be particularly suited are those coming from concurrency theory, ranging from workflow like models like Petri nets, to Graph Transformation systems and process calculi.

As suggested by the title of this contribution, we shall focus on the use of process calculi for modelling SOC systems. This choice is motivated by the more natural way in which process calculi can accommodate for SOC features such as open-endedness, dynamicity, compositionality, interaction and event handling w.r.t. the other afore mentioned models. Moreover, we shall favour the second strand of research outlined above, trying to distill some key aspects of SOC together with a small set of primitives associated with them and to expose some of the main causes (motivation) and consequences (benefits) of our approach.

Due to the particular nature of this volume, which contains the proceedings of a summer school, and the audience to which this paper is oriented, which for the most we assume to consist of young computer science researchers, we have decided to structure this paper as a tutorial, so that no prior acquaintance with process calculi is assumed on the reader. Being worried that the more expert readers can find some arguments of our survey not dealt with at the sufficient levels of details for their taste, we added, whenever necessary, suitable links to the more advanced papers and texts where the technicalities are exposed in their full glory. Furthermore, we have put some efforts in trying to accompany each calculus by original examples and modelling puzzles in the hope they will provide an enjoyable reading experience by themselves, possibly reusable as course material.

1.1 What You Will Not Find Here

The level of abstraction at which we intend to model SOC systems disregards the technologies and the implementation details, hence we are to some extent disconnected from current WS standards. More precisely, we disregards those aspects related to the so-called semantic web, like ontologies for classifying services, XML coding and standardisation issues. In fact these aspects can be superimposed later, on the concrete realisation of our techniques.

For analogous reasons, we are not concerned with the exact ways in which services and their descriptors are made public available, queried and located, even if some of these issues can be reasonably encoded in the same formalisms we shall present. Instead we handle service publication, discovery and linking in terms of name-handling á la pi-calculus, i.e. the scope of certain service names can be restricted, new services can be dynamically deployed and updated, their names can be communicated and extruded to enlarge their scope, etc.

Moreover, we abstract away from non-functional aspects (like Quality of Service and Service Level Agreement) and quantitative analysis, which also constitute themselves an active area of research. On the other hand, some preliminary ideas in this field have already led to process calculi extensions that are compatible with the proposals discussed here and we give relevant pointers to the related literature.

We also deliberately omit the exact formulation of many useful theorems (and any proof sketch) from the literature, which we try to replace by more intuitive descriptions of their underlying properties and consequences, at the informal level.

1.2 Aspects of Interest

The common trait of all issues we aim to encompass here is the handy, disciplined composition of services. This includes: the possibility to extract service descriptors that carry some behavioural information rather than mere syntactic information as those found in WSDL documents; the possibility to carry long-running conversation between the service caller and its callee, which are far more general than limited one-way and request-response patterns of WS; suitable techniques for checking the behavioural conformance of the service to be invoked w.r.t. the application requirements; the way in which service invocations and their outcomes can be orchestrated; the way in which the system can foresee at the design time the actions to undergo in case some unexpected event will happen during a conversation, like a peer abandoning a business transaction. More precisely, we briefly discuss below different alternatives proposed in the literature on the above topics, and outline our preferred design choices.

Orchestration and choreography: The terms *orchestration* and *choreography* were coined to describe two different flavours of service compositions: orchestration is about describing and executing a single view point model, while choreography is about specifying and guiding a global model. Though the difference between the two terms can be sometimes abused or blurred, substantially orchestration is usually associated with an executable flavour, for which a centralised orchestration engine is responsible (although distributed engines can be also considered), as opposed to the fully distributed vision of choreography, usually associated with some sort of protocol narration. Roughly, from a formal modelling viewpoint, orchestration is mainly concerned with governing the control and data flow between services, while choreography is concerned with interaction protocols between single and composite autonomous services. Our presentation shall privilege orchestration, but out approach is compatible with the choreography perspective, as the type systems defined to check the conformance of services w.r.t. the requirements of the invoker share some similarities with the use of so-called *contracts* to express choreographies.

Interaction: Process calculi can exploit different forms of interactions, ranging from shared data-space, to event-based (subscribe-notify), and message passing. We shall rely on synchronous message passing, that is best suited for the level of abstraction of this tutorial.

Sessions and correlation sets: When long-running conversation with services are established, different instances of the same service can be running concurrently to serve different requests. Therefore it is important to route interaction between the correct pairs, avoiding any interference. Web service standards exploit the idea of *correlation sets*, i.e., pre-defined subsets of the invocation parameters that are used each time to choose the corresponding service instance (e.g., requests are routed according to usernames). Though correlation sets offer a good expressiveness, we argue that they might complicate static analysis, because all interactions rely on data values. For example, applications can interfere with each other if they know (or use by chance) the right values. A different school of thought advocates the notion of a *session* as a more convenient abstraction mechanism for enclosing arbitrarily complex interactions between peers. Session keys are data-independent and can be created implicitly when the service is first invoked. This way, type systems can be more easily developed to check properties like the presence of exactly one peer. Session can come in different flavours: nested, interleaved, with delegation, used recursively, dyadic, multy-party, mergeable, closeable, permeable, etc. We shall focus on primitives for a well-disciplined use of nested, dyadic and closeable sessions.

Compensations and session handlers: Each service has full autonomy in denying a request or abandoning a pending interaction. It is then important to rely on standard mechanisms for programming such decisions and to handle their consequences in a *safe* way. For example, the classical travel agency scenario may involve a complex interaction between the customer and the travel agent, to let the service learn the customer preferences, let the customer select one among available packages, confirm or cancel the choice, and the service may need to invoke third-party services to get, say, up-to-date flight or hotel information. By *safe*, we mean that, in principle, the involved parties should always be able either to complete the interaction or to recover from errors that prevent its completion, like when a time-out expires or when one of the third-party services unexpectedly abandon the conversation because its server is overloaded. In the area of transactions, compensation mechanisms have to do with the programming of suitable counter-actions that are installed after a certain activity has been executed to compensate for its effects in case the rest of the interaction cannot be completed successfully. Of course, it is often the case that the previous actions cannot be simply undone (e.g., a sent message cannot disappear, booking cancellation can require some fees) hence full recovery is simply not possible. In the case of sessions, we shall consider a simple built-in mechanism for the graceful closure of nested sessions upon the abandon of a peer.

The outcome of the above consideration language was a new calculus, called CaSPiS (*Calculus of Sessions and Pipelines*) [8], which is the main objective of this tutorial.

1.3 Related Work

CaSPiS has been developed inside the SENSORIA project [57], as part of a larger research effort aimed to develop *core calculi for SOC* at three different levels of abstraction: (i) the service middleware level (close to current networking technologies to be directly implementable, but sufficiently expressive to support service oriented applications), (ii) the service description level (favouring more abstract formalisation of basic concepts such as service definition, invocation, instantiation, and communication), and (iii) the service composition level (with mechanisms for the modelling and analysis of qualitative and quantitative aspects of multiparty service compositions).

At the middleware level we find, e.g., the *signal calculus* (SC) [28]: it is based on a flexible and dynamical reconfigurable network of components communicating via the publish-subscribe message delivery paradigm. Sessions and message correlation are supported through a type system [29]. This calculus revealed easily implementable (in terms of a Java library) as well as expressive enough to support a high-level graphical programming environment.

At the service composition level we find, e.g., λreq [3] and *concurrent constraint pi-calculus* (cc-pi) [16]. The former has been exploited to support the development of techniques for the analysis of service compositions (such as statical analysis of the access to protected resources) within the so-called "call-by-contract" paradigm, while the latter integrates name handling features with constraint semirings to deal more effectively with quantitative aspects of negotiations (such as the so-called service level agreement).

CaSPiS lies at the service description level, where several other interesting proposals are also present, which can roughly be divided in two families: correlation-based and session-based.

The first group comprises COWS [42] (based on message-passing and stateless components) and SOCK [17] (based on shared data spaces and stateful components). The former can be seen as an extension of the pi-calculus with correlation-based communication mechanism and primitives for activity cancellation and preservation, while the latter is closer to WS standards like BPEL and it includes an explicit modelling of processes obtained as service instantiations, process memory, etc..

The second group comprises the so-called SCC-family of calculi [7, 8, 12, 20, 39], spawned by a first proposal of a basic calculus with nested session, the *Service Centred Calculus* SCC, later enriched and refined with different mechanisms for inter-session communication, like *data streaming* [39], *context-sensitive message passing* [20], *locations* and *dynamic multiparty sessions* [12], and *pipelines* [8].

While the above calculi are closer to the orchestration perspective, the *global calculus* [21] is closer to the choreography perspective and allows for static multiparty sessions, where session identifiers are modelled just as pi-calculus channel names (freshly created and distributed to participants during the initialisation phase of the service protocol). In [6] multiparty sessions are considered, but they are required to include one master endpoint and one or more slave endpoints, and direct communication is allowed only between the master and any slave.

It is important to remark that communication mechanisms are somehow orthogonal to sessions. In fact, while CCS-like communication [46] is the obvious choice

when only two-party sessions are considered, in the presence of multiparty sessions a more natural and more sophisticated alternative would be some variant of multicast (like broadcast [27] or CSP-like interaction [33], or even some combination of different policies [11]).

Behavioural type systems can also play a crucial measure for evaluating the various proposals, because they offer a mean to establish the compatibility of peers [1, 6, 15, 21, 26, 30, 34, 35, 36, 39, 44]. In this sense, it is interesting to relate behavioural types and the language independent approach based on contracts [9, 22] along the ideas in [40]. More generally, there are some interesting analogies between the way in which behavioural types resemble orchestration mechanisms and contracts resemble choreography descriptions.

1.4 Structure of the Paper

Section 2 gives some background on the basics mathematical ingredients of process calculi, like labelled transition systems, operational semantics, structural congruence, reduction systems, bisimilarity equivalences. We illustrate such concepts by simple and detailed presentation of the main sources of inspiration for CaSPiS. Step by step, we go from the basic interaction primitives of CCS, to the more advanced name handling features of the pi-calculus, to the use of explicit sessions and to the orchestration primitives of Orc. Section 3 introduces the main principles of CaSPiS, its syntax and reduction semantics and some modelling examples. Section 4 relates CaSPiS with other well-known formalisms by presenting several intuitive encoding. Some concluding remarks are in Section 5.

2 Setting the Context on Interactive and Orchestrated Systems

2.1 CCS, Labelled Transition Systems and SOS Rules

An elementary *action* of a system represents the atomic (i.e., that cannot be interrupted at the given level of granularity) abstract step of a computation that is performed by a system to move from one state to the other.

Ordinary computational models like Turing machines, register machines, several kinds of automata, the lambda-calculus and many imperative programming languages all rely on basic activities like reading from or writing on some kind of (passive) storage device or invoking a procedure with actual parameters.

Milner's *Calculus of Communicating Systems* [46] (CCS) introduced a model whose basic activities rely on some sort of handshake between two autonomous *processes*. Hence, in the case of concurrent systems, actions represent activities such as sending a message and receiving a message, exposing some alternatives and picking one alternative, producing a resource and consuming a resource, etc. On the one hand, when studying one process in separation from the others it is important to observe the kind of handshake it is willing to perform with other processes. On the other hand, when an handshake is performed between two entities, it constitutes a special *silent* action that has no further interaction capability.

To convince yourself about the ease of CCS in modelling concurrent systems and communication protocols, try writing down the solution to the puzzle below, adapted from [54], using first your favourite formalism, and then, after having learnt CCS basics, using CCS processes for modelling the various interacting entities (the light, the special room, and the strategies followed by humans). We shall show later some bits of the solution for the case where the light is initially on.

Exercise 1. 50 young, bright computer scientists are kept in Bertinoro until all exams will be completed, each locked in her/his own room. Their chance to be released is as follows: from time to time, one of them will be carried in a special room (in no particular order, possibly multiple times consecutively, but with a fair schedule to avoid infinite wait) and then back to her/his room. The special room is completely empty except for a switch that can turn the light either on or off (the light is not visible from outside and cannot be broken). At any time, if one of them truthfully asserts that all of them have already entered the special room at least once, then they all pass the exam and are released, but if she/he is wrong, then the chance ends and they will never pass the exam. Before the challenge starts, they have the possibility to discuss together some "protocol" to follow. Can you find a winning strategy when the initial state of the light in the room is known? And if it is not?

In CCS, we assume given a set A of activities, ranged by a, and let $\overline{A} \triangleq \{\, \overline{a} \mid a \in A \,\}$ be the set of co-activities (disjoint from A), with $\overline{\overline{a}} = a$. The set of CCS *labels* is $\mathcal{L} \triangleq A \cup \overline{A}$, ranged by λ, and the set of CCS *actions* is $\mathrm{Act} \triangleq \mathcal{L} \cup \{\, \tau \,\}$, ranged by α, where τ is the special *silent action*. Then, a CCS *processes* P is composed via a number of primitives, that we sketch below in an incremental way. Though the syntax may slightly vary in the literature, we let CCS processes generated by the grammar:

$$P \quad ::= \quad \textstyle\sum_{i\in I} \alpha_i.P_i \quad | \quad P[\phi] \quad | \quad P_1 | P_2 \quad | \quad (\nu a)P \quad | \quad X \quad | \quad \mathrm{rec}\, X.\, P$$

The meaning of each such process is given by a suitable *Labelled Transition System* (LTS) defined by structural induction on the syntax of the process, following Plotkin's *Structural Operational Semantics* (SOS) scheme [51, 52, 53].

We recall that an LTS $T = (S, L, \rightarrow)$ consists of: a set S of *states*, a set L of *labels*, and a *transition relation* $\rightarrow \subseteq S \times L \times S$. Sometimes a distinguished initial state $s_0 \in S$ is also considered. As usual, we shall write $s \xrightarrow{\lambda} s'$ instead of $(s, \lambda, s') \in \rightarrow$, with the meaning that there is a transition leading from state s to state s' and exposing label λ. The label gives some abstract information about the nature of the evolution. For a given label λ we denote by $\xrightarrow{\lambda}$ the binary relation $\{\, (s, s') \mid s \xrightarrow{\lambda} s' \,\} \subseteq S \times S$.

Formally, in the case of CCS, the states of the LTS are CCS processes, the set of labels is Act, and the transition relation is the least one satisfying all SOS inference rules. When a particular process P is considered, then the initial state s_0 of its LTS is P itself and the LTS can be restricted just to the states reachable from P (after any number of transitions). The elegance of SOS relies on the fact that few inference rules define the LTS of any process that can ever be specified. Moreover, SOS rules allow for proofs by structural induction, where the interaction of complex systems is defined in terms of (the behaviour of) their components and proofs by rule induction, where a property can

be proved to hold true for the whole LTS if whenever it holds for the premises of each rule, it holds also for the conclusions.

The simplest process is the *inactive* process, written $\mathbf{0}$ and called *nil*: it is not capable of performing any action. Trailing $\mathbf{0}$s are often omitted. No inference rule is needed for $\mathbf{0}$. *Action prefix*, written $\alpha.P$, prefixes a process P by an action α: the process $\alpha.P$ can perform α and then behave as P. The inference rule for action prefix is the axiom:

$$(\text{ACT})\ \frac{}{\alpha.P \xrightarrow{\alpha} P}$$

Non-deterministic choice, written $P_1 + P_2$, composes two processes in mutual exclusion: process $P_1 + P_2$ can behave as either P_1 or P_2. The inference rules for choice are:

$$(\text{LSUM})\ \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 + P_2 \xrightarrow{\alpha} P_1'} \qquad (\text{RSUM})\ \frac{P_2 \xrightarrow{\alpha} P_2'}{P_1 + P_2 \xrightarrow{\alpha} P_2'}$$

Sometimes *guarded summation* $\sum_{i\in I} \alpha_i.P_i$ is preferred to choice, prefix (single sum) and nil (empty sum). The corresponding inference rule is:

$$(\text{ACT})\ \frac{j \in I}{\sum_{i\in I} \alpha_i.P_i \xrightarrow{\alpha_j} P_j}$$

Renaming, written $P[\phi]$, renames any action α performed by P to $\phi(\alpha)$, where $\phi : \text{Act} \to \text{Act}$ is any renaming function such that $\phi(\overline{\lambda}) = \overline{\phi(\lambda)}$ and $\phi(\tau) = \tau$. The corresponding inference rule is:

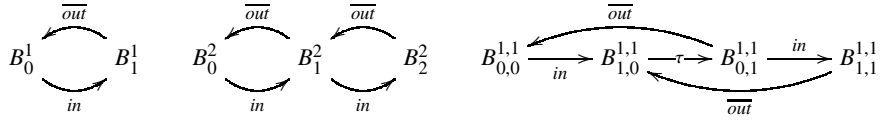
$$(\text{REN})\ \frac{P \xrightarrow{a} P'}{P[\phi] \xrightarrow{\phi(a)} P'[\phi]}$$

Parallel composition, written $P_1 \,|\, P_2$, composes two processes in parallel: P_1 and P_2 evolve autonomously by interleaving their actions, but with the possibility to handshake on complementary actions, in which case $P_1 \,|\, P_2$ performs a τ action. The corresponding inference rules are:

$$(\text{LPAR})\ \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \,|\, P_2 \xrightarrow{\alpha} P_1' \,|\, P_2} \qquad (\text{RPAR})\ \frac{P_2 \xrightarrow{\alpha} P_2'}{P_1 \,|\, P_2 \xrightarrow{\alpha} P_1 \,|\, P_2'} \qquad (\text{COMM})\ \frac{P_1 \xrightarrow{\lambda} P_1' \quad P_2 \xrightarrow{\overline{\lambda}} P_2'}{P_1 | P_2 \xrightarrow{\tau} P_1' \,|\, P_2'}$$

Restriction, usually written $P\backslash a$, but here written in pi-calculus style as $(\nu a)P$, restricts the scope of activity a to process P: the process $(\nu a)P$ is allowed to perform neither action a nor $\overline{a}$; however, if P comprises two parallel processes P_1 and P_2 that can perform a and $\overline{a}$, respectively, then they can still handshake on a "under" the restriction. As usual, we abbreviate $(\nu a_1)(\nu a_2)P$ by $(\nu a_1, a_2)P$ (and similarly for three or more consecutive restrictions). The corresponding inference rule is:

$$(\text{RES})\ \frac{P \xrightarrow{\alpha} P' \quad \alpha \notin \{\, a, \overline{a} \,\}}{(\nu a)P \xrightarrow{\alpha} (\nu a)P'}$$



(a) LTS for B_0^1 (b) LTS for B_0^2 (c) LTS for $B_{0,0}^{1,1}$

Fig. 1. Labelled transition systems associated with some simple buffers

It should be obvious that the above operators can define only finite behaviours. There are several ways to introduce some form of interation and recursion. *Replication*, written $!P$ or also $*P$ accounts for making an unlimited number of copies of P available. Sometimes it is restricted to some guarded form, like $!\alpha.P$ or $!\sum_{i\in I}\alpha_i.P_i$. The usual inference rule for replication is:

$$(\textsc{rep})\ \frac{P\,|\,!P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

However, (rep) has a couple of drawbacks: 1) it makes the transition relation not *image-finite*, i.e. there are processes P that can reach infinitely many syntactically different processes by performing the same action α, 2) it disallows proofs by structural induction, which is maybe a minor issue. If needed, rule (rep) can be safely replaced by the following two rules, that account for the possibility of one copy of P to evolve alone, or for two copies of P to handshake:

$$(\textsc{rep}1)\ \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'\,|\,!P} \qquad (\textsc{rep}2)\ \frac{P \xrightarrow{\lambda} P_1 \quad P \xrightarrow{\overline{\lambda}} P_2}{!P \xrightarrow{\alpha} P_1\,|\,P_2\,|\,!P}$$

A more flexible alternative to replication is given by the *recursion operator* $\mathrm{rec}\,X.\,P$, where X can appear as a process variable in P. The corresponding rule is:

$$(\textsc{rec})\ \frac{P\{\mathrm{rec}\,X.\,P/X\} \xrightarrow{\alpha} P'}{\mathrm{rec}\,X.\,P \xrightarrow{\alpha} P'}$$

where $\{t/x\}$ stands for the substitution of x by t. Alternatively, one can assume a set of *mutually recursive definitions* $\Delta = \{\, A_i \triangleq P_i \,\}_i$ is available, that defines suitable constants A_i. The corresponding rule is:

$$(\textsc{def})\ \frac{A_i \triangleq P_i \in \Delta \quad P_i \xrightarrow{\alpha} P'}{A_i \xrightarrow{\alpha} P'}$$

To acquire some confidence with the notation, let us consider a classical and simple example from CCS textbooks, namely the modelling of buffers with limited capacities.

Example 1. A process B^n modelling an initially empty buffer of capacity n can be defined by letting:

$$\begin{aligned} B_0^n &\triangleq in.B_1^n \\ B_i^n &\triangleq in.B_{i+1}^n + \overline{out}.B_{i-1}^n \qquad (0 < i < n) \\ B_n^n &\triangleq \overline{out}.B_{n-1}^n \end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\overline{out}.B_0^1 \xrightarrow{\overline{out}} B_0^1}(\text{ACT})}{B_1^1 \xrightarrow{\overline{out}} B_0^1}(\text{DEF})}{B_1^1[\phi_1] \xrightarrow{\overline{a}} B_0^1[\phi_1]}(\text{REN}) \quad \dfrac{\dfrac{\dfrac{}{in.B_1^1 \xrightarrow{in} B_1^1}(\text{ACT})}{B_0^1 \xrightarrow{in} B_1^1}(\text{DEF})}{B_0^1[\phi_2] \xrightarrow{a} B_1^1[\phi_2]}(\text{REN})}{B_1^1[\phi_1] | B_0^1[\phi_2] \xrightarrow{\tau} B_0^1[\phi_1] | B_1^1[\phi_2]}(\text{COMM})}{(\nu a)\Big(B_1^1[\phi_1] | B_0^1[\phi_2] \Big) \xrightarrow{\tau} (\nu a)\Big(B_0^1[\phi_1] | B_1^1[\phi_2] \Big)}(\text{RES})$$

Fig. 2. Proof of transition $B_{1,0}^{1,1} \xrightarrow{\tau} B_{0,1}^{1,1}$

taking $B^n \triangleq B_0^n$. The LTS for B^1 is in Fig. 1(a) and for B^2 in Fig. 1(b). A process P put in parallel with B^n can handshake by performing actions $\overline{in}$ and out. If renaming ϕ_1 maps out to a and renaming ϕ_2 maps in to a, then two buffers B_0^1 could be composed in series by writing the process $(\nu a)\Big(B^1[\phi_1] | B^1[\phi_2] \Big)$. The corresponding LTS is illustrated in Fig. 1(c), where we write $B_{i,j}^{n,k} = (\nu a)\Big(B_i^n[\phi_1] | B_j^k[\phi_2] \Big)$ for brevity. Figure 2 shows the proof of transition $B_{1,0}^{1,1} \xrightarrow{\tau} B_{0,1}^{1,1}$.

Exercise 2. Draw the LTS for the processes $B^1 | B^1$ and $B_{0,0}^{2,2}$.

Coming back to the puzzle from Exercise 1, the light could be modelled as a buffer of capacity one, where action in corresponds to "switch the light off" (it is initially on) and action out to "switch the light off". Then the scientists could agree to use the light as a counter: 49 of them will switch the light on only the first time they enter the room and find it off, while one distinguished scientist will count the number of times that she/he finds the light on (and will switch it off). Since the light is initially on, the count can start only after the distinguished scientist has switched the light off for the first time. A first solution is therefore:

$$\begin{aligned}
Bertinoro &\triangleq (\nu\, swOff, swOn)(\ LightON | C_0 | S | \cdots | S\ ) \\
LightON &\triangleq swOff.LightOFF \\
LightOFF &\triangleq swOn.LightON \\
C_i &\triangleq \overline{swOff}.C_{i+1} + \overline{swOn}.\overline{swOff}.C_i \quad (0 \le i < 50) \\
C_{50} &\triangleq \overline{freeAll}.\mathbf{0} \\
S &\triangleq \overline{swOn}.\mathbf{0} + \overline{swOff}.\overline{swOn}.S
\end{aligned}$$

where C_0 models the counting scientist and S any other scientist. Note that a scientist can wish to perform two consecutive interactions with the light just to leave its state unchanged. Unfortunately, this way there is no guarantee that consecutive interactions like $\overline{swOff}.\overline{swOn}$ are executed atomically, therefore it is better to modify the protocols in order to constrain the scientists to access the light in mutual exclusion. This can be done by modelling the special room as a one-capacity buffer, where action in corresponds to "enter the room" and action out to "leave the room": only after the room has been

entered it is possible to interact with the light. To make the model more faithful, we also introduce the process for representing a "waiting scientist", i.e. a scientist who does not need to interact any more with the light but can keep entering and leaving the room.

$$
\begin{aligned}
\mathit{Bertinoro} &\triangleq (\nu\, \mathit{in}, \mathit{out}, \mathit{swOff}, \mathit{swOn})\,(\ \mathit{Room} \mid \mathit{LightON} \mid C_0 \mid S \mid \cdots \mid S\ ) \\
\mathit{Room} &\triangleq B^1 \\
\mathit{LightON} &\triangleq \mathit{swOff}.\mathit{LightOFF} \\
\mathit{LightOFF} &\triangleq \mathit{swOn}.\mathit{LightON} \\
C_i &\triangleq \overline{\mathit{in}}.\left(\overline{\mathit{swOff}}.\mathit{out}.C_{i+1} + \overline{\mathit{swOn}}.\overline{\mathit{swOff}}.\mathit{out}.C_i\right) \quad (0 \le i < 50) \\
C_{50} &\triangleq \overline{\mathit{freeAll}}.\mathbf{0} \\
S &\triangleq \overline{\mathit{in}}.\left(\overline{\mathit{swOn}}.\mathit{out}.\mathit{WS} + \overline{\mathit{swOff}}.\overline{\mathit{swOn}}.\mathit{out}.S\right) \\
\mathit{WS} &\triangleq \overline{\mathit{in}}.\tau.\mathit{out}.\mathit{WS}
\end{aligned}
$$

We leave to the reader finding a solution for the case where the initial state of the light is not known in advance, e.g. when the light is modelled as the process $\tau.\mathit{LightON} + \tau.\mathit{LightOFF}$.

A vast literature on CCS has established different criteria for when two processes should be considered as "equivalent". Without entering into the details, we mention two of the most widely used notion of equivalence, namely *strong bisimilarity* and *weak bisimilarity*. Contrary to trace equivalence, bisimilarities can take into account the branching structure of the transition systems, i.e. the points where choices are made. We refer the interested reader to [31, 32] for a wider range of options.

Definition 1 (Strong Bisimilarity). *A binary relation $\mathcal{R}$ over processes is a* strong bisimulation *iff whenever $(P, Q) \in \mathcal{R}$ then for each $\alpha \in \mathrm{Act}$:*

- *if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\alpha} Q'$ for some Q' such that $(P', Q') \in \mathcal{R}$*
- *if $Q \xrightarrow{\alpha} Q'$ then $P \xrightarrow{\alpha} P'$ for some P' such that $(P', Q') \in \mathcal{R}$.*

Two processes P and Q are strongly bisimilar*, written $P \sim Q$, iff there exists a strong bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$, i.e. $\sim \triangleq \bigcup\{\, \mathcal{R} \mid \mathcal{R}$ is a strong bisimulation $\}$.*

Strong bisimilarity is an equivalence relation and it is a congruence w.r.t. all CCS operators, meaning that if we replace any subterm $P'$of P with a strongly bisimilar term Q' then the result is guaranteed to be strongly bisimilar to P. Notably, $\sim$ admits a logical characterisation in terms of Hennessy-Milner logic, a modal logic of actions for the analysis and verification of reactive systems [2].

Exercise 3. Prove that $B^2 \sim B^1 \mid B^1$ and $B^2 \not\sim B^{1,1}_{0,0}$.

Strong bisimilarity is coarser than LTS isomorphism, but it still distinguishes too many processes that have essentially the same behaviour. In particular, it is often the case that some additional silent transitions may arise or not depending on different attitudes to modelling the same system. Weak bisimilarity, denoted by $\approx$, relaxes the notion of strong equivalence by allowing to simulate a move also performing additional

silent transitions beforehand and afterwards: roughly, letting $\rightarrow^*$ denote the reflexive and transitive closure of $\xrightarrow{\tau}$ (i.e. $\rightarrow^*$ is the relation such that $P \rightarrow^* P'$ iff P' is reachable from P via any number of consecutive silent transitions, possibly none), in the weak case a step $P \xrightarrow{\lambda} P'$ can be simulated via a sequence of steps $Q \rightarrow^* \xrightarrow{\lambda} \rightarrow^* Q'$ and silent transitions $P \xrightarrow{\tau} P'$ can be simulated via a possible empty sequence of silent steps $Q \rightarrow^* Q'$. For example, $B^2 \approx B^{1,1}_{0,0}$. Weak bisimilarity is an equivalence relation that includes strong bisimilarity (in the sense that $P \sim Q$ implies $P \approx Q$ for any processes P and Q), but it is not a congruence (because it is not preserved by the choice operator).

2.2 Pi-Calculus, Structural Congruence and Reduction Semantics

CCS is Turing powerful, and it can be used at several level of analysis, as a specification language, as a programming language, as a description language, as a type language, etc. However, when one wants to model interactive systems with dynamic changes in connectivity, or networks where processes can move between physical or virtual locations, then the representation distance is quite increased and the modelling activity can become cumbersome.

Milner, Parrow and Walker's *Calculus of Mobile Processes* [48, 49, 56] (i.e., the π-calculus or also pi-calculus) introduces a key ingredient: the possibility to communicate channel names. This way, a process can acquire new communication links, pass its own private channels to other processes, create fresh channels, and much more. Even if the required extension to CCS syntax is to some extent minimal, it opened a still flourishing research thread. Nowadays, there are many variants of π-calculus (monadic, polyadic, synchronous, asynchronous, with mixed choice, higher-order, to name a few) each with a consolidated theory on its own. To appreciate the key difference w.r.t. CCS, let us consider the following puzzle, adapted from [24].

Exercise 4. 100 young, bright computer scientists are kept awake in Bertinoro until all exams will be completed. Their chance to have some sleep is as follows: first each of them is assigned a different id from 1 to 100 and a different room (assume rooms are also numbered from 1 to 100); then the ids are randomly distributed one per room; each scientist is given the possibility to open 50 rooms of her/his choice and look at the ids contained there; if all scientists are able to find their own id, then they are all given access to the rooms, otherwise (even if only one of them is not able to find her/his own id) they will not be able to sleep until all exams have been given. Each scientist is not allowed to look at the ids found in the rooms by her/his colleagues, and they are not allowed to speak to each other once the procedure is started. Before the challenge starts, they have the possibility to discuss together some "protocol" to follow. Can you find an optimal strategy to let them have some sleep with highest probability?

The best possible strategy leaves almost 1/3 of probability to get some sleep. It is based on a simple protocol, equal for all participants: the first room opened by scientist with id i must be room i; at each stage, if an id k different from her/his own is found in a room, then the next room to be opened is room k. The idea is that rooms and the id they contain define a set of permutation cycles: the strategy is "winning" iff all such cycles have length less than or equal to 50. As there can be at most one cycle of length greater

than 50, the probability to win coincides with the probability that such a long cycle is not present. If modelled in CCS, the protocol should consider 100 different continuations for each room, one for each possible id contained therein. Using pi-calculus instead, the next room to open can be just communicated.

From the point of view of the syntax, the only primitives to be changed are action prefixes. In the following we assume an infinite set of names $\mathcal{N}$, ranged by x, y, z, is available. The action prefixes of the pi-calculus represents either the sending $\overline{x}\langle y\rangle$ of a name y along x, or the receiving $x(y)$ of a name y along x, or the silent action τ. Sometimes the matching prefixes $[x = y]$ and mismatching $[x \neq y]$ are also considered: they represent ordinary test for equality and inequality of names, and can be used to follow different alternatives depending on the received names. The use of mismatch prefixes is discouraged because their presence can violate useful monotonicity properties of processes, like the fact that name-substitution does not decrease action capabilities of a process.

Unfortunately, the inference rules of CCS cannot be smoothly extended to pi-calculus and some additional care and machinery is needed. To see why, consider the straight extensions of inference rules for action prefixes, where in the case of input one simply guesses the name z that will be received:

$$(\text{INP})\ \frac{}{x(y).P \xrightarrow{xz} P\{z/y\}} \qquad (\text{OUT})\ \frac{}{\overline{x}\langle z\rangle.P \xrightarrow{\overline{xz}} P}$$

Now we should decide which actions should be forbidden under restriction. Take the process $(\nu n)P$ and suppose $P \xrightarrow{xz} P'$:

- if $n \notin \{ x, z \}$ then we can let $(\nu n)P \xrightarrow{xz} (\nu n)P'$;
- if $n = x$ then we must forbid the move;
- if $n = z \neq x$ then we must forbid the move, because n is a private name that cannot be received from the outside.

Now suppose $P \xrightarrow{\overline{xz}} P'$:

- if $n \notin \{ x, z \}$ then we can let $(\nu n)P \xrightarrow{\overline{xz}} (\nu n)P'$;
- if $n = x$ then we must forbid the move;
- if $n = z \neq x$ then what? If we forbid the move, then private names cannot be extruded to other processes, which would be a severe limitation. If we allow the move, then we would like to extrude the scope of n only to the processes that handshake on $\overline{xn}$, hence we should have $(\nu n)P \xrightarrow{\overline{xz}} P'$, where the restriction disappears from the target. On the other hand, when handshake is accomplished, we would like to restore the restriction.

The so-called *early operational semantics* solves the problem by introducing different labels for the free output $\overline{x}z$ and the bound output $\overline{x}(z)$ (where the name z is extruded). This in turn have several consequences on the rules for parallel composition: some side conditions are needed in order to avoid that an extruded name captures a free name of a process running in parallel, and two kinds of handshakes are possible, depending on the kind of output that is considered: the handshake between actions xz and

$S + \mathbf{0} \equiv S$ $\quad S_1 + S_2 \equiv S_2 + S_1$ $\quad S_1 + (S_2 + S_3) \equiv (S_1 + S_2) + S_3$ $\quad [a = a]\pi.P \equiv \pi.P$

$P \mid \mathbf{0} \equiv P$ $\quad P_1 \mid P_2 \equiv P_2 \mid P_1$ $\quad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$ $\quad P \equiv P \mid !P$

$(\nu a)\mathbf{0} \equiv \mathbf{0}$ $\quad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$ $\quad P \mid (\nu a)Q \equiv (\nu a)(P \mid Q)$ if $a \notin \mathrm{fn}(P)$

Fig. 3. Structural congruence laws for the pi-calculus

$\overline{x}z$ is the ordinary one (as in CCS); the handshake between actions xz and $\overline{x}(z)$ move the restriction (νz) on top of the parallel composition. In general, it emerges the necessity to take into account which are the free names of a process (denoted by $\mathrm{fn}(P)$) and which are its bound names (denoted by $\mathrm{bn}(P)$). In the case of pi-calculus, the only binders are input prefix and restriction, i.e. in both $x(y).P$ and $(\nu y)P$ the name y is bound and its scope is restricted to P. There are further consequences also on the definition of strong bisimilarity, when the actions to be simulated depend on the free names of a process. These caveats make the formal presentation of pi-calculus semantics more complicated and less intuitive than CCS one, when encountered for the first time.

For the above reasons, a different style of presentation is sometimes preferred for pi-calculus (and for many other calculi with name-handling features). It has two main ingredients: a *structural congruence* relation, used to write processes in some canonical form, easier to manipulate; a *reduction* relation that represents only completed interactions, roughly the τ moves.

Let us consider the following syntax for pi-calculus processes:

$$\begin{array}{rrcl} \text{(Processes)} & P & ::= & S \mid P_1 \mid P_2 \mid (\nu x)P \mid !P \\ \text{(Sums)} & S & ::= & \mathbf{0} \mid \pi.P \mid S_1 + S_2 \\ \text{(Prefixes)} & \pi & ::= & \overline{x}\langle y\rangle \mid x(y) \mid \tau \mid [x = y]\pi \end{array}$$

The structural congruence $\equiv$ of pi-calculus is the least congruence relation that satisfies the equalities in Fig. 3 plus alpha-conversion of bound names.[1] The structural congruence allows one to rearrange the syntax of processes so that any two possible interacting entities can be put side by side (in parallel composition). Note in particular that: the order in which we compose processes in sums should not matter; the order in which we compose processes in parallel should not matter; the order in which we restrict names should not matter. Moreover, the scope extrusion law (the rightmost equality in the bottom row of Fig. 3) can broaden the scope of a restricted name before it is communicated. It is not difficult to see that each π-calculus process P can be put in a canonical form like $P \equiv (\nu x_1)...(\nu x_k)(S_1 \mid ... \mid S_n \mid !P_1 \mid ... \mid !P_m)$ for some suitable names $x_1, ..., x_k$, sums $S_1, ..., S_n$, and processes $P_1, ..., P_m$ in canonical forms. Thus, all interactions can now be expressed by considering only a small number of reductions over canonical forms. Essentially there are two rules for basic reductions:

$$\text{(Rtau)}\ \frac{}{\tau.P + S \xrightarrow{\tau} P} \qquad \text{(Rcom)}\ \frac{}{(x(y).P_1 + S_1) \mid (\overline{x}\langle z\rangle.P_2 + S_2) \xrightarrow{\tau} P_1\{z/y\} \mid P_2}$$

[1] The laws for alpha-conversion allow for the arbitrary renaming of bound names, but avoiding clashes with free names. In the case of pi-calculus, alpha-conversion means that for any process P and any names x, y, z with $z \notin \mathrm{fn}(P)$ we have $x(y).P \equiv x(z).(P\{z/y\})$ and $(\nu y).P \equiv (\nu z)(P\{z/y\})$.